WORKING PAPER
ALFRED P. SLOAN SCHOOL OF MANAGEMENT

NEC: Standardization Strategy for a Distributed "Software Factory" Structure

Michael A. Cusumano

November 1987 WP #1954-87

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139

NEC: Standardization Strategy for a Distributed "Software Factory" Structure

Michael A. Cusumano

November 1987 WP #1954-87

Michael A. Cusumano
Sloan School of Management

8 November 1987
Software Project Paper #5

NEC: STANDARDIZATION STRATEGY FOR A DISTRIBUTED "SOFTWARE FACTORY" STRUCTURE

INTRODUCTION

This paper is part of a larger study examining the question of whether or not companies are choosing to manage a complex engineering activity such as large-scale software development with a range of strategic considerations and organizational as well as technological approaches that corresponds to the spectrum usually associated with "hard" manufacturing, i.e. job shops, batch organizations, and factories exhibiting various degrees of flexibility in product mixes and technologies. The research project includes the proposal of technology and policy criteria defining what a factory environment for software might look like; a survey of major software facilities in the U.S. and Japan to determine where firms stand in relation to these criteria; and detailed case studies examining the technology and policy implementation process followed at firms identified as being close to the factory model.[1]

There are several interrelated conclusions: (1) This spectrum, including "factory" approaches, is clearly observable in the sample of software facilities in the U.S. and Japan. (2) There appears to be nothing inherent in software as a technology that prevents some firms from creating strategies and organizational structures to manage product and process development more effectively, even with a relatively new and complex technology such as software. (3) The basic technological infrastructures to aid software process management are not significantly different between Japanese and U.S. firms. (4) But, Japanese firms -- led by the NEC group and Toshiba, and followed by Hitachi and Fujitsu--

are significantly ahead of most U.S. competitors in implementing "flexible factory" type of strategies focused on reusing standardized components (modules of code) and then customizing end products.

This paper extends the survey approach to analyze what is probably the most difficult aspect of the software factory -- the implementation process and the benefits or disadvantages this environment might offer in operation. The NEC case is significant for the following reasons. One, it adopted a software factory strategy around 1976, but, unlike Hitachi or Toshiba, did not attempt to construct a single major facility and use this as the main site for developing factory tools and methods. Rather, top management at NEC used a centralized committee and then laboratory organization to direct the development and dissemination of factory-like software tools, standards, procedures, and control methods among all the divisions and subsidiaries writing programs.

This approach appears to have reflected NEC's organization around product divisions rather than factories, as in the case of Hitachi and, to some extent, Toshiba. It has also led to a remarkably high level of standardization and consensus regarding software development despite NEC's production of various types of software, including telephone switching and transmissions systems, computer operating systems, and business applications programs.

The organization of this paper is as follows: Part I presents an overview of NEC and its development of the computer business, and then discusses in detail top management's factory strategy for software development and organization implementation in terms of divisional structures, committee programs, R&D organization, corporate-wide programs, and subsidiaries. Part II focuses on the tools, methodologies, and management-control systems developed to support the factory strategy. The conclusion makes some preliminary comparisons of NEC to other Japanese and U.S. firms, as well as offers several

suggestions about the general lessons regarding technology management that can be drawn from this study.

I. SOFTWARE-DEVELOPMENT STRATEGY AND ORGANIZATION

Corporate History

NEC was founded in 1899 as a joint venture between two independent Japanese businessmen and AT&T's Western Electric subsidiary, to manufacture and import telephone equipment. It later became an independent company affiliated with Japan's Sumitomo group and, over time, expanded into a variety of electronic and electric components and products. As a $16.8 billion dollar multinational corporation in 1987 (including consolidated subsidiaries), NEC revenues came from a range of product divisions. Each was a profit center organized within nine operating groups covering computers and industrial electronic systems (41% of 1986 sales), communications (switching and transmission systems as well as radio and microwave) products and services (29%), semiconductors and other electron devices (17%), and home electronics (8%).[2]

NEC OPERATING AND MARKETING GROUPS, 1986

Operating Groups	Marketing Groups
Research and Development	NT&T Sales
Production Engineering Development	Government Sales
Switching	Domestic Sales
Transmission and Terminals	International Operations
Radio	Advertising
Information Processing	
Electron Devices	
Home Electronics	
Special Projects	

Source: NEC Corporation, "This is NEC, 1986," p. 9.

Like Hitachi, Toshiba, and Fujitsu, NEC has a long history of computer and software development, dating back to the 1950s. NEC was one of the earliest producers of a transistorized computer in the world, completing a scientific-use model in 1958. To catch up to the United States in commercial computer technology, NEC entered into a licensing arrangement with Honeywell in 1962, through which it manufactured Honeywell computers for sale under the NEC label in Japan and upgraded its hardware and software design technology. While this technical relationship continued until 1979 and the two companies still cooperate in marketing, the technology transfer situation reversed in the 1970s as NEC started supplying mainframe hardware to its American partner.[3] By 1986, NEC was clearly one of the world's largest computer and software manufacturers, ranking, among Japanese firms, first in software, microcomputer, and data communications sales, and second in mainframe sales, behind Fujitsu.[4]

Early Learning and Technology Transfer

NEC's history of computer development is a combination of independent, in-house efforts, beginning in the mid-1950s, with hardware design and some software technology acquired from Honeywell in the 1960s. The key figure in software development from NEC's very earliest efforts has been Mizuno Yukio, who entered the company in 1953 after graduating from the Tokyo Institute of Technology, where he also received an engineering doctorate in 1960. In 1987 Mizuno was a senior vice-president in charge of software strategy.

The first programming efforts in NEC, according to Mizuno, were in 1955-1956, when he and several other engineers built an analog computer and then a message and accounting data sorter for a non-stored program machine -- a

computer that was, essentially "hard-wired." Their next project required programs for storage in memory -- the NEAC 2201, a transistorized computer NEC completed in 1958, and the 2203, an enlarged version introduced for commercial sale in 1959. The NEAC 2201 and 2203 prompted NEC managers to establish a programming section in 1959, with 26 engineers, including Mizuno and a leading Japanese computer experts at that time, Okazaki Bunji, who had joined NEC after building one of Japan's first electronic computers for Fuji Film. One of their first accomplishments was the completion of Japan's first compiler.[5]

Several Honeywell machines supplanted the NEAC models in 1962-1963. These gave way after 1965 to the NEAC 2200 series, which contained repackaged versions of three small and medium-size mainframes Honeywell had designed to compete with the IBM 1400, as well as several larger machines NEC designed independently to compete more directly with the IBM 360 family. The 2200 series remained as NEC's basic product line until NEC introduced the ACOS series beginning in 1974, to compete with IBM's 370 family of mainframes.[6]

Despite the technical arrangement with Honeywell, a decision in 1965 to build a large-scale computer for the Japanese market -- later dubbed the NEAC 2200-500 -- led NEC not only to use integrated circuits (ICs), printed circuit boards, and large-capacity core memory for the first time, but also to develop independently an advanced operating system capable of on-line processing and time sharing similar to the Multics system developed at the Massachusetts Institute of Technology.[7] The hardware and software tasks presented major technical and organizational challenges to NEC engineers, who had never tackled such extensive projects, and did much to advance the technical skills of company personnel in both areas.

The 2200-500 machine was based on a hardware design project completed in 1964 at NEC's central research laboratories. To develop the central processing unit into a commercial model, in 1964 NEC management established a computer factory at Fuchu, in the outskirts of Tokyo, and in 1965 a computer development group with about 30 engineers, including Mizuno as head of the programming section. Honeywell managers tried to dissuade NEC from taking on this project, arguing that its models were adequate and that the software for the proposed 500 would be too difficult to write. Much to the surprise of Honeywell executives, NEC completed the machine and operating system on time in 1968 -- delivering the 2200-500 to Osaka University and receiving an award from the Nikkan Kogyo Newspaper for having developed Japan's first IC-based computer. Honeywell thereafter brought NEC formally into its product planning system and entered into a cooperative development program with NEC for future machines.[8]

The operating system for the 2200-500, named MODE IV, NEC patterned directly after the IBM 360. NEC provided about 90% of the engineers working on the system; Honeywell contributed some 10%, who came to Japan and worked under Mizuno in Tokyo. Honeywell was quite helpful in teaching NEC managers how to plan for and control such a large software project, although Mizuno was disappointed to find that the Americans' level of technology regarding bug, cost, and productivity estimation was low. The need for improvement in forecasting led NEC managers to collect their own data and devise new planning models during the 1970s, which became the basis for software production control at NEC in the 1980s.[9]

A major step in standardization of control methods as well as of tools and development procedures was the creation of a new computer development group in 1971 and then the decision to build a new operating system for the ACOS

series. This decision led to the separation of basic software (operating systems and related programs) into an independent division in 1974, centered at the Fuchu plant.[10] From 1974, the Fuchu plant, which produced mainframe hardware and operating systems, was referred to within NEC as a "software factory," and became the center of software technology development until the establishment of a separate laboratory for this purpose in 1980. At this point the information processing group contained separate divisions for basic software centered at Fuchu, and, in the Mita section of Tokyo, near NEC headquarters, for large-scale customized software (system engineering projects) and user (business) applications software.[11]

Unlike Hitachi and Fujitsu, NEC's software development tasks did not include trying to be compatible with IBM. NEC thus continued with the non-IBM compatible Honeywell operating system. Mizuno recalls that because RCA in the U.S. and Unidata in Europe had both tried to build IBM-compatible machines and then failed to match IBM's subsequent 370 models, he and other NEC executives decided "there had to be a limit to the compatible" approach. But NEC still had to offer competitive performance with its machines to attract customers. Thus, even though he was head of product planning, in 1974 Mizuno moved from the comforts of NEC headquarters to the Fuchu plant and spent three years directing the new operating system development effort. Because IBM's 370 operating system did not have a strong time-sharing capability and was not well set up to handle data bases, the NEC managers decided to try to add these functions to their new system, as well as a "task-level" virtual machine capability. It turned out that developing such a sophisticated operating system was extraordinarily difficult, although they succeeded, using a high-level language, HPL, a PL/I subset, to write the system, in contrast to IBM, which was still using assembler.[12]

For future product development, two events in 1975 had long-term significance. One was the decision to train programmers in structured programming methods and to design standards and tools based on this methodology for newly written application and system programs.[13] Another was the decision to create subsidiaries specialized in software development, beginning with NEC Software, Ltd., and to recruit software houses to serve as subcontractors.[14] Subsidiaries and affiliated software houses also became members of NEC's Software Parts Company Association (headed by Vice-President Mizuno), in addition to being subject to controls imposed by procurement departments in each division.[15] The Fuchu Software Factory and NEC Software thus became the basis of a geographically distributed, product-centered "software factory" scheme, based around structured programming technology, that Mizuno and another NEC manager, Fujino Kiichi, began promoting more aggressively from 1976.[16]

NEC SOFTWARE ORGANIZATION AND SUBSIDIARIES, ca. 1986-87

Information Processing Group	Main Factories	Software Employees
Basic Software	Fuchu	2,000
System Engineering	Mita	2,500
Other Divisions		
Transmission	Tamagawa	1,500
Switching	Abiko	1,500
Domestic Software Subsidiaries		
NEC Software		2,100
NEC Information Service		900
Nippon Electronics Development		1,000
Nihon Computer Systems		800
NEC Communication Systems		800
NEC Engineering Information Systems		150
Kansai NEC Software		500
Chubu NEC Software		300
Kyushu NEC Software		200
Tohoku NEC Software		100
Hokuriku NEC Software		na
Chugoku NEC Software		na
Hokkaido NEC Software		na
Shikoku NEC Software		na
NEC Security Systems		na
Shizuoka NEC Software		na
Okinawa NEC Software		na
NEC Microcomputer Technology		na
NEC Engineering		na
NEC IC-MicroComputer Systems		na
NEC Telecom Systems		na
NEC Koku-Uchu (Aerospace) Systems		na
NEC Robot Engineering		na
NEC Ocean Engineering		na

Note: All employee figures are estimates by the author. An approximate total for the number of software engineers and programmers in the NEC facilities and subsidiaries noted above is 16,000.

Sources: NEC, Konpyuta sangyo no genjyo to NEC (1985); Joho shori: sofutouea kaisha roku (1985); Kiriu Hiroshi, Sofutouea sangyo no jitsuzo (1986); Nikkei konpyuta, 13 October 1986, p. 89; company data; author estimates.

Software Factory Strategy

NEC's software strategy evolved gradually. Mizuno began discussing publicly his view of the software factory as early as an October 1975 article in a leading Japanese journal on information processing. Then the manager of NEC's Basic Software Development Division, he stressed that the software industry had evolved to where cost and quality mattered to customers, therefore, he felt producers had to move beyond "individual" or "craft" approaches to production, and develop "modern" engineering and manufacturing practices and tools:

> As the use of computers has become more diverse and at higher levels, software -- quantitatively and qualitatively -- has become increasingly important... Thus, software is no longer the individualistic, craft-like product it once was; there is evidence that it has become a modern product with a market value and market distribution. Accordingly, we have reached the point where the development process for software must move beyond the individual or craft stage to a more rational, modern production system. Namely, we have reached the age where there is a strong demand for high quality, maintainable software as well as for software cost reduction and productivity improvement...
>
> In general, consciousness of software as a highly modern and major product is still extremely low among both software producers and users. The result is a tendency to pay too little attention to software inspection or quality assurance, and not to invest enough time and resources in the software development and production process...
>
> It seems we have reached the stage where current methods for software production, which resemble a cottage industry, may be moving toward 'manufacture' (factory-method industry)... [But] the tools used at the present for software development -- paper, pencils, machines, basic language processors -- are extremely primitive... To improve productivity in software development, to produce software of higher quality, it is necessary to go beyond cottage-industry type production methods. This higher level of technology for the software production process can be referred to by the term 'software engineering.' [17]

This 1975 article, published only a few months after System Development Corporation in the U.S. released information describing its "Software Factory" project, elaborated on the concept of a physically distributed software factory system using standardized procedures, tools, and components, with time-sharing

capabilities link several sites, such as Fuchu, Tamachi/Mita, and Shinjuku in Tokyo. Mizuno estimated that, with factory-type standardized methods and tools, productivity should rise from 2 to 5 times.

Several basic tools, concepts, and systems, in Mizuno's view, were necessary to support successful implementation of a software factory system. First, was the development of expandable, versatile high-level languages suitable for system description using structured programming methods. Second, was the use of structured programming techniques such as the top-down approach, abstraction techniques, introduction of segmented and layers structures, reduction of GO TO statements in coding, listing of program modules, or standardization of subroutine input/output interfaces. Third, was standardization of the production process, as had been achieved in other industries, to improve productivity and quality: "In the average production process, through Taylor's methods, standardization of tasks led to dramatic increases in productivity. In the same way, standardization in the software development process of the construction flow and documentation should make it possible to improve productivity. In addition, standardization should help reduce careless mistakes and increase reliability." Fourth, was a system for software development-process control. Mizuno offered a "phase-plan" system, which he claimed would add some visibility to the development process through checkpoints and reviews of standardized documentation for each step (see table). Fifth, Mizuno felt the software factory should have a comprehensive system for quality assurance. In addition, he identified nine elements fundamental to the technological infrastructure of an effective software factory (see tables).[18]

MIZUNO'S INITIAL SOFTWARE FACTORY CONCEPT

METHODOLOGY AND MANAGEMENT

1. High-level structured programming language for system description
2. Disciplined structured programming and top-down modularization techniques
3. Standardization of the development process
4. System for process control
5. System for quality assurance

TECHNOLOGY

1. (High-Level) Language processing
2. Program and data commonality
3. On-line memory and control of program components and information units
4. Simple system for integrating parts of programs being developed separately
5. High-reliability file system
6. High-level virtual memory
7. Suitable development language and program control function
8. "Boot-strap" function (capability of testing software developed in the factory in the actual mode or on the actual machine it will run on, to ease transfer)
9. Tools for debugging, dynamic link functions, project control, and continuous operation.

Source: Mizuno 1975, pp. 837-838.

PHASE-PLAN SYSTEM

Phase 0: PLANNING

Definition: Determination of product based on market needs, development costs, and technical capabilities

Documentation: Planning Specifications

Review: Schedule, Cost

Phase I: BASIC DESIGN

Definition: Analysis of the objective of the planned project and establishment of programming objectives

Documentation: Basic Design Specifications

Review: Performance, Cost

Phase II: DETAILED DESIGN

Definition: Design of detailed functions and structure based on the programming objectives

Documentation: Functional Specifications (Language Specifications)
Logic Specifications
Manual Draft

Review: Cost, Functions

Phase III: IMPLEMENTATION

Definition: Coding and debugging of each component based on the detailed design

Documentation: Programming Completion Document
User Manual

Review: Schedule, Cost, Functions

Phase IV: SYSTEM INTEGRATION

Definition: Test of debugged modules in combination

Documentation: System Integration Test Document

Review: Performance

Phase V: INSPECTION

Definition: Examination of whether the completed system meets the planned specifications

Documentation: Inspection Completion Document
Release Announcement

Phase VI: MAINTENANCE
Definition: Support and improvement of completed system

Documentation: Program Lists
Internal Documents

Source: Mizuno 1975, p. 838.

As they made progress in process standardization, NEC managers gradually changed emphasis to quality control and reusability of code as key ways to improve overall productivity. This shift in thinking can be seen in a 1986 interview, where Mizuno discussed the two components of his strategy for achieving continuous increases in productivity: (1) a rigorous quality control program to prevent errors, which become difficult and expensive to fix after a program is released to the customer; and (2) "accumulation of knowledge" about software engineering and passing this on to others through (a) reusable software modules, (b) reusable program patterns, and (c) bug or failure analysis and the development of better test rules and check points. SEA/I (alternately known as "Software Engineer's Arms" and "System Engineering Architecture") Mizuno considered a "first step" in the development of an automatic programming tool using this "accumulation of knowledge" concept.[19] Thus, building upon process standardization and control as the foundation of a factory approach, Mizuno now explained the software factory as a "concept" for maximizing the "accumulation of experience" and capturing this experience in control systems, tools, and process inputs (reusable code or designs):

> The term 'software factory' does not indicate a physical building. It is a method of producing software, or the tools used in this method, for example a control system. The software factory refers to the integration

of these types of things. It must be understood as a concept.

Another point that should be emphasized is that it is important for a software factory to be a place where systems are introduced that incorporate the experience of people who have made software in the past with new methods or particularly effective techniques. It is an accumulation of knowledge. Even if it doesn't go as far as a knowledge data base, it should be something where there is an accumulation of knowledge.

For example, in coding inspection, a particular group keeps making mistakes in register manipulation. Or they forget to close a table. In a software factory, it would be important to have a system for development-process control that, relying on a history of these mistakes, would prevent them from reoccurring.[20]

During the 1980s, the manager most responsible for implementing NEC's software factory strategy has been Fujino Kiichi, a vice president under Mizuno who headed the Software Product Engineering Laboratory in 1987. Fujino entered NEC in 1968 after working for eleven years in the Production Engineering Laboratory of Waseda University, and had been one of Waseda's first graduates in computer science, receiving a B.S. degree in 1955 and a M.S. degree in 1957, in addition to a doctorate in 1973.[21] In NEC he joined and later headed the Computer Science Research Laboratory, founded under the central R&D labs to study software engineering, and subsequently managed the Basic Software Development Division established in 1974.[22] He became a particularly avid proponent of the distributed, product-centered factory concept.

In a 1984 article, for example, Fujino noted that, in reviewing its communications and computer businesses, NEC managers in the mid-1970s came to the conclusion that five types of programs were "fundamental" to serve the needs of their customers: "basic software for host computers; distributed systems application software; on-line real-time control software; industry-oriented application software; and built-in microcomputer software."[23] They also concluded that, "NEC must meet different customer needs, so it is difficult

and not really desirable to standardize completely."[24] At the same time, Fujino and other NEC managers felt factory-type rationalization was essential "to accommodate the expanding use of computers." This led to the decision to view the factory strategy in terms of "global and distributed software production environments."[25] The "modern software factory" system Fujino attempted to implement consisted of five elements, with a centralized laboratory directing tool and method development for closely linked operating divisions and "satellite offices," including subsidiaries:[26]

FUJINO'S SOFTWARE FACTORY CONCEPT

Central Laboratory for Development of Standardized Tools and Methodologies

Software Work Stations

Communications Utilities

Appropriate Physical Environment

Management Engineering Techniques

a) Productivity Metrics
b) Quality Metrics
c) Cost Control System
d) Management Visibility

A key manager responsible for developing tools and methods was Azuma Motoei, who joined NEC in 1963 and was manager of the software management engineering department in the Software Product Engineering Laboratory before joining Waseda University in 1987. Azuma supported the view that treating software as an R&D-type of activity only would not lead to substantial improvements in productivity or quality. Therefore, he and other NEC managers have tried to develop "factory-like" systems, while recognizing that aspects of software development which are unique make it inappropriate to transfer

manufacturing practices too directly from hardware plants.

Azuma's work focused on devising standards, methods, tools, and environments -- in essence, different factory structures -- for the different types of software NEC produced. For example, he believed that development of large operating systems was more like R&D than manufacturing and required relatively flexible environments. But smaller software products like facsimile or PBX software closely resembled hardware design projects and, he thought, could be controlled more precisely. Business applications programs were often very similar across different projects and, therefore, contained modules that could be reused and offered possibilities for an automated production process. Following the standardization and structured methodology efforts of the 1970s, and the quality control program of the early 1980s, process development at the Software Product Engineering Laboratory during the mid-1980s thus pursued the dual goals of automation and reusability, leading to automated program-generator tools such as SEA/I for applications software and DECA (Detailed Design and Coding Assistant) for systems software.[27]

Implementation of the Factory Strategy

Implementation of a factory strategy to meet diverse product needs began with the founding of the Basic Software Development Division in 1974 and the centralization of operating systems development in the Fuchu Plant. Subsequent efforts centered not on any single facility but were company- and even group-wide, with the participation of corporate staff, operating-group line personnel, and subsidiaries personnel. There was centralization, with most operating-system developed located at Fuchu, applications at Mita, transmission systems at Tamagawa, and switching at the Abiko plant northwest of Tokyo. But, due to lack of physical space and other factors, some groups as well as the work of

subsidiaries and software houses continued to be distributed in different areas. What was perhaps most important, however, is that, in contrast to the System Development Corporation's Software Factory, NEC created permanent centers of development managed by departments and not projects. Work, therefore, flowed steadily through the new facilities, and provided managers with a rationale for continued development of the factory's technological and management systems.

NEC implemented the concept of standardized but distributed development through the use of several sets of common techniques and tools, as well as by linking physically separated sites through communication networks, a process under continual development. File transfers among different sites even made reuse of the same code possible in more than one facility.[28] The following table presents an overview of key projects and dates in the factory-strategy effort:

IMPLEMENTATION OF NEC'S SOFTWARE FACTORY STRATEGY

Year	Project/Designation	Focus
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

Source: Interviews with Fujino Kiichi and Azuma Motoei, 7/28/86

SOFTWARE STRATEGY PROJECT, 1976-1979

The Software Strategy Project was organized in 1976 on a group-wide basis (including in-house divisions and subsidiaries) rather than in the computer group because NEC had large numbers of programmers in other divisions, such as switching systems, as well as in subsidiaries. Mizuno headed the committee overseeing the 3-year project. There were two sub-projects -- one for applications software and one for systems software, the latter directed by Fujino.

The project resembled manufacturing initiatives taken at nearly all large Japanese companies after the 1950s in the explicit focus on, in Fujino's words, "eliminating waste." In practice, this meant collecting and analyzing *performance data, and then developing standardized tools, procedures, and methodologies for all phases of software production -- dealing with customers (requirements analysis), as well as design, programming, documentation, testing, installation, maintenance, and user service and support. The major thrust of the project turned out to be standardization and tool development to support structured programming techniques. Among the systems developed, in Fujino's opinion, most important in facilitating division of labor and cooperation among large numbers of programmers has been STEPS (Standardized Technology and Engineering for Programming Support) for applications software.[29] (More detailed discussion of STEPS and other tool and methodology systems at NEC will follow in a subsequent section.)

SOFTWARE PRODUCT ENGINEERING LABORATORY, EST. 1980

A problem that Azuma identified with the Software Strategy Project was

the lack of permanent manpower staff. Members worked on the tasks assigned when they had a chance, but meetings were infrequent and progress was slow. Mizuno felt at the end of the project that "we had to be more systematic" to raise productivity and quality as well as to improve control.[30] For these reasons, NEC managers decided in 1980 to establish the Software Product Engineering Laboratory to head the software development effort, organizing this as part of NEC's central research laboratories.

Mizuno launched the new laboratory then delegated the post of director to Fujino in 1981. This made Fujino responsible for managing about 30 full-time researchers and another 30 affiliated members. By 1986, the number of researchers had grown to about 170 full-time members (out of about 900 total assigned to the central research labs), including those in a newly-separated laboratory for microcomputer software development. The mission of the lab, in Fujino interpretation, was to link software production technology with management techniques. To accomplish this they organized it into three departments covering software technology, management methods, and interface architectures:

NEC SOFTWARE PRODUCT ENGINEERING LABORATORY

Departments	Areas of Research
Software Engineering	methodologies and tools for software design, production, and maintenance
Software Management Engineering	methodologies and tools for software production and products management
Interface Architecture	network architecture, information portability

Source: Fujino and Azuma interview, 7/28/87.

Funding of tool and method development in the lab came from corporate R&D funds as well as from internal NEC users, rather than from specific projects, as was often the case for software tool development in U.S. companies. Management policy considered research on tools that could be sold with computer hardware as part of product development, and corporate R&D funds assigned to the lab covered their expenses. Tools that would not be sold outside the company as products but had specific in-house applications NEC funded partially by R&D expenses and partially by costs charged to in-house divisions that eventually used the tools.[31]

Transferring technology developed at the lab to divisions was done in two ways. One was to plan and develop tools and methods in conjunction with engineers from line divisions. Another was for lab researchers to develop tools and then teach division employees how to use them.[32] This was done through lectures in large groups and through quality circles. A staff-level Software Education Department assisted the lab in education planning and implementation, especially in areas such as quality control.[33]

In addition to tools and methods, laboratory researchers also worked on identifying optimal "software factory environments." This effort resembled studies done in hardware factories, but took into account the greater "mental effort" required in software development:

> The work environment has been considered to be an important factor for productivity and quality improvement in hardware production. This field of research is called Industrial Engineering or Plant Engineering... The same approach is expected to increase software productivity. However, the mental work utilization rate in software work is greater than in hardware work because of the difficulty of machine utilization in software work. Therefore, not only is the same approach taken, but the effects on programmer mentality must be taken into account.[34]

Particularly important were motion factors, such as the layout of

equipment (desks, chairs, filing cabinets, terminals), which required unnecessary movements; physical factors, including furniture layout or lighting, which contributed to fatigue; and mental factors affecting concentration and motivation, such as noise levels or color coordination of surroundings. Researchers designed work environments both for a "team oriented work style," where "programmers in a team are constrained to develop software through close mutual communication," and an "individual oriented work style," where individuals worked on separate parts of a program independently, and therefore might desire partitioned spaces. Both were considered "work environments for software factories," although the team-oriented approach has proved most common in NEC facilities.[35]

SOFTWARE QUALITY CONTROL (SWQC) PROGRAM, EST. 1981

The origins of NEC's SWQC program date back to 1978, when NEC established a software quality control (QC) study group. There was not at this time much QC training in software and Japanese firms, in Azuma's estimation, were considerably behind the U.S. NEC Chairman Kobayashi proposed they initiate a formal study of software QC, and enlisted as a consultant the help of Professor Moriguchi from Tokyo University, an expert on quality who was knowledgeable about software. Mizuno was the leader of this study group, Fujino the sub-leader, and Azuma the chief secretary. The group remained active for 3 years, and included hardware people as well. As with the original Software Strategy Project, this was not a full-time activity for the committee members and there was no regular staff. Meetings lasted a few hours once or twice a month, although they managed to cover a wide variety of quality-related problems, and began drawing up better guidelines for procedures

required in software development, for example, on how to write documentation properly.[36]

Several conclusions came out of this study. One, according to Fujino, was that there appeared to be a close connection between software quality and software productivity, and that any "revolution" in software productivity would require equally revolutionary improvements in software quality control practices. While similar observations had come in other studies of software quality and productivity, this type of thinking had pervaded NEC's QC activities in hardware, which dated back to 1945 (under U.S. Army guidance). The best strategy as Fujino interpreted it was to "pursue quality in software, and productivity will follow."

Second, the group concluded that software was quite different from hardware in that it required more intellectual activity and "desk work." This led to a decision not to rely on hardware QC experts but to work with several U.S. software quality experts such as Gerald Weinberg to develop a system for quality evaluation and design review. Third, they felt it was impossible simply to "inspect out bugs," and that quality had to be "built in at each phase of development," with department-level managers fully involved in any QC effort, as well as lower-level employees.[37]

Fourth, was the observation that "human factors" seemed to be most important in achieving innovations in software development, determined through surveys of in-house software projects (see table). While they believed NEC had to develop automation techniques, it clearly was not possible to automate all phases of software development. Therefore, since human "motivation [should come] before automation," they decided the laboratory and other tool-development groups should develop automation tools, while a quality control program focused on motivation, teamwork methodologies, and other "human

factors."[38] This philosophy was clearly reflected in a 1983 article Mizuno published in **Computer**:

> Software projects tend to overemphasize technical aspects. After all, people organized into teams do the work. For projects to proceed smoothly, the human factor must receive adequate consideration. The capability of those involved in software work varies remarkably. Many claim that the relative abilities of programmers can vary by as much as 30 to one... It will become ever more vital to structure projects so that software quality and productivity do not directly reflect these differences. Training can develop skills, and we can produce tools that yield quality and productivity within given ranges regardless of the worker. We can also assemble teams and assign work in ways that compensate for individual differences.
>
> We need to develop better techniques for software production, but we must also carefully study our organizational frameworks for software development and maintenance. We must devise control methods that match the frameworks.
>
> People are at the heart of software work; obviously, human error cannot be allowed to lead to major bugs or malfunctions. Here, teamwork will be the key. Team members working together are, in their collective wisdom, far more effective than individuals working alone.[39]

PRODUCTIVITY FACTORS (NEC APPLICATIONS SOFTWARE)

Key: Scale of 0 to 2, with 2 indicating highest impact

SCALE	
1.80	Human Factor [Programmer Capability]
1.48	User [Program] Complexity
1.42	Development Tools
1.28	Quality
1.23	Generality
1.22	Hardware
1.13	Contract [with Customer]
1.11	Specification Volatility
0.98	Development Methods

Source: Mizuno 1983, p. 69.

Fifth, since it seemed that getting all workers to think together in groups how to solve quality problems was the best way to try to make progress in

software quality control, they decided to adopt the manufacturing practice of quality circles.[40] In a 6-month trial during 1980, NEC also experimented with applying other techniques used in hardware QC, including statistical quality control and studies of worker behavior.[41]

Finally, the study group determined that to continue their efforts they needed a formal, company-wide program covering all aspects of software production, management, services, sales, and training. The program took form in 1981 as the "SWQC" (Software Quality Control) organization.[42] This was incorporated into NEC's zero-defect (ZD) structure, as indicated in the table below, with Mizuno heading the SWQC Group Activity Steering Committee and Fujino the Administration Committee. The SWQC Information Center directed the educational programs. Quality circles, meeting once per week for two hours, were the chief vehicles used to carry out QC training and functions.[43] (More detailed discussion of the quality control system at NEC will follow in a subsequent section.)

NEC'S SWQC ORGANIZATIONAL STRUCTURE, 1986

Structure	Focus
Zero-Defect Steering Committee	General Hardware and Software Quality Policy Formation, Corporate-Director Level
Company-Wide SWQC Group Activity Steering Committee	Software QC Policy Formation, Division-Manager Level
Administration Committee	Planning for Training and Education
SWQC Information Center	Training and Education
Study Group	Research and Recommendations
SWQC Group Activity Management Committee	Policy and Implementation, Department-Level and Suppliers
SWQC Groups	Quality Circle Activities, All Employees

Source: Mizuno 1983, p. 71; Fujino interview, 7/28/86.

SOFTWARE PROBLEM STRATEGY PROJECT, 1982-1985

Unlike the SWQC program, which has remained a permanent organization within NEC, but similar to the 1976-1979 Software Strategy Project, the Software Problem Strategy Project (SPSP) launched in 1982 was a three-year effort. The earlier project had developed new methods and tools, relying heavily on structured programming techniques, and began the process of standardization. The follow-up attempted to impose more top-down organization on all divisions involved in software development, enforce standardization, quality control, and productivity-improvement measures, and establish or formally designate a series of software factories to serve NEC's different product divisions. NEC President Sekimoto was the formal head of the project, with Mizuno as the sub-leader.[44]

The 1982-1985 project focused on three major activities. First, was what NEC called "software production mapping." This involved a logical and organizational layout of information processing activities by product (basic software, system engineering programs, user applications, transmission, switching systems, microcomputers) to determine which software houses divisions were using, or which subsidiaries NEC should create to serve as "software factories" to assist divisions in program development. Second, was a more formal attempt to systematize procedures for managing software subcontractors. Third, was another effort to improve and link software productivity and quality assurance measures. The last element included the establishment of a Software Productivity Committee, which worked on documentation control, quality control,

software productivity and quality measurements, cost estimation, education, project management, tools, and software production environments.[45] (Some of the results from these efforts will be discussed in subsequent sections.)

The project also dealt with two specific trends in software development that Mizuno, Fujino, Azuma, and other NEC managers believed they had to confront: "decentralization" -- development and support of programs by geographically distributed groups; and "asynchronism" -- development of parts of programs at different times.[46] For example, due to the cost of physical space, the shortage of programmers living or willing to live in the city, and the need to service customers from all parts of the country, NEC's software operations became increasingly spread around Japan after 1980, rather than being centralized.[47]

NEC did not emphasize the separation of design from program implementation (detailed design and coding) as formally as Fujitsu (Kamata Software Factory), Hitachi (Omori Software Factory), or Toshiba (Fuchu Software Factory). Nonetheless, it was still a common practice in NEC to divide up work between NEC divisions and subsidiaries or affiliated software houses.[48] Implementation of this type of factory concept whereby design was performed in one location and programming in another, or where parts of programs were developed at different times and in different places, depending on manpower availability and expertise, clearly required methods of ensuring that groups in different places followed the same procedures and used the same tools. Tools used throughout the NEC group, such as STEPS, SEA/I, and SPD (Structured Program Diagram) for applications software, and HEART (Hopeful Engineering for Advanced Reliability Engineering) and SDMS (Software Development and Maintenance System) for systems software, were designed to support a distributed but standardized approach to development.

II. TOOLS, METHODS, AND MANAGEMENT SYSTEMS

Product Technology and Process Choice

Characteristic of all the software factory approaches in Japan was the development of specific process techniques and tools -- in effect, specific factories -- for different types of software. In the case of Hitachi, which established a software factory in 1969, systems and applications programs were initially done in the same facility, although separate tools, methods, and standards evolved gradually and made it relatively easy to divide the factory into two, one for systems software and another for applications. In NEC's case, managers did not centralize software development in a single site or standardize around a single set or tools or procedures, but tended to emphasize different product types and the constraints or opportunities in process technology they felt these different products (and markets) offered.

Two main kinds of programs are general purpose systems software, and user software. Azuma and Mizuno adopted the position that general purpose systems software, like control programs for operating systems, usually had to be compact, use little memory, and offer a variety of functions for different users. Therefore, it seemed to them that programmers writing this type of software "are required to have a high degree of skill," and that reuse of standardized modules or patterns was not an appropriate process choice, although they still emphasized factory-type techniques for process standardization, quality assurance, and project control. On the other hand, they noted that language processors and applications software contained more similarities and less functional constraints, and thus offered more opportunities for using software

engineering methods such as modularization and structured programming, and even manufacturing-type techniques such as assembly of standardized program components.[49]

Azuma and Mizuno provided several examples of how process R&D in NEC has taken into account the characteristics of different types of programs. For example, one application area they called "Large-Scale and/or Complex Software Used Repeatedly," including train seat reservation systems, process control programs for steel mills, factory production control systems, and on-line banking programs. These had a common primary requirement, i.e. that "they operate accurately." But NEC did not focus on tools to verify logical correctness or to reuse code, but concentrated on "techniques, documentation, nomenclature, etc. [which are] important for harmonizing work by many programmers."

Medium-scale, complex applications software such as for scientific and engineering calculations required accuracy and reliability. Since the algorithms implemented by the code were complex, verification technology was the main area of NEC research. Azuma and Mizuno felt that "Programs of this type have a character close to the arts, and standardization is not a significant factor."

A third type of application program was "Small-Scale, Simple Software Used Only Once," such as management reports from databases or simple engineering calculations. Unlike complex applications, Azuma and Mizuno considered these easy and inexpensive to write and, therefore, not in need of factory-type tools or standards.

A fourth type of applications software -- "Small-Scale, Simple Software Used Repeatedly" -- was the focus of the STEPS system, although NEC also applied STEPS to other applications programs. Small-scale software of this fourth type included batch processing programs such as for inventory updates

and common business applications generally averaging about 500 lines of code or steps in COBOL, ranging to as high as 3000. Azuma and Mizuno noted that, "When these program are classified according to process patterns such as updating and inquiries, about 90% of them belong to any one of about 20 types of patterns. Programs involving entirely new patterns are few." Since there was so much similarity, the NEC managers found it relatively easy to develop standardized techniques, nomenclature, and documents, as well as standardized components.[50]

For all types of software, since 1980 or so NEC has followed a relatively consistent strategy for process development, according to Fujino, by focusing on computer-aided design and manufacturing techniques (CAD/CAM); reuse of existing software to build new programs; cross-product planning, and better requirement definition techniques, to reduce duplication and wasted effort; as well as standardization, quality assurance, and education programs directed at raising both quality and productivity:

THE NEC APPROACH TO SOFTWARE PRODUCTIVITY AND QUALITY

GOAL	METHOD
Improvement in development	CAD/CAM, Standardization
Reuse of existing software	Reuse technology
Waste prevention	Cross-product planning
Elimination of excess functions	Methodology and tools for requirement definition
Product quality improvement	Software quality metrics, tools for quality assurance
Personnel quality improvement	Education programs for new and old employees and managers

Source: Fujino 1984, p. 58.

Applications Software Standardization and Reuse

Most important for standardizing work procedures and documentation in order to promote division of labor for general applications software written in COBOL has been STEPS (Standard Technology and Engineering for Programming Support). Azuma and Mizuno, in a 1981 paper, described how their motivation to develop this stemmed from a desire to standardize practices so that people can work together effectively in teams, with minimal problems due to incompatible software-engineering tools or methods, as well as restricted communication among engineers: "Regardless of how excellent these [software engineering] technologies are, the desired objective cannot be accomplished if each member of a team or an organization developing software adopts them at his own will. This problem occurs because some technologies conflict with each other and communication between engineers is restricted. Hence there is necessity for standardization of software."[51]

After development began in 1971, NEC released an initial version of STEPS for in-house use in 1972. Azuma and Mizuno recall, however, that "The first version was intended to be comprehensive, fool proof and was compiled in the form of a manual 30 cm thick. Therefore it was never used." This experience led to another effort to produce a system easier to use, which NEC staff completed in 1974. Revision of the STEPS methodologies and tools has continued every year since, with approximately 1200 NEC and NEC-customer sites using STEPS by 1984.[52] NEC engineers claim that the recent success of STEPS was due to their two-fold approach: offering of a technological component of standardized tools, procedures, and "prefabricated standard program" modules accessible through a program library; and emphasis on a

philosophy outlining how programs should be developed through each stage of the software life cycle.

The philosophy underlying STEPS, according to Azuma and Mizuno, drew on the historical experience of companies in "hardware" manufacturing industries which had found ways to improve productivity through standardization and division of labor, as well as process rationalization through analysis of "raw materials," "intermediate products," final "products," "work methods," and "product utilization." All of these process inputs, they insisted, had conceptual counterparts in software:

> The productivity of industry has greatly increased after the Industrial Revolution. The basic concept of the Industrial Revolution was to achieve high volume production of standardized products ...drastically reducing the manufacturing cost ... by thoroughly incorporating standardization not only of the final products but also their parts...
>
> In spite of essential differences between software and hardware, a number of similarities can be found where standardization is concerned.
>
> (1) Program language, macros, subroutines, etc. which correspond to raw materials in producing software...
>
> (2) Documents and specification documents corresponding to intermediate products. Their standardization facilitates division of labor and automation.
>
> (3) Programs and documents correspond to products. Mass production of standard products are [sic] similar to general purpose applications of computer manufacturers and software houses. The needs and circumstances of users are identical to other examples were production of orders are tailored to the customer. Products should perfectly be made with parts in units which are as large as possible, that is, by combining standard modules.
>
> (4) Tools correspond to the software used for software production. Compilers, test data generators, etc. are examples of these.
>
> (5) Work methods represent standardization of methods such as application system designs, software designs, and coding.
>
> (6) The product utilization method concerns which data is processed by the software. To efficiently design individual items of software and to eliminate contradictions among them, standardization of data will be important. In other words, this concerns standards of languages, modules, program structures, tools, methods, documentation, and data, and standardization must be conducted systematically under a consistent

philosophy.[53]

The STEPS procedures required program development to be divided into five distinct "phases," with subsets of "activities" and then "work sets" for each activity, consisting of specific procedures and "basic work elements," all of which managers monitor as part of scheduling and progress control. Process standardization came from standardizing each work set and accompanying documents, using form sheets or simple headings for "work of a more creative type." NEC managers considered standardization and documentation of this degree essential for communication in general and division of labor and program maintenance in particular (see table).[54] Division of labor also included the separation of design from production to the extent that NEC divisions sometimes "handed off" high-level design documents to subsidiaries, which then did detailed design, coding, and testing.[55]

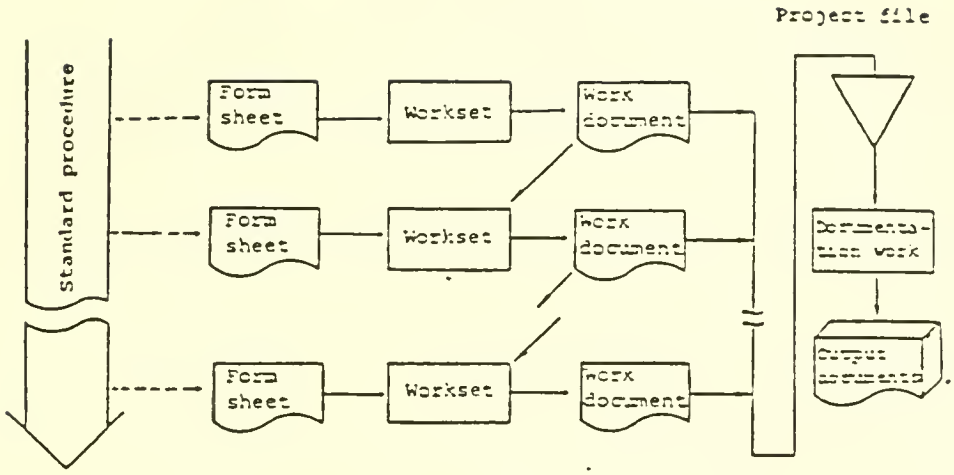


Fig. 2 Concept of System Development Standards

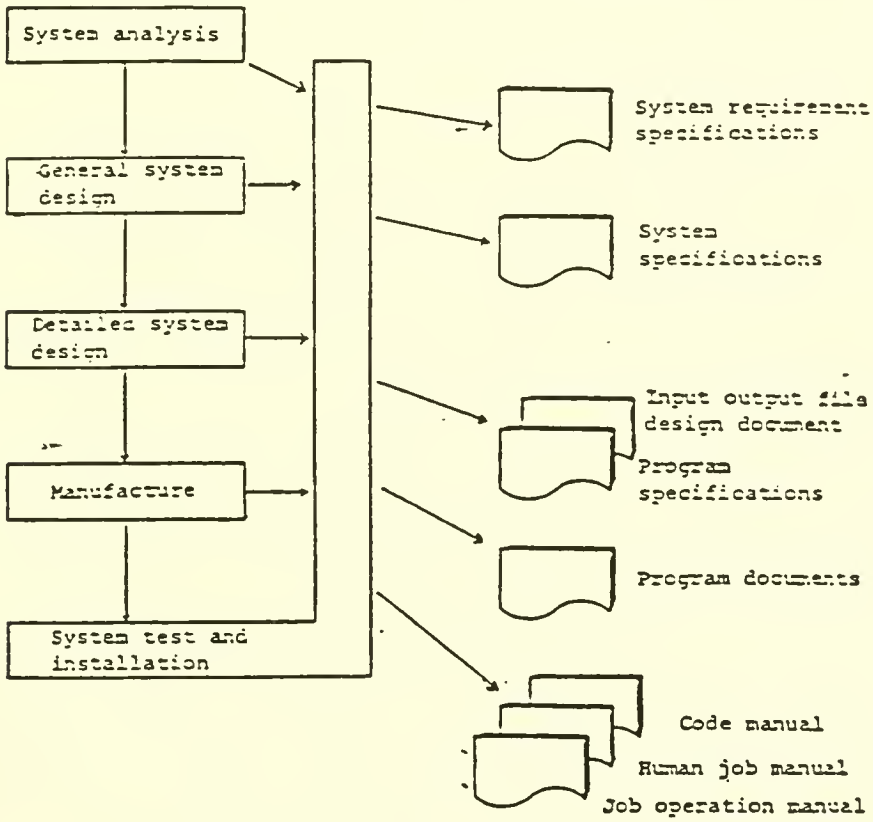


Fig. 3 Phases and Output Documents

STEPS PROGRAMMING STANDARDS

Programming Standards	Work	Outputs
Standard Patterns	General System Design	Processing Flow
Standard Specifications (Structured Program Diagrams)	Detailed System Design	Program Specifications
Program Standards (Structured Program Coding)	Manufacture	COBOL Source Coding

Source: Azuma and Mizuno 1984, p. 88.

Reusability stemmed from the similarity of business applications; the ability to modify existing code to suit new users -- semi-customization -- was made possible by modularization, standardization of design techniques and the programming language (COBOL), and extensive documentation. Azuma and Mizuno also felt confident they could construct almost any type of business program easily from standard patterns modified for individual customers: "It should be possible to modify standard patterns to suit every user, and modification of standard divisions for every program should be easy in order to be able to apply standard patterns to a larger number of programs. Modifications of standard divisions for every program should be easy." Accomplishing this required several techniques and procedures:

1) clear classification of standard source codings intrinsic to common patterns
2) easily understandable lists of program patterns

3) short documentation accompanying lists

4) clear program structures designed for easy maintenance and modification

5) COBOL as the standard coding language

6) structured programming techniques sufficiently versatile to handle complex as well as simple programs

7) standards for design, documentation, coding, module nomenclature, work areas, etc. (see figure).[56]

There were various ways to reuse code that Azuma and Mizuno had considered. One was to locate and utilize similar programs; they rejected this strategy for STEPS because it was not versatile, and modifications of programs were time consuming and led to errors. Another was for programmers to memorize recurring, specific patterns; experienced personnel seemed to do this as a matter of course, although it was not a practice that fit into a set of engineering techniques. A third way was to prefabricate programs for common applications, with parameters entered in place of specific data. This they found useful for simple patterns such as medium conversion and extraction, but not for more complex functions. A fourth was to generate programs from parameters or from simple language descriptions; for complex programs, however, the language needed to do this came near to the complexity of COBOL. A fifth approach was to place frequently used modules in macros; this did not cover all types of coding divisions, however.

The technique STEPS promoted was to create what NEC called "standard program patterns," written in structured COBOL for relatively easy modification and maintenance (see figure). Development of a "new" product for a particular customer involved writing a general outline that corresponded to a standard pattern, examining the more detailed program specifications, and then

identifying eliminations or additions as needed in the detailed design phase before coding:

> A program developer first writes a process flow to match a standard pattern in general system design phase. There is a standard program specification corresponding to each standard pattern. Program Designer compares it with a requirement specification in detailed system design phase. This eliminates unnecessary functions, and defines processing intrinsic to business that is to be added and inserted. Next, in the programming phase, standard programs are modified if necessary, and additions and insertions of processes intrinsic to application are performed.[57]

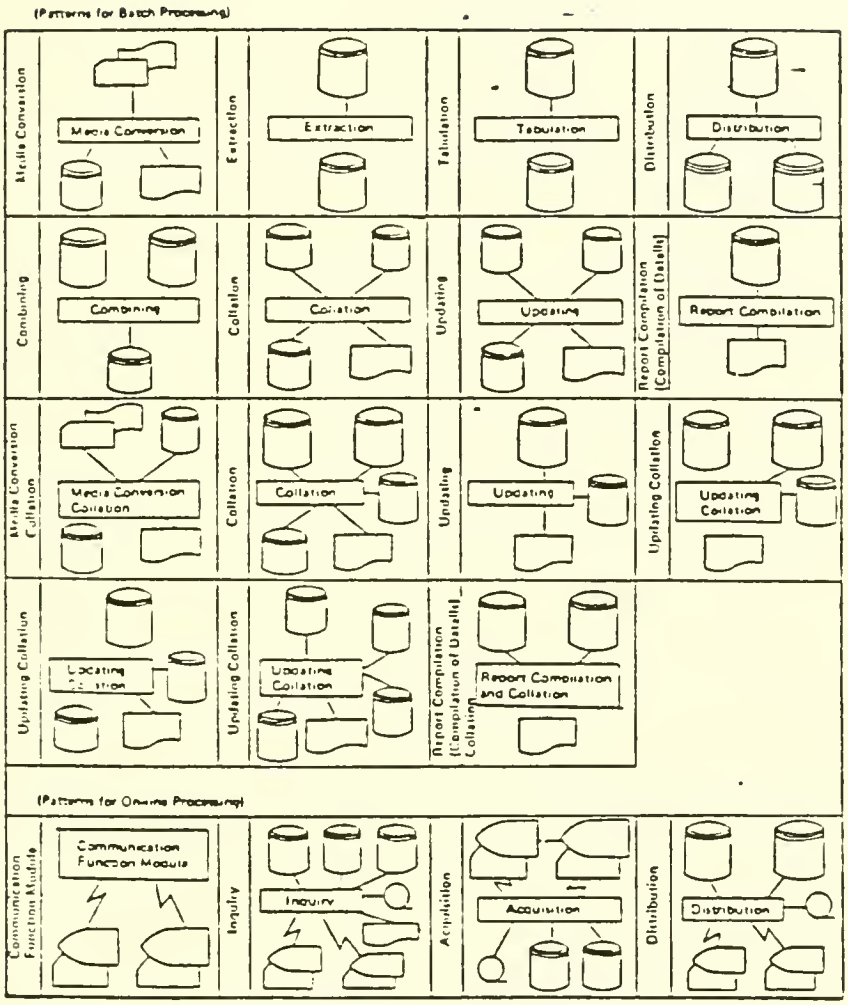


Fig. 7 STEPS Standard Patterns

Unlike Toshiba's Fuchu Software Factory, NEC did not impose formal requirements on designers to register a certain number of reusable modules per month. The STEPS system itself promoted reuse, as did program-generator tools such as SEA/I and DECA. NEC did, however, provide bonuses to designers who produced software modules that turned out to be frequently used.[58] This procedure was formally part of the SWQC program, which also gave out awards for quality and productivity. [59]

To facilitate the design and production process outlined in STEPS, NEC also developed a structured programming diagram technique called SPD (similar to Hitachi's PAD, Fujitsu's YAC-II, and NT&T's HCP diagrams). SPD represented program modules hierarchically and showed interrelationships in the flow of control, with diagrams written in Japanese or any other language. NEC introduced its first version of this diagramming technique in 1974, with considerable resistance from software personnel -- usually, the more experienced engineers -- who preferred to write sophisticated but unstructured "spaghetti programs." The earlier versions of SPD also had only simple lines and notations, and thus lacked representations such as symbols for node control (input/output paths) and module names. Additions to the system, as well as the use of "logic tables" to describe process specifications, significantly increased acceptance of STEPS, according to Azuma and other NEC engineers (see figures).[60]

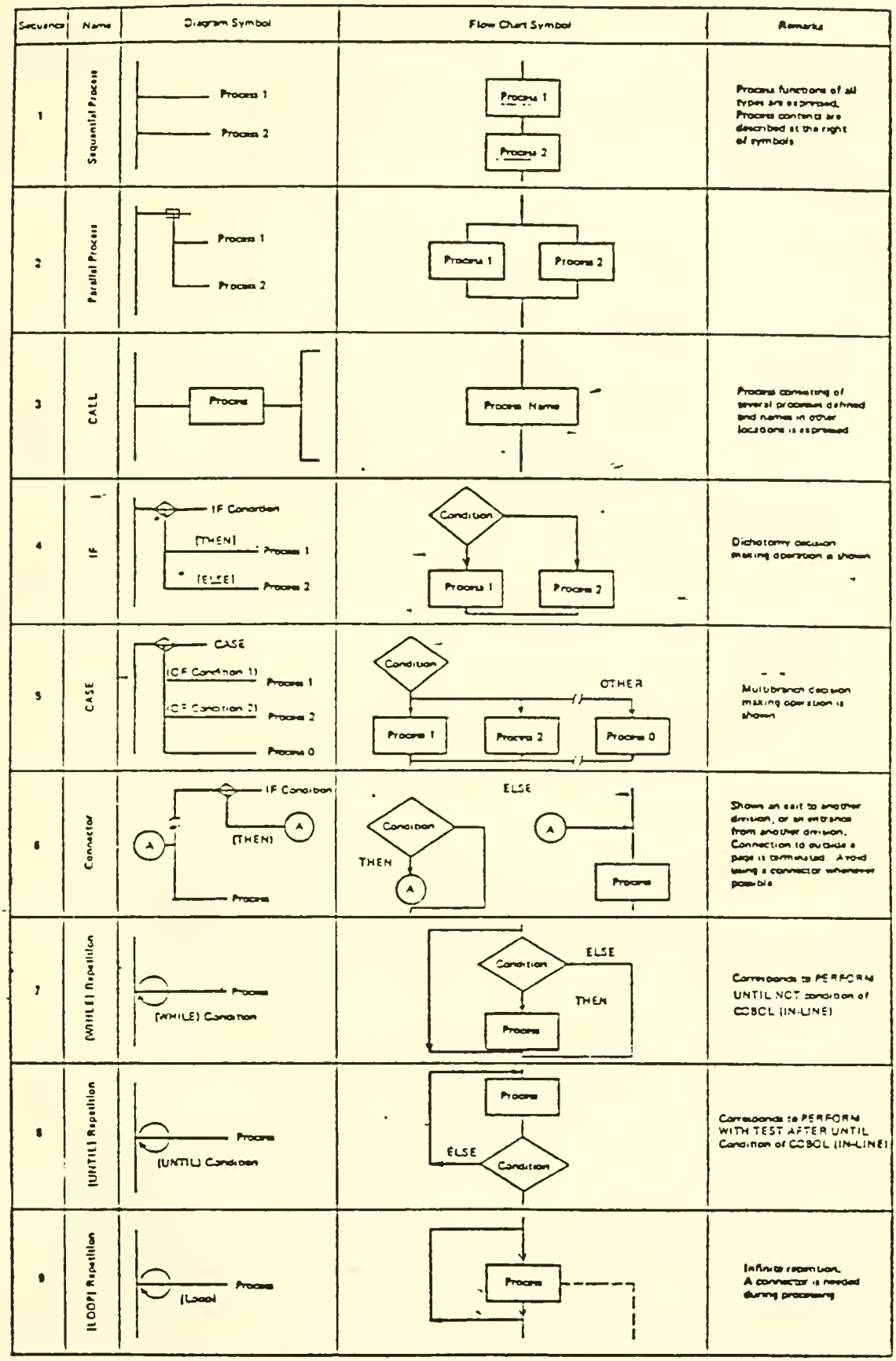


Fig. 9 Symbols Used in SPD

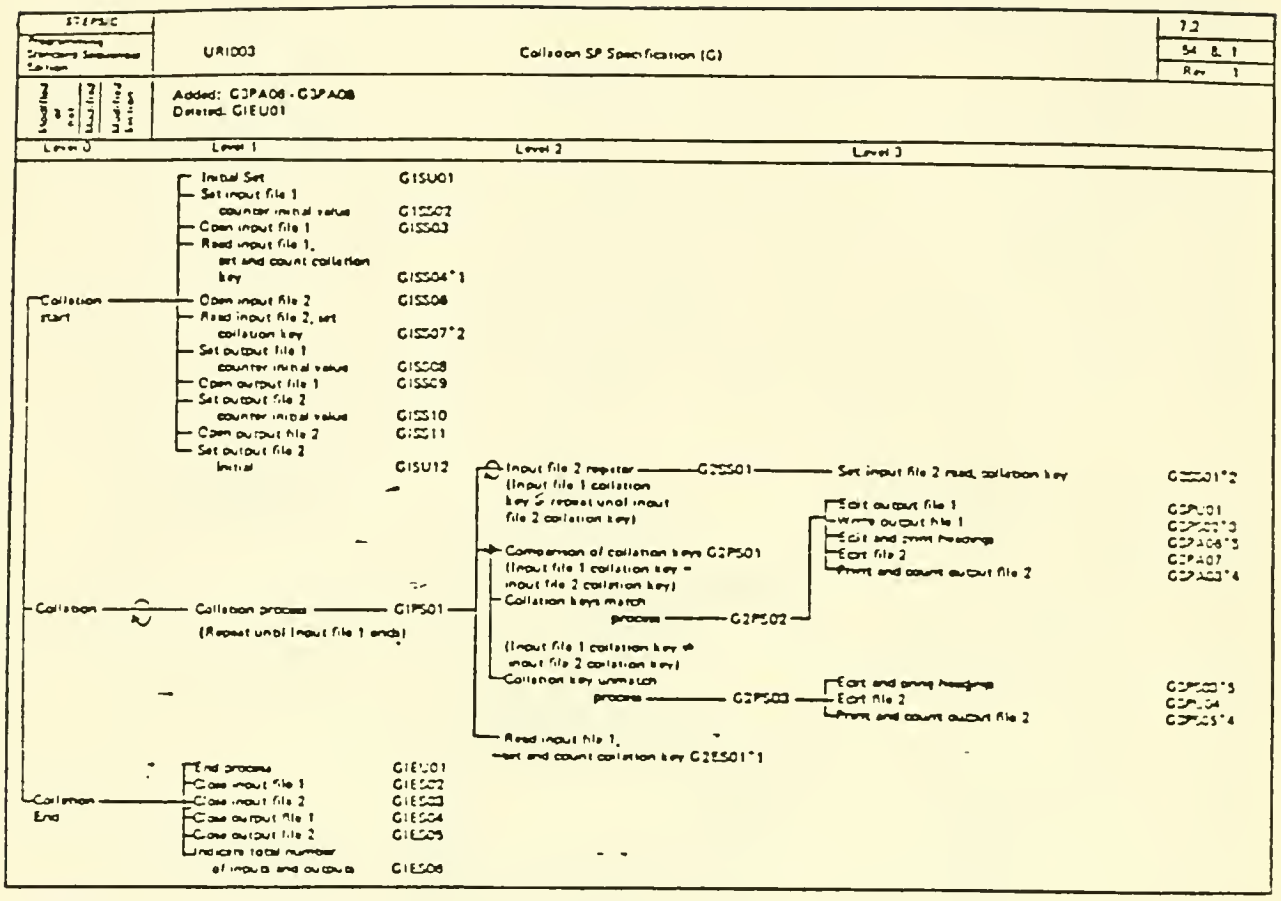


Fig. 10 Example of Collation SP Specification

Program Name	Date	Edition	Approval	Supervisor	Person in Charge	Reservation No.
URI003	LOGIC TABLE 78/04/03	1				Reference No.
						Computer John Doe

Process Function	Input	Process	Output to
G1SU01 Initial setting		Accept process date	Date of first heading item
G1SU12 Initial setting of output file 2	Constant 0	To set initial value of page counter	Page count item (work region)
G2PU01 Editing of Output File 1	Input File 1 • Slip No. • Date • Customer Code • Commodity Code • Quantity • Commodity Class/Accounts Code	Posting	Output File 1 • Slip No. • Date • Customer Code • Commodity Code • Quantity • Commodity Class/Accounts Code
	Input File 2 • Commodity Name • Unit Price	Posting	Output File 1 • Commodity Name • Unit Price
G2PA06*3 Editing and Printing of Headings	———	To execute a common subroutine "Heading Editing, Printing "HEAD—RTN""	
G2PA08 Editing of Output File 2	———	To execute a collation key match record edit routine EDIT—MATCH—R—ROC"	

Standard program codings or procedures corresponded to SPD and were divided into 5 levels. The first three treated as "black boxes" processing divisions, conditions for selecting the primary processing functions (modules), and specific processing functions. Another level consisted of common subroutines. The fifth was the "users level," where customers defined procedures unique to their needs. Programmers using the STEPS methodology only had to write specifications and code for this fifth level.[61]

STEPS STANDARD CODINGS

- Identification Division
- Environment Division
- Data Division
  - File Selection
  - Working Storage
  - Section
- Procedure Division
  - Level 1 (Process Division)
  - Level 2 (Primary Process Function)
  - Level 3 (Specific Processing of the Primary Process Function)
  - Subroutines
  - Users Level

Source: Azuma and Mizuno 1984, p. 91.

Despite some resistance in the mid-1970s, NEC managers considered the introduction of STEPS as a major success. Among programmers surveyed at 88 in-house and customer sites in 1981, according to Azuma and Mizuno, about 88%

found STEPS "very easy" or "easy" to learn; 55% found it made coding and debugging "very easy"; and 81% felt it had a "considerable" or "big" effect on productivity. Productivity improvements measured by man-days required for similar programs developed with and without STEPS ranged from 26% in the specification phase, 91% in coding, 35% in compile time for debugging, and 53% overall for man-days. The average STEPS program, however, was slightly longer than average non-STEPS programs by about 6% more lines of source code (see table). Offsetting this, according to 1984 data on STEPS users at 1200 sites, was generally a 20 to 50 percent cost reduction in analysis and design phases, and a 50 to 80 percent reduction in program manufacturing phases.[62]

STEPS PRODUCTIVITY COMPARISON (1981)

Sample: Project data from 1200 STEPS user sites

Key: s = source code lines; md = man-days; t = compile time for debug

PHASE:	STEPS	NON-STEPS	IMPROVEMENT
Specification	286 s/md	361 s/md	1.26
Coding	218 s/md	416 s/md	1.91
Debug	68 s/t	92 s/t	1.35
Total	64 s/md	101 s/md	1.53
Average Program Size	458 s	431	

Source: Azuma and Mizuno 1984, p. 95.

Automated Production of Business Applications Programs

The most important automation tool used for business applications programs in COBOL was SEA/I (System Engineering Architecture). Development began around 1980, when NEC established the Software Product Engineering Laboratory, and in 1984 NEC began offering SEA/I as a product priced at $10,000 and running on an NEC minicomputer, with support for up to 10 engineers.[63] The tool consists of three subsystems: an Empirical Information Base (EIB), a "tool machine" set, and a work methodology, covering proposal, design, implementation, testing, and installation. EIB attempted to capture programming experience by providing set formats and ready access to previously written system definitions, layout designs, system and program structures, program modules ("source parts"), number of tested programs, and test data. The tool set provided specific software to aid in all phases of development, from proposal formation and prototyping through maintenance. The work methodology was based on STEPS.[64]

SEA/I served as a program generator in two ways. One method NEC called "auto synthesizing." This produced complete programs in structured COBOL through a menu system which pulled out software modules or "assembly parts" from a Source Parts Library and put the parts together automatically. The second method NEC called "interactive synthesizing." This allowed the user of the tool to specify parts to be recalled from the parts library and where they were to be located in the program, and to insert newly written code anywhere in between. In both cases, the synthesizer function of SEA/I checked syntax validity, data-name consistency, and attribute consistency, to make sure the modules would work together.[65]

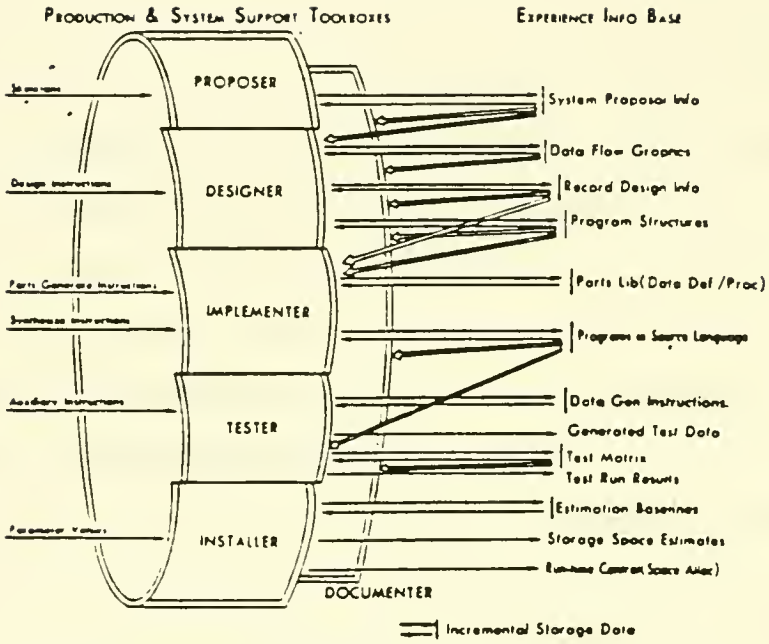


Fig.3 SEM/I Tc . Machine and EIB

An example of developing a data entry program with the various tools included in SEA/I is shown in the figure below. This example requires one storage file (Item Master File) and one screen form through which the user enters data. The program designer uses the Data Record Design Aid (DDA/1) tool to create a record structure and screen format shown in the upper left-hand corner. The Data Definition Source Parts Generator (IMA/2) analyzes the design and then generates two sets of source codings (file structure and screen format), which are put together into a single COBOL program by an "implementor" tool called IMA/3. A testing tool analyzes the program to extract test data, while a configuration tool calculates how much storage space is needed and generates job control commands to allocate the file space. For other types of business applications, a menu-based tool called Program Design Tool (PDA/1) performs similar functions.[66]

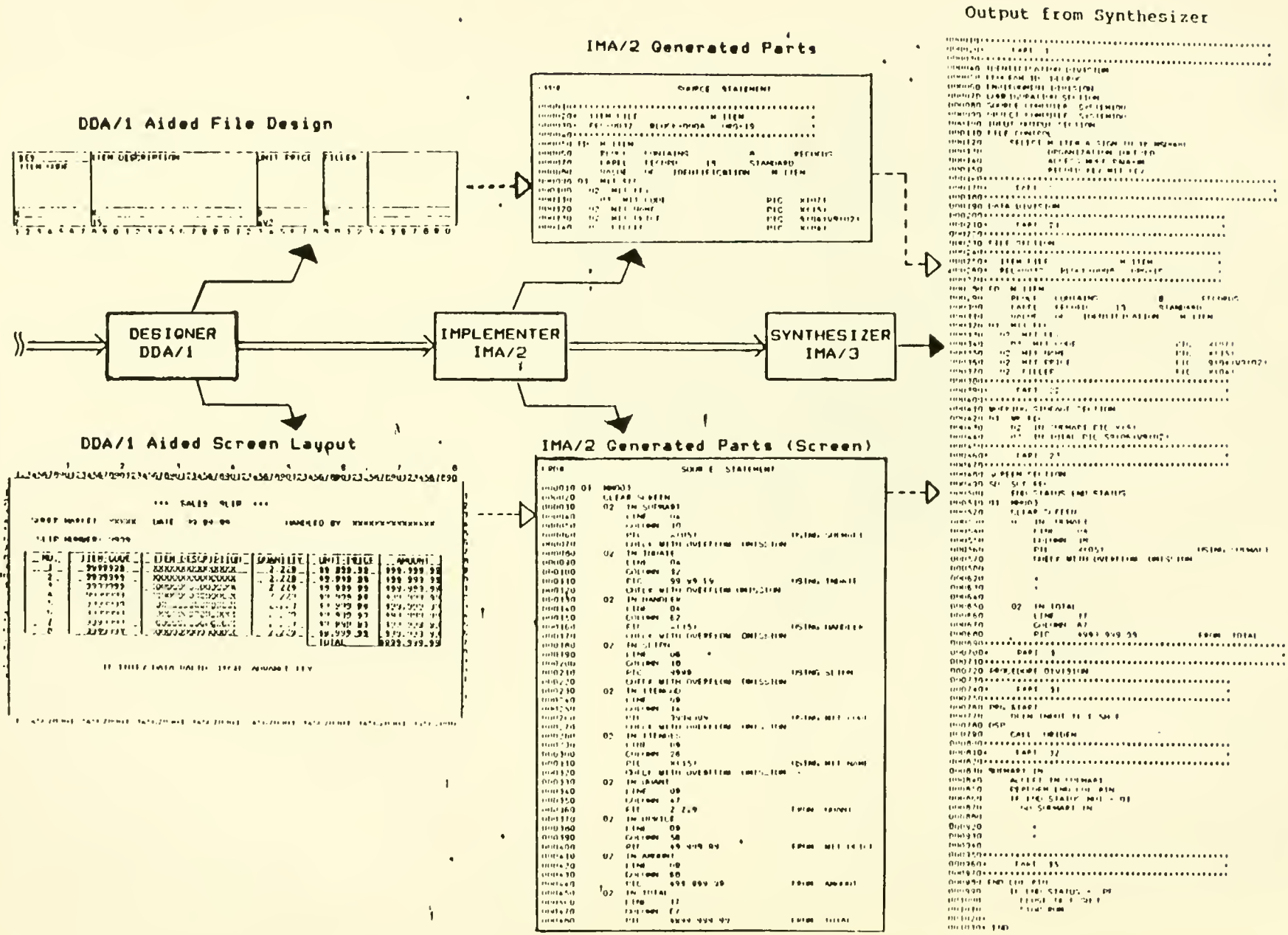


Fig.4 Program Generating Process in SEA/I

Of particular significance, in the view of NEC managers such as Fujino and Masao Matsumoto (the latter has been directly involved in developing the tool), is that SEA/I "learns" in the sense that the number of elements in its Empirical Information Base -- system designs, documentation, code -- increase every time a new program is designed on the system. Improved designs and documentation thus become accessible automatically to future users. More importantly, the procedures and tools accompanying SEA are completely standardized, making it possible to divide labor among the different development stages or have different individuals up-date programs in the future without, according to Fujino, the usual problems that come with changes in personnel. Users have reported as much as a 7-fold increase in productivity due to the automated functions.[67]

Another automation tool used primarily for applications software requirements definition and documentation was SPECDOQ, also developed at the Software Product Engineering Laboratory. This provides graphics, menus, and diagramming functions linked to a relational database that allow engineers to design programs through standardized documents, and then change these documents easily for different applications. The tool was designed to run on the Unix operating system and, according to NEC developers, was also able to process the STEPS documentation.[68]

Systems Software Development and Process Control

A major issue in NEC's basic software development was standardization. Because the company supported 4 incompatible operating systems, including one inherited from Toshiba in the late 1970s, managers had to think about standardization even more deliberately than other divisions to get any economies

of scale across these different lines. Even though the architectures were different (despite being merged gradually), planning and standardization allowed NEC to reuse code across the different systems, especially for language utilities and data base programs.[69] But the incompatibility of different machines and the presence of old software and documentation made it necessary to (1) formally recognize and try to improve existing tools and procedures; and (2) delay the introduction of more advanced design-support and process control tools.

All of NEC's systems software divisions, subsidiaries, and affiliated firms relied on tools developed along with different operating systems. Most important was HEART (Hopeful Engineering for Advanced Reliability Engineering), used for operating systems development. The procedures, standards, concepts, and tools associated with HEART had been used for several years in NEC until, as part of the SWQC activities at the Fuchu Software Factory, managers decided to compile a formal manual and promote HEART as a "scientific" system for production and quality control. The quality assurance department at the Fuchu software facility has been the formal promoter and continues to assist NEC subsidiaries and affiliated software houses in adopting the recommended techniques and standards.[70]

The philosophy behind HEART was that it was possible to use data collection and analysis to link efforts in process control to quality control. This was based on the notion that improving quality will lead to higher productivity.[71] To implement this concept, HEART relied on 5 basic elements:

1. Process Control System (Standardization, Team Methodologies)
2. Design Techniques and Tools (SPD, DECA)
3. Program Analysis System
4. Modularization Methodology for Reuse

5. Testing Technology.[72]

HEART defined 7 phases of development and testing, and required data collection during each: requirement analysis and definition; functional design; detailed design, which used the SPD charts; coding, mainly done in a PL/I subset, HPL, with some also in C and assembler; unit and function test; system integration; and system test. Development item history sheets were required from requirement analysis through coding; quality accounting sheets were required from unit/function test through system test. NEC automated some of the data input; other data entry was done manually, although all the data went into a central process/quality control data base accessible to managers through on-line terminals (figure). The kinds of data required for each phase concentrated on scheduling and productivity (man power) (table).[73]

PROCESS-CONTROL DATA COLLECTION

1. Requirements Analysis/Definition Process

-- estimated date of completion of each process
-- estimated program size
-- estimated man power
-- person in charge of development and years of experience
-- language in use
-- development form (new development/modification/division transplant)
-- difficulty level of development [type of program]

2. Functional Design Process

-- completion date
-- actual man power used and break-down by persons in charge
-- difference between the standard times and the man power used
-- quantity of functional specifications and revision history
-- scale of design review (number of workers/times) and the number of corrections

3. Detailed Design Process

-- completion data
-- actual man power used and break-down by persons in charge
-- difference between the standard times and the man power used
-- quantity of design specifications and revision history
-- scale of logic inspection (number of workers/times) and the number of corrections

4. Coding Process

-- completion data
-- actual man power used and break-down by persons in charge
-- difference between the standard times and the man power used
-- the development size
-- detailed information for each program to realize functions
-- scale of code inspection (number of workers/times) and the number of corrections

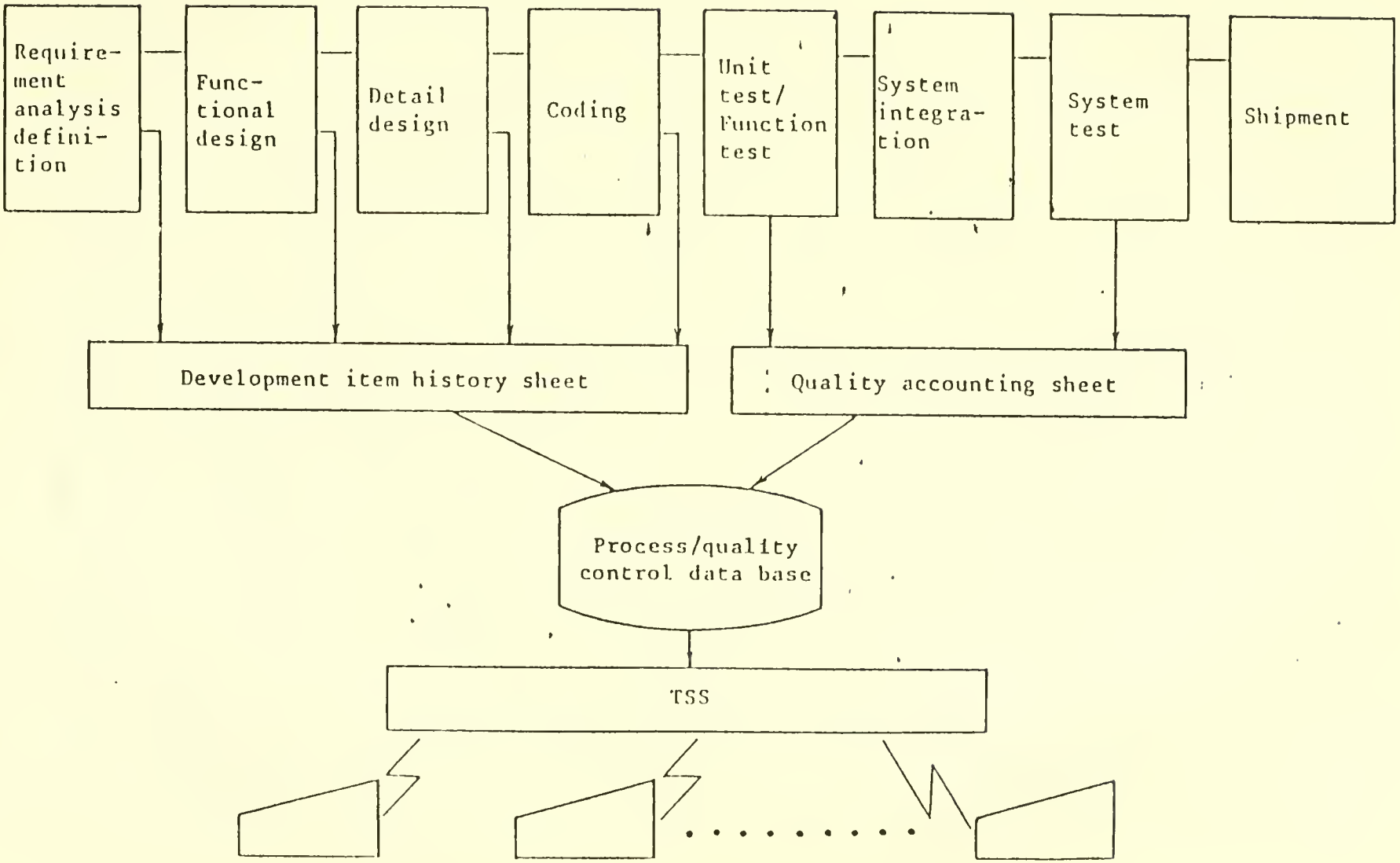
5. Unit Test/Function Test Process

-- number of test cases to be executed
-- target bugs to be detected
-- required man power
-- number of test cases executed previously in a certain period of time
-- number of bugs detected previously in a certain period of time
-- man power used in the past

6. System Test Process

-- number of test cases to be executed
-- target bugs to be detected
-- required man power
-- number of test cases executed previously in a certain period of time
-- number of bugs detected previously in a certain period of time
-- man power used in the past

Figure 1 Development Process and Data Collection



An SPD translator tool called DECA (Detailed Design and Coding Assistant) translated SPD diagrams into program source codes, or program codes into SPD diagrams.[74] This eliminated much of the tedium (and potential errors) of coding in a variety of languages, including C, Fortran, HPL (similar to PL/I), and COBOL (for applications software), although DECA was only "semi-automated" (compared to SEA-I) in that users still had to write code to create interfaces between different modules.[75] By 1987, with the exception of specific programs following different customer specifications, all NEC software facilities -- producing systems software as well as applications programs -- used versions of SPD and DECA to write detailed designs and code.[76]

HEART procedures called for "process management meetings" to be held periodically to discuss primarily two issues: adherence to the promised delivery (release) date; and quality of the final product. Analysis for this included data from several "process management models," primarily a 12-factor cost model. Differences between estimated and actual man-power figures required a review of the managers in charge.[77] According to Hirai Yozo, Manager of the Quality Assurance Department in the Basic Software Division,, NEC was generally able to deliver system software products within a month of planned release dates; while NEC included some "buffer time" to allow for slippage of schedules, this represented an improvement of approximately 20 to 30% in planning accuracy over the past five years. There was no organizational division of labor along with the different phases of development, as might occur in applications software development, although less-experienced programmers did detailed design and coding, and the more experienced personnel focused on requirements analysis.[78]

With regard to project control, it is interesting to note that NEC did not follow the advice of Frederick Brooks at IBM, who had found that adding

people to an already late project made it later, due to the communication time required. Hirai claims that NEC was able to add people to any stage of the process and effectively reduce lateness, although managers tried not to add people to testing, because this meant they were not catching bugs and eliminating them during development.[79]

SDMS (Software Development and Maintenance System) was a more advanced and integrated set of tools and techniques originally intended to serve as a software factory infrastructure for all basic software. Due to introduction problems, NEC has limited its use to new switching systems and communications portions of operating systems.[80]

Basic planning for SDMS started in 1975, at about the same time information was becoming available on SDC's Software Factory experiment. In many ways, NEC managers intended to use SDMS as a total, factory support system for the development and maintenance of modularized systems software. In fact, a 1979 article published in Japan's leading journal on information processing compared SDMS to the SDC Software Factory, as well as to other tools developed at Mitre (Simon), Softech (Software Engineering Facility), TRW (SREP), Fujitsu (SDSS), and Toshiba (SWB). According to the NEC authors, these other tools concentrated either on design support or project management. SDMS was superior in its attempt to integrate both functions.[81]

The first practical version of SDMS was released for in-house use in 1980; NEC has continued to introduce improved versions every year or two. The basic system remains the same, consisting of a software development data base and three subsystems: (1) for design, including a standardized design language (SDL) and a methodology emphasizing modularization and sophisticated data flow and abstraction techniques, with facilities for design modifications, automated error checking, and automatic design document generation; (2) for product

management (configuration, updating, retrieval); and (3) for project management, including progress control and productivity data.[82] The design subsystem is considered the most important part of SDMS.

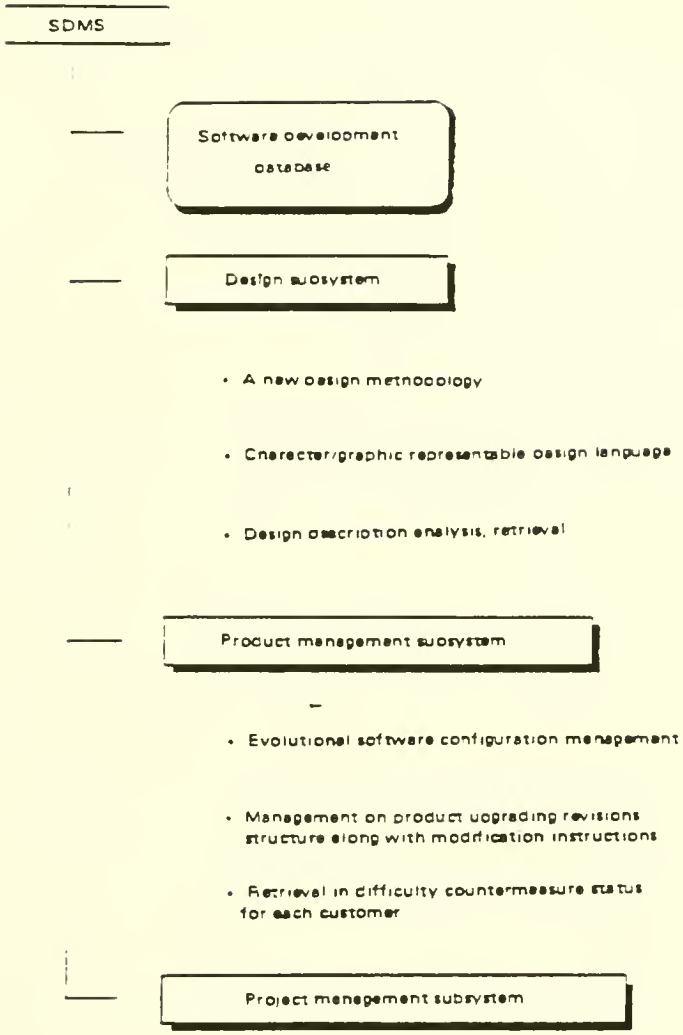


Fig. 1. SDMS System Structure

Several experimental projects have reported significant improvements in productivity and quality. For example, in the design of a comptroller system with a data base, engineers using SDMS showed twice the output rates of comparable projects, including the discovery of 90% of the design errors before the programming phase, and automatic generation of more than 90% of the design documents, which previously were written by hand. In maintenance of an overseas switching system, SDMS helped reduce man-hours in producing upgrades and revisions by 90%.[83]

Despite these results, NEC developers have acknowledged serious resistance or problems in adopting the standardization and tools required by SDMS. Therefore, NEC has restricted its use to new selected new programs, especially in the switching division. SDMS is also the required tool for new software development projects done within the Software Product Engineering Laboratory.[84] But examination of why SDMS has not quite fulfilled the early goal of serving as a comprehensive "software factory" infrastructure provides several lessons regarding the introduction of new standards and tools into any existing development and production system.

One problem has been the time required to learn the new design methodology. This learning tended to delay the start of actual work on projects, as well as slow down design progress because programmers were not familiar with the new methodology. In some cases, this type of experience led to the suspension of SDMS introduction plans. A second problem was that the new design techniques did not work well with existing code written using less emphasis on concepts such as data-flow and abstraction. The need to upgrade previous systems and the difficulty of replacing existing software entirely has also reduced the successful transfer to SDMS even in divisions that would like to adopt the system. A third, related difficulty has been that SDMS was

developed to run on NEC's large-scale ACOS-6 operating system. Programmers writing software for different computers found it difficult to mix more than one machine. Fourth, was the general problem that NEC divisions had developed their own tools and standards over the years [such as those now included in HEART]. Getting them to change over to SDMS standards took time and was particularly difficult if divisions were engaged in joint development efforts with other firms.[85]

Process, Cost, and Quality Control

Because Honeywell had been particularly successful developing software for its 200-model computer series, for the MODE IV project NEC's control section in the computer division decided to copy Honeywell's process-control system, which consisted of a PERT network model. "We were very influenced by Honeywell," Mizuno admits, but they soon realized that the PERT charts were almost useless. The initial planning parameters were so inaccurate that none of the schedules came out close to estimates. This prompted NEC managers to adopt another planning and forecasting model used at the System Development Corporation (SDC) in Santa Monica, California.

SDC had designed a model for the U.S. Navy during the 1960s that broke down the development process into phases such as specifications design, detailed design, coding, and component test. It was superior to Honeywell's technique, but, according to Mizuno, the SDC system also had serious deficiencies. The main problem was that SDC had fixed the parameters and coefficients used in the model, basing them on the average performance level of its engineers; users were unable to take into account different experience levels of programmers or changes over time. NEC managers decided they needed a more dynamic model

that could accommodate different experience levels, as well as additional factors they felt had an impact on software costs and personnel performance.

It was at this point -- around 1967 -- that NEC had enough of its own data to begin calculating rough standard times for different software development activities and different types of programs. This data, based on the MODE IV programming experience, became the basis for a planning model completed in 1969-1970. The standard times used were still primitive, in Mizuno's view, because they lacked a good metric for measuring productivity and did this simply by lines of code (steps) in a given period of time. "We came to the conclusion that we had to introduce more scientific measures for software productivity," he recalls, and this conviction gradually led to more precise distinctions in the productivity data collected between different types of programs and languages used, as well as the amount of documentation required. This work led to a totally new cost estimation system in 1977, based on a multi-factor regression model derived from the "meta model" developed at the University of Maryland by J.W. Bailey and V.R. Basili, which consists of several standard model expressions that allow users to select from their own data different factors that appear to impact cost. Users can also make adjustments for productivity improvements.[86] Accordingly, NEC revises standard-times data annually and has used comparisons of estimates and actual data for projects to improve the model continuously.[87] To make it easier to use, the expression of the NEC model follows the popular COCOMO format proposed by Barry Boehm and his staff at TRW.[88]

NEC COST ESTIMATION FACTORS AND MEASUREMENT SCALES

Man Power (man-hours)
Program Size (1000 lines of code, excluding comments and macros)
Development Tools (%)
Development Technologies/Methodologies (%)

OS/Hardware Interface (%)
Member Experience (years)
Outside-Order Dependency (%)
Specifications Change Frequency (scale of -2 ~ +2)
User Characteristics (-2 ~ +2)
Product Flexibility (-2 ~ +2)
Novelty/Complexity (-2 ~ +2)
Quality Requirements (-2 ~ +2)
Team Ability (-2 ~ +2)

Source: Mizuno 1985, p. 5.

NEC used its multiple regression model for both cost estimation and productivity analysis. By establishing a productivity range and mean productivity coefficient value for each factor, it was possible to determine the effect of each factor on productivity and, therefore, to what extent improvement was possible in each area. For example, recent data suggested that factors with the "most room for improvement" were development tools (52.6%), quality requirements (43.7%), outside-order dependency (procurement) (36.1%), product flexibility (25.0%), and frequency of specifications changes (22.3%).[89]

In the early 1980s, the Software Product Engineering Laboratory developed a version of the cost model to be used on a personal computer, for groups of 10 to 20 programmers. This system, called TOMATO (Table-Oriented Manager's Tool), used a relational database format like LOTUS 1-2-3 to allow managers to track schedule progress, compare actual times to estimates, and do some simple productivity factor analysis.[90]

NEC used in conjunction with these cost and planning tools a "quality accounting system," developed by the Quality Assurance Department at the Fuchu software factory. Department literature explained this as a "system in which bugs generated into a program are regarded as a debt, this debt is repaid through bugs detected with a test and the shipment is performed when the debt

becomes 0."[91] More specifically, the accounting procedures called for (1) predictions of the number of potential bugs; (2) establishment of a bug control curve; and (3) execution of tests and quality evaluations.

Bug control utilized a simple "reliability prediction model." This estimated the number of bugs likely to be in a certain type of software based on past data on the number of bugs generated in similar programs, adjusted for 8 factors: (1) program size; (2) degree of inspection and design review (number of corrections in functional specification, detailed design, and coding, divided by the number of documentation pages or program size); (3) development form (new, modified, or transplanted code); (4) language of development and degree of difficulty; (5) type of operating system/hardware interface; (6) years of experience of development team members; (7) frequency of specification changes; and (8) sufficiency of man power. As test data accumulated for a current program, NEC substituted these in the model for the estimated figures. A bug control curve determined whether or not the detection of bugs through testing was going according to predicted levels, based on a Gompertz curve reliability growth model. If test results were significantly different from predicted bug levels, causes of the discrepancy had to be analyzed. Testing was completed only "[w]hen all predicted bugs are detected."

BASIC FORM OF RELIABILITY PREDICTION MODEL

$$B = a \cdot \prod_{i=1}^{n} \alpha_i \cdot S^{b_0}$$

$$\alpha_i = 10^{b_i x_i} \qquad (i = 1, 2, \ldots\ldots, n)$$

Here, B: Number of potential bugs when the test starts. (This is substituted by the number of detected bugs during the test.)

S: Development/modification size (KL) of program

X_i: Various development factors

a, b_i: Proportional coefficient

b_0: Exponential coefficient

While the Quality Assurance Department in the Basic Software Development Division tended to direct QC activities for software development throughout the NEC group, much of its directives were channelled through the SWQC (Software Quality Control) program and quality circles established in 1981 (see discussion above).[92]

NEC gradually extended the SWQC program through a series of steps.[93] First, the SWQC Information Center staff determined which low-level managers should form groups. Higher level managers were then asked to allow the employees to devote some time, usually once a week for two hours, to this activity. Once formed, groups set targets, collected and analyzed data, reported on their results, and participated in the establishment of measures to prevent similar errors from recurring (see table). SWQC teams even performed internal reviews of programs in development, in order to catch mistakes early., although these reviews were in addition to formal "third-party" technical reviews conducted in later stages of development. To guide group activities, the SWQC Information Center held training workshops for group leaders and published various handbooks and manuals, such as "The Seven Tools of SWQC." This was based on convention QC techniques used in other industries, and outlined how to use data sorting, control charts, and Pareto charts for data analysis, and then brainstorming, cause-effect diagrams, and other methods to implement solutions.

<u>SWQC PROGRAM</u>

1) SWQC grouping
2) Target setting
3) Orientation
4) Data collection
5) Cause analysis
6) Consideration of radical measures
7) Filling in the SWQC report
8) Proposal and implementation

9) Reporting
10) Publication of excellent results
11) Management evaluation and awards
12) Repetitive enforcement

Source: Mizuno 1983, p. 70.

According to Mizuno, NEC programmers at first resisted the idea of joining quality circles. "Every programmer was against me," he recalled in a recent interview. "They're white-collar workers and they protested that quality control was a blue-collar problem." Eventually, however, he and other NEC managers persuaded 95% of NEC's software developers to join the groups.[94]

NEC managers, such as Azuma, promoted the use of quality circles--essentially teams of people studying together -- as a means of reducing differences in skills among programmers and improving the performance of weaker members. The groups were only part of this effort, however. Also used to minimize differences among programmers were extensive training in structured programming techniques, encouragement of proofreading, program standardization, and use of software tools.[95]

Mizuno devoted so much attention to the SWQC program and quality circles because he considered them essential not only for quality control but also for individual and team motivation. In addition to helping people communicate and cooperate in teams, the program provided considerable recognition for achievement through SWQC group conferences held twice a year and an awards conference held annually, as well as through a series of prizes given to teams. In a 1983 article, Mizuno reported significant results from SWQC activities in improving quality and productivity in each of NEC's divisions developing software:

SWQC PROGRAM RESULTS

GROUP	DIVISION	TARGET	RESULTS
1	Switching	Machine time for debug	Down 1/3
2	Transmission control	Bug ratio	1.37/KS to 0.41/KS
3	Minicomputers	Bug Ratio	0.35/KS to 0.20/KS
4	Large OS	Bug ratio Source Size Object Size	6/Month to 0.9/Month 20KS to 8KS 72KB to 26KB
5	Large Applications	Spec changes	Down 40%

Source: Mizuno 1983, p. 71.

Along with conventional QC techniques, NEC trained its programmers in a standardized set of metrics, methods, and tools for quantitative software quality control it called SQMAT (Software Quality Measurement and Assessment Technology).[96] This was based on the SQM (Software Quality Metrics) system developed by Gerald Murine, a U.S. consultant. Developing SQMAT was formally the responsibility of the Quality Assurance Task Group organized in 1982, which worked under the previously established Software Productivity Committee. SQMAT procedures called for the determination of quality targets before each phase of development; planning meetings to discuss quality criteria; checkpoints for measuring quality; and action plans for corrective measures before proceeding on to the next phase. Factors analyze included an interrelated set of "Software Quality Design Criteria" (SQDC) and "Software Quality Requirements Criteria" (SQRC), as noted below. SQRC factors were ranked in order of importance, and measured through several quantitative methods.[97]

RELATIONSHIP BETWEEN SOFTWARE QUALITY REQUIREMENTS CRITERIA AND SOFTWARE QUALITY DESIGN CRITERIA

Key: C = Correctness; R = Reliability; M = Maintainability; F = Flexibility; U = Usability; E = Efficiency; S = Security; I = Interdependability

SQDC: \ SQRC:	C	R	M	F	U	E	S	I
Traceability	0							
Completeness	0							
Consistency	0	0	0					
Simplicity		0	0					
Accuracy		0						
Error Tolerance		0						
Modularity			0	0				
Self-Descriptiveness			0	0				
Conciseness			0					
Instrumentation			0					
Generality				0				
Expandability				0				
Training					0			
Communicativeness					0			
Operability					0			
Machine Independence				0				
Software System Independence				0				
Execution Efficiency						0		
Storage Efficiency						0		
Access Control							0	
Access Audit							0	
Data Commonality								0
Communication Commonality								0

Source: Sunazuka, Azuma, and Yamagishi 1985, p. 5.

CONCLUSION

Despite its early reliance on U.S. computer and software producers, such as Honeywell and SDC, NEC managers insisted on developing a distinctive approach to software production. Influence from hardware production and traditional factory models was strong, seen in the emphasis on standardization, automation, reusability of components, and linking of quality control to productivity. Mizuno and other managers also felt there were better process-management and control techniques possible for software, in contrast to the craft- or art-like approaches often followed in the industry. The historical evolution of software engineering in NEC proceeded roughly as follows:

-- data collection and process analysis

-- formulation of standard times to improve estimation and cost control

-- centralization of development for different product groups in permanent factories, rather than management by projects

-- processes (procedures and tools) optimized for different product groups

-- standardization of factory procedures and tools

-- training of workers in standardized procedures and tools

-- gradual decentralization of development through establishment of subsidiaries, satellite offices, and subcontracting

-- formalization and centralization of process R&D

-- formal quality assurance training and control program

-- development of automated tools

-- promotion of reuse of standardized components

-- flexible customization

-- continual evolution of tools, procedures, management systems

-- continual improvement of quality and productivity.

NEC did not try to model software development around hardware engineering and manufacturing processes as closely as firms such as Hitachi and Toshiba did. Rather, the company developed different factory strategies and structures for different types of software produced in numerous locations. The result was several factory-like systems for its main product groups -- STEPS for applications in general, STEPS with SEA-I for well-understood business programs, HEART for operating systems, and SDMS for switching systems. Distinctive about NEC's approach was also top-down direction from executive committees, a central laboratory, and the quality assurance department in the basic software development division. Progress occurred steadily even though distribution of development activities across so many different sites and subsidiaries, as well as the tendency of departments to prefer established tools and methods, made it difficult for the central committee and Software Product Engineering Laboratory to impose new systems such as SDMS.

The pattern of process development in NEC also occurred, more or less, at other Japanese firms that built software factories. Furthermore, this type of strategic and organizational evolution -- process analysis, centralization, and elimination of project management preceding advances such as standardization, distributed development, automation, quality assurance, standardization of components, and flexible customization -- is perhaps applicable to any firm that wants to move beyond job-shop approaches and "rationalize" the engineering and production process, but still retain flexibility in product development.

REFERENCES

1. The current version of the main paper from this research project is titled Michael A. Cusumano, "The Factory Approach to Large-Scale Software Development: Implications for Strategy, Technology, and Structure," Sloan School of Management, 1987 Working Paper #1885-87 Revised. Other completed papers are "A U.S. 'Software Factory' Experiment: System Development Corporation," Sloan School of Management, 1987 Working Paper #1887-87 Revised; "Hitachi: Pioneering a 'Factory' Strategy and Structure for Large-Scale Software Development," Sloan School of Management, 1987 Working Paper #1886-87 Revised; and "Toshiba's Fuchu Software Factory: Strategy, Technology, and Organization," Sloan School of Management, 1987 Working Paper #1939-87.

2. NEC Corporation Annual Report 1987.

3. Japan Economic Journal, 13 December 1986, p. 22; interview with Kaneda Hiromu, former NEC managing director and current corporate advisor, 9/26/85.

4. Datamation, 15 June 1987, pp. 30-32.

5. Interview with Mizuno Yukio, 9/26/85; and Usui Kenji, "NEAC kaihatsu shi (2)" (History of the development of the NEAC), Computopia, September 1975, p. 17.

6. Usui, p. 21; Nippon Denki Kabushiki Kaisha, "Konpyuta sangyo no genjyo to NEC" (The current state of the computer industry and NEC), NEC Information Processing Planning Office, July 1985. The initial ACOS models were designed in cooperation with Toshiba.

7. Usui, pp. 22-23.

8. Usui, pp. 22-23; Nippon Denki Kabushiki Kaisha, Nippon Denshi saikin 10 nen shi (A history of the last 10 years of Nippon Denki), NE$C, 1980, p. 86.

9. Mizuno interview, 9/26/85.

10. Kaneda interview, 9/26/85.

11. Fujino interview, 9/8/87.

12. Mizuno Yukio and Mitsugu Mamoru, "Sofutouea bijinesu no mirai" (Future of the software business), Konpyuta, April 1986, pp. 85-86.

13. Mizuno interview, 9/26/85.

14. Nippon Denshi saikin 10 nen shi, pp. 230-231; annual report, 1976.

15. Fujino interview, 9/8/87.

16. Fujino interview, 7/28/86.

17. Mizuno Yukio, "Sofutouea enjiniaringu no hitsuyosei" (The need for software engineering", Joho shori, Vol. 16, No. 10 (October 1975), pp. 836-837.

18. Mizuno 1975, pp. 838-839.

19. Mizuno 1986, p. 91.

20. Mizuno 1986, p. 92.

21. Kiichi Fujino, "Software Development for Computers and Communications at NEC," Computer, November 1984, p. 62.

22. Fujino interview, 7/28/86.

23. Fujino 1984, p. 57.

24. Fujino interview, 9/8/87.

25. Fujino, pp. 57, 62.

26. Fujino interview, 7/28/86.

27. Azuma interview, 10/1/86.

28. Hirai interview; also, Nobuyoshi Tsuchiya et al., "A Control and Management System for the C&C Satellite Office," NEC Research and Development, No. 81 (April 1986), pp. 19-23; and Sen Nakabayashi et al., "C&C Satellite Office and Networks," NEC Research and Development, No. 81 (April 1986), pp. 8-18.

29. Fujino interview, 7/28/86; Mizuno interview, 9/26/85.

30. Azuma interview, 7/28/86; Mizuno interview, 9/26/85.

31. Azuma interview, 9/26/85.

32. Fujino interview, 7/28/86.

33. Motoei Azuma interview, 7/28/87. Azuma at the time was manager of the Software Management Engineering Department in the Software Product Engineering Laboratory.

34. K. Honda et al. (NEC Software Product Engineering Laboratory), "Research on Work Environment for Software Productivity Improvement," COMPSAC'85, 1985. pp. 1-2.

35. Honda 1985, pp. 3-8.

36. Azuma interview, 7/28/86.

37. Fujino interview, 7/28/86; Mizuno 1983, p. 69.

38. Fujino interview, 7/28/86.

39. Yukio Mizuno, "Software Quality Improvement," Computer, March 1983, p. 68.

40. Fujino interview, 7/28/86.

41. Azuma interview, 7/28/87.

42. Azuma interview, 7/28/86.

43. Mizuno 1983, p. 69.

44. Interviews with Fujino and Azuma, 7/28/86.

45. Interviews with Azuma and Fujino, 7/28/86.

46. Fujino interview, 7/29/86.

47. Interviews with Horie Susumu, Engineering Section Manager, Basic Software Division, NEC Software Ltd., and Yamaguchi Hiroshi, General Manager, Basic Software and Application Software Development Division, NEC Software Ltd., 9/2/87.

48. Interviews with Horie and Yamaguchi (NEC Software) and Akira Yamada, Manager, Software Education Dept., NEC, 9/2/87.

49. Azuma and Mizuno 1984, p. 88.

50. Azuma and Mizuno 1984, pp. 88-89.

51. M. Azuma and Y. Mizuno, "STEPS: Integrated Software Standards and Its Productivity Impact," Proceedings of COMPCON 1981, IEEE, p. 83.

52. Fujino 1984, p. 59.

53. Azuma and Mizuno 1981, pp. 85-86.

54. Azuma and Mizuno 1984, p. 86.

55. Fujino interview, 9/8/87.

56. Azuma and Mizuno 1984, p. 87-88.

57. Azuma and Mizuno 1984, p. 91.

58. Interviews with Hirai Yozo, Manager. Quality Assurance Dept., Basic Software Development Division, NEC, 9/8/87.

59. Hirai interview.

60. M. Azuma et al., "SPD: A Humanized Documentation Technology," Proceedings of the IEEE Computer Society's International Computer Software and Applications Conference (COMPSAC), November 7-11, 1983, p. 308.

61. Azuma and Mizuno 1984, p. 91.

62. Fujino 1984, p. 59.

63. "NE Probes AD System for Software," Electronic Engineering Times, 11 February 1985, p. 66.

64. Masao Matsumoto, "SEA/I Application Software Productivity System," IEEE SoftFair '83 Proceedings, p. 378.

65. Masao Matsumoto, pp. 376-377.

66. M. Matsumoto, pp. 378-379.

67. "NE Probes AD System for Software," Electronic Engineering Times, 11 February 1985, p. 66.

68. Hiroyuki Kitagawa et al., "Form Document Management System SPEDOQ -- Its Architecture and Implementation," Proceedings 2nd ACM-SIGOA Conference on Office Information Systems, June 1984 (Toronto).

69. Hirai interview.

70. Interviews with Hirai and Fujino, 9/8/87.

71. Fujino interview, 9/8/87.

72. Hirai Interview.

73. NEC Corporation, "QA System in NEC: Scientific Control of Production and Quality in NEC -- Basic Software," unpublished internal document, September 8, 1987, pp. 3-9.

74. NEC Corporation, Software Product Engineering Laboratory, "SPD (Structured Programming Diagram)," undated manuscript.

75. Hirai interview.

76. Fujino interview, 9/8/87.

77."QA System in NEC," pp. 10-12.

78. Hirai interview.

79. Hirai interview.

80. Hirai interview.

81. Iwamoto Kanji and Okada Tadashi (NEC), "Purojekuto kanri tsuru" (Project control tools)," Joho shori, Vol. 20, No. 8 (August 1979), pp. 719-724.

82. Uenohara Michiyuki (NEC) et al., "Development of Software Production and Maintenance System," Research and Development in Japan Awarded the Okochi Memorial Prize (Okochi Memorial Foundation, 1984), pp. 26-32.

83. Uenohara et al., p. 32.

84. Hirai interview; Fujino interview, 9/8/87; Kanji Iwamoto (NEC), et al., "Early Experiences Regarding SDMS Introduction into Software Production Sites," NEC Research and Development, January 1983, p. 54.

85. Iwamoto 1983, pp. 54-59.

86. See J.W. Bailey and V.R. Basili, "A Meta Model for Software Development Resource Expenditures," Proceedings ICSE, Vol. 5, pp. 107-116 (March 1981).

87. Mizuno interview, 9/26/85.

88. Yukio Mizuno, "A Quantitative Approach to Software Quality and Productivity Improvement," NEC Corporation, ca. 1985, pp. 2-3. See also B. W. Boehm, Software Engineering Economics, Englewood Cliffs, N.J., Prentice-Hall, 1981.

89. Mizuno 1985, p. 6.

90. Mizuno interview, 9/26/85; Software Product Engineering Laboratory, "Sofutouea kanrisha shien tsuru TOMATO" (Software managers' support tool, TOMATO), undated manuscript.

91. This section is based on "QA System in NEC," pp. 14-25.

92. Fujino interview, 9/8/87.

93. This section is based on Mizuno 1983, pp. 69-71.

94. Bro Uttal, "Japan's Persistent Software Gap," Fortune, 15 October 1984, p. 154.

95. Azuma interview, 9/26/85.

96. Fujino interview, 9/8/87.

97. Toshihiko Sunazuka, Motoei Azuma, and Noriko Yamagishi, "Software Quality Assessment Technology," Proceedings ICSE, 1985, pp. 1-7.