日本科学技術

M.I.T. - JAPAN SCIENCE AND TECHNOLOGY PROGRAM

HITACHI: PIONEERING THE "FACTORY" MODEL FOR LARGE-SCALE SOFTWARE DEVELOPMENT

Michael A. Cusumano

Sloan School of Management
Massachusetts Institute of Technology

MITJSTP 87-07

HITACHI: PIONEERING THE "FACTORY" MODEL FOR LARGE-SCALE SOFTWARE DEVELOPMENT

Michael A. Cusumano

Sloan School of Management
Massachusetts Institute of Technology

MITJSTP 87-07

Michael A. Cusumano
MIT Sloan School of Management
Working Paper #1886-87

5/31/87
Software Project Paper #3

HITACHI: PIONEERING THE "FACTORY" MODEL FOR LARGE-SCALE SOFTWARE DEVELOPMENT

Contents:

Introduction
I. Comparison to the Survey Criteria
II. Hitachi: The Strategic and Structural Setting
III. Software Strategy: The Factory Model
IV. Policy and Management Infrastructure
V. Technological Infrastructure: Computer-Aided Tools and Systems
VI. Additional Flexibility through Subsidiaries
Conclusion
Appendix I: Survey Data Tables
Appendix II: Toshiba and NEC

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 38 software facilties 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 observable in a statistically significant sample of 38 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 managing the development process more effectively than others. (3) The approach to developing a technological infrastructure to aid software development is 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 applying a disciplined and flexible "factory" approach -- production-management concepts, general-use tools, standardized procdures, effective quality-control techniques -- to large-scale software development. Other U.S. firms relatively close to the flexible factory model are TRW and, to a lesser extent, Sperry and System Development Corporation (SDC), now both part of Unisys.

This paper presents the results of Hitachi's responses to the survey and then extends beyond this 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. Hitachi is significant for two reasons: One, its Software Works (originally a facility performing both systems and applications programming) was the first software factory established in the world, and Hitachi has made availabe extensive historical and technical documentation for this facility's technological and policy systems. Two, Hitachi has extended its factory approach to both applications and systems software development.

The organization of this paper is as follows. Continuing the introduction, the first section presents the survey criteria and results, and examines the Hitachi responses and comments for individual critiera. The subsequent sections focus on the motivations behind the founding of Hitachi Software Works in the late-1960s, and organizational and technological development between 1969 and 1986, including the institution of various controls and support-systems for production management and product engineering. The conclusion contrasts Hitachi's implementation process with System Development Corporation in the U.S. and notes the historical focus on development factory standards first, that is, before investing heavily in a factory technological infrastructure. It also reviews the Hitachi case in light of several theoretical benefits the factory approach might provide: dissemination of good technologies and practices; enhanced focus on productivity and product cost/performance; reduction in individual dysfunctional behavior; improvement in process management and control.

I. COMPARISON TO THE SURVEY CRITERIA

In the survey of 38 software facilities in the U.S. and Japan, utilizing criteria extrapolated from System Development Corporation's Software Factory Experiment in the mid-1970s, Hitachi Omori Works and Hitachi

Software Works were just above average on the scale toward the "flexible factory" model (see Appendix data tables). Overall, Omori ranked 12th (75%, 8% above the sample mean but average for Japanese facilities); and the Software Works 14th (73%). In the technology area, they ranked, respectively, 15th and 19th; in the policy/methodology area, 13th and 15th. Among the 17 Japanese facilities responding, they were about average, ranking 9th and 11th. Thus, though Hitachi has been the historical world leader in promoting the factory approach, judged by the survey criteria, it was not pursuing this strategy as rigorously as other firms. The specific responses to individual criteria examined below are abbreviated as Applications (indicating Omori Software Works) and Systems (indicated Hitachi Software Works).[2] Later sections will elaborate on the tools or capabilities described.

SURVEY ANSWERS KEY:

4 =	CAPABILITY OR POLICY IS FULLY USED OR ENFORCED
3 =	CAPABILITY OR POLICY IS FREQUENTLY USED OR ENFORCED
2 =	CAPABILITY OR POLICY IS SOMETIMES USED OR ENFORCED
1 =	CAPABILITY OR POLICY IS SELDOM USED OR ENFORCED
0 =	CAPABILITY OR POLICY IS NOT USED

n = 38 (Jap. = 17, U.S. = 21)

I. TECHNOLOGY/FACILITY INFRASTRUCTURE

ALL COMPANIES/FACILITIES

	All Companies		Japanese		U.S.	
Question	Mean	S. Dev.	Mean	S. Dev.	Mean	S. Dev.
A	3.47	0.62	3.38	0.65	3.55	0.58
B	3.45	0.71	3.69	0.48	3.25	0.79
C	3.07	1.02	2.97	0.87	3.15	1.13
D	2.55	1.04	2.99	0.67	2.20	1.14

E	2.68	1.18	2.44	1.17	2.88	1.15
F	3.04	1.08	3.40	0.86	2.75	1.14
G	2.67	1.25	2.94	1.08	2.45	1.34
H	1.85	1.06	2.37	0.82	1.43	1.04

APPLICATIONS COMPANIES/FACILITIES

	All Companies		Japanese		U.S.	
Question	Mean	S. Dev.	Mean	S. Dev.	Mean	S. Dev.
A	3.35	0.62	3.15	0.71	3.50	0.50
B	3.31	0.76	3.58	0.52	3.12	0.84
C	2.97	1.10	2.70	0.93	3.16	1.18
D	2.54	0.93	2.83	0.67	2.33	1.03
E	2.65	1.25	2.20	1.27	2.96	1.14
F	2.84	1.18	3.23	0.98	2.56	1.24
G	2.54	1.36	2.75	1.66	2.39	1.39
H	1.86	1.04	2.38	0.64	1.50	1.12

SYSTEMS COMPANIES/FACILITIES

	All Companies		Japanese		U.S.	
Question	Mean	S. Dev.	Mean	S. Dev.	Mean	S. Dev.
A	3.68	0.55	3.71	0.36	3.64	0.69
B	3.68	0.52	3.86	0.35	3.50	0.60
C	3.25	0.84	3.36	0.58	3.14	1.03
D	2.57	1.19	3.21	0.59	1.93	1.29
E	2.75	1.05	2.79	0.92	2.71	1.16
F	3.39	0.74	3.64	0.58	3.14	0.79
G	2.89	1.00	3.21	0.59	2.57	1.21
H	1.82	1.08	2.36	1.03	1.29	0.84

A. Centralization of development for a distinct software product family in a single location or directly linked sites operating as an integrated unit, rather than decentralizing development in independent sites.

Applications:	4
Systems:	4

These answers were above the sample average, though systems companies and facilities averaged nearly 4. It was Hitachi company policy to centralize both systems software development (in the Software Works) and large-scale applications (at Omori).

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

Applications:	4
Systems:	4

These answers were above the sample average, though common for Japanese firms, especially systems facilities. Both Hitachi facilities used established "factory standardards" for design documentation, coding methods, testing methods, etc.

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

Applications:	3
Systems:	3

Average responses. Hitachi did not centralize these for the entire facilities but they did for each project.

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

Applications:	3
Systems:	4

Above average. Centralized control was provided through the Computer-Aided Production Control System (CAPS). [Use of this was not mandatory in Omori, apparently to provide development groups with more flexibility in project management for customized software.]

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

Applications:	1
Systems:	2

Below average responses. The low score at Omori reflected the wide variety of projects and customers, but was low even for an applications facility.

F. A specific group or groups designated to develop and disseminate methodologies and tools to automate tasks such as requirements specification and design, coding, documentation, system testing and debugging, as well as to facilitate standardization of practices and division of labor, and effective managerial control over all programming activities.

Applications:	4
Systems:	4

Above average, though systems facilities usually scored 4. In both Hitachi facilities, there were production engineering groups that performed this function.

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

Applications:	4
Systems:	2

The Systems response was below average and especially low for a Japanese firm. The Applications response was especially high for an applications facility. According to the survey respondent, Omori was more advanced in this area due to the development of EAGLE (Effective Approach to Achieving High Level Software Productivity), an integrated program-generating tool and methodology that provided a standardized interface for applications software development.

H. Automated or semi-automated integration of applicable data from support tools and development data bases with management control systems (project data bases and the central production data base), for each phase of program development; and the utilization of this capability to facilitate budgeting, forecasting, maintenance, and overall life-cyle cost control on current and future projects.

Applications:	2
Systems:	2

These responses were slightly above average for the sample, though somewhat low for Japanese facilities. The Hitachi managers noted that project progress control, and historical recording of program corrections or changes, were "mechanized" functions.

II. METHODOLOGY & POLICY INFRASTRUCTURE

ALL COMPANIES/FACILITIES

	All Companies		Japanese		U.S.	
Question	Mean	S. Dev.	Mean	S. Dev.	Mean	S. Dev.
A	1.77	1.20	1.85	1.14	1.71	1.24
B	2.50	1.30	2.98	1.08	2.16	1.34
C	3.33	0.89	3.55	0.65	3.18	0.99
D	2.00	1.11	2.05	1.21	1.96	1.03
E	2.81	1.07	3.20	0.95	2.54	1.06
F	2.67	0.98	3.30	0.60	2.21	0.94
G	2.46	1.16	2.75	1.17	2.25	1.11
H	2.07	1.28	2.88	0.85	1.50	1.22
I	1.99	1.04	2.58	0.87	1.57	0.94
J	2.53	1.25	2.90	1.24	2.26	1.19
K	1.94	1.27	2.33	1.43	1.67	1.05
L	2.90	1.17	2.75	1.29	3.01	1.06
M	3.18	1.10	3.65	0.63	2.85	1.24
N	2.71	1.03	3.13	0.75	2.42	1.10
O	2.61	1.13	3.48	0.53	2.00	1.04

APPLICATIONS COMPANIES/FACILITIES

	All Companies		Japanese		U.S.	
Question	Mean	S. Dev.	Mean	S. Dev.	Mean	S. Dev.
A	1.77	1.20	1.85	1.14	1.71	1.24
B	2.50	1.30	2.98	1.08	2.16	1.34
C	3.33	0.89	3.55	0.65	3.18	0.99
D	2.00	1.11	2.05	1.21	1.96	1.03
E	2.81	1.07	3.20	0.95	2.54	1.06
F	2.67	0.98	3.30	0.60	2.21	0.94
G	2.46	1.16	2.75	1.17	2.25	1.11
H	2.07	1.28	2.88	0.85	1.50	1.22
I	1.99	1.04	2.58	0.87	1.57	0.94
J	2.53	1.25	2.90	1.24	2.26	1.19
K	1.94	1.27	2.33	1.43	1.67	1.05
L	2.90	1.17	2.75	1.29	3.01	1.06
M	3.18	1.10	3.65	0.63	2.85	1.24
N	2.71	1.03	3.13	0.75	2.42	1.10
O	2.61	1.13	3.48	0.53	2.00	1.04

SYSTEMS COMPANIES/FACILITIES

	All Companies		Japanese		U.S.	
Question	Mean	S. Dev.	Mean	S. Dev.	Mean	S. Dev.
A	2.21	1.03	2.43	1.08	2.00	0.93
B	2.50	1.04	2.86	0.95	2.14	0.99
C	3.86	0.40	3.93	0.17	3.79	0.52

D	2.43	1.13	2.79	1.19	2.07	0.94
E	3.21	0.65	3.36	0.69	3.07	0.56
F	2.75	0.96	3.29	0.52	2.21	0.99
G	2.54	1.13	2.71	0.84	2.36	1.33
H	2.21	1.47	3.57	0.42	0.86	0.69
I	2.21	1.24	2.64	1.19	1.79	1.13
J	2.54	1.20	3.36	1.03	1.71	0.70
K	2.18	1.42	3.14	1.16	1.21	0.92
L	3.36	0.61	3.36	0.79	3.36	0.35
M	3.61	0.43	3.71	0.36	3.50	0.46
N	2.54	1.19	3.07	0.94	2.00	1.16
O	2.39	1.18	2.86	1.38	1.93	0.68

A. Use of a standardized design language

Applications:	3
Systems:	2

Hitachi's Systems response was average for the sample. Omori's response was high for an applications facility.

B. Use of a standardized module-specification language

Applications:	3
Systems:	3

These were slightly above average for the sample and average for Japanese facilities.

C. Use of a standardized coding language

Applications:	4
Systems:	4

Slightly above average. Most firms standardized this.

D. Emphasis on high-level abstraction (data-type or procedure abstraction; object rather than variable orientation)

Applications:	2
Systems:	2

Average, though the systems response was low for a Japanese facility.

E. Planning for maintainability at the module-design level

Applications:	3
Systems:	3

Average for the sample.

F. Planning for reusability at the module-design level

Applications:	3
Systems:	3

Average for the sample.

G. Planning for portability at the module-design level

Applications:	3
Systems:	2

Applications was average; Softwar Works was somewhat low for a systems facility.

H. Monitoring of how much code is being reused

Applications:	3
Systems:	4

These were high for the sample but average responses for Japanese applications and systems facilities. Omori used an automatic monitoring tool, although it did not place as much emphasis on keeping track of reuse as the Software Works, which recently initiated an effort to monitor and promote reuse of code.

I. "Layering" of reused modules from the program library, along with newly written code, to create new programs

Applications:	3
Systems:	2

These were average responses for the sample, though Software Works was somewhat lower than other Japanese systems facilities.

J. Cataloging for the program library of common functional modules (e.g. a date verification routine)

Applications:	3

Systems:	4

Somewhat above average for the sample, especially the Systems response.

K. Cataloging for the program library of data abstraction modules (e.g. table or linked-list managers)

Applications:	3
Systems:	3

Above average for the sample, although the Software Works' response was average for a Japanese systems facility.

L. Writing of documentation to accompany modules placed in the program library

Applications:	2
Systems:	2

Below average for the sample.

M. Requirement that, if changes are made in the code of a module in the program library, the documentation must also be changed

Applications:	4
Systems:	4

Slightly above average for the sample, though average for Japanese facilities or systems facilities.

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

Applications:	2
Systems:	2

Below average responses.

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

Applications:	3

Systems: 2

Somewhat below average responses.

II. HITACHI: THE STRATEGIC AND STRUCTURAL SETTING

A. Corporate Organization and Products

Hitachi originated in 1908 as the machinery repair section of a mining company in the town of Hitachi, Japan, a couple hours by train north of Tokyo. In 1986 it had approximately 80,000 employees and sales in the neighborhood of $20 billion dollars. Hitachi's major area of business was communications and electronics equipment, including computers and software (36% of fiscal 1985 sales), although the company also sold heavy machinery (21%), home appliances (24%), industrial machinery (9%), and telephone exchange equipment and other products (10%).[3] In computer sales among Japanese companies during 1985, Hitachi ranked fourth, behind Fujitsu, IBM Japan, and NEC, but was traditionally the market leader in large mainframes and second to IBM in very-large mainframes.[4]

For most of its history, Hitachi's organization has centered around factories, of which the company operated 28 domestically in 1986. These belonged to 7 groups: Computers, Electronic Devices, Consumer Products, Industrial Components and Equipment, Industrial Processes, Power Generation and Transmission, and International Operations. Group headquarters retained responsiblity for sales, but factories have been responsible for product engineering and manufacturing. Factories also operate as independent profit centers, with financial management based on 6-month budgets for each factory. Plant managers are thus responsible for engineering and production costs, the setting of production amounts, and any related expenses; and company policy has required factory managers to institute standardized controls and procedures for budgets as well as engineering and manufacturing management. There have been no exceptions, not even in the case of software.[5] This is what led to the birth of the world's first software factory.

B. The Computer Group

Computer exports for Hitachi have been relatively small in comparison to domestic sales (about 17% in 1985).[6] Mainframes are designed to compete specifically with IBM models as well as to be fully compatible, and appeared to be of unique designs. For example, the AS/9000, introduced around 1982 to compete with IBM's 3081 model, used denser circuitry and a shorter data-flow path than IBM to provide considerably more computing power for the dollar. It was also, according to **Datamation**, "a more expandable and more cost-effective mainframe, with expanded performance levels when compared to a similar IBM system."[7] Hitachi computers introduced to compete with IBM's new 3090 Sierra series, the AS/XL 60 and 80, also achieved computing speeds equivalent to the IBM machines with half the number of processors and at a lower price.[8]

The 1982 incident in which Hitachi engineers were caught by the FBI attempting to buy information on the 3081 operating system, particularly new features that IBM had decided to imbed in microcode ("firmware"), suggests that Hitachi has actively sought information on IBM machines and software to help its hardware and software engineers design compatible products.[9] This process of information gathering or even "reverse engineering" may have also aided the performance of Hitachi mainframes and software.

It is also the case, however, that underlying Hitachi's apparent technical success in hardware and software is a long history of computer development. Hitachi engineers began experimenting with this technology in the mid-1950s and completed their first model in 1957, using parametrons (a solid-state device used primarily in Japan during the 1950s), and then a transistorized business computer in 1959. The model for this machine was developed at a Ministry of International Trade and Industry research institute, the Electrotechnical Laboratory (ETL), and largely completed during 1956 -- two years before the commercial introduction of transistorized computers in the United States. The main ETL designer, Takahashi Sigeru, helped transfer this technology to Hitachi and moved to the company formally in 1962, where he headed hardware product development until 1980.[10] Along with in-house research and product development, Hitachi also benefited from a licensing agreement with RCA between 1961 and 1970, through which it manufactured RCA-designed computers, as well as sold RCA software, for resale under the Hitachi label in Japan.

The production of computer products before 1961, along with the arrangement with RCA, reflected a dual strategy within Hitachi: independent development of new technology as well as direct purchasing of technology from abroad. This two-fold approach turned out to be extremely important: When RCA failed to introduce competitive new products in the late 1960s and then withdrew from computers in 1970, Hitachi had sufficient internal expertise to design machines that would eventually compete with the IBM 370 and subsequent mainframes. Equally important, Hitachi engineers had an opportunity to cultivate independent skills and develop a distinctive, factory-centered approach to software development.[11]

Hitachi's computer group in the mid-1980s consisted of six main facilities, two for software and four for hardware. The Software Works, which started with 348 employees in 1969, grew to nearly 3000 before being separated into two sites during 1985. It has continued to produce operating systems for mainframes, mini-computers, and office computers; related systems software (such as language processors); and on-line data-base programs. The smallest programs were several thousand lines of code and the largest several hundred thousand.[12] Omori Software Works produces large-scale customized applications programs such as real-time banking or factory control software. Research and development on new hardware technologies as well as software development tools and design concepts took place in two corporate facilities.[13] In addition, Hitachi had numerous subsidiaries producing computer-related products and services, including 23 software companies.[14]

HITACHI COMPUTER GROUP, CA. 1986

LINE FACILITY	EMPLOYEES	PRODUCTS
Hardware		
Kanagawa Works	2,800	Mainframe Computers
Odawara Works	2,400	Peripherals
Asahi Works	1,100	Small-Scale Computers
Device Development Center	80	Semiconductor Devices
Software		
Software Works	1,400	Systems Software

Omori Software Works	1,500	Applications Software
CORPORATE R&D		
Central Research Labs	1,200	Basic Research
Systems Development Laboratory	350	Systems and Applications Software Technology

C. Corporate Programs for Engineering and Manufacturing Improvement

The Software Works, as a Hitachi factory, takes part in all company-wide efforts at analyzing and improving various aspects of engineering and manufacturing operations. Several corporate programs appear to have led to an increasing refinement and improvement of the factory concept (technology and procedures) for software, and of engineering performance in this area. For example, a company-wide movement among Hitachi factories since 1968 has been the "Management Improvement" (MI) program. The major focus of this has been to promote the establishment and implementation of specific standardzation, "zero defect," and productivity-improvement objectives.[15] At the Software Works, during 1969-1971, this movement took the form of setting standards for design, programming, and testing activities, introducing a document control system and a zero-defect program, and launching a study of how to reduce personnel costs and increase programmer performance. As a next step, in 1973 managers asked all planning personnel to submit suggestions on how to improve productivity; this resulted in 1437 proposals, some of which were adopted quickly -- such as structured programming techniques and better debugging methods.

Also under the Management Improvement program, during the later 1970s, the Software Works launched studies of design productivity, reliability, profit generation, and market share.[16] Management formally organized these efforts through the factory staff structure, such as a Rationalization Promotion Center in 1975 (headed by the factory manager).[17] The company-wide focus in recent years has centered on three specific areas and potential ways to integrate and improve productivity in product engineering and production; Hitachi has equally applied the concepts and recommendations in hardware and software facilities:[18]

Area	**Main Objective**	**Solutions**
Design technology	Shorter times	CAD Standardization
Production engineering	Labor reduction	Automation Process improvement
Control technology	Less work-in-process	Inventory control

Another example is quality assurance. Since the founding of the company, Hitachi has followed a practice called "gleaning," which involves picking out product- or system-design errors, analyzing them, and then formally recommending solutions and making reports to colleagues. Factories have case reports once a month; there are also reports at the division level approximately once every other month. The Software Works adopted this practice in 1977, with the particular objective of developing design and analysis procedures that would reduce the recurrence of system problems identified by customers, such as not meeting user specifications or designing programs that were not "user friendly."[19]

III. SOFTWARE STRATEGY: THE FACTORY MODEL

A. The 1960S: Product Proliferation and Programmer Shortages

The first Hitachi computers of the late 1950s and early 1960s used drums for main memory, and paper tape for entering programs and data as well as receiving output. Thus, they did not require software except for simple input/output programs and a few subroutines for scientific calculations. With the inclusion of core memory and card readers during 1963-1965, it became possible to use higher-level languages such as FORTRAN and to write more sophisticated programs. Yet the hardware still had no interrupt features, so control programs were small. The first program resembling a modern operating system for a Hitachi computer was a Fortran "monitor" system introduced with the HITAC 4010 in 1965.[20] But this was actually an RCA product (model 401), which Hitachi produced from imported knock-down kits; Hitachi required little product engineering or software knowledge, except to be able to service the machine.[21]

An in-house project, on the other hand, provided Hitachi engineers with extensive experience in both hardware and software development, as well as began to strain engineering resources. In the early 1960s, Hitachi's Central Research Laboratory took on contracts with Tokyo University, the Japanese Meteorological Agency, and Nippon Telegraph and Telephone's main laboratory to build a very-large scale computer capable of time sharing, dubbed the HITAC 5020. The Central Laboratory completed one unit for its own use in 1964 and then, under the direction of Shimada Shozo, set out to produce an operating ("monitor") system that would allow the 5020 to perform input, output, and computation functions simultaneously. Laboratory engineers had previous experience developing an assembler and FORTRAN compiler for Hitachi's parametron computers; between 20 and 30 were assigned to work on software for the 5020. The Central Laboratory was one of two sources of computer expertise in Hitachi at the time; the other was the Totsuka Works, which produced telecommunications equipment and had led the company's entrance into computers during the 1950s.[22]

Shimada's major source of ideas for the operating system software was MIT, where he and several other Hitachi engineers visited in 1965 on the introduction of a Tokyo University professor to the head of MIT's electrical engineering department. MIT researchers were then developing their own time-sharing system, **Multics**, using a GE mainframe. Shimada received a copy of the manual, which discussed several new approaches and ideas such as 2-level addresses and virtual memory. In Shimada's words, the **Multics** manual "actually made our mouths water." As soon as he returned to the Central Research Laboratory, he made the development of a comparable operating system his next project, in cooperation with Tokyo University's Computing Center. The first delivery of the 5020 was in 1965, to Tokyo University.[23] They finished a Japanese version of Multics in 1968, a couple years before MIT.[24]

The 5020 was not suited for businesses and the project team became short-handed as Hitachi management gave priority to developing system software for the HITAC 8000 series.[25] Introduced during 1967-1969, the 8000 family was a Japanese version of the RCA Spectra series (which was partially compatible with the IBM 360). The 8000 also provided a major incentive to create a formal strategy and mechanism for program development, because RCA was not developing adequate system software. Hitachi decided at first to use the RCA operating system, TDOS, but this required at least two magnetic-tape stations for compilers and the program library. In contrast, a major feature of the IBM 360 was that all functions were available on a faster and larger disc drive system. While RCA hesitated over whether or not to develop a disc system, Japanese customers insisted on this, prompting Hitachi to start modifying RCA's TDOS around 1966 and create a new "disc operating system," DOS.[26]

Designing an effective disc operating system capable of on-line processing exacerbated the strain on software-engineering resources in Hitachi. The manager of the project, Sakata Kazuyuki, found 80 engineers to work on the system, with assistance from Hitachi's Central Research Laboratory, the Totsuka Works, two subsidiaries (Hitachi Electronics Service and Hitachi Electronics Engineering), and a subcontractor, Yoshizawa Business Machines. (The groups from Hitachi Electronics Engineering and Yoshizawa remained together and formed the basis of the company's largest software subsidiary, Hitachi Software Engineering, established in 1969.)[27] Both TDOS and DOS provided the basic structure of EDOS, which allowed for greater volume on-line and large-scale batch processing and was completed in 1969; this became the foundation for Hitachi's current operating system for large-scale computers.[28]

Yet another software project Hitachi tackled in the 1960s was an operating system for a project sponsored by MITI and Nippon Telegraph and Telephone (NT&T) to build a very-large scale computer, called the HITAC 8700/8800 within Hitachi (the NT&T version, the 8800, was to be used for telecommunications data processing). Development work for the software started in 1968 at the Kanagawa Works and was then taken over by the Software Works in 1969. The commercial operating system that resulted from this project, OS7, had multi-processor, multi-virtual memory capabilities, as well as supported large-scale batch processing, time sharing, and on-line real-time computing.[29] The first commercial deliveries came in 1972-1973, primarily to universities and research institutes.[30] The computer fell short of several performance goals and was not nearly as powerful as the 370 series, which IBM introduced while the 8700/8800 was in development. Nonetheless, the project provided Hitachi with extensive experience in hardware architecture design, integrated-circuit logic chip design, and large-scale software engineering.[31]

Systems programs were not the only software orders to Hitachi during this period. Since few companies in Japan outside of the computer manufacturers had in-house software expertise, Hitachi and the other mainframe producers had to design several large applications programs. In Hitachi's case, these included a series of real-time reservations systems for the Japan National Railways (the first programmable system Hitachi delivered in 1964, with 1100 terminals throughout Japan); on-line currency-exchange and deposit systems for the Tokai Bank (1965) and Sanwa Bank (1967); and real-time production control systems for Toyo Kogyo (Mazda) and Nissan (1968).[32]

The banking systems were particularly important, because most Japanese banks at the time were buying these from IBM; Hitachi's system marked the beginning of a shift to more domestic systems.[33] Developing the Tokai software, which connected 200 remote terminals around Japan to a central processing center in Nagoya, was a particularly difficult but valuable learning experience, according to Sakata. Before taking on the job, he and other Hitachi engineers spent nearly two months in the U.S. during 1963-1964 to observe several American airline and banking systems, including Howard Savings in New Jersey and Continental Illinois in Chicago. They were thoroughly dismayed at how difficult the programming looked and, once they completed the initial Tokai system, it took a full year to get the software working properly. Due to the contract terms, Hitachi was not paid for this extra debugging time and had to absorb the costs itself.[34] The cost and frustrations of this project made Sakata and other managers particularly concerned about improving their ability to control schedules and time estimates, as well as bugs.

B. Evolution of the Factory Strategy

During the late 1950s, engineers at Hitachi's Totsuka Works, including Sakata, believed that, since computers relied on digital technology similar to the telephone exchange equipment they were already manufacturing, Hitachi would be able to manufacture computers independently.[35] This factory thus began hardware design in Hitachi, and also established the first group officially responsible for software development (mainly language processors and utility programs), the engineering service section. The section started in 1960 with about 10 engineers and in 1963 was divided into two planning sections, one for government and university business, and another for private contracts. The concept of "service" was intimately linked to software since, to sell computers, Hitachi had to learn from potential customers what their needs were and then be able to provide adequate programs. Closely related to this section was another group which trained technicians for maintenance.[36]

Hitachi management next decided to establish a separate computer division in 1962 along with a new factory to manufacture hardware, the Kanagawa Works (separated from the Totsuka Works). The hardware design section in the new plant took charge of writing or revising software for the new RCA machines Hitachi was planning to offer. But managers worried that the dispersion of a scarce resource -- software engineers -- would make it difficult to write software for the new 8000 series. This situation then led to the creation of centralized system program department in the Kanagawa plant, headed by Sakata and modeled after a similar department in RCA.[37] The new department formally brought together the group in the Central Research Laboratory that had been developing software for the 5020; the software engineers already in the Kanagawa Works; and a program section at the division level (although physically located in the Kanagawa Works) that had been studying programs for pre-Spectra series RCA machines produced in Japan. The core group consisted of about 60 engineers; Hitachi hired another 20 personnel, for a total of 80.[38]

Underlying the establishment of this department, according to the head of the design section and Sakata's successor as department manager in 1969, Fujinaka Satoshi, was also "the anticipation that we would develop software as a factory product." With the 8000 series going on sale and software for the 5020 yet to be delivered, noted Fujinaka, "Work was increasing so rapidly that the new structure couldn't catch up with it. Every day was a struggle with time."[39]

The next logical step was a software factory. In fact, rapid growth of the computer division caused acute shortages of space and personnel in both the hardware and software areas. The building housing the Kanagawa Works, located in Totsuka-cho, Yokohama, was expanded but this was still insufficient. As a result, Hitachi established a separate facility for peripherals (Odawara) in 1966 and purchased another site in Hadano, an hour or so by train from Totsuka, where it built its current mainfram plant (Kanagawa Works). The design and production departments moved to Hadano in August 1968, leaving most of the company's software departments at Totsuka (a few others, for some systems engineering and small-scale computers, remained within the division's staff organization until later in the 1970s). In February 1969, the Totsuka building was officially upgraded to a separate factory -- the first software facility in the world referred to as a "factory."[40]

According to the managers who operated the new factory, Komoto

Yukio (the first head of the Software Works), Nakatani Nobuo (his successor), and Sakata Kazuyuki (who served as deputy general manager during the 1970s), there were two reasons for following this strategy: One was the acute shortage of software engineers and the hope that centralizing software development in a single facility would bring about an increase in productivity. (Despite a nation-wide search for software engineers, they still had considerably less than their target to staff the new factory in 1969.) A second reason was their decision to stop treating software as simply an engineering service that went along with hardware but to view it as a separate product that could and should be produced and inspected in a disciplined, factory environment.[41] This was not a casual decision; managers discussed it at the highest levels of the company. Hitachi President Komai made the final decision to establish the Software Works as an independent factory, insisting that they maintain the tradition of factory profit centers. A debate within Hitachi ensued over the nature and name of the new facility; some engineers wanted to establish a "Software Center." But President Komai intervened and ordered that they "call it a factory."[42] This was despite the fact that no other company in the world had established a factory for software production, and Japanese university professors criticized Hitachi, maintaining that software was not sufficiently understood to be produced through engineering and factory methods.[43]

C. The Factory Architects

The key figure in the development of the factory's management-control system was Sakata Kazuyuki, the individual who had served as manager for several key projects as well as for the system program department (currently he is a senior managing director of Nippon Business Consultants, a Hitachi software subsidiary). Sakata had entered Hitachi in 1941 from a technical high school and gone to work as a machine operator in the Totsuka Works. After additional training at Hitachi's in-house engineering school and a two-year stint in the army, he joined Totsuka's production engineering department in November 1945 and began developing standard times for machining operations as well as studying job tasks, scheduling, and conveyor systems to improve productivity. In 1957, Sakata moved to the accounting department and got his first glimpse of a computer -- an IBM 421 tabulating machine. In 1960, he was reassigned and made the manager of a new computer inspection section, which had about 30 members. When Hitachi management separated computer development from the Totsuka Works in 1962 and established the Kanagawa plant, Sakata continued as manager of the inspection section, which was now located within the engineering department. The following year he became head of the computer division's engineering service department, which did systems engineering for Hitachi customers. Then, in 1965, with the establishment of the system program department, Sakata became responsible for software production and quality control.

Sakata's major frustrations were bugs in the software Hitachi was receiving from RCA, and the shortage of programmers, which he had to divide among the RCA machines, the 5020 project, and applications programs. A dozen Hitachi hardware engineers not working on the 5020 learned how to write software by studying and translating RCA's COBOL, FORTRAN, and ASSEMBLER manuals for the HITAC 3010 and 4010 machines; seven or eight then continued in the system program department reviewing the new programs from RCA amd correcting bugs before shipping the software to Hitachi customers. This experience, as well as his background in hardware production management and inspection, and in computer engineering service, convinced Sakata that there had to be a better way to produce programs and prevent breakdowns due to errors: setting the same quality standards for software as for other Hitachi products, and rejecting the notion that the nature of software was such that there would always be bugs. "Thus," Sakata recalled, "even though it was software, we called [the new facility] a factory."44

The key figure who became responsible for implementing Sakata's basic ideas was Shibata Kanji, currently the head of the engineering department in the Software Works. He first joined the engineering service section of Hitachi's computer division in 1964 after majoring in electrical engineering at Shinshu University, and later moved to the system program department and then the production administration section of the Software Works.

Shibata quickly became the in-house expert on software-engineering management soon after he joined Hitachi. One aspect of the company's training program for new engineers required them to take several months during their second year to write a paper on a theme related to their work, and then give a presentation. Shibata chose to collect data on programmers working on software for the RCA machines and the 5020 -- how much time they spent each day on different activities and different types of programs. Programmers did not like being watched closely or keeping records, recalled

Shibata, so they stopped doing this in 1966. But Sakata, Shibata's supervisor in the system program department, read his paper and decided this data was too valuable not to collect. Sakata then hired female employees to keep the records, which became the basis of the Software Works' production planning and control system.[45]

IV. POLICY AND MANAGEMENT INFRASTRUCTURE

A. Conceptualizing the Development Process

Hitachi engineers, despite the factory environment, conceived of the software development process in much the same way as other software engineers around the world: as primarily composed of design and testing activities (see figure below). Data on man-power allocations (total number of workers) per process at Hitachi Software Works ca. 1985 indicates this is an accurate conceptualization: roughly 50 to 55% went into planning and design, 5% to coding, 30-35% to debugging, and about 10% to inspection.[46]

HITACHI SOFTWARE WORKS:[47]

A. SYSTEM SOFTWARE PROCESS FLOW

Development Process	Inspection Process
Basic Design	Initial Inspection Planning
Functional Design	
Structural Design	
Coding	Documentation Planning
Stand-Alone Debugging	
Combinational Debugging	
Comprehensive Debugging	Inspection Program Compilation
Final Product Inspection	

B. CUSTOM APPLICATIONS SOFTWARE PROCESS FLOW

System Proposal Compilation
Demonstration
Estimate
System Construction/Consultation
  System Design
  Program Implementation
  Conversion
  System Test
Follow-Up Service

But a distinguishing development at Hitachi was that, with the decision to establish a software factory, managers became obligated to adopt company-wide accounting and management procedures and thus innovate in software engineering by devising systematic controls on the process flow, costs, and product quality. In Hitachi's hardware factories, the management systems evolved centering on standardization and components control. Sakata and Shibata believed it was possible to apply the same concepts to software.

In particular, Shibata wanted programmers to design modules that would serve as the equivalent of standardized hardware parts. "Around 1970 we came to believe that we had to introduce a components control system similar to what you find for hardware, and in the Software Works we established a committee to take this up as a special project." The committee members soon realized, however, that "software is not sufficiently standardized to be treated the same as hardware components control." They changed their priorities and decided that, first, they had to find a way to standardize product design, just as this had been done in hardware manufacturing, and then worry about components control. A special committee then started establishing standards for all activities, while the original committee adopted the name "Structured Programming Methods Committee," believing that structured programming techniques would provide a way to standardize the software design process. Company engineers next spent several years studying these techniques from available literature, as well as analyzing programs Hitachi had already written to find ways to improve the design structure. This was before structured programming became discussed more widely in industry journals and adopted by other companies.[48]

In addition to their central objective of standardization, the experience of Sakata and other Hitachi managers in hardware production had encouraged them to believe that improvements in productivity and quality (reductions in bugs) were most likely to come from better tools and management systems. In software, they viewed these as higher-level languages and modularization for long-term maintenance, as well as visible charts and documentation for better process control. Developing a "visualized" production control system became an especially important goal, because software engineering did not involve visible raw materials. To pursue these objectives, they decided to analyze, and then attempt to manage, the overall process of software development in much the same way as they saw hardware engineering and manufacturing: They developed factory systems that provided controls for production management, including man-power, process, quality, and product controls; and for product engineering, including standardization, design, and inspection. The controls involved a mixture of manual and automated support-tools and systems, with strong efforts during the late 1970s and 1980s toward computer-integrated production.[49] (See Table)

Initially, the motivation for pursuing factory-type production and product-engineering systems was to be able to inspect software products, like any other product Hitachi manufactured, and thereby be able to guarantee quality. But a side benefit of the system of controls, according to Shibata, turned out to be the "minimization of problems." This has resulted in significant improvements in productivity, thus indirectly addressing the shortage of skilled programmers.[50]

B. The Factory Organization

The Software Works began with three design departments for distinct types of programs: system development (systems engineering), system programs (basic software), and on-line programs (large-scale real-time applications programs for the National Railways, NT&T, banks, and other industrial or insitutional customers). Administrative functions were similar to those in hardware manufacturing plants: product planning, inspection, engineering, accounting, and general affairs, as well as training. Hitachi managers did not view the factory organization as static; over time, they have consolidated some of these functional areas, added design departments as software technology evolved (such as for artificial intelligence and computer graphics), and centralized all large-scale custom applications (railways, banking, industrial) development in a second facility, the Omori Software Works (officially separated in 1985).

HITACHI SOFTWARE WORKS: ORGANIZATION CHART, 1969[51]

DESIGN DEPARTMENTS

SYSTEM DEVELOPMENT
- Design Groups (6)

SYSTEM PROGRAMS
- Planning Group
- Design Groups (2)

ON-LINE PROGRAMS
- National Raliways (2 Groups)
- NT&T (4 Groups)
- Banking (2 Groups)
- Government etc. (1 Group)

ADMINISTRATIVE SECTIONS

Administration
Inspection
Engineering
Accounting and Control
- General Affairs

COMPUTER TECHNOLOGY SCHOOL

HITACHI SOFTWARE WORKS: ORGANIZATION CHART, 1986[52]

Product Planning Department

DESIGN DEPARTMENTS:
No. 1 Systems Programming
No. 2 Systems Programming
Database Programming
Data Communications Programming
Language Processor
Artificial Intelligence
Computer Graphics
Small-Scale Systems Programming

OTHER DEPARTMENTS:
Engineering
Documentation/Manual Development
NT&T Information Processing Systems
Inspection
Computer Center Service
Software Education Center
General Administration
Purchasing
Accounting and Control
Software Technology Center

In addition to overall organization, another flexible aspect of the factory was the ability of managers to move personnel freely between among groups within a department, such as if problems arose on a given project. (Departments generally had between 500 and 600 people, with the NT&T department being the largest (around 900); departments were then organized into groups of about 100 programmers, with sub-groups of about 30 members.) Managers could also appeal to engineering groups within each department to add manpower to help solve project-related difficulties, or they could appeal to the Engineering Department and the Software Technology Center for problems or develop tools considered to have factory-wide relevance.[53]

PRODUCT PLANNING

The Product Planning Department in the Software Works was launched in 1970 to centralize planning activities dispersed among the system program department, the large-scale program area, and the administration (control) section. Responsibilities included planning for products such as new operating systems, beginning with OS7, but also for exports. In 1974, for example, the department set up promotion conferences to determine policy for Hitachi's M series, which Hitachi was designed to compete directly with the IBM 370 family. These activities included preparing for OEM exports to Itel in the U.S. and studying how to make the Hitachi hardware fully IBM compatible. To assist in this effort, Hitachi also established in 1972 a Computer Liason Office in Mountain View, California, which the Product Planning Department in the Software Works administered directly. This office served as an "information pipeline" on IBM, replacing RCA, and in 1978 was absorbed by the San Francisco office of Hitachi America.[54] In the late 1970s, the department became involved in product pricing as Hitachi unbundled software from hardware, and in administration of overseas technical contracts.[55]

ENGINEERING (PRODUCTION ADMINISTRATION)

This department originated in the engineering department, software section, of the Kanagawa Works, and was moved in 1970 to the Software Works. It began with two sections (engineering and administration) and 36 members. The engineering section served largely as a liason group for the computer division, other Hitachi factories, and subcontractors, providing explanations and documentation on software-product pricing and progress on product development. The administration section was responsible for production management and process control (scheduling), as well as administration of the computer and software centers attached to the factory. This group set standard times and was responsbile for cost control and studying software productivity. It also helped develop control systems to monitor design and inspection, as well as other tools, in conjunction with the System Development Laboratory (established in 1975) and the Software Technology Center (established in 198X). General-use tools were always paid for out of the factory budget. Other sections of the original engineering department later became full departments: procurement, which purchased software from overseas and Japanese subcontractors; and the documentation/manuals section.[56]

INSPECTION

This department originated with the Software Works and has followed the strategy of developing inspection and control techniques based on actual operating data. The manager of this department reports directly to the factory head, as in all Hitachi factories. In addition to overseeing all testing and debugging, an important tool has been the "needle probe," to identify bugs while a program is in development and provide data to revise estimates and formulate countermeasures to correct the problems. The inspection department also operated the SST, established in 1977, which simulated user conditions and used input/output and circuit defect generators to detect bugs; organized the design review task forces, which include reviewers from several different departments, including inspection; evaluated the performance of programs at customer locations; took charge of maintenance; compiled information on bugs and developed methods of testing for potential defects and developed the factory's bug forecasting system. In addition, the department had responsibilities for programmer training and helped develop support tools.[57]

ACCOUNTING AND CONTROL

The members of this department set up and have maintained the factory's cost accounting system. A major problem in the beginning was how to treat orders, sales, and income. Management then decided to total development expenses for systems software after a project's completion and then charge these back to the Kanagawa Works. Payments for applications programs for customers were included with the hardware; the Software Works received payments by submitting in-house order tickets. Essential to the calculations were the standard times for all software development activities, set by the engineering (production administration) department.[58]

SYSTEMS ADMINISTRATION (OMORI SOFTWARE WORKS)

This department originated in 1973, from the computer division's systems consultation department, which developed large-scale applications programs for the government and private customers. Fujimoto, the head of the Software Works, and Sakata, the deputy general manager, moved this divisional department to the Software Works as part of their effort to centralize and standardize design activities, in preparation for the introduction of the first M-series mainframes in 1974, which required additional personnel to rewrite programs to run on them. The institution of SC (system consultation) standard times corresponded to the move of this department to the Software Works. The department also took over responsibilities for financial controls for leased systems.[59] Hitachi then moved the systems administration department to the Omori, Tokyo site, and in 1985 made this a separate factory for applications programs.

C. Product Engineering and Production Management

MANPOWER CONTROL

The basic tool for manpower control in Hitachi's software factories is standard times for all phases of the development process, beginning with basic design. Since the establishment of the Software Works, a committee of Hitachi managers has revised these once per year, to keep up with improvements in programmer productivity. This attempt to study and discipline an engineering activity began in the mid-1960s, when programmers in the Kanagawa Works began recording data on computer time and personnel required to develop particular programs. Guided by Shibata, Hitachi engineers had enough actual data and confidence by 1967 to establish formal procedures for estimating both labor and machine hours for program development, initially placing this information on job tickets. The inauguration of the Software Works in 1969 then made it necessary to adhere to Hitachi's company-wide budgeting and cost-accounting procedures, which used standard times as basic accounting units for all engineering and production activities.

"We were perplexed," recalled Sakata, but they set up a committee that succeeded in drawing up formal standard times for each activity and for each class of programmers, based on their training and experience. The standard times consisted of debugged program steps per day or month, and took into account factors such as the type of program being written and the language being used. They collected the data in a "red book" that became, in the early 1970s, the basis for the factory's current cost accounting and planning systems. Hitachi instituted these controls for software several years before IBM began promoting the use of standard times, although the System Development Corporation had published some materials discussing how to devise standard times for software during 1967-1968 and these provided several suggestions to Shibata.[60]

Hitachi managers early on realized that systems software development required a different set of activities than customized applications software. To deal with this problem, they established SC (system consultation) standard times in 1973.[61] The systems engineering groups also developed separate tools and systems such as HIPACE (Hitachi Phased Approach for High

Productive Computer System Engineering) to standardize their proposals and to aid in design automation. The evolution of different standards and tools made it relatively easy to move the applications departments to the independent Omori site.

Planning and scheduling improved significantly soon after the factory was established, through the compilation of actual data for each phase of the development process, and continual refinement of planning techniques. Accurate programmer classifications were considered essential to both the planning and budgeting systems, which, for each project, took into account the experience and potential output of team members. Programmer classifications were determined largely but not entirely by their length of service in the company. Seniority was a fairly accurate indicator of performance, according to Sakata, because actual data showed that coding speed increased markedly with experience. But, at any given time, only between 20 and 30 percent of programmers actually met standard times, and it generally took 2 to 3 years to reach standard times for coding (and longer for design).

Programmers were also tested before managers raised their official classifications. They were made to take certain courses, and then tested at the completion of each course. In addition, all programmers twice a year were tested through competion in contests, where they had to write flow charts and code to solve certain problems. The contests involved both individuals and groups, and management recorded the results in a data base as a reference for future planning and scheduling.[62]

PROCESS CONTROL

Establishing a capability for accurate planning and process control were his most enduring problems, claimed Sakata. These were especially important because Hitachi's computer division sales department would always announce new computer products and give out specifications before the Software Works had developed the systems software. This placed a tremendous burden on software managers to meet their targets. Customers also wrote their own applications programs, based on the advance specifications, and became very upset if the systems software was delayed or if Hitachi changed the specifications.

The Software Works' process-control system monitors the status of each project through documents covering the first half of the development process. Large projects include several hundred people, divided into teams of about thirty with responsibilities parcelled out equally to team members. Time and manpower estimates rely on actual data for past projects, including the annually revised standard times.

For example, scheduling for a given project first involves the system architects determining the general functional specifications and how many program steps this will take. This is the most difficult part of scheduling, according to Shibata, and thus the most error prone. Then they look to standard times data for each phase of the development process and calculate a standard time objective for the entire project. They next look at the skill levels and particular programming experiences of the employees available for the project. Using the standard times as a reference, they work out a schedule based on required program steps, man-months adjusted for skill and experience levels, and computer time.

At the inception of the factory, managers began using simple arrow diagrams and then a computerized diagramming tool, PERT Network System (PNET), to keep track of projects and draw up a master schedule. Hitachi linked this experimentally with a planning simulation program in 1971, HICAP (Hitachi Computer-Aided Planning). It did not prove to be especially accurate, however, and involved other problems. Most serious was that letting the computer control the schedule was too lax, since parts of a program often have to be completed before other work can continue; managers preferred to deal with these types of problems through personal negotiations. In addition, during project progress meetings (formalized in 1973-1974 and which met once a month to discuss problems and potential solutions), they found it convenient to use arrow diagrams on paper, because of all the schedule changes they usually made. Hitachi thus went back to manually written arrow diagrams for process planning, until the factory introduced CAPS as an on-line production control system in 1978 to track the actual progress of completing modules and documentation, and of debugging and design review. This system involved the assignment of every programmer to a specific terminal as well as a central data base to keep track of the process flow, mainly through the completion of documentation.[63]

In the debugging process, the basic control mechanism has been check lists, which indicate problems and progress toward solutions, including from 1973-1974 a system of tickets (or tags) for accompanying documents.[64] Hitachi used PX tickets to accompany daily control charts during debugging, and PY tickets for daily control charts during inspection. PZ tickets accompanied control charts identifying specific defects or bugs, while PW tickets designated control charts used during planning, design, and coding. Other tickets indicated the state of work-in-process, and progress in correcting defects. Overall monitoring of the debugging process was also incorporated within CAPS.[65]

To determine what percentage of a project has been completed, managers submit reports on completion of modules. If a program is designed to have 100 modules, for example, and 80 are completed, then they consider the project 80% done.[66] One of the results of the control systems used at the Software Works is that, if projects are falling behind, managers can add people -- not just anyone, but the best people available -- and generally finish close to the target. This was because the factory environment facilitated rapid understanding of projects.[67] In contrast, IBM's experiences with the 360 operating system development was that adding people tended to make projects later, due to the communication time needed to familiarize new personnel.[68]

QUALITY CONTROL

Hitachi engineers defined quality control for software as, first, preventing the creation of bugs in the design stage, and, second, meeting performance specifications. These two factors directly impact the customer and so, according to Shibata, are given primary importance. Other features of quality that affect the manufacturer are maintainability and portability of a program.[69] To improve quality control, Sakata, Shibata, and other factory managers decided to focus on the adoption of structured design methods, high-level languages, and formal systems for quality control, process control, and product engineering. These tools facilitated long-term maintenance as well as short-term productivity by making it possible to divide job tasks more easily and to test completed modules and programs.

From 1971, the factory instituted control charts indicating the generation of bugs and status of corrections, as well as a "needle probe" technique, developed by Sakata, to test a program being developed when about 60% was completed, and then revise the overall bug estimates.[70] In 1972, Hitachi added another ticket-control system: P tickets to designate program changes, and B tickets to indicate corrects of bugs. In 1974, these were linked to the PX-PW-PZ process-control ticket system, to simplify project control and estimating. At the same time, the needle probe tool and data on actual bug generation for different types of programs became the basis of a time-series statistical program for forecasting bugs instituted in 1975, FORCST. This also provided programmers with examples of bugs in different types of software, to help them avoid making similar errors.[71] Included as part of the quality control effort were also design review sessions from around 1974 (particularly used for applications programs where Hitachi had to meet customer specifications).[72] In addition, the system-gleaning practices, focusing on particular system design problems and solutions that seemed instructive to present to all employees in the Software Works, supplemented other quality assurance activities.

PRODUCT CONTROL

This consisted of a formal system, started by the system program department in the Kanagawa Works, for both storing program source files and accompanying documentation for future reference, either to correct bugs or to add enhancements. In 1976, Hitachi started the practice of keeping copies of all programs in a separate location to guard against destruction from earthquakes or accidents.[73]

STANDARDIZATION

In addition to standard times, from the inception of the Software Works, Hitachi managers made "job standardization" a top priority. They did not establish long-term fixed standards, because personnel and technology were changing continually. But the standards establishment committee initially met almost weekly to determine what short-term work standards should be and codified these as the Hitachi Software Standards (HSS). This entire effort involved a deliberate attempt to standardize software as Hitachi factories standardized the development and production of material products; specifically, managers tried to prevent programmers from designing, coding, and documenting software products in different ways. In Shibata's opinion, standardization of the entire process flow was probably the most important technique Hitachi found to raise productivity, particularly when combined with high-level languages and group-programming support systems such as CAPS. The fewer bugs that resulted was one advantage; another was that standardized documentation made inspection easier.[74]

According to the official factory history, the standardization effort was extremely difficult and not very successful at first. Over time, however, factory managers succeeded: The effort depended on the establishment of a structured design method in 1973 to facilitate program maintenance and portability, despite the lengthier programs that often resulted.[75] At the same time, the factory instituted a standardized "components [modules] control system" and then in 1977 a general-use software tools registration and control system. Meanwhile, the factory published standard coding manuals for each programming language used in the facility.[76]

The structured programming method Hitachi adopted began with a standardized approach to design: (1) determination of user requirements; (2) determination of external specifications (the program's logic structure); (3) determination of internal specifications (the program's "physical" structure); (4) manufacturing (coding); and (5) inspection (testing and debugging). The logic structure represented the functional layers of a program and the interconnections (input/output) between those layers. First, programmers wrote specifications in natural language and used an in-house tool, CEG (Cause-Effect Graph), and decision tables to identify inconsistencies or flaws in the logic structure. They then broke down the object function into partial functions to develop algorithms to implement the desired specifications. The physical structure represented the actual modules making up the program (their hierarchical arrangement as well as interconnections) and data (data hierarchies, data and module interconnections, and data-point relationships).

Hitachi's major design objectives were (1) to match the physical structure as closely as possible to the logic structure; (2) to standardize the physical structure; and (3) to make the elements of the physical structure as independent as possible. This latter principle required each module to have only one input and one output, and each module to be in effect a "closed subroutine." Documentation also followed a standardized format. In addition, several support tools relied directly on the standardized design structures. AGENT (Automated Generator of External Test Cases), for example, automatically generated test items from the cause-effect diagrams and served as a logic-design support tool. ADDS (Automated Design and Documentation System) served as a design-support tool for the physical structure by analyzing design information and generating documents in graphic form.[77]

Related to standardization was the capability to reuse modules developed for both system and applications programs. Since standard times did not assume a programmer would reuse software, doing this allowed programmers to meet or surpass standard times, and helped managers meet cost or scheduling targets more easily than without reusing modules. Potential reusability was considered at the very beginning of designing a program, and facilitated through the standardization of design through structured programming.

The tradeoffs, according to Shibata, involved performance and enhancements; structured programs designed to contain reusable parts did not always perform as well as newly written programs. In fact, a general rule Hitachi used was that, if they had to revise 30% of the code in a module, then, in terms of functional performance, it was better to write the module needed from scratch. Only in 1985 did Hitachi managers require programmers to start keeping data on how much code they were reusing, although survey data in the previous paper ranked Hitachi in the high category for this measure, exceeded only by Toshiba, which was over 50% (See Appendix table on "Reusability Analysis"). For new releases of systems software, the reusability rate was about 90%.[78] The most opportunities for reusability, however, Hitachi viewed as being in applications programs.

In addition, considered as part of the standardization and reusability effort were a series of systems to facilitate the transfer of programs among different machines. HIPAL (Hitachi Program Application Library), an on-line data base of applications programs and utilities, was first set up within the computer division in 1968 and then transferred to the Software Works in 1970. A tool for translating programs for different machines, HITRAN (Hitachi Translation Program Library), was separated from the HIPAL system in 1977. Both of these were for in-house use. HILINK (Hitachi Users' Program Library Network) was a separate system launched in 1975 that made it possible for Hitachi customers to exchange programs they had written.[79]

TRAINING

Training was integral to standardization and the general success of the factory effort. Consequently, an education program was set up along with the Software Works. Hitachi hired both software engineers who had studied in the U.S., as well as high-school graduates which the company had to train itself. But the expansion of Software Works personnel from 348 in 1969 to over 900 in 1971 created a severe strain on instructors; a temporary solution was to make greater use of large meetings, as well as use the results of tests based on the Hitachi Software Standards and contests among the programmers, to judge the abilities of programmers.[80] Managers recorded the results of these tests and contests and used them (along with length-of-service information) to classify programmers as an aid in estimating time and cost schedules.[81]

The education and classification scheme extended for 12 years in Hitachi, after which individuals received the grade of chief programmer or system engineer, depending on whether they were in systems software or applications software. During the first year in the Software Works, employees were classified as trainees and took courses in introductory computer science and basic programming. They remained classified as "junior" programmers or system engineers from the second through fifth years in the factory, during which time they received additional courses. Programmers with between six and ten years of experience were designated junior leaders and received middle-level courses. Between their nineth and tenth years, programmers rose to the status of planners or sub-leaders, while undergoing advanced training. Chief programmers continued their education and studied subjects such as software reliability, use of design review, semiconductors, and computer network technology.[82]

V. TECHNOLOGICAL INFRASTRUCTURE: COMPUTER-AIDED TOOLS AND SYSTEMS

The initial essence of the factory infrastructure at Hitachi was primarily a combination of policies to promote standardization and the use of "good" practices such as structured design. A technological infrastructure to facilitate division of labor and group programming evolved afterward, largely in response to several problems in software management that appeared to defy complete solution through management alone. A Hitachi memorandum cited six areas of specific concern:

1. The invisible nature of the production process
2. Increasing scale, complexity, and diversification of program functions
3. Pressure for higher reliability
4. Difficulty of improving production efficiency
5. Shortage of software designers, especially experienced designers, and managers
6. Increased work hours for management.[83]

The response engineers at Hitachi's Systems Development Laboratory and at the Software Works arrived at during the late 1970s was to develop computer-aided systems for functional support (such as design, coding, and testing) and group programming in general. The factory's policies for standardization, design, and inspection functions for systems software were integrated through CASD (Computer-Aided Software Development System); and the policies for man-power, process, and quality control through CAPS (Computer-Aided Production Control System for Software). Both CASD and CAPS relied on various subsystems or support tools, and standardized methods. They themselves also evolved into a broader effort labelled ICAS (Integrated Computer-Aided Software Engineering System), aimed at fully integrating product-engineering and production-management tools and activities.[84] The System Development Laboratory has directed the development of these technologies, with the cooperation of the Engineering Department in the Software Works in areas related to production and quality control.[85]

A. Computer-Aided Software Development System (CASD)

CASD was mainly a response to the increasing size and complexity of software programs and grew out of a design and debugging tool Hitachi developed during 1975-1977 to centralize controls for supporting and standardizing design through the use of structured programming, high-level languages for system construction, multiple and remote computer sites, and a central program library.[86] After 1979 it became also a tool for reliability improvement, evolving increasingly in parallel and with linkages to CAPS, which standardizes a variety of technologies and activities to improve control over manpower planning and scheduling as well as cost, process flows, and quality.[87]

There were two basic assumptions underlying CASD: One was that labor productivity in software could be improved by standardizing the tasks in each phase of development and then utilizing support tools. Second was that performance could be improved not only through standardization but also through automating these support tools for each phase of the development process and then integrating them into a single system. This was an attempt, in the words of the architects of the system, to "modernize" as much as possible of what has usually been considered a manual activity, supported only with tools for discrete parts of the development process.[88]

Structurally, CASD included three interconnected support subsystems, for design, programming, and testing. The design-support subsystem constructed the design documentation and analyzed the design specifications. The programming-support subsystem made it possible to write the system using a high-level language, and analyzed the results. The testing-support subsystem then helped devise the test items and run the program being tested, as well as evaluate the comprehensiveness of the tests after they were run.

The design-support subsystem relied on a structured-programming tool called Automated Design and Documentation System (ADDS) as well as Hitachi's Module Design Language (MDL). MDL made it possible to standardize and formalize design specifications at the module level; the ADDS system placed this documentation, as well as corrections or changes, into a central database, and checked for obvious errors, thereby assisting in the design review process. Printers and terminals provided the capability to "visualize" the design documentation. The tables and charts produced by ADDS covered areas such as the functional layered structure of a program, module specifications, the data flow path, module connections, and summaries of the modules, functions, and changes.[89]

The programming-support subsystem relied on a standardized language for coding, the Hitachi Programming Language (HPL). This also facilitated design review. HPL's main components were a compiler and what Hitachi called the Static Code Analaysis (SCAN) system. Coding reviews were supposed to catch program bugs as early as possible and also provide a way to examine the program logic. Hitachi engineers were frustrated because this was largely a manual process, and was affected significantly by the different levels of ability of the programmers. To address these problems, the SCAN system received static-code analysis data from the HPL compiler and put out various reports for use in the coding review. These reports analyzed the program control structure in graphic form, the program's data structure, and the module control structure and relationship between modules and data. Then, SCAN checked the results of these analyses with design information from ADDS.[90]

The testing-support subsystem was intended to tackle problems of quality control and productivity simultaneously by facilitating the identification of bugs and, correspondingly, the reduction of man-hours devoted to testing. This subsystem had four objectives. One was to clarify in detail the design specifications, on the assumption that the test items had to determine the conformance of the program to its specifications. Another was to establish testing standards for a given program, recognizing that it was impossible to test all potential operating conditions. In addition, the subsystem was designed to automate as much of the testing process as possible, as well as evaluate the comprehensiveness of the tests. Several other tools -- Cause and Effect Graphs (CEG), Automated Generator of External Test Cases (AGENT), HPL Test and Debugging system (HPLTD), and Test Coverage Manager (TESCO) -- were integrated within the system to perform these objectives.[91]

B. Computer-Aided Production Control System (CAPS)

As with CASD, CAPS development was inspired by several persistent problems. Hitachi managers wanted greater standardization and control of the process flow, delivery times, quality, and overall costs. In particular, CAPS focused on the following areas:

1. Collection and analysis of management or process-control data in accordance with the structure of a program
2, Chronological analysis of each type of process-control data
3. Imposition of controls on the process limits of design documentation
4. Japanese character and graphic output through a non-impact printer
5. Multifaceted quality analysis
6. Construction of a data base for actual data and standard times
7. Automatic collection of data
8. Capability for on-line instantaneous utilization of the automated

output.[92]

The manual procedures and computer-aided production-control tools introduced at the Kanagawa Works and then the Software Works between 1967 and 1976 provided the foundation for CAPS, of which Hitachi completed an initial version between 1977 and 1980, by establishing a formal means of collecting and analyzing both historical and current data on programmer productivity and project management. The incremental evolution of these management policies and tools is clearly evident in the following chronology of major milestones preceding the start of CAPS development:[93]

1967 Completion of a system for computing labor and machine hours for software development (Kanagawa Works)

1969 Establishment of standard times for software development (Software Works)

1971 Establishment of programmer ability coefficients and amendments of standard times

Completion of an estimation and budget system using standard times and a simulation system for resource planning

Completion of a PERT Network System (PNET), an automatic diagramming system for schedule control

Implementation of a manual system for time-series analysis of test processing and quality control

1972 Completion of a system for budget-vs.-actual expenditure control for man-hours and machine-hours for each project and department

Implementation of a manual system for document schedule control

1973 Implementation of a manual system for job process control

Implementation of a manual system for productivity analysis and control

1974 Completion of a productivity analysis system for each project and department

1975 Implementation of a manual system for defect factor analysis and quality control

1976 Development of a time-series statistical program for forecasting defects (FORCST)

Establishment of a standard scheduling system

Implementation of structured design methods

Central to CAPS was the standardization and clarification of program structures; this the factory accomplished by requiring programmers to use structured design methods. But successful implementation of the computer-aided features of the control system depended equally on several improvements in hardware technology. One was high-performance computing power, so that numerous programmers could be on terminals connected to the same data bases; this was done by installing Hitachi's largest mainframes, the M-180 and M-200H. CAPS also demanded increased storage capacity for the historical data base recording past data and present data, and comparing these with standard times; this came through another Hitachi product, MSS (Mass Storage System). To use MSS efficiently required a large-scale data-base management system; Hitachi filled this gap with the development of several systems, most notably ADM (Adaptable Data Manager). Managers also wanted simple visual graphic output, to make it easier to follow the process flow; this was achieved through the use of non-impact laser beam printers that printed Japanese characters as well as English. In addition, managers wanted to formalize the development process for new software products and then shorten the time needed for program development; Hitachi has been most successful in accomplishing this in the applications area, with a series of procedures and tools such as EAGLE and HIPACE, as well as

CORAL (Customer-Oriented Application Program Development System), and CANDO (a prototyping tool). These are used primarily in Omori and applications subsidiaries.[94]

An example of this mixing of management policy and computer technology can be seen in the incorporation of standard times into CAPS. Controlling programmer time and machine time was considered critical because, according to Shibata and Yokoyama, these accounted for over 90% of software production costs. Sakata had earlier decided it was necessary to establish standard-time estimates for man-days and computer time. Shibata and his contemporaries, however, wanted to incorporate these into a computerized system that would enable Hitachi to follow the time estimates more closely in the actual production process. Standard times required, first, determining job standards and, second, classifying programmers by ability; managers such as Shibata wanted to be comprehensive but stressed that standard times be as simple as possible, so they would be easy to revise as well as to simulate, while still covering most of the appropriate criteria. To facilitate the accuracy and utilization of the standard times, for each project, estimates and actual data were fed as the project progressed into a central production data base from on-line terminals. This made it possible to compare progress to estimates and revise estimates during the project. Under CAPS ca. 1980, data points included the following:

1. type of object machine (large, medium, small, peripheral)
2. type of program (control program, on-line user control, generator, simulator, etc.

3. process phase (basic design, functional design, etc.)
4. degree of difficulty (newness)
5. production volume (number of steps)
6. language being used (assembler, COBOL, FORTRAN, etc.)
7. machine being used

In addition, cost overruns and late deliveries were the result, Shibata and Yokoyama believed, of inaccurate daily scheduling and planning. To correct this problem, Hitachi engineers wrote an algorithm to calculate manpower needs and schedules automatically. This took two factors into consideration: (1) actual working hours of the committed programmers; (2) the minimum necessary times required for different phases for each type of software program and standard times. Another assumption at Hitachi was that there was a relationship between the progress of a software project and its quality; an ideal system would thus integrate production management and quality control data. Therefore, they designed CAPS to estimate automatically the number of defects likely for each phase of development, according to the type of program, number of steps, items tested, and other factors, based on actual data.[95]

CAPS relied completely on the use of structured programming techniques, and then used this design technology to make the structure of programs visible. A data base control system designed for structured programs divided into modules, ADM (Adaptable Data Manager), automatically checked actual progress versus estimates, as each module of a program was completed. ADM then produced a detailed printout tracking the schedules for design, testing, and inspection, with additional information on documentation and quality (errors in the specifications, design documents, or manuals; bugs found in test items and coding; analysis of the causes of bugs and countermeasures). Three subsystems -- for documentation daily-schedule control, testing preparations and programming progress control, and testing, bugging, and inspection progress control -- were also integrated within CAPS and provided additional printouts with information and anlysis. In this sense, CAPS was more than just a system for production management that provided a visual capability for process monitoring; it also analyzed data and served as a tool for quality control.[96]

As a production and quality control system, CAPS was not fully integrated with CASD but was developed in parallel. For example, CASD output files were not automatically sent to the CAPS production database source file; nor did CASD automatically send corrected modules to CAPS. Automating the information flow was, however, a major area of development and central to the ICAS program.[97] For example, between 1980 and 1983, Hitachi completed links between the two systems making it possible to register program modules in the CAPS program library automatically from CASD, and to automatically feed data on bugs from CASD to CAPS.[98]

C. Integrated Computer-Aided Software Engineering System (ICAS)

ICAS also represented a mixture of technology and standardized methods and procedures), but was aimed at incorporating even more advanced methods and computer-aided tools. It contained four main features: (1) an integrated methodology for the structuring and abstraction of software, using a formal language and graphic notation, throughout a life cycle defined as need analysis, requirement definition, planning, programming, testing, operation and maintenance; (2) interactive tools for each phase of the life-cycle; (3) an "intelligent" workbench, using a personal computer, allowing programmers to use the tools by having dialogues with the computers; and (4) complete management of information for all phases using a relational database. The basic philosophy of this approach was to develop not "methodology-free" tools, leaving it up to the user to decide on which development methodology to employ, but to present computer-aided tools with a "fixed development methodology of multi-purpose use," allowing users to develop software quickly by using the tools "without having to worry about which methodology to apply."

Requirements definition involved stepwise refinement in the procedural, functional, and logical description of the system being designed. From the conceptual model, programmers determined the control structure, abstracted data and formed data modules, and defined functional algorithms with only three control elements -- sequence, repetition, and selection -- using PDL or PAD (Problem Analysis Diagrams). ICAS then automatically converted the functional algorithm into statements written in a programming language. Several tools simplified needs analysis and description (PPDS--Planning Procedure to Develop Systems and FRAME--Formalized Requirements Analysis Method), and requirements definition (RDL/RD (Requirement Definition Language/Analyzer).

Design-aid tools included ADDS (Automated Design and Documentation System) and MDL (Module Design Language), already part of CASD, as well as PDL/PAD (Problem Design Language/Problem Analysis Diagram). PDL/PAD was intended to automate coding and completely integrate design documentation and source programs. It consisted of a program logic design tool, based on structured programming, and a tool to convert automatically design documents into high-level language source programs (PL/1), or vice-versa. In addition, DBDS (Database Design System) was a tool for designing databases. Test-aid tools included AGENT, TESCO, and CEG, discussed above. A Software Engineering Workbench (SEWB) and a Software Engineering Database (SEDB) provided an infrastructure to use these tools in an integrated manner. The relational database stored all data from each tool input into the computer.[99]

The quality control portions of ICAS were in actual operation in the Software Works as of 1986 as part of CASD. Other subsystems of ICAS being refined at the Systems Development Laboratory, Hitachi Software Works, and Omori Software Works were already in use for applications software. Most important were HIPACE, the set of procedures and methodologies to guide system design, and EAGLE (Effective Approach to Achieving High Level Software Productivity), an automated system-development support tool.[100] Even before complete integration within the ICAS project, HIPACE provided a factory-type methodological infrastructure and EAGLE a factory-type technological interface for applications development; managers at Hitachi's Omori facility and at Hitachi Software Engineering also viewed these tools as factory systems.[101]

Hitachi intended HIPACE to reduce costs in customized applications development by facilitating communication between the company's system engineers and customers and applying well-defined, standardized procedures to system development and project management. The process flow was simply analysis; sytem planning; system design; program design; program construction; test; transfer (to customer); operation and evaluation. First, engineers used Structured Data Flow diagrams (SDF) to analyze customer needs. Planning Procedure to Develop System (PPDS) and Standard Procedure to Develop Systems (SPDS) then provided documentation and project-management standards for each phase of the development process. A set of worksheets referred to as Work Breakdown Structure (WBS) provided the format for planning of the actual design, programming, and testing tasks. To facilitate long-term maintenance and reliability (and reusability), engineers then used HIPACE-SA for structured analysis, HIPACE-SD for structured design, and HIPACE-SP for structured programming (usually in COBOL or PL/1).[102]

The EAGLE system extended the HIPACE methodology by adding four computer-aided functions: (1) conversational language processing from system design through testing, using easily understandable menus; (2) a central database on design specifications and program implementation, as well as project management (tracking information from the standardized work sheets defined in the SPDS manual) to facilitate system development and maintenance; (3) automated construction of new source programs from standardized patterns ("skeletons") and components (sub-routines); and (4) automatic compilation of maintenance documentation.

The process flow in using EAGLE was also clearly defined. The first two steps are the analysis of data types and interrelationships and their recording in a "data dictionary" database; this is followed by registration of the system design and program documentation in a specifications database. At this point, new standardized modules are identified and registered in a central program parts library, and existing components are identified for the system being developed, if applicable. This makes it possible to "assemble" programs using the new and reused modules. (Hitachi also makes these standardized modules available as products with the EAGLE systems it sells to customers, although company engineers have found that "data modules" are easier to use than processing algorithms, which tend to be more difficult to standardize.) EAGLE next generates an outline of the program from the detailed (module-level) specifications and then produces a source program. The source program is then edited to add particular functions wanted by individual users. Finally, test commands are automatically generated and carried out in conversational language (Japanese).[103]

Recent efforts (1984-1986) to develop the EAGLE system have focused on making it both more flexible for meeting customer needs as well as more appropriate to a factory environment stressing division of labor and maximal use of standardized components. One the one hand, the conversational interface has been improved; and the system now handles PL/1 and CORAL (Customer-Oriented Application Program Development System -- a Hitachi language for writing specifications in Japanese), in addition to COBOL. New software makes it possible for customers to design their own menus, rather than use only the ones Hitachi provides, to add unique features to programs being constructed. The system also can now be used to construct data-base and data-communications programs, in addition to business-applications programs.

On the other hand, EAGLE has been modified to work more smoothly in a time-sharing environment, to allow more people to divide up the tasks of system development and have better access to the library of reusable components.[104] The overall result, according to Hitachi data, is that programs designed with EAGLE generally show a 2.2-fold improvement in "productivity" (Hitachi usually measures this by lines of code per programmer in a given time period). As indicated in the table below, EAGLE also has shifted more effort into system design and substantially reduced necessary for testing. For a hypothetical program taking a year to develop without EAGLE, this would mean a reduction in development time to 5.4 months, with testing being reduced from 4.8 to 1.4 months and program implementation from 4.8 to 1.9 months.

	Without EAGLE	With EAGLE[105]
Development	100 (12 months)	45 (5.4 months)
System Design	20% (2.4)	38% (2.1)
Program Implementation	40% (4.8)	36% (1.9)
Test	40% (4.8)	26% (1.4)

VI. ADDITIONAL FLEXIBILITY THROUGH SUBSIDIARIES

Where Hitachi has needed more organizational or geographic diversity to meet customer needs than its two software factories provided, it has relied on approximately 23 subsidiaries. The largest were Nippon Business Consultants (ca. 2500 employees), established in 1959; Hitachi Software Engineering (ca. 2400 employees), established in 1969; and Hitachi Micro-Computer Engineering (ca. 1500 employees), established in 1982.[106] Hitachi classified these firms into ten groups, with several overlapping:

(1) General systems and applications software houses
(Nippon Business Consultants, Hitachi Software Engineering, Hitachi Information Networks, Hitachi Computer Consultants, Hitachi Computer Engineering; and the regional companies Hitachi Chugoku Software, Hitachi Tohoku Software, Hitachi Chubu Software, Hitachi Nishibu Software)

(2) Industrial-use control systems
(Hitachi Industrial Engineering, Hitachi Process Computer Engineering, Hitachi Control Systems)

(3) Semiconductor and micro-computer software
(Hitachi VLSI Engineering, Hitachi Micro-Computer Engineering)

(4) Information-processing and telecommunications systems
(Hitachi Electronic Service, Hitachi Communications)

(5) Video and audio equipment, and personal-computer systems and software
(Hitachi Video)

(6) Semiconductors and electronic devices
(Hitachi Electronic Devices)

(7) Precision instruments software
(Hitachi Instruments Engineering)

(8) Automotive electronics
(Hitachi Automotive Engineering)

(9) Robotics, control equipment, and business personal computers
(Hitachi Kyoba Engineering)

(10) Personal Computers
(Hitachi Micro-Software Systems)

A brief discussion of Hitachi Software Engineering reinforces the notion that the Hitachi group has managed to combine flexibility in serving customer needs with a disciplined engineering and manufacturing approach to software development. In contrast to Hitachi's two in-house software factories, this subsidiary ranked low on the factory scale for both the technology and policy criteria, reflecting the diverse nature of the company: Some sections worked as part of the permament workforce in Hitachi's in-house software factories, while other groups did customized systems development for a wide variety of Japanese customers. In a sense, Hitachi Software Engineering served primarily as a manpower facility for Hitachi Software Works and Omori Software Works, following the factory practices at these facilities, or applying them to projects it did independently. Overall, however, it incorporated Hitachi technology such as CASD, CAPS, and HIPACE, as well as developed modified in-house systems. Hitachi Software Engineering was also remarkably efficient in project control. The company reported in 1981 that it was able to complete 98% of projects on time and 99% at an actual cost between 90% and 110% of the original estimates. This compared to 300%-overruns during the early 1970s. The average project size was 50,000 lines of code; the largest were about 500,000 lines.[107]

As did the Hitachi factories, Hitachi Software Engineering emphasized extensive programmer training (1-year training periods, including 2 months of off-the-job training when they entered the company), as well as strict implementation of top-down, structured design and careful controls on budgets and project management, including standard times for programmers (design and coding). Historically, managers at the subsidiary focused first on setting up a project and production auditing system (1969-1975); applying structured programming techniques and software tools (1976-1978); productivity improvement and quality assurance through the standardization of methodologies and tools (1979-1981); and productivity and quality improvement through the generation of reusable modules ("standard patterns"), their cataloging in program libraries, and utilization in the writing of new programs. Reusable modules included patterns for mainframe operating systems and related programs, as well as for functions such as message reception, message format and contents checking, data-base management system processing, message editing and switching, screen mapping, line-overflow, error displays, program-function key code analysis, screen editing, and table lookup. Managers also assigned programmers exercises on a monthly basis to make them familiar with subroutines stored in the program library.[108]

Despite the lack of centralization and standardization for the subsidiary as a whole, an integral and clearly stated component of management strategy at Hitachi Software Engineering was rigid discipline. In fact, the two managers who spearheaded the development of production technology at this subsidiary, after moving over from Hitachi Software Works, openly admitted to the use of extensive training techniques to make programmers comply with company standards for program design: "To meet our production criteria, project procedures have been standardized and imposed on employees. If any modules deviate from the coding standard, they are returned to the production line."[109]

CONCLUSION

The survey revealed that Hitachi management was not as rigorous in promoting certain factory-type concepts or even technologies as several other firms, notably the NEC group, TRW, and Toshiba. In any case, Hitachi is significant as an historical leader in introducing a strategic approach toward the management of what is largely an engineering activity; and the history of this firm's efforts in large-scale software reveals how a major process innovation -- the software factory -- was conceived and implemented.

Interviews with managers and a study of technical and historical documentation indicate that Hitachi's movement toward the factory model resulted from three interrelated strategies or decisions:

1) A company policy of establishing independent factories (which included both product engineering and mass-production functions) for each major product area.

2) Belief on the part of managers responsible for corporate- and division-level strategy, as well as for software development, that a centralized

and disciplined factory environment, integrating product engineering and mass-production activities, offered for any product the potential of improving worker productivity and quality, as well as project and cost control.

3) Top management decision and commitment (including divisional executives and, especially, the company president) to treat software as a product whose development could and should be controlled in a factory, as any other product the company manufactured -- making it necessary to apply company-wide, standardized accounting and management controls to all software engineering activities.

The history of Hitachi Software Works also indicates that the foundations of the factory were primarily policy-oriented. Somewhere in between technology and policy was the introduction of a structured programming technique during the mid-1970s. Structured programming is really a methodological tool; since it was necèssary to train and require programmers to use this as standard procedure, the use of this new technology or tool involved critical policy decisions and implementation. This was especially important because structured programming as defined by Hitachi became the foundation of the factory's standard-times, cost-accounting, and general production-control systems, as well as specific support tools. The rather sophisticated (computer-aided) technological infrastructure evolved after the basic policy infrastructure, but rapidly, from a few tools at first to an extensive set of interrelated systems that are increasingly being further integrated.

A contrast between implementation of the factory model at Hitachi and the experiment at System Development Corporation in the mid-1970s also offers some suggestions regarding why one company might succeed better than another at this approach. SDC attempted to introduce simultaneously a factory system containing both a technological infrastructure and a policy infrastructure (set of standard procedures covering system-analysis, design, implementation, testing, and project-management). While the technology (central production database, program library, automated documentation and testing tools, etc.) was there, programmers seemed to dislike the standardized environment and reusing other programmers' code. Perhaps more important was that project managers disliked giving up control to the factory and work for the facility dwindled; this led eventually to the dismantling of the factory infrastructure through the dispersal of systems engineers to different sites to develop programs, with little capability to divide labor or reuse modules as once envisioned in the factory concept.[110]

Hitachi, on the other hand, incrementally developed and imposed a policy infrastructure over a period of several years, thereby training programmers and managers to operate within a highly standardized, factory-like environment. Hitachi modified these procedures gradually and then from the mi-1970s began investing heavily in tools and large-scale computer-aided systems -- the factory-type technological infrastructure. This shift in focus to technology-based tools and automation thus came after successfully innovating in process management by applying a standardized approach to both system and applications software development.

One might also identify a parallel here with Toyota, a company often cited for its excellence in production management. The largest Japanese automaker has consistently demonstrated the world's highest levels of physical productivity in automobile manufacturing by deemphasizing the use of sophisticated automation or computer-based systems, preferring instead to focus on process control and innovation, as well as flexible tools, to extend the performance of human workers. Only after these policy innovations have Toyota managers agreed to introduce more automation, but only if the technology is sufficiently flexible (such as programmable robots) to fit into its manufacturing system and supplement the efforts of human workers.[111]

The comparison with Toyota, as well as the SDC case, reinforces the notion that technological advances alone do not necessarily bring as many benefits in productivity as simply better process management. The Hitachi case in software suggests that, with the proper mixture of policy and technology, including a strategy to assure the compliance of managers and engineers or other programmers, the factory approach should offer several advantages in efficiency. As suggested in the first paper from this research project, these might include the following:

Institutionalization or dissemination of "good" technologies and programming or management practices.

The production engineering departments in Hitachi's software factories, as well as the factory training programs, were responsible for introducing techniques and tools that, in textbooks on software programming and engineering management, are widely considered to be fundamentally good practices. These include structured design methods; bug-forecasting data collection and models; documentation standardization and control systems; formal project-management and design-review systems; program libraries; wide use of computer-aided design, coding, and testing tools; and promotion of standardization of practices and reusability of code where possible.

Providing a sufficient scale of people and operations to justify research and development for improving process technology and techniques.

In addition to engineering departments in the software factories, Hitachi also used the System Development Laboratory to perform R%D activities related to programming support tools and methods. The centralization of software production at the Software Works and Omori provided an in-house core of 3000 programmers and supporting staff; Hitachi Software Engineering and Nippon Business Consultant added another 5000.

Improving process efficiency through teamwork and better inter-group communication.

This can be seen in two examples. One, is that the percentage of late projects in the Software Works dropped dramatically within a few years of the founding of the factory, from over 72% in 1970 to 13% in 1973 and to a remarkably low 7% in 1974 and 1979, with an average of 12% during 1974-1985. The figure for 1986 was about 5%. These numbers placed Hitachi Software Works along with other firms leading in this category. Variations in these figures reflect the level of activity within the factory, with more late projects when Hitachi was completing new large new projects -- for example, a new mainframe operating system. But, in general, reporting procedures, as well as CAPS (Computer-Aided Production Control System for Software), made it possible to integrate manpower-control, process-control, and quality-control functions and support tools. Another example of the factory benefits is Hitachi's overturning of Brook's law about more programmers added to a late project causing projects to be later. The factory environment allowed Hitachi managers to add people (albeit the best people available and not just anyone) to help finish projects on time.

HITACHI SOFTWARE WORKS: PERFORMANCE DATA

Year	Workers	Sales/Worker	Projects Late	Reported Bugs
1969	348	100(Index)	-- %	N.A.
1970	675	202	72.4	N.A.
1971	926	178	56.6	N.A.
1972	1107	190	36.3	N.A.
1973	1169	204	13.0	N.A.
1974	1079	331	6.9	N.A.
1975	1093	313	9.8	N.A.
1976	1141	360	16.8	N.A.
1977	1288	386	16.1	N.A.
1978	1395	468	9.8	100(Index)
1979	1398	505	7.4	79
1980	1398	594	10.7	48
1981	1419	678	14.0	30
1982	1437	792	12.9	19
1983	1666	943	18.8	13
1984	1833	1257	16.3	13
1985	2018	N.A.	18.0	14

Source: Based on data provided by Shibata Kanji, Manager, Engineering Dept., Hitachi Software Works, 23 July 1986.

Notes: Sales per worker reflects the value of software products sold by Hitachi Software Works to other Hitachi profit centers as well as to outside customers. Outside sales represented approximately 90% of revenues; in-house customers were charged prices below market rates. Reported Bugs refers to major bugs reported by outside customers per machine per month.

Organizational focus on raising engineering productivity and product quality (defect control).

As indicated in the table, sales productivity of Hitachi employees in the Software Works doubled within one year of the factory's opening and has increased significantly overtime, although direct productivity figures for the facility are proprietary. Company-wide and factory programs, such as the Management Improvement movement and system gleaning practice, formally promoted the analysis and implementation of measures to improve labor performance and product quality. The central production data base for the factory started in the mid-1960s also made it possible to track programmer productivity as well as defect measures, and thereby have precise data to use in determining specific methods or in developing new tools to be used throughout the factory. Perhaps most important, the administrative and technological infrastructures of the factory -- manpower, process, quality, and product control in the area of production management; standardization, design, and inspection in the area of product engineering -- facilitate performance analysis and improvement. Individuals do not have to expend time and energy, for example, in deciding which languages or methods to use, or whether to develop a particular support tool. Systems such as EAGLE also save extensive manpower by automating much of testing and by recycling program components.

In quality, Hitachi employs a measure of user-reported major bugs and has reduced bugs from an index of 100 in 1978 to 14 in 1985. One unofficial estimate is that this figure represented approximately 0.01 defects per program package per machine installation per year, and placed Hitachi in the low category, along with other leaders in this measure that particpated in the survey.[112] According to Shibata, the decrease in bugs reflected several factors: a new System Simulation Tester (SST) completed in 1977-1979; CASD (Computer-Aided Software Development System), another group-programming tool to facilitate product standardization, design support, and inspection functions; reused code, reaching approximately 90% in new releases of products such as operating systems; and increasing sales of essentially bug-free programs.[113]

In addition, in terms of price/performance measurements and user satisfaction, Hitachi-manufactured mainframes and accompanying software sold through National Semiconductor (NAS) in the U.S. have been rated consistently higher than IBM machines and software. For example, the Hitachi/NAS AS/5000 and AS/7000, which competed with the IBM 3033 and 4341 mainframes, in a DATAPRO survey achieved overall satisfaction ratings of 3.55. compared to 3.19 for the IBM machines. In terms of software in particular, the Hitachi/NAS products rated 3.15, compared to approximately 3.03 for IBM (see table).

COMPARISON OF IBM AND HITACHI MAINFRAMES AND SOFTWARE[114]

System Ratings	HITACHI AS/5000 AS/7000	IBM 3033 4341
Ease of Operation	3.78	3.32
Reliability of Mainframe	3.63	3.61
Manufacturer's Software		
Operating System	3.17	3.10
Compilers/Assemblers	3.17	3.19
Application Programs	3.00	2.85
Ease of Programming	3.38	3.01
Ease of Conversion	3.38	3.01
OVERALL SATISFACTION (includes maintenace & technical support)	3.55	3.19
Number of Systems	10	318

Source: DATAPRO RESEARCH CORP.

Reducing waste and redundancies due to dysfunctional behavior of individuals and the lack of an organizational strategy.

High-level factory managers such as Sakata, as well as middle managers from the engineering department such as Shibata, have set a clear direction for personnel and technological development in the Software Works. Their initial focus has been on gathering information on software technology and then standardizing methods and tools, and setting factory-level performance goals. Later efforts have foused on automation and reusability. The factory infrastructure and strategy they have created has reduced the possibility of individuals duplicating the efforts of others in tool or method development, as well as writing code, and lessened the likelihood of workers engaging in practices that are contrary to the organizational goals such as to develop reusable modules, or use factory-wide tools and standardized methods that facilitate reusability, maintenance, testability, and the like.

Maintenance of organizational and technical "flexibility."

Both the technological and policy infrastructures of the Hitachi software factories have been evolving since 1969. Most of the tools developed for Hitachi Software Works (and then introduced in other Hitachi facilities or subsidiaries), such as CAPS, CASD, HIPACE, and EAGLE, have been adaptable enough to incorporate important technical advances, such as additional linkages between systems, the addition of other support tools for documentation, testing, and the like, or increased capabilities of adapting to non-standardized customer needs. Structured design has also endured for well over a decade. High levels of reusability indicate as well that Hitachi programmers find modules in the program library are not obsolete. While the factory approach might seem to make it difficult for a particular facility to respond to a unique customer need or a specific type of technology, Hitachi has been addressing these concerns through continued development of customer-oriented design systems such as EAGLE, as well as the establishment of numerous subsidiaries and new factory departments such as for artificial intelligence.

APPENDIX

SURVEY RESULTS: DATA SUMMARY TABLE

I	= Technological Infrastructure	(32=100%)
II	= Policy/Methodology Infrastructure	(60=100%)
III	= Total Factory Model	(92=100%)

@ indicates two responses and averaged or joint responses.

COMPANY/FACILITY	I	II	III
*NEC@	89%	89%	89%
*Toshiba Software Factory	84	87	86
*NEC Information Service	81	88	86
*NT&T Comm. & Info. Proc. Lab.@	81	77	78
*Hitachi Omori Works	78	73	75
*Fujitsu Info. Proc. Sys. Lab.@	77	73	75
*Nippon Systemware	72	70	71
*Nippon Business Consultant	56	67	63
*Hitachi Software Engineering@	53	50	51
*Nippon Electronics Development	41	48	46
TRW	97	83	88
Unisys/Sperry@	91	72	78
Unisys/SDC	72	77	75
Control Data@	84	67	73
Martin Marietta/Maryland	59	76	70
Hughes Aircraft	83	63	70
Boeing Aerospace@	84	53	64
AT&T Bell Labs	72	58	63
Cullinet	64	59	61
Martin Marietta/Denver	69	43	52
Electronic Data Systems@	61	43	49
Honeywell/Defense Systems@	44	42	42
Draper Laboratories@	34	17	23
Computervision@	28	25	26
Applications Means	**69**	**62**	**65**
Japanese (*)	71	72	72
U.S.	67	55	60

*NEC/Switching Systems	98%	99%	99%
*NEC/Operating Systems	92	92	92
*Toshiba Software Factory	84	87	86
*NEC Software	84	87	86
*Hitachi Software Works@	78	70	73
*Fujitsu Numazu Factory@	77	68	71
*NT&T Comm. & Info. Proc. Lab.@	58	48	51
Control Data@	78	67	71
IBM Endicott	78	62	67
Data General Westboro & N.C.	61	63	62
Boeing Aerospace@	77	53	61
Unisys/Sperry@	61	62	61
IBM Raleigh	84	43	58
DEC (VMS)	41	48	46
Systems Means	75	68	70
Japanese (*)	82	78	80
U.S.	69	57	61
OVERALL MEANS	71	64	67
JAPANESE (*)	76	75	75
U.S.	68	56	60

Statistical Analysis of Sample:

Variable:	**Totals**	**Technology**	**Policy**
Sample Size:	38	38	38
Average:	66.89	71.18	64.45
Median	70	77	67
Mode	86	84	43
Standard Deviation:	17.26	17.40	18.59
Range	76	70	82
Kurtosis:	0.31	0.03	0.02
Standardized Kurtosis:	0.40	0.03	0.03

RANKINGS: TECHNOLOGY/FACILITY INFRASTRUCTURE

8 Questions; 32=100%

Key:
A.J. = Applications Japan
A.U. = Applications U.S.
S.J. = Systems Japan
S.U. = Systems U.S.
* = Japanese firms

COMPANY/FACILITY		%	
S.J	*NEC/Switching Systems	98	Flexible
A.U.	TRW	97	Factory
S.J.	*NEC/Operating Systems	92	Approach
A.U.	Unisys/Sperry	91	
A.J.	*NEC	89	
A.J.	*Toshiba Software Factory	84	
S.J.	*Toshiba Software Factory	84	
A.U.	Boeing Aerospace	84	
S.U.	IBM Raleigh	84	
S.J.	*NEC Software	84	
A.U.	Hughes Aircraft	83	
A.J.	*NEC Information Service	81	
A.J.	*NT&T Comm. & Info. Proc. Lab.	81	
A.J.	*Hitachi Omori Works	78	
S.U.	IBM Endicott	78	
A.U.	Control Data	78	
A.J.	*Fujitsu Info. Proc. Sys. Lab	77	
S.U.	Control Data	77	
S.J.	*Hitachi Software Works	78	
S.J.	*Fujitsu Numazu Factory	77	
S.U.	Boeing Aerospace	77	
A.J.	*Nippon Systemware	72	
A.U.	AT&T Bell Labs	72	
A.U.	Unisys/SDC	72	
A.U.	Martin Marietta/Denver	69	
A.U.	Cullinet	64	
S.U.	Data General Westboro & N.C.	61	
A.U.	Electronic Data Systems	61	
S.U.	Unisys/Sperry	61	
A.U.	Martin Marietta/Maryland	59	
S.J.	*NT&T Comm. & Info. Proc. Lab.	58	
A.J.	*Nippon Business Consultant	56	
A.J.	*Hitachi Software Engineering	53	
A.U.	Honeywell/Defense Systems	44	
S.U.	DEC (VMS)	41	
A.J	*Nippon Electronics Development	41	
A.U.	Draper Laboratories	34	
A.U.	Computervision	28	Job Shop

RANKINGS: POLICY/METHODOLOGY INFRASTRUCTURE

15 Questions, 60=100%

	COMPANY/FACILITY	%	
S.J.	*NEC/Switching Systems	99	Flexible
S.J.	*NEC/Operating Systems	90	Factory
A.J.	*NEC	89	Approach
A.J.	*NEC Information Service	88	
A.J.	*Toshiba Software Factory	87	
S.J.	*Toshiba Software Factory	87	
S.J.	*NEC Software	87	
A.U.	TRW	83	
A.U.	Unisys/SDC	77	
A.J.	*NT&T Comm. & Info. Proc. Lab.	77	
A.U.	Martin Marietta/Maryland	76	
A.J.	*Fujitsu Info. Proc. Sys. Lab.	73	
A.J.	*Hitachi Omori Works	73	
A.U.	Unisys/Sperry	72	
S.J.	*Hitachi Software Works	70	
A.J.	*Nippon Systemware	70	
S.J.	*Fujitsu Numazu Factory	68	
A.J.	*Nippon Business Consultant	67	
A.U.	Control Data	67	
S.U.	Control Data	67	
S.U.	Data General Westboro & N.C.	63	
A.U.	Hughes Aircraft	63	
S.U.	IBM Endicott	62	
S.U.	Unisys/Sperry	62	
A.U.	Cullinet	59	
A.U.	AT&T Bell Labs	58	
S.U.	Boeing Aerospace	53	
A.U.	Boeing Aerospace	53	
A.J.	*Hitachi Software Engineering	50	
S.J.	*NT&T Comm. & Info. Proc. Lab.	48	
S.U.	DEC (VMS)	48	
A.J	*Nippon Electronics Development	48	
S.U.	IBM Raleigh	43	
A.U.	Martin Marietta/Denver	43	
A.U.	Electronic Data Systems	43	
A.U.	Honeywell/Defense Systems	42	
A.U.	Computervision	25	
A.U.	Draper Laboratories	17	Job Shop

RANKINGS: TOTAL FACTORY MODEL

23 Questions, 92=100%

COMPANY/FACILITY		%	
S.J.	*NEC/Switching Systems	99	Flexible
S.J.	*NEC/Operating Systems	91	Factory
A.J.	*NEC	89	Approach
A.U.	TRW	88	
A.J.	*NEC Information Service	86	
A.J.	*Toshiba Software Factory	86	
S.J.	*Toshiba Software Factory	86	
S.J.	*NEC Software	86	
A.J.	*NT&T Comm. & Info. Proc. Lab.	78	
A.U.	Unisys/Sperry	78	
A.U.	Unisys/SDC	75	
A.J.	*Hitachi Omori Works	75	
A.J.	*Fujitsu Info. Proc. Sys. Lab.	75	
S.J.	*Hitachi Software Works	73	
A.U.	Control Data	71	
S.J.	*Fujitsu Numazu Factory	71	
A.J.	*Nippon Systemware	71	
A.U.	Martin Marietta/Maryland	70	
S.U.	Control Data	70	
A.U.	Hughes Aircraft	70	
S.U.	IBM Endicott	67	
A.U.	Boeing Aerospace	64	
A.J.	*Nippon Business Consultant	63	
A.U.	AT&T Bell Labs	63	
S.U.	Data General Westboro & N.C.	62	
S.U.	Boeing Aerospace	61	
A.U.	Cullinet	61	
S.U.	Unisys/Sperry	61	
S.U.	IBM Raleigh	58	
A.U.	Martin Marietta/Denver	52	
S.J.	*NT&T Comm. & Info. Proc. Lab.	51	
A.J.	*Hitachi Software Engineering	51	
A.U.	Electronic Data Systems	49	
S.U.	DEC (VMS)	46	
A.J	*Nippon Electronics Development	46	
A.U.	Honeywell/Defense Systems	42	
A.U.	Computervision	26	
A.U.	Draper Laboratories	23	Job Shop

APPENDIX II: TOSHIBA AND NEC

Toshiba

Toshiba, since the mid-1970s, has been the other Japanese leader in pursuing factory practices, focusing on large, real-time control programs and related systems software for industrial applications. The Fuchu Software Factory in Tokyo had about 2300 software personnel in 1985.[115] A major article written in 1981 by the manager primarily responsible for developing the Toshiba facility also cited the SDC factory experiment as a precedent.[116]

Toshiba's Software Factory contained about 200 terminals in three buildings connected by high-speed data buses. One building focused on software design, another on software and hardware design, and the third on systems testing. The factory infrastructure itself revolved around "SWB," a software workbench system first developed in 1977, comprising five main tools: SADT (structured analysis for requirements definition); CASD (computer-aided specification analysis and documentation); HIPO (hierarchy plus input-process-output); PROMISS (program information management system, to facilitate reuse of proven programs); and SYSGEN (system generator, to generate software combining newly written modules and standard modules).[117]

The reuse of software modules was central to Toshiba's factory strategy. To create an environment fostering reusability, Toshiba relied mainly on management, with a supporting factory infrastructure. For example, it was an official policy to "promote registration of proven programs and reuse." As an incentive to programmers and managers, reused code counted the same as newly written code in productivity measurements.[118] In terms of design technology, Toshiba stressed data abstraction, clearly defined interfaces and parameters, as well as careful cataloging of modules for the program library.[119] The languages used at Toshiba did not in themselves facilitate reusabiity, however. In 1981, about 83% of the code was written in real-time FORTRAN and PL/I, 6% in assembler, and 11% in a machine-oriented system description language.[120] None of these languages were specifically designed for data or procedural abstraction (which is useful for reusability), although, in 1984, Toshiba was using Ada to describe program structures, before building prototypes using languages such as Prolog, APL, and Basic.[121]

Based on the extensive reuse of code, Toshiba doubled productivity at the Fuchu Software Factory after 1976, with programmers producing 3100 assembly-language equivalent instructions per month by 1985.[122] In contrast, U.S. programmers, according to a U.S. Deptartment of Commerce study, typically produced about 300 lines of code per month.[123] At Toshiba, over 50% of the 3100 lines of code produced per programmer month in 1985, and as much as 70 to 80% in some years (such as 1981), was reused from other programs.[124] Along with the high rates of nominal productivity, moreover, this extensive recirculation of debugged modules allowed Toshiba to report merely 0.3 bugs per 1000 lines of code.[125] By comparison, a study of 60 large IBM projects showed an average of 3 errors per line of code -- 10 times greater than Toshiba.[126]

PRODUCTIVITY AT TOSHIBA'S FUCHU SOFTWARE FACTORY[127]

Year	1000 Instructions/Programmer/Month	New Code
1972	1.23	
1973	1.39	
1974	1.37	
1975	1.21	
1976	1.39	
1977	1.69	1.00
1978	1.94	
1979	2.30	
1980	2.60	
1981	2.87	
1985	3.10	1.60

NEC Corporation

NEC launched a "software strategy project" in 1976, focusing on standardization, automation, elimination of waste, and quality control, in addition to promotion of reusability. NEC also established a Software Product Engineering Laboratory in 1980 to experiment with software-development and management technologies, and to oversee their application in company facilities and subsidiaries.[128]

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, p. 58.

For systems software and complex on-line programs, NEC developed a factory-type system called SDMS (Software Production and Maintenance System). NEC formulated the basic plan for SDMS in 1975, intending to design a total support system for the development and maintenance of modularized systems software. The first practical version was released for in-house use in 1980, consisting of a software development data base and three subsystems: for design, including a standardized design language and programming methodology; for product management (configuration, updating, retrieval); and for project management, including progress control and productivity data.[129]

NEC reported some resistance to adopting the standardization required by SDMS, but by 1984 it was installed in all NEC computer centers. Several experimental projects have also demonstrated significant improvements. For example, in the design of a comptroller system with a data base, engineers using SDMS showed twice the productivity rates compared to 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%.[130]

For applications software, NEC began developing what it calls STEPS

(Standardized Technology and Engineering for Programming Support) in 1971, completing a version for in-house use in 1972 and revising the system each year thereafter, with considerable input as well from outside users. The basic concepts of STEPS are (1) to promote integrated standards covering the entire software development life cycle, with standards for methodology, documentation, etc., and then (2) the use of a "prefabricated standard program library" to facilitate the writing of new programs.[131] At 800 sites within and outside NEC through 1980, an NEC survey reported productivity improvements in specification, coding, and debugging of between 26 and 91% (Table). At 1200 sites by 1984, NEC reported 20% to 50% cost reductions in analysis-design phases and 50% to 80% in program manufacturing phases.[132]

NEC: STEPS PRODUCTIVITY SURVEY, 1981

Key: S = Source Code Number
MD = Man-Days
T = Compile Time for Debug

	STEPS	NON-STEPS	PRODUCTIVITY IMPROVEMENT
Average Program Size	458 S	431 S	
Specification	286 S/MD	361 S/MD	1.26
Coding	218 S/MD	416 S/MD	1.91
Debugging	68 S/T	92 S/T	1.35
TOTAL	64 S/MD	101 S/MD	1.53

Source: Azuma and Mizuno (1981), p. 95.

NEC's application of its "traditional" manufacturing techniques to software can be seen most clearly in quality control. Administration of this function is linked directly to computer hardware "zero defect" activities and extends down to programmers organized, as in hardware production facilities, into quality circles. Since around 1981, these have been meeting 2 hours per week, and applying tools such as data sorting, control charts, pareto charts, and cause-effect diagrams to software projects.[133] Some results are as follows:

NEC SOFTWARE QUALITY CONTROL 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 Appli-cations	Spec changes	Down 40%

Source: Mizuno (1983), p. 71.

Along with developing their own technology, NEC engineers extensively studied American software techniques, including SDC's estimating model and then the Software Factory during the 1970s as approaches to cost-estimation and project control.[134] During the mid-1980s, NEC also launched a study of factory-type layout environments for software engineers to improve productivity.[135] In addition, the manager of NEC's Software Product Engineering Laboratory publicly stated in a 1984 article that "software factory realization" was one of the most important areas "[c]ompanies that expect to meet future demands must direct their efforts to."[136]

REFERENCES

1. The main paper from this research project is Michael A. Cusumano, "The 'Software Factory' Reconsidered: An Approach to the Strategic Management of Engineering." Sloan School of Management, 1987 Working Paper #1885-87. Another completed case is "A U.S. 'Software Factory' Experiment: System Development Corporation." Sloan School of Management, 1987 Working Paper #1887-87.

2. The respondents for Hitachi Software Works were Shibata Kanji, Engineering Department Manager, and Yokoyama Yoichi, Senior Engineer; for Omori, Tsuda Michio, Senior Engineer.

3. Toyo Keizai, Kaisha shikiho (March 1986).

4. Japan Economic Journal, 7 June 1986, p. 14; and, for market share data, "Nihon no konpyuta setchi jokyo," Computer Report (in Japanese), January 1985, p.78.

5. Hitachi Ltd., "Introduction to Hitachi and Modern Japan" (International Operations Group, 1983); Tadao Kagono et al., Strategic vs. Evolutionary Management: A U.S.-Japan Comparison of Strategy and Organization (Amsterdam, North-Holland, 1985), pp. 103-105; Takahashi interviews.

6. Japan Economic Journal, 7 June 1986, p. 14.

7. Dale F. Farmer, "IBM-Compatible Giants," Datamation, December 1981, pp. 96-97, 104.

8. "2 New Computers from IBM Rival," The New York Times, 12 March 1985, p. D5.

9. A useful book in Japanese on the details surrounding this incident is Nano Piko, Nichi-Bei konpyuta senso: IBM sangyo supai jiken no teiryu (The Japan-U.S. computer war: underlying the IBM industrial spying incident) (Tokyo, Nihon Keizai Shimbunsha, 1982).

10. RCA, Control Data, IBM, and NCR all delivered transistorized computers in 1958. See Franklin M. Fisher, James W. McKie, and Richard B. Mancke, IBM and the U.S. Data Processing Industry: An Economic History (New York: Praeger, 1983), pp. 50-51. For the Japanese story, see Sigeru Takahashi, ""Early Transistor Computers in Japan," Annals of the History of Computing, Vol. 8, No. 2, April 1986. I have also interviewed Dr. Takahashi extensively on computer development in Hitachi, beginning on 9 and 21 January 1985.

11. Takahashi interviews.

12. Shibata interview, 9/19/85.

13. Hitachi Seisakusho, Yuka shoken hokokusho, March 1985, pp. 16-17; Hitachi Ltd., "Outline of Hitachi. Ltd." (1984); "Introduction to Hitachi and Modern Japan," p. 12.

14. These will be discussed in a later section.

15. A recent book on the MI movement at several Hitachi factories (although not including the Software Works) is Iwai Masakazu, Hitachi-shiki keiei kakushin: MI undo no kenkyu (Tokyo, Daiyamondo-sha, 1983.

16. Hitachi Seisakusho, Sofutouea kojo no 10-nen no ayumi (10-year history of the Software Works) (Kanagawa, 1979), pp. 118, 179-184.

17. Sofutouea Kojo, p. 198, 202.

18. Nihon Noritsu Kyokai (Japan efficiency association), ed., Hitachi no seisan kakumei-- MST seisan shisutemu no zenbo (Hitachi's production revolution--the full story of the MST production system) (Tokyo, 1982), p. 32. MST stands for "Minimum Stocks/Minimum Standard Time.

19. Shibata and Yokoyama interview, 7/23/86.

20. Sofutouea Kojo, pp. 51-52.

21. Takahashi interview. See also Sofutouea Kojo, tables on pp. 49-50.

22. Sakata interview.

23. Minamisawa Noburo, Nihon konpyuta hattatsu-shi (Nihon Keizai Shimbun-sha, 1978), chronology.

24. Shimada Shozo, "Hitachi no gijutsu (2): kaihatsu-ki (HITAC 5020)--sofutouea," in HITAC Yuza Henshu linkai, ed., 20 nen no ayumi (Hitachi Ltd., 1983), pp. 27-29. Also, Murata Kenro, "Hitachi no gijutsu: kaihatsu-ki (HITAC 5020, 8800) -- hadouea," in HITAC Yuza linkai, p. 22.

25. Usui Kenji, "HITAC kaihatsu shi (2), p. 37.

26. Takahashi interview, 1/9/85; Usui Kenji, "HITAC kaihatsu shi (2)," Computopia, July 1975, p. 37; Sofutouea Kojo, p. 53.

27. Hitachi Seisakusho, Kanagawa kojo 15-nen no ayumi (1978), p. 40. Since EDOS continued to build upon the RCA operating system, the mainframes Hitachi has sold in Japan after 1970 are close to IBM but not compatible, although machines Hitachi produces for export and overseas sales through National Semiconductor are modified to be fully IBM compatible.

28. Sofutouea Kojo, pp. 54-56.

29. Sofutouea kojo, pp. 56, 130.

30. Hitachi memo to Cusumano, 21 August 1985.

31. Cite Takahashi, documents on this, etc.

32. Hitachi Seisakusho, **Hitachi Seisakusho shi**, Vol. 3, 1971, pp. 55-56, 223-224; Usui Kenji, "HITAC kaihatsu shi (2)," pp. 36-38; **Kanagawa kojo 15 nen no ayumi** (1978), pp. 45-47.

33. Minamisawa, pp. 154, 163.

34. Usui, "HITAC kaihatsu shi (2)," p. 36; Sakata interview.

35. Usui, "HITAC kaihatsu shi (2), p. 30. Footnote other sources.

36. Usui (2), p. 31; **Sofutouea kojo**, pp. 50-51; Takahashi interview, 1/9/85.

37. Sakata interview, 9/10/85.

38. Usui (2), pp. 36-37; **Sofutouea kojo**; pp. 17, 129.

39. **Sofutouea kojo**, p. 23.

40. **Sofutouea kojo**, pp. 21-22; **Kanagawa**, pp. 18-22, 69. The English translation Hitachi uses is "Software Works," although the Japanese word for "works" (**kojo**) is usually translated as "factory" and has this connotation in Japanese. An analysis of the Odawara Works can be found in Hitachi Seisakusho Odawara Kojo, **Odawara Kojo 10 nen shi** (Kanagawa-ken, Hitachi Odawara Kojo, 1977).

41. **Sofutouea kojo**, pp. 4-5, 8.

42. Sakata interview.

43. Sakata interview, 9/10/85.

44. Sakata interview, 9/10/85.

45. Shibata interview, 9/19/85.

46. Shibata interview, 9/19/85; and **Sofutouea kojo**, p. 119.

47. **Sofutouea kojo**.

48. Shibata interview, 9/19/85. See also **Sofutouea kojo**, p. 5.

49. **Sofutouea kojo**, p. 113.

50. Shibata interview, 9/19/85.

51. **Sofutouea Kojo**, p. 192.

52. Hitachi Software Works, Memo, July 1986.

53. Shibata interview, 7/23/86; and <u>**Sofutouea kojo**</u>, pp. 192-202, which contains organization charts from 1969 to 1979.

54. <u>**Sofutouea kojo**</u>, pp. 127-128.

55. <u>**Sofutouea kojo**</u>, p. 128.

56. <u>**Sofutouea kojo**</u>, pp. 164-165; Shibata interview, 9/19/85 and 7/23/86.

57. <u>**Sofutouea kojo**</u>, pp. 160-161; 118-119; Shibata interview, 7/23/86.

58. <u>**Sofutouea kojo**</u>, pp.4-11, 166.

59. <u>**Sofutouea kojo**</u>, p. 156.

60. <u>**Sofutouea kojo**</u>, pp. 113-114; Hitachi Memo, "Table 1: History of Production Control at Hitachi's Software Works"; Sakata interview; Shibata interview, 9/19/85 and 7/23/86.

61. <u>**Sofutouea kojo**</u>, p. 114.

62. Sakata interview; Shibata interview, 9/19/85 and 7/23/87.

63. Interviews with Shibata and Yokoyama

64. Shibata interview, 9/19/85.

65. <u>**Sofutouea kojo**</u>, pp. 114-115.

66. Shibata interview, 9/19/85.

67. Sakata interview.

68. Frederick P. Brooks, Jr., <u>**The Mythical Man-Month: Essays pm Software Engineering**</u> (Reading, MA, Addison-Wesley, 1975), pp. 16, 31.

69. Shibata interview, 9/19/85.

70. Sakata interview. Also see Sakata Kazuyuki, "Sofutouea no seisan kanri ni okeru yosoku giho no teishiki-ka -- sei-teki na yosoku oyobi koshoritu suii moderu" (Formulation for Predictive Methods in Software Production Control-- Static Prediction and Failure Rate Transition Model), pp. 277-283, and "Sofutouea no seisan kanri ni okeru yosoku giho no teishiki-ka -- doteki na yosoku: sakidori hyoka giho" (Formulation for Predictive Methods in Software Production Control -- Dynamic Prediction: Quality Probe), pp. 284-291, in <u>**Denshi tsushin gakkai ronbun shi**</u> (Transaction of the institute of electrical and communications engineers, May 1974, Vol. 57-D, No. 5.

71. Shibata interview, 9/19/85.

72. **Sofutouea kojo**, p. 115.

73. **Sofutouea kojo**, p. 115; Shibata interview, 9/19/85.

74. Shibata interview, 9/19/85.

75. Sakata interview.

76. **Sofutouea kojo**, pp. 117-118. For applications software sold outside the company, the design departments took until 1975 to set their own standards for both customer estimates and the program-development process.

77. Kataoka Masanori, Domen Nobuyoshi, and Nogi Kenroku, "Sofutouea kozo sekkei giho" (Software Structure Specification Method), **Hitachi hyoron**, Vol. 62, No. 12 (December 1980), pp. 7-10.

78. Definitions of reused code: Project Reuse Rate = Number of Reused Steps divided by reused steps plus new steps plus revised steps times 100. Cumulative Reuse Rate = Cumulative Number of Reused Steps from Version I divided by Number of Steps in Current Version times 100. Shibata interview, 7/23/86.

79. **Sofutouea kojo**, p. 117; Yokoyama interview.

80. **Sofutouea kojo**, pp. 7-8, 124.

81. Shibata interview

82. **Sofutouea kojo**, p. 124.

83. Hitachi memorandum, "Table 1.2 Background and Conditions Affecting CAPS Development."

84. Regarding ICAS, see M. Kobayashi et al., "ICAS: An integrated Computer Aided Software Engineering System," **IEEE Digest of Papgers--Spring '83 COMPCON** (IEEE, 1983), pp. 238-244; and Kobayashi Masakazu and Aoyama Yoshihiko, "Sofutouea seisan gijutsu no saikin no koko" (Current Topics in Software Engineering), **Hitachi hyoron**, Vol. 68, No. 5 (May 1986), pp. 1-6.

85. Interviews with Shibata and Yokoyama, 7/23/86 and 9/19/85.

86. Kataoka Masanori, Hagi Yoichi, and Nogi Kenroku, "Sofutouea kaihatsu shien shisutemu (CASD shisutemu)" (Computer Aided Software Development System, CASD), **Hitachi hyoron**, Vol. 62, No. 12 (December 1980), p. 33. Kataoka and Hagi were from the Software Works; Nogi was from Hitachi's System Development Laboratory.

87. Interviews with Shibata and Yokoyama, 7/23/86.

88. Kataoka Masanori, Hagi Yoichi, and Nogi Kenroku, "Sofutouea kaihatsu shien shisutemu (CASD shisutemu)" (Computer Aided Software Development System, CASD), Hitachi hyoron, Vol. 62, No. 12 (December 1980), p. 33.

89. Kataoka et al., pp. 33-34.

90. Kataoka, p. 34.

91. Kataoka et al., pp. 35-36.

92. Shibata Kanji and Yokoyama Yoichi, "Sogo sofutouea seisan kanri shisutemu 'CAPS'" (Computer Aided Production Control System for Software), Hitachi hyoron, Vol. 62, No. 12 (December 1980), p. 37.

93. Hitachi memorandum, "Table 1.2 Background and Conditions Affecting CAPS Development."

94. Shibata Kanji and Yokoyama Yoichi, "Sogo sofutouea seisan kanri shisutemu 'CAPS'" (Integrated computer-aided production control system for software 'CAPS'), Hitachi hyoron, Vol. 62, No. 12 (December 1980), p. 37; Hitachi memorandum, "Table 1.2 Background and Conditions Affecting CAPS Development"; Shibata interview, 7/23/86. Omori makes greater use of prototyping, which produces overal specifications of a program before the actual modules and code are fully written. In systems software, Shibata has felt that prototyping is not practical, since it requires too much time and effort.

95. Shibata and Yokoyama, pp. 39-40.

96. Shibata and Yokoyama, p. 40-42. A version of CAPS was available for microcomputers and used at Hitachi's Omika factory, which designed systems software for control computers, as well as at subsidiaries such as Hitachi Software Engineering and Nippon Business Consultants. The system was not sold commercially. (Shibata interview, 7/23/86.)

97. Shibata and Yokoyama, p. 42.

98. Interviews with Shibata and Yokoyama, 7/23/86.

99. Kobayashi et al.

100. Interviews with Shibata and Yokoyama, 7/23/86 and 9/19/85. Regarding HIPACE, see "Apurikeshon shisutemu no koritsu-teki sekkei giho 'HIPACE'" (Software Engineering Methodology for Development of Application Systems 'HIPACE'), Hitachi hyoron, Vol. 62, No. 12 (December 1980), pp. 15-20; for EAGLE, see Hagi Yoichi et al., "Shisutemu kaihatsu shien sofutouea 'EAGLE'" (Integrated Software Development and Maintenance System 'EAGLE'), Hitachi hyoron, Vol. 68, No. 5 (May 1986), pp. 34.

101. Tsuda Michio, Senior Engineer at Omori Software Works, written response to "Large-Scale Applications Software" survey, 2/20/87; and Takahashi Tomoo, Department Manager at Hitachi Software Engineering, written response tot "Large-Scale Applications Software" survey, 3/6/87.

102. Miyazoe Hidehiko (Hitachi Software Works) et al., "Apurikeshon shisutemu no koritsu-teki sekkei giho 'HIPACE'" (Software Engineering Methodology for Development of Application Systems 'HIPACE'), **Hitachi hyoron**, Vol. 62, No. 12 (December 1980), pp. 15-20. Also see Nakao Kazuo Hitachi System Development Laboratory) et al., "Shisutemu keikaku no tame no shisutemu yokyu bunseki shuho 'PPDS' no kaihatsu" (System demand analysis procedures for system planning 'PPDS'), **Hitachi hyoron**, Vol. 62, No. 12 (December 1980), pp. 21-24.

103. Hagi Yoichi (Hitachi Software Works) et al., "Shisutemu kaihatsu shien sofutouea 'EAGLE'" (Integrated Software Development and Maintenance System 'EAGLE'), **Hitachi hyoron**, Vol. 66, No. 3 (March 1984), pp. 19-24.

104. Hagi Yoichi et al., "Shisutemu kaihatsu shien sofutouea 'EAGLE'--EAGLE kakucho-han 'EAGLE 2'" (Integrated Software Development and Maintenance System 'EAGLE' - the Enhanced Version of EAGLE, 'EAGLE 2'), **Hitachi hyoron**, Vol. 68, No. 5 (May 1986), pp. 29-34.

105. Source for this table is Hagi et al. (1986), p. 34.

106. See Hitachi Seisakusho, "'86 shitemu/sofutouea no Hitachi gurupu" (The '86 Hitachi group for systems and software)

107. Denji Tajima and Tomoo Matsubara (Hitachi Software Engineering), "The Computer Software Industry in Japan," **Computer**, May 1981, p. 95.

108. Denji Tajima and Tomoo Matsubara, "Inside the Japanese Software Industry," **Computer**, March 1984, pp. 36-39.

109. Tajima and Matsubara (1984), p. 40.

110. See Michael A. Cusumano and David E. Finnell, "A U.S. 'Software Factory' Experiment: System Development Corporation." **M.I.T. Sloan School of Management**, Working Paper #1887-87, 1987.

111. For a discussion of Toyota see Michael A. Cusumano, **The Japanese Automobile Industry: Technology and Management at Nissan and Toyota** (Cambridge, MA: Harvard University Press, 1985). This comparison is also examined in more detail in "Toward the Strategic Management of Engineering: The 'Software Factory' Reconsidered."

112. McNamara estimate.

113. Shibata interview, 7/23/86.

114. Abbreviated from Farmer, p. 97.

115. See Yoshihiro Matsumoto, "A Software Factory: An Overall Approach to Software Production" (2 December 1986), forthcoming in Peter Freeman, ed., **Software Reusability** (IEEE Tutorial, 1987); Y. Matsumoto et al. (Toshiba), "SWB System: A Software Factory," in H. Hunke, ed., **Software Engineering Environments** (Amsterdam: North-Holland, 1981), pp. 305-318; "New Toshiba Software Facility Spurs Productivity," **Toshiba Newsletter**, No. 256, November 1983, p. 1; K.H. Kim, "A Look at Japan's Development of Software Engineering Technology," **Computer**, May 1983, pp.31-33; and Shimoda, pp. 99-109.

116. Yoshiro Matsumoto (Toshiba), "Management of Industrial Software Production," **Computer**, February 1984, p. 318 fn 2.

117. Kim, pp. 31-32; Matsumoto (1981), pp. 305-318; and **Toshiba Newsletter**.

118. Matsumoto (1981), p. 308.

119. Matsumoto (1984), p. 68.

120. Matsumoto (1981), p. 308.

121. Matsumoto (1984), p. 65.

122. Kim, p. 33; Matsumoto (1981), p. 308.

123. **A Competitive Assessment of the U.S. Software Industry**, p. 11. Other productivity data for a variety of U.S. projects can be found in Martin Shooman, **Software Engineering: Design, Reliability, and Management** (New York: McGraw-Hill, 1983), pp. 438-445; and Frederick P. Brooks, Jr., **The Mythical Man-Month: Essays pm Software Engineering** (Reading, MA, Addison-Wesley, 1975), pp. 88-94.

124. Calculated from data in Matsumoto (1981), p. 308.

125. Kim, p. 32; Robert Haavind, "Tools for Compatibility," **High Technology**, August 1986, p.41.

126. Cited in Shooman, p. 326. **A Competitive Assessment of the U.S. Software Industry**, p. 11, also cites data that Japanese programmers produce one-tenth the bugs of U.S. programmers.

127. Sources: Kim, p. 33; and Yoshihiro Matsumoto, "A Software Factory: An Overall Approach to Software Production" (Toshiba Corporation, December 2, 1986), p. 5.

128. Kiichi Fujino (NEC), "Software Development for Computers and Communication at NEC," **Computer**, November 1984, pp. 57-58.

129. 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. 26-27; and Kanji Iwamoto (NEC), et al.,

"Early Experiences Regarding SDMS Introduction into Software Production Sites," NEC Research and Development, January 1983, p. 51.

130. Uenohara et al., p. 32.

131. M. Azuma and Y. Mizuno (NEC), "STEPS: Integrated Software Standards and its Productivity Impact," Proc. Compcon Fall 81 (IEEE), September 1981, p. 83.

132. Fujino, p. 59.

133. YukioMizuno, "SoftwareQuality Improvement," Computer, March 1983, pp. 69-71.

134. Mizuno interview; and Iwamoto Kanji and Okada Masashi (NEC), "Purojekuto kanri no tsuru" (Project control tools), Joho shori (Information processing), Vo. 20, No. 8, p. 721. Iwamoto was a member of the Computer Systems Research Laboratory, part of the Central Research Laboratories; he joined the Software Product Engineering Laboratory when NEC created this in 1980. Okada was then in the systems engineering department of NEC Software Ltd., a subsidiary.

135. See K. Honda et al. (NEC), "Research on Work Environment for Software Productivity Improvement," paper submitted at the Compsac '85 conference.

136. Interviews with Azuma (NEC), 10/1/86, and Yamaji (Fujitsu), 7/31/86. See also Kiichi Fujino (NEC), "Software Development for Computers and Communication at NEC," Computer, November 1984, pp. 61-62.