Fujitsu Software: Process Control and Automated Customization

by

Michael Cusumano

WP 2044-88 Aug, 1988

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139

Fujitsu Software:

Process Control and Automated Customization

by

Michael Cusumano

WP 2044-88 Aug, 1988

FUJITSU SOFTWARE: PROCESS CONTROL AND AUTOMATED CUSTOMIZATION

Contents:

Introduction
The Corporate Setting
Systems Software Development
Applications Software Development
Conclusions

INTRODUCTION

Fujitsu followed Hitachi, NEC, and Toshiba in adopting factory-type approaches and organizations for software development, beginning with extensive controls over project management and product inspection for systems software during the early and mid-1970s. This culminated in the centralization of systems software development at Fujitsu's Numazu Works in Shizuoka prefecture during 1982-1983. Fujitsu also began centralizing applications programming in 1970, and then issued a set of standardized methods and tools before establishing a Software Factory Department in Kamata, Tokyo, in 1979, which performed detailed design, coding, and testing for specifications done in system engineering departments outside the factory.

The first section of this case discusses the background of Fujitsu as a company, its entrance into the computer business from a basis in telephone switching and electro-mechanical equipment, and elements of the software strategy in place by the 1970s. This included product-market objectives such as IBM-compatibility, and technical goals such as improving software development and control technology, cultivating specialized skills and subsidiaries, and building tools with mechanization and automation capabilities. The next two sections focus on development experiences, tools, methods, products, and performance in the systems and applications software divisions. Systems software development resembled Hitachi and NEC, and was characterized by a gradual refinement and integration of manual systems and automated support tools for process control, product registration and inspection, and quality assurance and testing. In the applications area, Fujitsu again resembled its Japanese competitors, responding to large demand for customized programs with extensive investment in program-automation tools and software reuse, especially packaged subsystems, to facilitate the efficient production of semi-customized and new programs.

THE CORPORATE SETTING

Company Outline and Organization

Fujitsu was established in 1935 when Fuji Electric, Ltd., spun off its telephone equipment division as a separate company. The new firm developed Japan's first digital calculator in 1935 and expanded into switching systems and other electric and electro-mechanical equipment, before introducing a primitive non-programmable computer in 1954. Fujitsu gradually expanded product development and marketing for a range of telecommunications equipment, office equipment, computers and computer peripherals, and data processing services.

In the year ending March 1987, Fujitsu had over 50,000 employees in the parent corporation, with approximately 11,000 personnel involved in software operations. Consolidated sales (including some 70 subsidiaries) totalled over $12 billion, and were divided among computers and data processing systems (67% of total revenues), communications systems (15%), semiconductors and electronics components (12%), and car audio and other electronics equipment (5%).[1] Software accounted for at least 10% of these revenues, and was expected to rise to 14% by 1990.[2] Fujitsu managed these areas through 13 operating or marketing and service groups. Two contained factory-type facilities for software development: one for systems software (operating systems, control programs, network software, data base systems, language processors, Japanese language and graphics or voice processing software, and automatic translation programs);[3] and another for customized applications software (Table 7.1).

Table 7.1: FUJITSU ORGANIZATION AND SOFTWARE FACTORIES[4]

Operating Groups	Software Factories
Computers (mainframes, minicomputers)	Numazu Works, Software Division
Printed-Circuit Board Products	
Information Equipment (peripherals)	
Transmission Systems	
Switching Systems	
Telecommunications Systems	
Semiconductors	
Electronic Devices	
NT&T Division	
Marketing and Service Groups	
Hardware Systems Sales	
Office Automation Sales	
Hardware Maintenance	
Systems Group	Software Factory Department

Systems software was developed in the computer groups's software division, which Fujitsu separated from other divisions for hardware design and manufacturing, integrated systems development, medical systems, and factory automation. Both systems software and hardware manufacturing operations were located in the Numazu Works, a factory located near Mt. Fuji in Shizuoka prefecture that Fujitsu established in 1974 to produce the FACOM-M series mainframes and minicomputers, designed to compete with IBM's 370 models. The software division was housed in a separate building (connected to the hardware building) constructed in 1981.

Fujitsu management did not publicize Numazu as a software factory.[5]

Nonetheless, centralization of most systems software development at this facility by 1984 made it possible to institute a level of standardization and control that clearly resembled factory-like organizations in other Japanese firms.[6] The software division in 1987 consisted of two software engineering departments, eight development departments, an inspection department, and a Field Support Center.[7] Of the approximately 3000 software personnel, about 60% were college graduates and the rest high-school graduates.[8]

For applications programming, Fujitsu opened a Software Factory Department in 1977 at the Information Processing Systems Laboratory in Kamata, Tokyo, continuing a decade-long effort to centralize system engineering and programming for software required by banks, securities firms, and manufacturing and distribution companies. Fujitsu had established the laboratory in 1970 to house the Systems Group, which, in the mid-1980s, consisted of separate departments for system engineering, software package planning, and applications development.

The implementation sections in the Software Factory Department had approximately 1500 programmers in 1987, including some employees of subsidiaries and subcontractors working full time in the factory.[9] Nearly all were college graduates and performed detailed design, coding, and testing, based on specifications received from approximately 3000 system engineers and other staff in the Systems Group.[10] In addition to these in-house employees, 53 software subsidiaries employed nearly 11,000 programmers, and affiliated software houses added several thousand more, bringing total staff resources for the Systems Group to about 20,000 (Table 7.2).[11] As of 1986, Fujitsu was also adding personnel at the rate of about 1000 college graduates per year in the Systems Group, while its subsidiaries were adding 1500 to 2000 people per year.[12]

Table 7.2: FUJITSU SOFTWARE SUBSIDIARIES AND EMPLOYEES (1985-1986)[13]

Software Area	Companies	Employees	Major Companies
System Engineering	34	7,000	Fujitsu AIB
Basic Software	9	1,800	BIC
Communications	7	1,300	Fujitsu Dai-Ichi
Microprocessors and Personal Computers	3	700	Fujitsu Micon
TOTAL	53	10,800	

Note: Employee figures for 38 subsidiaries are based on Fujitsu data; other figures are author estimates, based on average company size in each category.

Entry into Computers

Fujitsu entered the computer business in 1954 with a dedicated accounting machine, the FACOM 100. This was not a programmable computer but was hardwired, using electro-magnetic relay switches adapted from telephone switchboards. Fujitsu introduced several other relay computers before adopting parametron circuits in 1958, following the lead of Hitachi and NEC. Fujitsu's parametron computers also were primarily dedicated calculating machines. Again following Hitachi and NEC, Fujitsu began working on a transistor-based computer in 1958, and introduced its first programmable commercial models in 1961, for business applications. Management also signaled a firm commitment to the new industry by establishing a computer division in 1963.[14]

While Hitachi, NEC, and Toshiba during the 1960s relied heavily on U.S. firms (RCA, Honeywell, and General Electric, respectively) for computer hardware and much software technology, Fujitsu management tried to link up with IBM. When this initiative failed, management had little choice but to pursue independent development. Prospects for Fujitsu seemed dim, since it was already behind Hitachi, NEC, and Toshiba due to a late switch to transistors and the assistance U.S. firms were providing Japanese competitors But the flow of new models from the U.S. largely stopped during the later 1960s, as GE and RCA prepared to exit the computer business, and as Honeywell reduced its product development efforts. This created market opportunities.

Fujitsu in 1965 introduced its 230 series of small, medium, and large mainframes, using transistors initially and integrated circuits (ICs) from 1968. These attracted many Japanese customers, especially in the banking industry and at universities, due to low prices and powerful processors. The Japanese government also aided these sales by placing restrictions on purchases of non-Japanese computers. Due to the popularity of the 230 models, Fujitsu became Japan's largest computer manufacturer in 1968, surpassing Hitachi and NEC. In 1970, for the first time, computer revenues exceeded 50% of sales and became the largest part of Fujitsu's business.[15]

A critical factor in Fujitsu's ability to develop computers independently was the skill of Ikeda Toshio (1923-1974), a graduate of the Tokyo Institute of Technology who had entered Fujitsu in 1946. After designing telephone switching equipment, he moved into R&D and computer development, quickly gaining recognition as perhaps Japan's most talented computer engineer. In 1969 Ikeda also met Gene Amdahl, who had designed IBM's 360 family and then left IBM in 1970 to found the Amdahl Corporation, to produce IBM-compatible mainframes. This meeting began a relationship with the Amdahl Corporation that greatly strengthened Fujitsu's efforts in computer development during the 1970s.

One reason for pursuing ties with Amdahl was that Ikeda and other Fujitsu managers were concerned that Fujitsu could not export the 230 machines because they were not compatible with IBM. This issue prompted Fujitsu in 1972 to adopt IBM compatibility for all mainframes developed during the 1970s. Fujitsu made this decision in conjunction with Hitachi, and both firms agreed to standardize their mainframe architectures, although without jointly developing hardware or software. The IBM-compatible strategy for both Fujitsu and Hitachi thus began with the M-series, announced in 1974 and designed to compete with IBM's 370 family.

To support this move, Fujitsu again approached IBM to license its operating system. After IBM declined, Fujitsu made its first investment in Amdahl in 1972. The premature death of Ikeda in 1974 then persuaded Fujitsu executives to become the majority investor in Amdahl and seek design assistance.[16] Assisted by Amdahl in developing IBM-compatible logic circuits, using large-scale semiconductor integration technology (LSI), Fujitsu by the mid-1970s was able to deliver hardware comparable to IBM in performance at given price ranges. Fujitsu also managed to increase exports after the mid-1970s by manufacturing mainframe central processing units for Amdahl in the U.S. and ICL in Great Britain, based on their specifications, and selling Fujitsu mainframes to Siemens for marketing in Europe under the Siemens label.[17]

Software Strategy: Products and Production

The decision in the 1960s not to import American computer technology forced Fujitsu to develop independently not only hardware designs but also systems and applications software. This provided important challenges and learning experiences for company engineers, particularly since Amdahl, with the exception of its version of UNIX, did not provide software to Fujitsu.[18] Major systems software and packages developed at Fujitsu between 1962 and 1984, listed in Table 7.3, indicate that the company was able to produce a wide range of software, at least after 1970.

MONITOR V, completed in 1968-1971 for the larger 230 series models, was Fujitsu's first modern operating system. Problems in finishing this led to a revamping of the structure to manage systems software development, although not until the M-series operating system in the mid-1970s did Fujitsu undertake the type of systematic data collection and integrated tool and methodology development that has characterized Hitachi, Toshiba, and NEC. However, to improve control over released products, Fujitsu did introduce a series of inspection and quality assurance procedures in the early 1970s, initially to meet NT&T's software procurement requirements, and then transferred these procedures to other divisions.

Table 7.3: FUJITSU SYSTEMS SOFTWARE AND PACKAGES, 1962-1984[19]

1960	ALGOL compiler
1961	ASSEMBLER FORTRAN compiler
1963	Data communications software
1965	FORTRAN IV MCP (control program for the FACOM 230-20/30 mainframes MONITOR II (proto-operating system for the FACOM 230-50)
1966	KEMPF (econometrics modeling and analysis system)
1968	MONITOR V (large-scale mainframe operating system) STAT (statistical package)
1969	BOS (medium-size mainframe batch operating system) ROS (medium-size mainframe real-time processing operating system) PL/I (programming language version) EPOCS (medium-size mainframe data control program) ADSL (continuous simulation system)
1970	RAPID (data base system)
1971	MONITOR V TSS (time-sharing function added to MONITOR V) BOS II (successor to BOS and ROS operating systems) OS II (operating system for FACOM 230-45/S and /55 mainframes ASTRA (structural analysis program)
1972	UMOS (mini-computer operating system) TIMS (time series analysis program)
1973	MDS (management decision-making support tool) MPS (mathematical planning tool)

1974	BOS/VS and OS II/VS (medium-size operating systems with virtual memory control function) MONITOR VI/VII (large-scale mainframe operating system)
1975	OS IV/F4 (operating system for largest M-series mainframe) FEM (finite element analysis program)
1977	AIM (on-line data base system) FNA (Fujitsu Network Architecture system) OS IV/X8 (medium-size operating system for FACOM M series) OS IV/F2 (small-size operating system for FACOM M series) PDL/PDA (performance measurement and analysis tool)
1978	OS/UAS (mini-computer operating system) FAIRS-I (document search system) KING (Japanese-language line printer support program) EPG (executive planning guide) C-NAP (customer needs analysis program)
1979	JEF (Japanese language text and information processing system) INTERACT (end-user support system) FAIRS-II (management information search system) FDMS (document control system) GEM (library control program) SSOPTRAN (FORTRAN source program optimizer)
1980	FORTRAN77 (scientific- and technical-use language) AOF (computer center operations support tool) ARIS (Japanese language information system) HOPE (hospital administration system) SDSS (software development support tool)
1981	OS IV/F2 ESP (small-scale operating system for FACOM M-series) CADAM (computer design support system) ICAD (computer design support system) HYPER COBOL (high-productivity version of COBOL) DOCK/FORTRAN77 (FORTRAN debugger for system displays) ANALYST (statistical data processing package) PLANNER (planning and control information system) ESTIMA (business forecasting system) AXEL (conversational language data analysis system)
1982	OS IV/F4 MSP (large-scale operating system for FACOM M-series) AIM/RDB (relational data base system) DRESSY/P (simplified graphics system) ADAMS (application software development system) LIMS (books control system)
1983	FOS (Fujitsu Office System Concept) OS IV/X8 FSP (medium-size operating system for FACOM M-series) UNICUS (minicomputer operating system) JEF II (Fujitsu Japanese language processing system) MACCS/REACCS (information analysis and search system)

ODM (documents processing system)
ELF (electronic file system)

1984 OS IV/ESP III (small-scale operating system for FACOM M-series)
IMPRESS (image information system)
IPS (small-scale printing control system)
UPLINK III/PS (personal computer network system)
FCAD-11 (personal computer CAD system)
ATLAS (automatic translation system)

A critical element shaping the software strategies of Fujitsu and Hitachi was this commitment to IBM compatibility. This was especially true for Fujitsu, which intended to follow IBM standards for its domestic and export mainframes, whereas Hitachi allowed the architecture for its domestic computers to depart slightly from IBM compatibility. Both Japanese companies proceeded on the assumption that, since IBM's 360 operating system had been in the public domain, they could duplicate the 370 software freely, without provoking legal action from IBM. This strategy worked in the 1970s, but brought on several challenges from IBM in the 1980s.[20]

After charges from IBM that Japanese firms were stealing IBM code, and the arrest of several Hitachi and Mitsubishi Electric engineers in 1982, it became imperative for Fujitsu to maintain compatibility without directly copying IBM code. While Japanese companies reached agreements with IBM that allowed them to market IBM-compatible software, IBM in 1984 again challenged the originality of Fujitsu's operating system. This delayed Fujitsu's delivery of a large-scale mainframe, the M-780, competing with IBM's 3090 machine, and prompted Siemens to stop purchasing Fujitsu's operating system for the Fujitsu mainframes it marketed in Europe. In response to these events, Fujitsu management reorganized and expanded software operations at Numazu, and shifted the division's focus to producing new operating systems with more original designs. These efforts as of 1988 have avoided expensive patentinfringement suits and licensing fees with IBM, and appear to have created within Fujitsu the capability to design operating systems independently and move away from IBM compatibility, should this prove necessary.[21]

In the applications area, Fujitsu's history resembled that of its major Japanese competitors. In the mid-1960s, it began developing large-scale customized programs for industrial and government customers. Rapid growth in demand led to greater emphasis on reusability and "semi-customization" of programs rather than full customization, and the introduction of numerous automated tools.[22] Fujitsu was distinctive for achieving a major success in package sales with the introduction of JEF (Japanese Processing Extended Feature) in 1979. This quickly became the market leader in Japanese-language word processing packages, and was followed by an equally popular program in 1983, JEFII, which added integrated text, image, and voice processing capabilities.[23]

On the production side, the stated concerns of Fujitsu managers resembled those of counterparts elsewhere: to improve control over development costs while maximizing product functionality, performance, reliability, and maintainability, and delivering products on time to customers.[24] To accomplish these goals with demand for software rising faster than Fujitsu's ability to train programmers, management encouraged efforts to "accumulate and improve technology," both individual and organizational, in three main areas (Table 7.4). Furthermore, to encourage users to design their own software and spread the burden of programming, Fujitsu sold a large number of support tools and provided instruction in software engineering methodologies to buyers of Fujitsu hardware as well as to subsidiaries and affiliated software houses.

Table 7.4: FUJITSU PRODUCTIVITY IMPROVEMENT STRATEGIES

Development Technology	**Specialization**	**Mechanization/Automation**
Joint Development	Organization	Software Tools and other Computer-Aided Systems for Planning, Development, Testing, Maintenance
Modularization		
High-Level Languages	Subsidiaries	
Reuse		
Review Procedures	Packages	
Structured Design		
Structured Programming		

Support and Control Technology	
Project Management	Standardization
Development Planning and Phase Completion Reporting System	Quality Control Activities (High-Reliability Program)
Management Objectives	Training

To improve productivity, the first area of focus was **development technology**. This included more emphasis on higher-level languages and, to reduce the volume of new code that had to be written, reuse of designs and code, as well as what Fujitsu called "joint development." Joint development referred to software where there were multiple, interdependent components. Fujitsu's philosophy was to make sure managers in different areas developed these components once and reused them across different systems, by standardizing system description languages and then requiring that the software be modularized, designed to be reused, and catalogued. In addition, to reduce product complexity, Fujitsu stressed modularization, structured methods for design and programming, as well as the use of standardized in-house design and coding languages. Careful and quantified review, quality control, and inspection procedures also served to improve product reliability and marketability, and maximize manpower efforts by reducing time spent on fixing or altering programs.[25]

A second area was **specialization**. This included the establishment of development organizations dedicated to different types of software and functions, in particular the Numazu Works Software Division and the Kamata Software Factory, smaller development facilities linked by on-line networks,[26] as well as 53 subsidiaries specializing in various areas or providing regional system-engineering services throughout Japan (see Table 7.2). These subsidiaries, supplemented by long-term arrangements with independent software houses, provided not only regional service and specific skills, but allowed Fujitsu to add less expensive and temporary capacity for programming operations. Another strategy building on the concept of specialization was to have development facilities gradually place greater emphasis on writing software packages, especially for personal and office computers, and integrating these packages with new software to produce semi-customized systems.

A third area of emphasis was **mechanization and automation**, particularly the use of computer-aided tools to support design, coding, testing, and maintenance. These included automated program generators and reuse-support systems, which management emphasized to improve productivity and quality simultaneously. Supporting all three emphases was the application of automation and mechanization to control technology, in the form of tools for planning, project management, testing, and quality evaluation, integrated with standards, manual procedures, and practices such as quality circles.

Also of long-term relevance to Fujitsu's plans to continue using subsidiaries and software houses was the SIGMA project, initiated in 1985 by Japan's Ministry of International Trade and Industry (MITI). The project involved 128 companies in 1986, including Fujitsu, Hitachi, Toshiba, NEC, and Mitsubishi, as well as 8 foreign firms, and was attempting to increase the level of tool and library support for small software producers by developing a modified version of the UNIX operating system and an on-line data base of tools and subroutines accessible from specially developed engineering work stations.[27] Fujitsu and other major Japanese software producers were interested both in improving tool standardization and general reuse capabilities at the smaller software houses, as well as in selling software tools, particularly software engineering work stations.[28]

SYSTEMS SOFTWARE DEVELOPMENT

Early Experiences

Fujitsu's relay and parametron computers either did not employ software or required only minimal programming. As a result, Fujitsu's experience with software development did not really begin until the early 1960s, with the introduction of several transistorized computers. Yet even these machines, as well as the FONTAC, delivered in 1964 to the Electronics Industry Promotion Association of Japan, still did not have "modern" operating systems but only performed simple batch-processing operations.

The FONTAC project nonetheless initiated a new era, by requiring Fujitsu to develop a large-scale computer roughly equivalent to IBM's 7090 machine (introduced in 1960), and exposing company engineers to various types of programs and computer languages. MITI subsidized the project in an effort to get Japanese manufacturers to develop domestic computers comparable to IBM's. Fujitsu designed the main processor and the card punch equipment, while NEC and Oki Electric made the sub-processors and input/output devices. Fujitsu also took charge of programming, with assistance from NEC and Oki engineers, several Japanese professors, and several U.S. consultants.

The FONTAC software consisted of a "monitor," based on earlier control programs introduced by IBM and available in Japan. The FONTAC also used other software which the monitor program controlled directly: ALGOL, COBOL, FORTRAN, and ASSEMBLER compilers; a library editor; a sort generator; an executable coded tape editor; utility programs; and several object (applications) programs. Fujitsu later modified the FONTAC monitor and introduced this in 1965 as MONITOR II with the 230-50 mainframe, Fujitsu's commercial version of the FONTAC.[29]

Operating Systems Development

The first serious challenge Fujitsu engineers faced in systems software was to develop an operating system comparable to the IBM 360. Fujitsu needed this for the upgraded 230-series models, which incorporated integrated circuits and had, for the time, extensive processing capabilities. This was particularly true for the large-scale 230-60 model, because it utilized two central processing units. Fujitsu started planning for the 230-60 in 1966 and established a software development department to oversee the 100 employees who worked on the software. Fujitsu completed both the hardware and the operating system, MONITOR V, in 1968.[30]

The manager who headed the new software department was Fujitsu's president in 1988, Yamamoto Takuma, a 1949 graduate of Tokyo University's electrical engineering department. Like Ikeda Toshio, Yamamoto first worked in telephone switching equipment design, and in 1952-1953 briefly worked on relay computers. From the mid-1950s through 1966, Yamamoto concentrated mainly on switching and communications data-processing equipment, before heading the MONITOR V effort. The success of Yamamoto's group contributed to his promotion in 1968 as director of software development for systems and applications, and to a subsequent as company president.[31]

According to several Fujitsu engineers, MONITOR V was extremely difficult to develop due to the dual-processor architecture, then available only in U.S. military computers.[32] Furthermore, as Yamamoto recalled, Fujitsu had not yet designed a sophisticated operating system even for one processor, let alone two. Yamamoto, Ikeda, and other engineers made several trips to the U.S. to seek assistance from U.S. software houses and computer manufacturers. This was typical of Fujitsu, where management encouraged engineers to travel abroad frequently, especially to the U.S., to visit consultants, suppliers, and competitors.[33]

Fujitsu ended up using IBM's 360-series operating system as a basic model, with modifications to accommodate two processors, and delivered a first version of the system in January 1969 to the Kyoto University Computer Center. Due to numerous bugs, the system was late as well as incomplete, although Kyoto University personnel worked with Fujitsu engineers to correct existing problems and finish the programming. Based on this experience, Fujitsu had an easier time producing a successor operating system, MONITOR VII, completed in 1974 and also patterned on the IBM 360 software.[34]

The major task of the 1970s was to develop an operating system for the M-series, which incorporated large-scale integrated circuits (LSI) and competed directly with IBM's 370 series. The new system thus had to offer features comparable to IBM and remain compatible with IBM and Fujitsu's earlier 230 series. To minimize development, Fujitsu produced three operating systems to cover the small, medium, and large M-series mainframes, in contrast to the five separate operating systems IBM offered for its 370 series models.[35] The first version of the operating system for its largest mainframes, OS IV/F4, Fujitsu delivered in 1975, and versions for mid-size and small mainframes followed in 1977.[36] But problems completing the software delayed deliveries of the hardware and convinced management of the need for another reorganization and a more serious commitment to developing procedures and tools for software production, testing, and quality control.[37]

One of the key managers responsible for M-series systems software, as well as for the systematization of process and quality control in general, was Miyoshi Mamoru, in 1988 a Fujitsu executive director and head of the Systems Group. Miyoshi began his career in systems software management in 1973, taking charge of inspection for operating systems. Prior to this, he had learned a great deal about programming and quality control by inspecting software for DIPS (Distributed Information Processing System), telephone switching systems produced for NT&T. Not only was NT&T a leader in introducing software technology into Japan in the early 1970s, but developing the DIPS software in cooperation with Hitachi and NEC required Fujitsu, as well as the other Japanese contractors, to adopt rigorous standards and controls. Miyoshi would later make sure Fujitsu's systems and applications divisions adopted similar practices, although when he first moved to commercial software, simply trying to keep up with demand was preventing Fujitsu from instituting tighter standards for process and product, standardization, control, and documentation:

> At the time, NT&T was very advanced in its thinking. The DIPS software was really a large system. Since three companies, including us, had to develop the software, standardization was essential. My impression was that NT&T was able to devote more effort to software phase divisions and carrying out work standardization than we were able to do in the private sector. Of course, we influenced that thinking, too, since there was an interchange among us, an exchange of information...This was a period of rapid growth, and demand for software was increasing rapidly too. We had to write programs quickly to meet this demand, and worried about preparing documentation later. In other words, we gave top priority to writing programs. This tended to defeat documentation and program conformance. In the long run this is self-defeating, and, like NT&T, it is better to review and inspect every phase, and get rid of as many bugs as possible in each phase. But, in the private sector, since software development must have more constraints on time and costs, even though we were aware of this problem, there was not much we could do.[38]

Systematizing Process and Quality Control

Direction of Fujitsu's efforts in process and quality control in systems programming came from the inspection department in the Numazu Works's software division, which Miyoshi headed before becoming division manager in 1974. Yoshida Tadashi, in 1988 the deputy general manager of the division's quality assurance department, prepared for internal use an historical outline describing the evolution of these efforts (Table 7.5). An electrical engineer educated at Yamagata University, he joined Fujitsu in 1965 and worked in telephone switching systems inspection before moving to systems software inspection in 1969.[39]

Yoshida divided this history into three main phases: prior to 1970, when Fujitsu had no system for inspection and quality control, and allowed programmers to test software at their own discretion; 1970-1978, when Fujitsu set up its first systems for product and process standardization, as well as inspection and quality control; and the period from 1979, when Fujitsu introduced structured programming techniques, which assisted in design standardization, and established the procedures that formed the basis for inspection and quality control in the 1980s. Distinguishing the last phase was a broadening of inspection to include not simply testing and documentation conformance, or product evaluation, but analysis of the development process.

Table 7.5: SYSTEMS SOFTWARE PRODUCT-PROCESS CONTROL[40]

1968 Software Engineering Department established, including a Program Testing Section, which assisted in debugging.

1970 "Strengthen Inspection" effort begun (1970-1973). Concept of testing extended from merely debugging to inspecting software from the user's perspective, comparing performance to specifications stated in manuals.

Software Administration Department established.

1971 Software Product Registration System established, requiring new software to undergo inspection and receive product numbers before being released. Product Handling Regulations formalized procedures for software products, manuals, and documentation. Required all these product components to be brought in simultaneously, not one-by-one, to the inspection department for testing and evaluation.

1972 Planning Department established.

1973 Preliminary version of Development Planning Report System launched.

"Application of QC Techniques" effort begun (1973-1978). First serious attempt to forecast bugs, reduce them through consistent procedures, and thereby produce "bug-free" software.

1974 "Product and Process Standardization" effort for begun. Involved estimating what the phases would be for the life cycle of individual software products, and then determining what to design in a given period of time, as well as what programming techniques to use.

Bar charts instituted for project control, replacing PERT charts.

Development started of BSMS (Basic Software Project Management Support System).

Inspection procedures and forms standardized (Main Factor Analysis).

MTS (Multi-Terminal Simulator) and HTS (Hardware Trouble Simulator) tools developed for testing.

1975 SWN (Software **Normen**) Standards effort launched.

Process (phase) divisions formalized.

Procedures for bug analysis and data gathering formalized." OC Data Accumulation" effort formally begun. BSMS data base started.

Estimates Handbook drawn up, with actual data on past projects to aid managers in estimating project requirements, from BSMS database.

1976 Development Planning Report System (Version No. 1) instituted, software products delivery procedures formalized, and BSMS required for in-house use.

1977 "Inspection of Quality and Evaluation Indicators" (1977-1978) program.

1978 Structured programming techniques introduced for utility programs and similar types of software.

1979 "Consolidation of Inspection Ideology" -- Concept of software inspection formally promoted as extending beyond testing, to evaluation of final products and the development process.

"Quality Assurance through Organization" effort launched. Formalization of QA as an organizational function not restricted to inspection but extending to design and planning.

Phase Completion Reporting System instituted.

1980 Development Planning Report System (Version No. 2) instituted, adding productivity data and review data reporting.

Campaign started to increase quality three-fold and productivity 30% by 1982.

Software Division establishes QC circles, as part of the company-wide High-Reliability Program.

Software Division Quality Objectives established -- formalization of procedures for establishing project estimates and standard times.

"Diffusion of Inspection Ideology through Horizontal Development" effort begun to spread knowledge and "good practices" horizontally, i.e. beyond one's own department.

1981 "Advancement of Quality Concepts" movement begun, including ease of use evaluations (ESP) and QAL (Quality Assurance Liaison) initiated to facilitate inter-departmental sharing of data on bugs.

1982 "Performance checking" begun under direction of the inspection department.

Quality Objectives control system instituted and Software Quality Subcommittee established.

Six program development departments established at Numazu Works and located in new software building.

1983 New Software Engineering Division established, centralizing all systems software development in the Numazu Works.

1984 Software Division extends quality assurance and quality circle activities to software subcontractors, as part of the second company-wide High-Reliability Program.

According to this chronology and Yoshida's explanations, based on the MONITOR V experience, Fujitsu decided in 1968 to establish a program testing section, following the lead of IBM. Assisting designers in debugging was the group's primary function. At this time Fujitsu had approximately 800 people developing software, and was experiencing a growing shortage. During 1969-1971, Fujitsu management also increased the burden on programmers by trying to make them more sensitive to customer responses. This involved expanding the mission of testing to evaluate software for conformance to documentation in the customer manuals, to make sure programs performed as manuals said they would.

The next step was to create specific product-inspection procedures, as opposed to simple debugging. The most important measures required completed software to go through a formal inspection process before release to the customer. To manage this, Fujitsu established a software administration department and instituted specific procedures for product handling, evaluation, and registration. The product handling procedures covered the software itself, as well as manuals and other documentation, requiring all to be in conformance and brought to the inspection department together. Previously, software was sometimes completed and released without proper documentation. From 1971, however, no product received a registration number, which was necessary for release to customers, without meeting these requirements and gaining the formal approval of the inspection department.

During 1973-1978, Fujitsu made its first serious attempts to control bugs in software through three interrelated efforts: applying quality control techniques, standardizing software product controls and development practices, and collecting quality and performance data. A planning department organized in 1972 oversaw these initiatives for the first few years. This contained sections for operating systems, linear programs, and DIPS, and thus facilitated a transfer of controls already required for NT&T to commercial software.

In 1975 Fujitsu started a development planning report system, which set up formal work-estimation procedures for project managers to use in estimating man-power needs, quality objectives, product objects, and schedules. Table 7.6 lists the development planning system documentation and groups responsible for checking. This system was characterized not by divisions of authority or tasks, but by sharing of responsibility for quality and planning control in all development and testing phases.

In addition, to automate part of this system, especially data collection on quality and budget control, Fujitsu introduced BSMS (Basic Software Project Management Support System) in 1975, after two years of research and trials. The new procedures, supported by BSMS, introduced not only standards but also a tool for estimating schedules for different development phases and managing the development process, as well as testing and inspection.

BSMS, which continued to be the centerpiece of Fujitsu's production-management system for basic software in the 1980s, worked as follows. First, managers had to submit a product number application to begin a project. If this was approved, they submitted a budget. The control system then centered on the development plan documents and monthly reports of actual work hours, computer time, and phase completions, which BSMS tracked and compared to the initial estimates.[41] Fujitsu had used PERT charts for developing MONITOR VII but they did not work well. The development planning system called for the use of bar charts for project control, which Fujitsu later combined with the on-line BSMS system.[42] The introduction of BSMS also allowed the software engineering department to begin compiling an "estimation handbook" containing actual data on past projects to aid managers in their estimates.

Table 7.6: DEVELOPMENT PLANNING ITEMS AND RESPONSIBILITIES[43]

Key: C = Control, P = Planning, E = Engineering, D = Development,
T = Integration Testing, I = Inspection, o = Included in Activities,
X = Major Responsibility

		Groups Responsible for Checking					
Items	Check Points	C	P	E	D	T	I
QUALITY OBJECTIVES:							
Development Objectives	Program positioning, development results, marketability	o	X	o	o	o	o
Desired Functions	User needs and program functionality		X		X	o	X
Performance	Objectives, measures, comparative analyses, appropriateness of the measurements		o	o	o	X	X
Compatibility	Objectives, appropriateness and completeness of the objectives		X	o	o	o	X
Reliability	Appropriateness of the objectives, likelihood of implementation		o	o		o	X
PLANNING IMPLEMENTATION							
Development Size, Language	Appropriateness of size given functions, modularization/reuse	X		X		o	o
Development Process	Appropriateness of delivery time to user, feasibility	X	o		X	X	X
Man-Power Machine Time	Match of productivity and budget, man-power allocations, subcontracting percentage, progress	X					o
Review & Test Plans	Amount of reviews & test, appropriateness of bug detection (compared to phase standards)					X	X
Work Objectives	Sufficiency of measures for improving quality and efficiency	o		X	o		o
Development Structure	Organization, work distribution	o				o	o

Fujitsu divided the development process into seven phases, following conventional life-cycle models: basic design; structural and functional design; detailed design and coding; unit and combined test; component and system test; product inspection; delivery and maintenance.[44] Personnel in each phase were responsible for standardized documentation and reports. At the initial phases, the development and planning documents included a market survey report and a basic design document, which listed functional, performance, and reliability objectives, as well as estimates of manpower and machine time.

Completion of functional and structural design required documents detailing the function of each system component, module structure, and interfaces between each module, plus a testing plan. Detailed design required flow-chart and tabular documentation on the internal structure of each module, and input and output structure. At the completion of unit and combined test, Fujitsu required a programming report, which included the detailed design documentation, review reports, test specifications and results, as well as the source modules. After system test, the testing report documentation included the inspection specifications, test specifications and results, and the test set, while final inspection generated another set of results.[45]

The most important quality measures Fujitsu initially focused on were reliability in component test, conformance to documentation, number of bugs detected, and mean time between failures. Main-factor analysis techniques, introduced in 1974 and influenced by practices at Hitachi, required testers to sort through intermediate test data to identify sources of bugs, before final testing. Standardizing testing and review procedures was particularly important, because Fujitsu had no consistent approach for predicting and eliminating bugs. Individual programmers decided which tests to run, without formal planning or supervision. Better control in this area then allowed Fujitsu in the later 1970s to track other measures, such as software functionality and portability.[46]

The period from 1979 Yoshida characterized by three themes: quality assurance through organization, diffusion of inspection ideas through horizontal development, and advancement of quality concepts. The central notion in assuring quality through organization was to make quality assurance activities part of the formal organizational and job structure, especially in design and planning, rather than a function restricted to the inspection department. Fujitsu implemented this new emphasis by instituting in 1979-1980 a "phase completion reporting system," resembling the design-review systems used at other firms, and then a new version of its development planning report system. The new reporting system required, for the first time, programmers and managers to collect productivity data. Standardized productivity data then allowed Fujitsu to introduce in 1980 standard times for worker performance, based on actual previous data for different types of software, and incorporating quality objectives. Although Fujitsu was several years behind Hitachi, NEC, and Toshiba in instituting standard times as well as structured programming techniques, these became a major feature of project control and tool development. Like other Japanese companies, Fujitsu also adopted the practice of revising standard times annually.

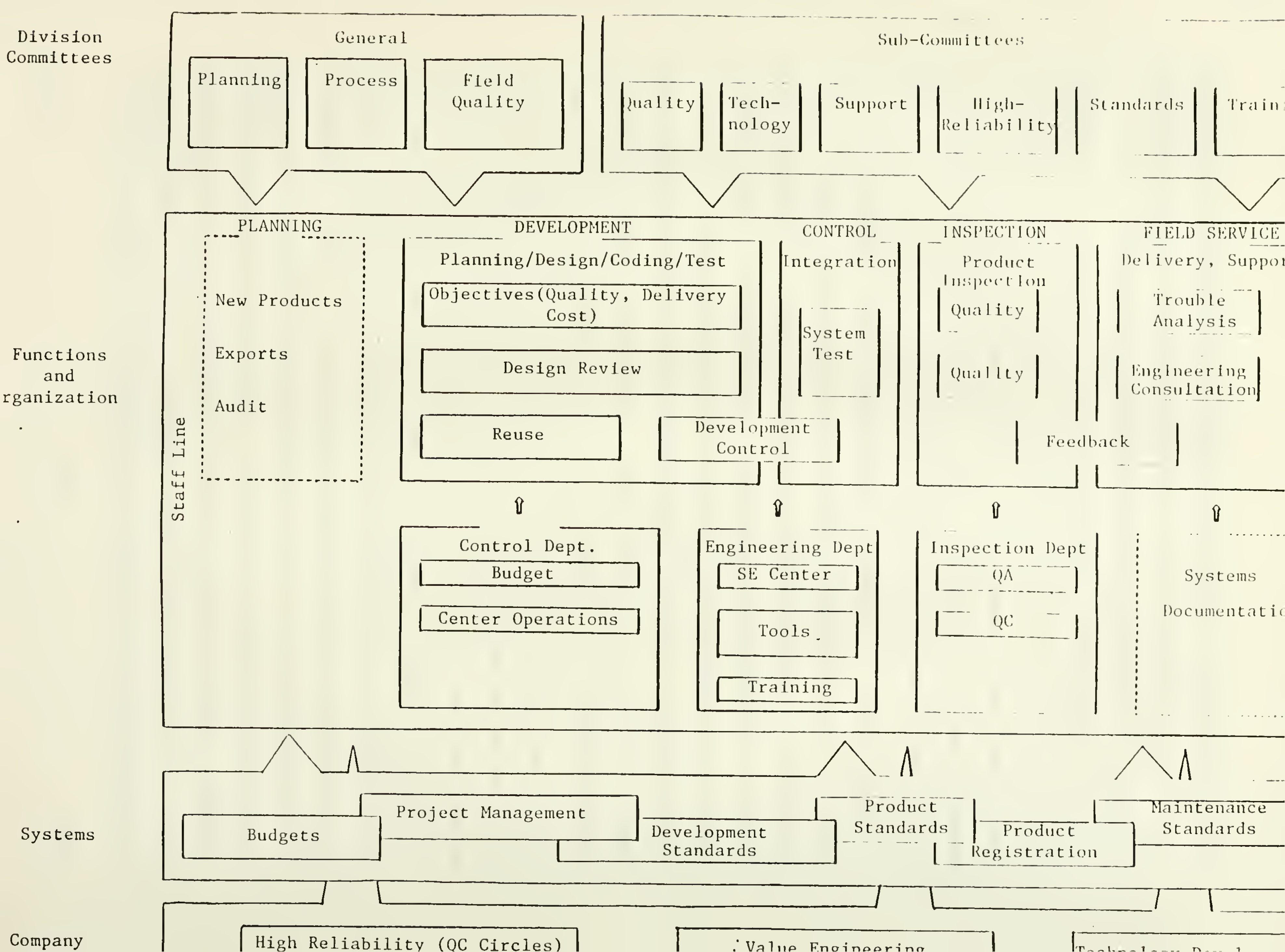
Other measures included in the quality assurance effort were the institution of practices to check for bugs introduced in one part of a program when fixing bugs in another part; and participation of the division in the company-wide "High-Reliability Program." Fujitsu managers also set a goal of raising productivity 30% by reducing bugs one-third; even though the division did not meet. this objective, this established the policy of connecting productivity to quality control, and setting specific goals to motivate employees. To support this initiative, in 1982-1984, Fujitsu introduced new testing tools and techniques to help select test items, and purchased additional equipment to insure that testing could proceed 24 hours a day, if necessary.

What Yoshida called "horizontal diffusion of inspection ideas" referred to deliberate efforts to spread good thinking and practices regarding quality control beyond one's small group. This involved institution of a formal "quality assurance liaison" (QAL) structure, which set up meetings for different departments and projects to share information on bugs and other quality-related data. As part of this initiative, Fujitsu also established other quality control activities and meetings, including quality circles, initiated for software in 1980. The software division also participated in the second High-Reliability Program, which promoted the teaching of quality control techniques to subsidiaries and subcontractors. "Advancement of quality concepts" centered on expanding interpretations of product quality beyond "zero bugs" to other characteristics, such as product design, function, performance, ease of use, and maintainability, as encouraged by U.S. experts like Barry Boehm.

Also in the early 1980s, Fujitsu began applying to software the term "TQC" or "Total Quality Control," a term coined in the 1950s in the U.S. and applied widely by Japanese firms since the 1960s, including Fujitsu, to comprehensive quality assurance efforts covering product planning, design, manufacturing, and service, rather than simply inspection after production. Software TQC goals in Fujitsu focused on improving control over bugs, delivery dates, and costs, which management hoped to achieve through division-level committees coordinating lower-level activities in all areas affecting software products, from planning through maintenance (Figure 7.1). A series of committees for product and process planning, field quality evaluations, quality circles, technology, standardization, training, and high-reliability promotion set policies and supported the efforts of line and staff departments for planning, development, software engineering, control, inspection, and field-support. Procedures for project and budget control, work and product standards, product evaluation, registration, and maintenance completed the TQC system.[47]

The High-Reliability Program presents an example of how the software division in Fujitsu, like other Japanese firms with software factories, participated with conventional factories in company and group efforts to improve quality and productivity. Fujitsu started this program in 1966, initially in hardware development and manufacturing divisions, in an attempt to institute a comprehensive TQC system covering research, design, manufacturing, and service. The program gradually expanded the use of quality circles, which met once or more a month to discuss a range of issues related to productivity and quality, as well as work conditions. Managers also used the circles to supplement formal training required of new company employees.

The software division did not actively participate in the program until 1980, when Numazu organized its first software quality circles (a year before NEC). By 1985, the software division had approximately 300, averaging 7 members and meeting once per month. The quality assurance group within the inspection department oversaw circle activities, while section chiefs organized and managed them. The most frequently discussed themes in 1985 were software maintenance problems and techniques (30%) and testing (18%), followed by programming, bug analysis and correction, design, reviews, manuals, planning, and inspection. Management also encouraged the circles and individuals to submit suggestions, which in 1985 dealt mainly with maintenance, programming, control (quality, process, cost), testing, computer usage, and bug analysis and correction. Under the influence of the second High-Reliability Program, the software division in 1984 also began working with subsidiaries and contractors to establish similar TQC systems, including quality circles.[48]


Division Committees
General
Planning
Process
Field Quality
Sub-Committees
Quality
Tech-nology
Support
High-Reliability
Standards
Train
Functions and rganization
PLANNING
New Products
Exports
Audit
Staff Line
DEVELOPMENT
Planning/Design/Coding/Test
Objectives(Quality, Delivery Cost)
Design Review
Reuse
Development Control
CONTROL
Integration
System Test
INSPECTION
Product Inspection
Quality
Quality
FIELD SERVICE
Delivery, Suppor
Trouble Analysis
Engineering Consultation
Feedback
Control Dept.
Budget
Center Operations
Engineering Dept
SE Center
Tools
Training
Inspection Dept
QA
QC
Systems
Documentatio
Systems
Budgets
Project Management
Development Standards
Product Standards
Product Registration
Maintenance Standards
Company
High Reliability (QC Circles)
Value Engineering
Technology Devel

Quantifying Control Measures

Similar to other Japanese factories, Fujitsu's Numazu Works attempted to quantify and analyze basic measures of performance in a standardized manner, and then institute standardized procedures and tools. One reason was to reduce individual variability and thus dependency on high-levels of experience and skill, much as conventional factory organizations have attempted. Another, related objective was to "build in" quality at each phase of development, rather than trying to "inspect in" quality at the end of the process. Yoshida reflected this thinking in a recent article describing the software division's program for quality assurance:

> The prevailing opinions that high-quality software products can be produced readily by using certain methods, that quality can be ensured by sufficient testing, and that quality control can be achieved through the diligent work of a single control section are mistaken. Better development methods, appropriate standards, and various control activities must be applied throughout the entire software life cycle, and quality can be assured only if all the above are carried out under an integrated management system throughout the life cycle.[49]

To institute this type of management system, Fujitsu staff in inspection and quality assurance worked on four approaches. First, were a series of activities to quantify and analyze information on project progress and quality. Second, was the introduction of methods for predicting error occurrence in the programming phase, and institution of a "management by objectives" system aimed at error correction. Third, to determine test items, was the use of statistical "Design of Experiment" techniques, often referred to as the "Taguchi method" in the U.S. and generally applied only to product development and processing of conventional hard products such as automobiles. Fourth, was an attempt to evaluate quantitatively the concept of "user friendliness" of software through a variety of tests.[50]

Measuring design progress was problematic mainly because the most common way to evaluate schedules was to compare actual progress, such as the number of completed modules, or in Fujitsu's case, completed "design sheets" and "work items," with estimates made prior to starting work. Accuracy of the measure thus depended on the accuracy of the prediction, which managers could not fully determine until completion of a project. To maximize the accuracy of estimates, Hitachi kept extensive data on previous projects and the performance and experiences of individual personnel. Fujitsu chose not to keep such detailed data, but used two similar indices for project evaluations. One measured quality, based on points received in reviews, and another quantity, by how well predicted amounts of work fit actual results. This data helped managers define tasks in small, realistic segments, to minimize estimation errors.

The system worked as followed: In the planning stage, managers determined work items and entered these on work item sheets, which then were entered into the BSMS data base. Work items were segments of design or other activities that could be done in about two weeks. Projects were organized in groups, and each group leader was responsible for one work item at a time. Each member then had to fill out a "review trust sheet," as well as receive reviews from several other people, including a chief reviewer. Reviewers used a check-list to evaluate work, and members had to make changes until the chief reviewer was satisfied. At weekly progress meetings, managers and group leaders checked the progress of each work item. The quality index compared the number of quality points received in reviews with the number of predicted quality points, assigned through a simple scheme: Completion of materials for reviews received 50 points; completion of all reviews and changes required brought 100 points. Actual quality points were assigned by giving 50 points for completion of materials for reviews, and 50 or more points depending on the review evaluations.

Using these formulae, Fujitsu arrived at "comprehensive quantity figures for each member and project," as well as made an initial tracking of product quality. In addition, as part of its quantitative measurement efforts, Fujitsu regularly used statistical regressions and factor analysis to identify probable causes of errors, as well as to estimate numbers of bugs and man-power requirements, and to test the accuracy of estimates. Progress evaluations still depended on the accuracy of the predicted values, which sometimes were far from actual values. Overall, however, Fujitsu managers claimed to have solved the major problems with the system and, reported significant improvements in progress control as well as quality.[51]

Testing Techniques

Fujitsu's objectives in testing were to detect 65% of the bugs in a new program by the end of the coding phase review, 95% by the end of system test, and 99% by the end of the inspection process.[52] A problem encountered in meeting these goals, however, was that, as in the design of any complex product or process, large programs contained too many combinations of factors and conditions to test completely. Furthermore, company data showed significant improvement in the ability of personnel to detect errors in software as their experience increased, with a leveling off after about six years. As a result of these observations, Fujitsu managers decided to employ a method for selecting test cases that less experienced could understand, as discussed in a 1987 article by an engineer in the quality assurance department:

> In tests involving many inspectors, a means is essential to standardize the quality of testing conducted by each inspector, but there is a limit as to the extent to which one can share the knowledge and skill through training...The omission of test cases during test factor

> analysis is often associated with problems relating to the scope of knowledge and insight of the testing personnel... The omission of test cases during test case generation is often caused by omissions or careless errors... [T]he number of test factors [selected] increases in proportion to employed years up to 6 years and stays constant after that. Therefore, to improve the quality of test factor analysis as a whole, it is a significant importance [sic] to transfer knowledge to less experienced testing personnel.[53]

Two approaches appeared most useful in capturing and transferring knowledge regarding software testing. One was to develop tools that automated as much of the testing process as possible, particularly the generation of initial tests based on a program's external specifications or input characteristics. Such tools often could identify critical factors that were the source of most errors in a program. The conventional way of generating initial test cases was to use cause-effect graphs, but these required a great deal of knowledge to use effectively. A second approach was to create a data base on test-factor analysis methods and conditions, complete with an editing support function to help personnel utilize the data base to create test cases for new programs. As part of this latter technique Fujitsu introduced, beginning in the early 1980s, Design of Experiments methodology.

The Design of Experiments techniques relied on educated guesses of where problems were likely to exist (which could be listed for easy reference), and then specific tests selected and evaluated relying on tables of "orthogonal arrays" (statistically independent groups of factors). These made it possible to control the number of combinations that existed between any two factors and minimize measurement errors and other conditions, and identify correctable defects more quickly than conventional techniques such as cause-effect graphs. In fact, Fujitsu data indicated that the DE methods could actually detect 5 to 10 times or more the number of errors usually found with conventional methods, with fewer test cases.[54]

To quantify "ease of use" measures, Fujitsu employed manual reviews and inspections, and validation of accuracy by surveying users through questionnaires. The surveys indicated that users grouped features relating to ease of use into three categories: major (such as friendliness, efficiency, understandability), intermediate (such as productivity, ease of learning, maintainability), and minor (such as syntax flexibility, speed of operation). Scoring a product on various items, using simple "yes" or "no" responses and more points for major as opposed to minor items, led to quantified scores. Fujitsu claimed its evaluations covered all the basic characteristics of a software products (performance, reliability, operability, compatibility, coordination, ease of maintenance, quality of information, price and delivery), as well as design quality and consistency (defined as the "degree to which software targets and specifications meet user needs," and the "degree to which the finished software meets the design targets and specifications"), and sufficiency and attractiveness ("the extent to which products are acceptable to users"). Fujitsu also claimed to use 113 items merely to evaluate ease of use.[55]

Tool Support

Similar to Hitachi, Toshiba, and NEC, in the mid-1970s Fujitsu began developing numerous support and management tools; major ones (for systems software primarily) are listed in Table 7.7. Numazu was the center of tool development, with Kamata importing tools such as ADAMS and GEM. During the 1980s, however, Kamata and the central research facility, Fujitsu Laboratories, have become more involved in tool development, such as those directed at automating system design and construction.[56]

Table 7.7: MAJOR SYSTEMS SOFTWARE DEVELOPMENT TOOLS (1983)[57]

DESIGN

YAC II (new version 1987) (Yet Another Control Chart). Detailed design language, combining aspects of conventional flow charts and pseudo code.

PROGRAM EDITING/CODE GENERATION

GEM (1979) (Generalized Program Editing and Management Facilities). Automatically maintains a development history of a program. Linked to PRISM.

PRISM (1982) (Problem and Repair Interrelated System Management Extended). Maintains a data base of the program source and automatically tracks maintenance changes. Linked to GEM.

COMPACT (1982) (Compact Print-out Utility). Compacts and prints out compiled and assembled code.

TOOL 2 (1983) Allows on-line search, from time-sharing terminals, of program source code and lists.

YPS (1987) (YACII Programming System). Allowed the user to edit YACII diagrams and then automatically generated code in several languages.

TESTING

SAT/ART (1980) (Systematic and Automatic Testing/Automatic Regression Testing). Automated regression testing tool for operating system development.

TDQS (1977) (Test Tool for DQS). Automated regression testing tool for on-line display operations.

INDS (1983). (Interactive NCP Debugging System). Allows checking and analysis of NCP test results from time-sharing terminals.

MTS (1971, 1977) (Multi-Terminal Simulator). Simulates remote terminals for host load testing.

TIOS (1977). (Time-Sharing System Input/Output Simulator). Simulates remote terminals for host load testing.

HTS (1977) (Hardware Trouble Simulator). Simulators hardware bugs for functional test evaluations of software responses.

DOCK/FORT77 (1981). Debugging system for FORTRAN, using a "slow video display."

PIC (1984) Program Information Control System. Allows inspection and analysis of memory-dump lists from time-sharing terminals. Linked to the Kamata-Numazu Dump Transfer system.

ATOS (1982) AIM Test Oriented System. Automatically records on computer data base results from testing and inspection.

DOCUMENTATION

ODM (1983) (Office Document Manager). Document handling system for Japanese language.

MANUAL COMPILATION AUTOMATION SYSTEM (1979). Automated editing of electronic documentation files.

EGRET-4 (1983) (Easy Graphic Report Generator).

ATLAS (1982). Automatic translation of Japanese documents into English. English to Japanese system added in 1986.

MAINTENANCE

TDP II (1980) (Total System for Difficulties Information Processing). Data base for software report information.

ITS (1981) (Incident Tracking System). Control system for questions from customers.

KAMATA-NUMAZU DUMP TRANSFER (1982). On-line transfer of data and reports on bugs. Linked to PIC testing tool.

PDE (1983) (PTF Document Editor). Automatically edits documents on program corrections, based on TDP II data.

CONTROL

BSMS (1976) (Basic Software Project Management Support System). Data base and control tool for software project management.

NOA (1978) (Numazu Office Automation). Employment data control system.

IN-HOUSE TRAINING DATABASE SYSTEM (1982). Relational database system for in-house training administration.

Fujitsu's tools closely resembled those in use elsewhere in Japan, though they operated separately and were less integrated than the work-bench systems at Hitachi and Toshiba. There were, however, data-base links among TDPII, ITS, and the Dump Transfer system (maintenance), BSMS and TDP II (control), and GEM and PRISM (program construction).[58]

Particularly important among these tools were BSMS, described earlier, and GEM (Generalized Program Editing and Management Facilities), a library-management and data-collection tool that automatically generated graphic reports, for groups and individuals, on productivity (lines of code developed), progress status (by month and week), and quality (bug levels per module). The library functions included version control of modules and completed programs, as well as compressed data to reduce the volume of stored files.[59]

Another important tool set was YAC-II (Yet Another Control Chart), which Fujitsu used for functional design, and YPS (YACII Programming System), which automatically generated code. The YAC diagrams, first introduced by Fujitsu in the 1970s, resembled other flow-chart or problem-analysis diagrams used in Japanese firms since the early 1970s, after NT&T began developing an internal system and requiring its use by subcontractors. The current version, introduced in 1987 along with YPS, enhanced productivity in two ways. One was to eliminate the need for detailed documentation. Another was to reduce coding and debugging time. The YPS system contained an editor, compiler, debugger, and documentor, and received inputs in structured but conversational Japanese on work stations. A host computer then translated the inputs into machine-readable YACII charts and then executable, "bug-free" code in C, Fortran, Cobol, or SPL (System Programming Language, a PL/I derivative developed at Fujitsu).[60]

Performance Improvement

Fujitsu data indicated steady improvements in the decade after 1975 in quality (reliability), productivity (development costs), and project schedule control (lateness). Available data summarized in Table 7.8 cover systems software, where the average program ca. 1986-1987 was about 170,000 lines of code with comments (125,000 lines without comments), with a maximum of about 800,000 lines. These programs were primarily written in SPL or C.[61]

For all code in the field (new code plus maintained software), between 1977 and 1979, bugs reported by users dropped by one-third, and then another one-third between 1979 and 1982. By 1985, bug levels for outstanding code had fallen to practically zero (0.01 and below). For newly written code (annual data for which is confidential), Fujitsu showed similar gains. Bugs per 1000 lines of new code reported by users (generally about 10 times the level for all code) fell four-fold between 1980 and 1981 alone. Improvement since 1981 has leveled off, as bugs dropped to approximately 0.1 and below.

Table 7.8: SYSTEMS SOFTWARE DEVELOPMENT PERFORMANCE MEASURES[62]

Note: All data are estimates based on graphs, and therefore are approximate figures only. 1983-1985 is based on internal Fujitsu data. Bugs/1000 LOC refers to all bugs reported by users per 1000 lines of code over 6-month periods (averaged in the table below) in the field, i.e. newly delivered code plus supported outstanding code.

	Bugs/ 1000 LOC	Bug Detection Method (Estimates)			Development Costs per 1000 LOC (Index)
		Test	Source Code Review	Design Sheet Review	
1975	...	...	...	...	100
1976	...	...	...	...	87
1977	0.19	85%	15%	--%	61
1978	0.13	80	15	5	54
1979	0.06	70	20	10	50
1980	0.05	60	25	15	50
1981	0.04	40	30	30	50
1982	0.02	30	30	40	37
1983	0.02	...	...	...	35
1984	0.02	...	...	...	...
1985	0.01	...	...	...	...

Data on methods used to detect bugs reflects the growing use and effectiveness of design-sheet reviews. In 1977, approximately 85% of all bugs found were detected through testing, with 15% found through reviews of source code (coding phase review). With the institution of design sheet reviews in

1978, this has gradually become the major method for detecting bugs, rising from 5% to approximately 40% in 1982.

Fujitsu measured productivity by lines of code (steps) produced per man-month, number of documents per man-month, and total development costs per 1000 lines. Two other related indices were machine time divided by man-months, and pages of documentation per 1000 lines of code.[63] The data on development costs indicated a three-fold productivity improvement between 1975 and 1983, not adjusting for inflation or use of outside contractors to reduce expenses. The most dramatic improvements came with the opening of the Numazu Works in 1976, and then the centralization of system development in the Numazu Works after 1981. Lines-of-code data also suggested that, between 1975 and 1985, including reused code, Fujitsu experienced roughly a 5-fold increase in output per worker.[64] In-house studies indicated that factors most strongly affecting productivity on the positive side were programmer ability, followed by product complexity and software tools. On the negative side were reliability requirements.[65]

Improvements in quality and productivity corresponded to significant progress in scheduling accuracy. In the late 1970s, according to Yoshida, about 40% of projects were typically late, with "late" defined as reaching the inspection department after the scheduled time. By the early 1980s, Fujitsu had reduced this to about 15%, a level comparable to Hitachi. Delays most frequently came in the transition from functional design to coding, with coding too often taking more time than estimated.[66]

APPLICATIONS SOFTWARE DEVELOPMENT

Factory Origins and Organization

Fujitsu's decision to establish a software factory for applications programming stemmed from the need to produce a variety of nominally different programs for different customers. In the early and mid-1960s, users performed much of their own applications programming, with assistance from a service department Fujitsu set up with its computer division in 1963. Several large system orders, however, led Fujitsu to organize a systems department in 1964 (later expanded into the Systems Group). These orders consisted mainly of integrated hardware and software systems, such as on-line programs for securities firms and banks (first delivered to Nikko Securities in 1964 and the Norin Chuo Bank in 1965), and data processing software for Fujitsu mainframes delivered to government customers (such as the Ministry of Labor in 1965) and NT&T (beginning in 1967).[67]

Rapid increases in sales of Fujitsu computers brought continued demand for customized applications software, prompting Fujitsu management to establish in 1970 the Information Processing System Laboratory in Kamata, Tokyo.[68] The Laboratory was divided into a systems development area and an education area. An initial staff of 700 allowed Fujitsu to centralize program development for a variety of industrial, engineering, and scientific applications, as well as for computer-aided design, pattern recognition, and computer-aided instruction. By the 1980s, Fujitsu was using the Kamata facility to produce software for nearly all its mainframes, minicomputers, and super-computers, as well as for office computers, terminals (banking, postal), and personal computers.[69]

Although the U.S. software market shifted gradually toward software packages and away from fully customized systems, Japanese customers continued to prefer tailored software systems. In response, according to Fujitsu director

Yoshiro Nakamura, Fujitsu created a Systems Group, centered on the Kamata facility, that not only offered specialized engineering and programming departments, but also facilitated the development and reuse of software packages, which could be integrated with new code as necessary. The mission of the Software Factory and subsidiaries or software houses connected to the Systems Group was primarily to use these packages, as well as a factory-type methodology and tool set, to produce semi-customized systems rather than standardized products (packages) or fully customized products.[70]

As of 1986, the Systems Group consisted of three major areas, with a separate Development Planning Office directing tool and methodology research (Figure 7.2). The "joint development" area included the SE (System Engineering) Technical Center, which housed the Software Factory and departments such as for standardized technology promotion, office system and other system engineering done in the center, and the SIGMA project; the Package Planning and Control Division; and a research facility.[71] A second area was the industry-specific design departments, which performed system engineering for companies in finance, insurance, securities, and manufacturing and distribution, as well as for NT&T, scientific and technical applications, and government users. The third area consisted of functionally specialized departments, such as for management information systems, "value-added networks" for offering different on-line services, personal computer systems, and new telecommunication firms (NCCs).

Figure 7.2: 1986 SYSTEMS GROUP AND SUBSIDIARIES ORGANIZATION[72]

SYSTEMS GROUP (ca. 9000 Employees in 1988)

Development Planning Office

JOINT DEVELOPMENT AREA

--- SE Technical Center
-- **Software Factory Department**
-- Information Center Dept.
-- Standardized Technology Promotion Dept.
-- System Engineering Dept.
-- Office Computer System Engineering Dept.
-- SIGMA Project Office

---- Package Planning and Control Division
-- Package Planning Dept.
-- Software Distribution Dept.

- - Fujitsu Systems Research

INDUSTRY-RELATED DEPARTMENTS
--- Finance
--- Insurance/Securities
--- Manufacturing/Distribution
--- Scientific/Technical
--- NT&T
--- Government/Mass-Communication

FUNCTIONAL DEPARTMENTS
--- Management Information Systems
--- VAN (Value-Added Network) Systems
--- Information and Communications
--- Personal Computer Systems
--- NCC (New Common Carriers) Systems

SYSTEM ENGINEERING SUBSIDIARIES (ca. 53 firms and 11,000 Employees)
--- Industry and Functional
--- Regional

While Fujitsu made efforts to centralize applications development in the 1960s and early 1970s, only in the later 1970s did it adopt a factory-type organization separating system design from program construction (Table 7.9).

According to Murakami Noritoshi, one of the developers of the Kamata Software Factory and in 1988 a manager in the Software Development Planning Department, a preliminary step was the 1977 establishment of a Software Conversion Factory Department. Fujitsu used this to revise applications programs written for Hitachi and IBM computers and terminals to run on Fujitsu machines. Next, in 1979, Fujitsu expanded the department into a small software factory, with about 300 programmers charged with turning requirements specifications received from system engineering into code.

The major reason for upgrading the conversion facility, according to Murakami, was to centralize people and standardize process technology, with the objective of accumulating knowledge regarding tools, standardization, programming environments, and development techniques. The factory was able to handle about 25% of all commercial applications programming in Fujitsu.. Most other programming work Fujitsu channeled to subsidiaries and affiliated software houses.[73]

Prior to establishing the factory, a group headed by Yoshiro Nakamura began compiling the SDEM (Software Development Engineering Methodology) standards in 1976, and introduced a version for in-house use in 1977. As occurred with operating-system development, the larger programs needed for the M series computers required Fujitsu to standardize and integrate software engineering tools, methods, and management procedures.[74] Fujitsu also issued a tool set, SDSS (Software Development Support System), for programming in Fortran. As more business programs were written in COBOL, Fujitsu replaced SDSS with new tools, later integrated under the name SDAS (Systems Development Architecture and Support Facilities).[75]

Table 7.9: APPLICATIONS SOFTWARE FACTORY ESTABLISHMENT

1964	Establishment of Systems Department, later expanded to Systems Group
1970	Centralization of applications systems development at Information Processing Systems Laboratory in Kamata
1977	Establishment of Software Conversion Factory within the Laboratory to convert IBM and Hitachi programs to Fujitsu machines
	Introduction of SDEM standards
1978	Introduction of SDSS tool set
1979	Establishment of the Software Factory Department to perform detailed design, coding, and integrated test of customized applications programs specifications received from system engineering departments

Since specialized knowledge seemed essential for accurate system design, Fujitsu maintained specialized system engineering departments outside the software factory, covering manufacturing, distribution, financial systems, government systems, and scientific and engineering programs. The SE departments handed designs to subsidiaries and affiliated software houses, or programming sections in the software factory, where programmers also specialized in certain applications or industry areas.

The factory's (and subsidiaries') workers took system documentation and produced detailed designs (flow charts) and code (if necessary), and continued through integration test. System test was then done by the designers and programmers together at the customer site. Prior to the factory organization, system designers either did their own coding or worked in the same groups as programmers, rather than handing off design documents.[76]

The factory departments were organized in teams of 7 or 8 programmers doing coding for 4 or 5 system engineers. The system engineers wrote design specifications in a standardized format using conversational language (Japanese mainly), but providing input/output diagrams and mathematical equations as necessary, to minimize the amount of skill required of the factory personnel. Programmers who did not understand part of a design document would call the responsible system engineer and discuss the problem. When programmers finished a part of a system, they would send it back to the system engineers, who would check it with the customer. Most problems in specifications were thus ironed out during the design and coding process.[77] Fujitsu also minimized the type of "matrix management" problems that had plagued the SDC Software Factory by designating a chief system engineer and a chief programmer, and then giving the system engineers the main responsibility for dealing with customers and producing systems that correctly met customer specifications.[78]

System engineering departments ~~mi~~ght send designs either to the factory or subsidiaries for implementation, hence there was not a strong distinction between the type of programming these different organizations could handle. In general, however, the factory provided capacity for systems that subsidiaries might have a difficult time with, such as on-line banking software, which could reach millions of lines of code. For these types of large programs, the factory employed an "on-line" SDEM system that allowed up to a thousand people to work on a single project simultaneously, with 200 to 300 terminals connected to a single mainframe. (The Kamata factory maintained about one terminal for every two employees, twice the number at Toshiba.) The largest projects done in the factory might use 4 to 6 mainframes, and involve a thousand personnel over a period of one or two years. The factory also did some programming for packages designed in the system engineering area, if there was a work overflow in the SE departments' new-code development groups. In addition, the factory avoided jobs that required a great deal of expertise on the part of programmers, such as artificial intelligence programs, which Fujitsu did in R&D departments.[79]

Software houses that worked on a long-term basis with Fujitsu were particularly valuable when projects were running late. Fujitsu would often add personnel from these firms, primarily to help in coding (although program generators were gradually eliminating coding). As at other Japanese software factories, Fujitsu appeared able to add people without delaying projects further because of the standardized methods and tools, and training, even of subcontractors' personnel. Another advantage of using subsidiaries and software houses was that Fujitsu could meet demand peaks beyond its own capacity and without adding too many full-time (and higher paid) employees to the ranks of permanent employees.[80]

Training and Career Paths

While most employees in the Kamata Software Factory were college graduates, and usually had good backgrounds in science or mathematics, new employees generally had no computer engineering or software training.[81] This required Fujitsu to invest heavily in training, but allowed the company to teach workers, and managers, the factory methods and no others.

For example, after deciding on the SDEM standards, in 1977 Fujitsu instituted a three-day training seminar, followed with periodic workshops, for project managers, and a lengthier education program for new employees.[82] New employees were considered trainees for the entire first year and attended full-time classes for three months, learning Assembler, COBOL, machine I/O, and other areas of basic programming and computer architectures. After the classes, training continued on-the-job, through coding simple programs for commercial customized systems, and through another two or three days of additional instruction every month or two.[83] Until they became managers, employees continued to receive about eight days of training a year in workshops, in addition to self-study materials and quality-circle activities.[84]

Career paths at Fujitsu resembled those at other software Factories, though they were not as formally defined as at Hitachi and Toshiba. New employees usually did coding for several years, and then moved on to module design, and structural design. After 5 to 10 years, depending on an individual's ability (as determined by managers, rather than formal testing), a programmer might become a system engineer (designer). Programmers in the factory, like system engineers, specialized by application or industry area for at least their first 3 years, but then might move to other areas.[85]

There are some reports that lack of a formal training program for system engineers, again in contrast to Hitachi and Toshiba, made it possible for Fujitsu's competitors to provide better system-engineering services. To counter these claims, in 1986 Fujitsu established Fujitsu Systems Research as an in-house department for training personnel in system consultation and assisting customers in system planning and problem solving. Fujitsu staffed the new department with 30 experienced administrators and 70 experienced system engineers.[86]

The Factory Methodology: SDEM

Fujitsu introduced SDEM (Software Development Engineer's Methodology) and SDSS (Software Development Support System) during 1977-1978 to serve as standardized development and control procedures and automated support tools aimed at improving customer-requirements definition and techniques for design, programming, project control, and system maintenance. Managers were particularly concerned with the constraints on productivity created by design changes and documentation rewriting (usually 20 to 40% by the final version) and test-case generation.[87]

Fujitsu had standards prior to 1977, but they were only vaguely defined and weakly enforced. Development was very much in a "craft" mode, with programs more the property of individuals than the product of an organization. Documentation and standardization were so lacking that, as recalled by the engineers who developed SDEM and SDSS, it was difficult to understand a program written by someone else:

> We had software development standards before SDEM and SDSS were developed but the standards had no clear definition of the development phases. This lack of definition resulted in so much freedom that software systems often became the products of individuals. It was not easy for a person who had not designed a program to understand how it worked. To improve on this situation, we concluded that clear, reasonable definitions of the developing phases and activities were required. The kinds of tasks performed during each activity also needed standardization.[88]

According to Nakamura, they considered two options: improvement and standardization of procedures, techniques, and the overall development environment; and automatic program generation, relying on standardized formats for specifying user requirements and "state-of-the-art" tools. Due to the still-emerging state of program generation, they focused initially on standardization, and later on program generation. It was also their philosophy that, before proceeding with automation, they had to establish a methodology and a production system, and solve problems "rationally and systematically," relying on feedback from developers:

> There are two approaches to software engineering. One approach is to upgrade current development methods by standardizing development procedures and by improving both programming techniques and the development environment. In the other approach, programs are directly generated using formal specifications of user requirements, and this approach involves the application program generator or automatic programming. The latter approach is one way of accomplishing our ultimate goal of software engineering, and if we narrow the applied field this approach is feasible. However, it is very difficult to realize these latter technologies at the present state of the art, if we apply them generally....Therefore, we addressed the

> following items based on the evolutionary approach: 1) clarification of software development methods and standardization for development activities, 2) utilization of tools to computerize development activities.
>
> We developed SDEM around the first item. For the second, we developed SDSS as a set of tools to support the phases after system design...Our basic attitude toward software development is that it is more desirable to first clarify software development methods and a software development system, then to approach these problems and solutions rationally and systematically, depending heavily on feedback from the actual system developers.[89]

The SDEM methodology was set down in four manuals. The **SDEM Concept Manual** outlined the basic standards and development approach. The **SDEM Standards Reference Card** listed specific procedures and work items for each development phase and project-team member. The **Detailed SDEM Standards Manual** presented more details of each work item and related documents, with examples. The **SDEM Introduction and Application Manual** explained basic management techniques and guidelines for introducing and using the SDEM system. The development process as defined in these manuals divided the life cycle into 6 stages and 12 specific phases, from survey and planning through maintenance and evaluation. These included a series of work categories and specific work steps, as shown in Figure 7.3. The documentation flow for the major phases is shown in Figure 7.4. The Software Factory Department and programming subsidiaries covered the bottom half of this process, and system engineering departments the upper portion.

FIGURE : OUTLINE OF SDEM

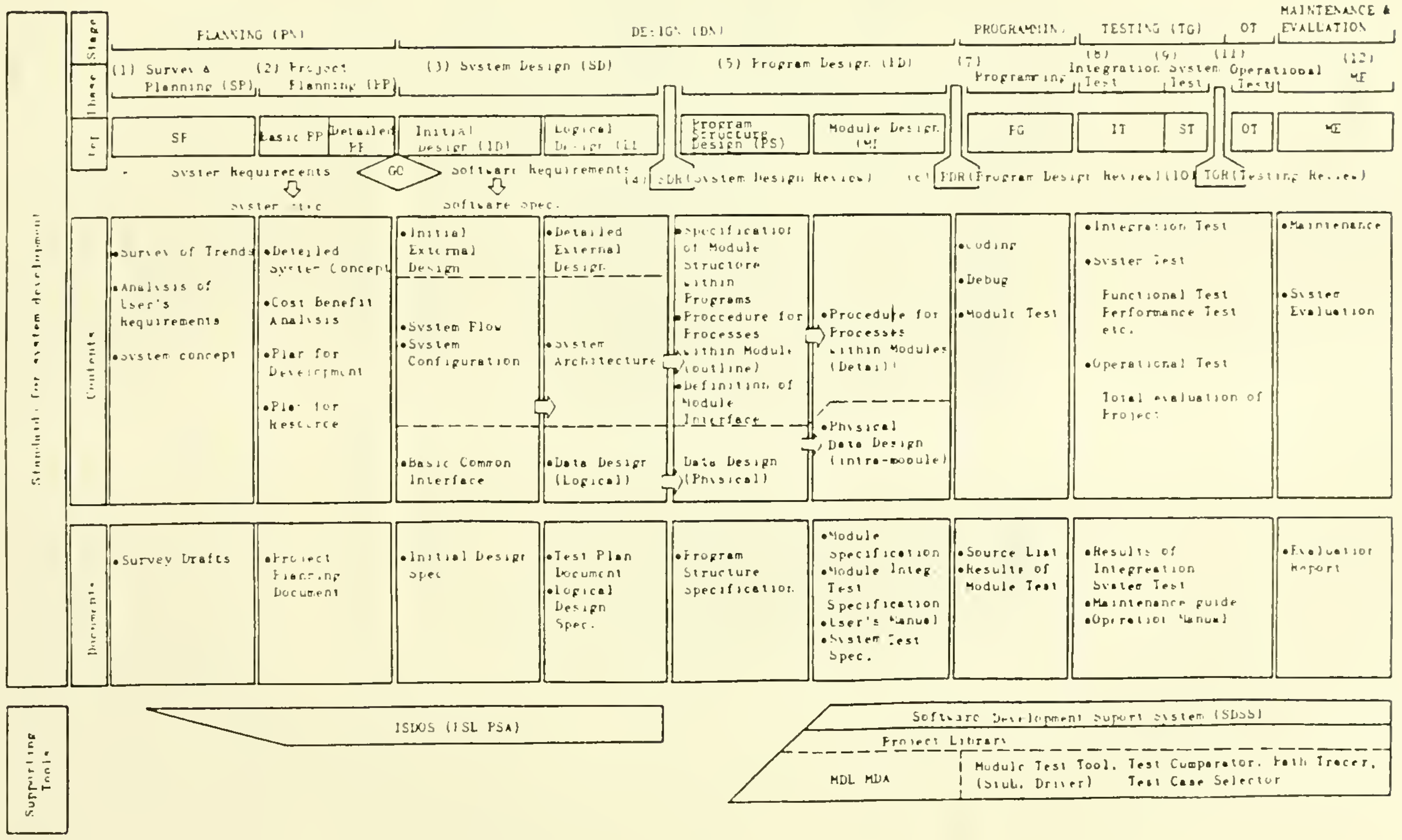
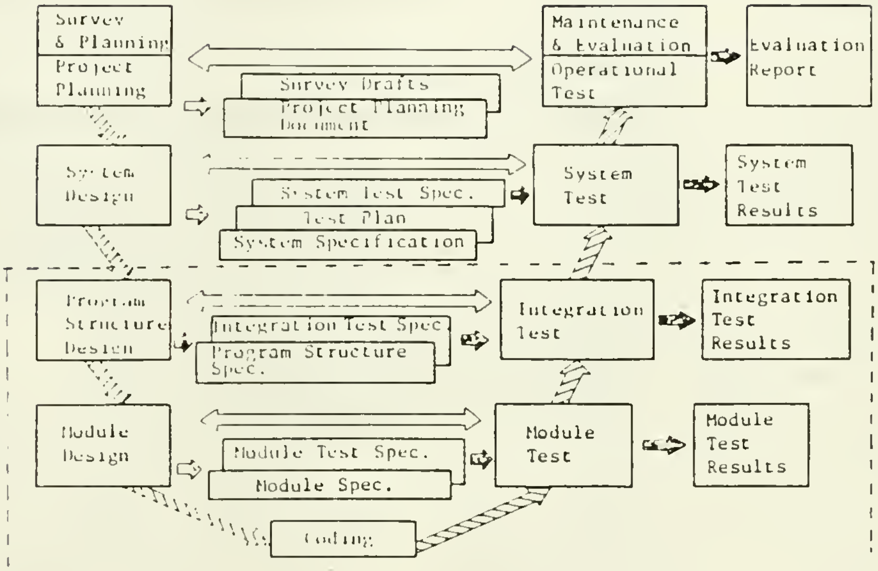


FIGURE : SDEM DOCUMENTATION AND PROCESS FLOW



Software Factory

A major objective of SDEM was to clarify and separate system design from program design, to facilitate division of labor as well as improve the "flexibility" (portability or reusability) of the designs. System design consisted of exterior specifications and logical designs that did not rely on specific machines (computers) or languages, whereas program design consisted of the physical program, which was machine and language dependent.[90] Developing the system designs separately created a risk that the entire implementation and testing effort might have to be redone, if a finished program did not meet customer requirements. On the other hand, use of SDEM and factory-support tools in all phases, from requirements through coding and test, minimized misunderstandings and errors. There was also extensive interaction between system engineers and programmers in determining the accuracy and meaning of design documents.

Two tools that grew out of the SDEM effort to standardize and simplify the specification and design process were EPG (Executive Planning Guide) and C-NAP (Customer-Needs Analysis).[91] EPGII formalized a set of planning procedures and work sheets for designing corporate information-management systems, mapping out a customer's needs against information-management systems already existing in the company. To define more formally the specifications necessary to write software, C-NAPII set forth another series of procedures and work sheets.[92]

SDEM procedures for quality control were similar to those used at Numazu but called for negotiations with customers in determining targets -- "maximum allowable number of errors per 1000 statements." If early tests indicated bug levels were above target, procedures called for validating the target levels and then instituting countermeasures such as more intensive debugging or program redesign.[93]

The Factory Tools: From SDSS to SDAS

SDSS, which automated aspects of documentation compilation and file editing and management, consisted of three elements: a project library for source programs, designs, test data, documentation; a module description language and analyzer containing information such as the name and module function, interfaces with other modules, and syntax; and test-support tools for module test, results analysis, module path tracing, and test-case selection. Although SDSS was gradually supplanted, Fujitsu used this system to set objectives for tool development.

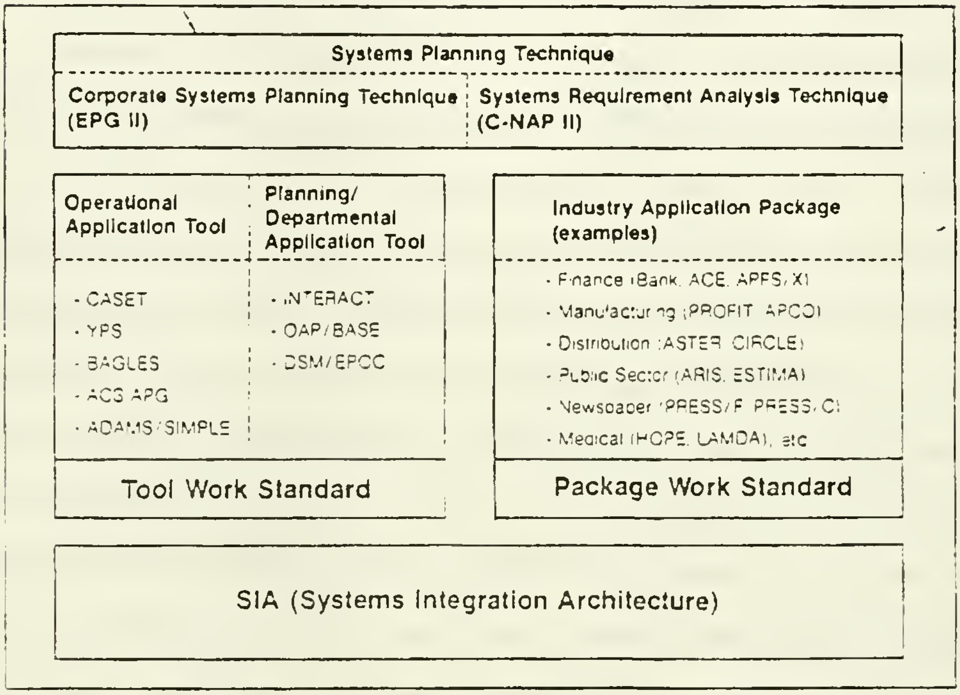
One of the first tasks, taken on by Murakami and other engineers, was to extend the system to handle languages other than FORTRAN; this became essential as more and more applications programs came to be written in COBOL. Another was to improve the module description language so that it could describe more complicated data structures and be used from the beginning of the program design phase, rather than from the module definition stage. Other objectives were to develop some automatic or mechanized program-generation capabilities, as well as reuse capabilities, and generate maintenance documents from source code, using the module description analyzer.[94]

Numazu provided solutions to some of these problems. Particularly important were the adoption of YAC flow-charts and then YPS, for editing and code generation. For program development and management, Kamata also imported GEM from Numazu, to serve as a program library system and production-control data base, and ADAMS, to serve as a project-development data base, with an interface to other tools, the project library, and management data bases.[95]

Because its basic mission was to produce customized programs, the most significant tool-development efforts at Kamata, with assistance from Fujitsu's central research labs, focused on automating aspects of program design and coding, and supporting the reuse of designs and code in the form of library routines and registered packages, which served as program parts for semi-customized systems. Fujitsu called the new set of tools, which it also began selling to its computer users in 1987, SDAS (Systems Development Architecture and Support Facilities).

SDAS continued to rely on the SDEM methodology but incorporated all the tools developed after 1980. In new-program development under SDAS, for example, system engineers used EPGII and C-NAPII to define the customer requirements, and then determined where they could reuse applications packages from the program libraries, either as they were or with changes. Process planning then centered on how much new code had to be developed, how much existing software had to be modified, and how much could be reused as is.[96] SIA (Systems Integration Architecture) created a standard interface for programming in COBOL, FORTRAN, C, LISP, and PROLOG, and for using the different SDAS tools on Fujitsu mainframes, office computers, and work stations (Figure 7.5). The SDAS methodology and tool set, along with the SIA architecture, also facilitated division of labor by making it easier to break up large programs into distinct units that could be done on different work stations and then combined later.

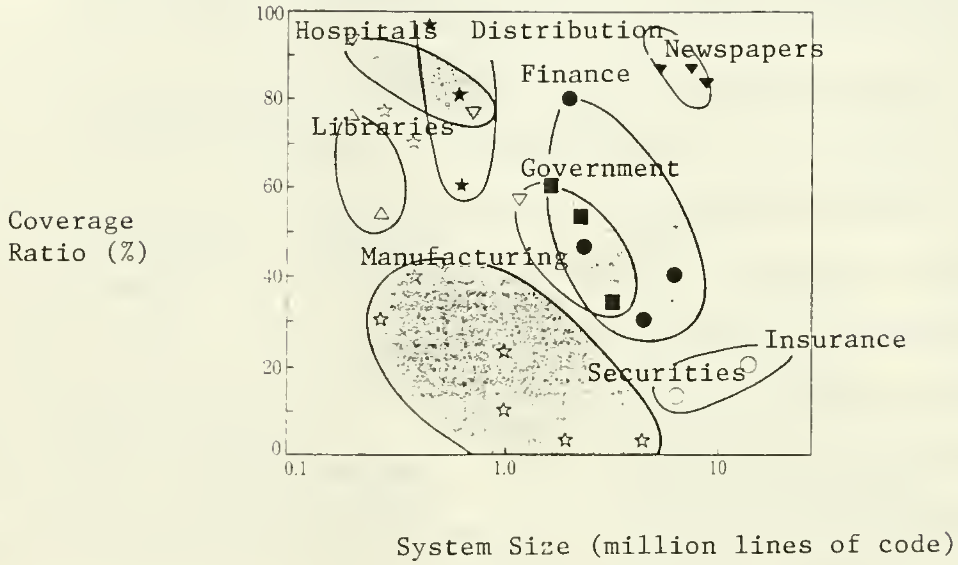
SDAS: Systems Development Architecture and Support Facilities


Systems Planning Technique
Corporate Systems Planning Technique (EPG II)
Systems Requirement Analysis Technique (C-NAP II)
Operational Application Tool
Planning/ Departmental Application Tool
· CASET
· YPS
· BAGLES
· ACS APG
· ADAMS/SIMPLE
· INTERACT
· OAP/BASE
· DSM/EPOC
Industry Application Package (examples)
· Finance (Bank, ACE, APFS, X)
· Manufacturing (PROFIT, APCO)
· Distribution (ASTER, CIRCLE)
· Public Sector (ARIS, ESTIMA)
· Newspaper (PRESS/F, PRESS/C)
· Medical (HOPE, LAMDA), etc
Tool Work Standard
Package Work Standard
SIA (Systems Integration Architecture)

The emphasis on packages reflected Fujitsu's attempt to improve productivity, quality, and cost control simultaneously by reusing design and coding know-how incorporated in previously written programs, and thereby reducing the amount of new designs and code needed to satisfy new customers. A major issue, however, was how to produce packages that could be easily tailored to fit individual user needs, and easily modified as user needs changed over time.[97] Fujitsu's solution was to keep package design, and some coding, in the specialized system engineering departments, to take advantage of specific application-related or functional expertise. Engineers in the departments reviewed systems registered in program libraries and, when it seemed appropriate, modified some of these for general sale as packages or for use in semi-customized systems built by Fujitsu or its subsidiaries.[98] Two package registration systems helped in this effort. The largest catalogued original Fujitsu packages, totalling about 1000 in 1986 and increasing 40% to 50% per year. Another library registered software developed by users or computer dealers, and in 1986 totalled about 300.[99]

Figure 7.6 summarizes Fujitsu data on how well these packages covered user needs. Manufacturing systems extended over the broadest range, with larger ones requiring total customization and some systems containing more than 40% existing code from packages. Small systems could often be covered almost entirely by packages, such as in the case of hospitals and newspaper companies.[100] Fujitsu estimated that SDAS-based applications packages, covering financial, manufacturing, distribution, government, newspaper, and medical applications, would account for 40% of applications systems delivered in 1988. Packages could not accommodate all demand, since many users continued to require unique features, although Fujitsu estimated that the systematic combination of packages and new code doubled overall productivity.[101]

FIGURE: PACKAGE COVERAGE RATIO



Automated Customization

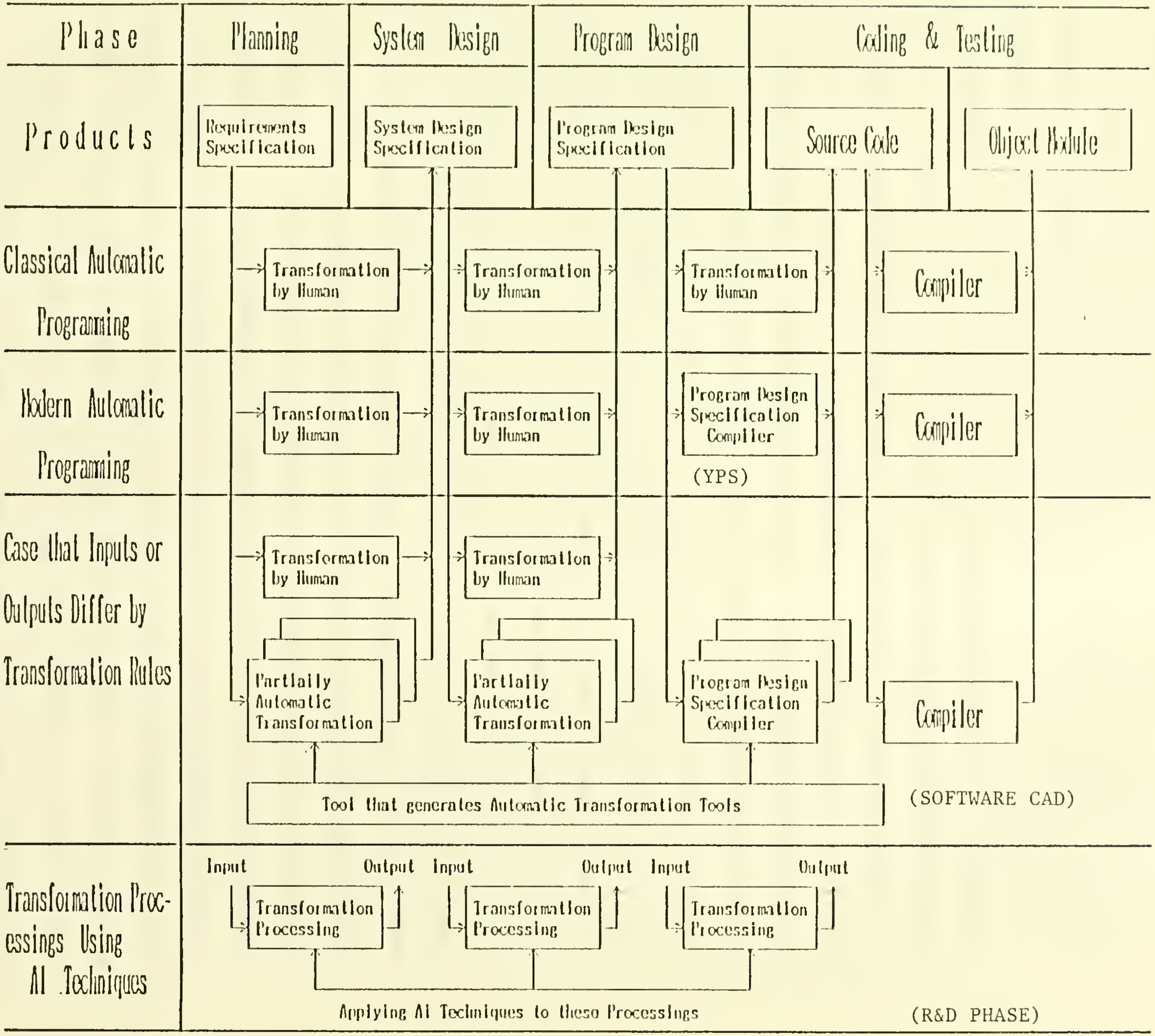
Developers of the automated programming tools Fujitsu sold as part of the SDAS set maintained that they were serving two groups: expert programmers, and non-experts. On the one hand, the tools were supposed to "free software engineers from the tedious and cumbersome job of writing run-of-the-mill computer programs and turn their talent to more creative work." This would occur because programmers no longer had to compose in computer languages but could focus on business problems and write programs "more finely attuned to the requirements of the end-user." On the other hand, SDAS tools were designed to "enable computer users to develop their own application software without the help of expert programmers."[102] Hitachi and NEC provided similar capabilities to their customers with EAGLE and SEA-I, although Fujitsu offered an even broader range of tools.

The motivation for selling this technology was simple: the high demand for customized programs continuing to outstrip the ability of Japanese firms to hire and train enough programmers. In fact, a Fujitsu survey of its large-systems customers in 1986 indicated a back-log for systems development of approximately 2.7 years per company -- which would have been far longer without reuse of designs and code, and program-automation tools.[103]

In its internal literature, Fujitsu presented an overview of automatic programming that placed the company's efforts within a broader spectrum of research and practice (Figure 7.7). In this interpretation, "classical automatic programming" consisted of compilers that turned computer code into machine-readable object languages. The "modern" technology -- represented by YPS in Fujitsu, according to Murakami, as well as similar systems in Hitachi, NEC, Toshiba, and NT&T -- focused on automating the transformation step from program design specification to machine-readable code. These "program design specification compilers" thus read structured designs or diagrams (PAD at Hitachi, SPD at NEC, and HCP at NT&T), and generated code more or less automatically (module interfaces might have to be written manually), usually in COBOL, FORTRAN, PL/I, or C. Two additional steps were to move automation further back in the development life cycle, to automate part or all of the system planning and design phases. Tools such as Fujitsu's Software CAD attempted this through transformation rules defined by the user, and fell into the category of a "partially automatic transformation tool." At the R&D level in Fujitsu (and in many other firms) were tools that tried to automate the transformation process through AI techniques.[104]

PARADIGM, issued around 1980 for in-house use and sold since 1981, was used for business applications software and supported module design, coding, and testing, as well as software reuse and program maintenance.[105] This relied on the same concept as NEC's STEPS library routines and Hitachi's EAGLE patterns. Users created program parts or patterns, consisting either of fully coded modules or design specifications (program outlines, module functions, detailed flow charts), in COBOL, using a standardized structured design methodology, and then registered them in PARADIGM's library data base, which grew with each newly designed piece of software. The parts and patterns were then accessible through a search program that used conversational language.[106] The library also contained utility programs for use on multiple operating systems, referred to as "black box" components.[107]

Overview of Automatic Programming


Phase
Planning
System Design
Program Design
Coding & Testing
Products
Requirements Specification
System Design Specification
Program Design Specification
Source Code
Object Module
Classical Automatic Programing
Transformation by Human
Transformation by Human
Transformation by Human
Compiler
Modern Automatic Programing
Transformation by Human
Transformation by Human
Program Design Specification Compiler
(YPS)
Compiler
Case that Inputs or Outputs Differ by Transformation Rules
Transformation by Human
Transformation by Human
Partially Automatic Transformation
Partially Automatic Transformation
Program Design Specification Compiler
Compiler
Tool that generates Automatic Transformation Tools
(SOFTWARE CAD)
Transformation Processings Using AI Techniques
Input
Output
Input
Output
Input
Output
Transformation Processing
Transformation Processing
Transformation Processing
Applying AI Techniques to these Processings
(R&D PHASE)

Fujitsu claimed that PARADIGM simultaneously improved productivity and quality in several ways. First, it reduced man-hours required in new program development through reuse of designs and code. It also boosted productivity due to minimal testing required of the reused components. Furthermore, PARADIGM and similar tools "enabled even inexperienced programmers to write high-quality programs," and "eliminated misunderstandings due to individual differences," by making design documents and code easier to read and maintain due to high levels of standardization.[108]

ACS/APG (Application Control Support System/Application Program Generator) was a refinement of PARADIGM that allowed users to write new programs from menus that the tool translated into COBOL. ACS was actually a package that eliminated the need to rewrite data-base and data-communications (DB/DC) portions of a software system, although it only handled low to moderate data-flow traffic. To deal with high-traffic programs, such as on-line banking software, Fujitsu developed another version of ACS, called AIM (Advanced Integrated Manager).[109]

Programming sections in the Software Factory used a similar tool, BAGLES (Banking Application Generator Library for Extensive Support), to produce the more complex software needed to control automated teller machines or to move funds among different bank locations through on-line transfers. Fujitsu began developing this tool in 1980 and introduced it for general use in Fujitsu in 1985, after several years of experimentation. The first version automatically generated COBOL but the design process was so similar to actual programming that it had to be done by skilled software personnel. In 1986, however, Fujitsu simplified the tool and added computer-aided design capabilities.[110] These allowed even non-software specialists to design new programs through menus that contained nearly all the operations banks performed. Similar to Hitachi's

EAGLE, NEC's SEA-I, and Toshiba's application generators, BAGLES translated the menus into machine-readable design documents, and then into COBOL code. The tool required factory-type hardware support, however, by occupying two million lines of code and using 10 to 15 giga-bytes of mass storage to generate software that covered 500 to 1400 terminals and 50,000 to 150,000 transactions per hour.

Still another related tool, CASET (Computer-Aided Software Engineering Tool), supported development of relatively simple business applications programs written in COBOL, consisting largely of relational data-base subsystems and information lists, such as for inventory or sales control. After the user defined specifications by menu in Japanese, the tool produced a program design and automatically coded the designs, while a debugging support subsystem and document generator facilitated testing and maintenance.[111]

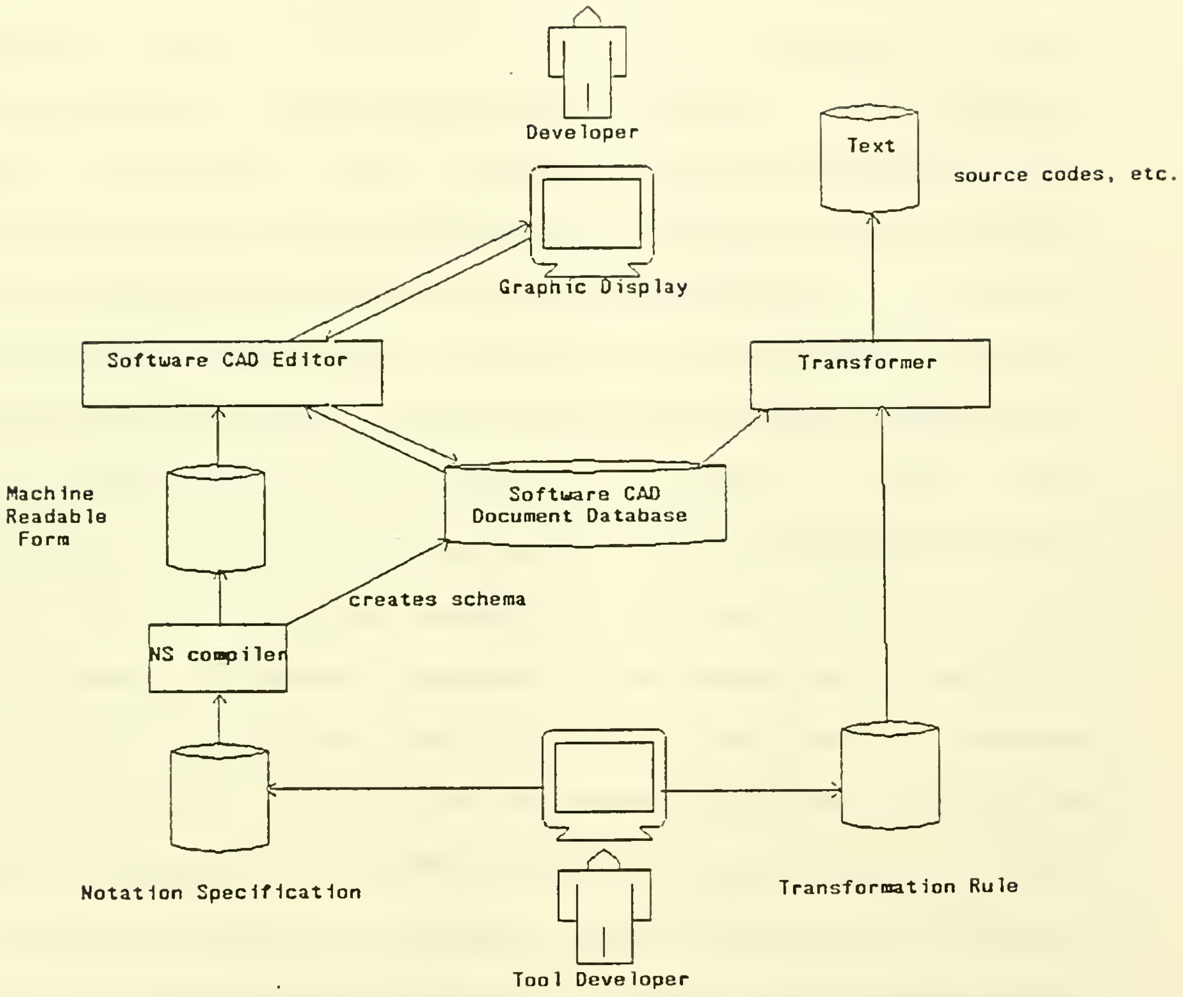
Tools like PARADIGM, ACS/APG, CASET, and BAGLES were useful for well-defined applications that could be served by menu-driven design sheets and libraries of patterns or subroutines. To automate development of software for new applications required tools that were not limited to pre-existing designs, library routines, menus, or development methods. Software CAD was the most advanced tool of this type used in Fujitsu's Software Factory.[112]

Fujitsu engineers described Software CAD as "a general tool for the drawing, verification, and transformation activities of software development...independent of any particular methods or development phases... Software CAD is a general tool that developers can customize to fit particular methods for their own circumstances." Fujitsu completed development during 1986-1987, and introduced the tool into Kamata in 1987, initially for writing segments of business applications programs. Fujitsu was gradually extending its use to other areas, such as development of operating systems, switching systems, communication software, and "firm-ware" (micro-code embedded in semiconductor chips).

The 1987 version of Software CAD used generalize-purpose notations combining text, tables, and diagrams, as well as an editor and compilers that transformed standardized notations into modifiable design documents and then different types of code. The notation system used six types of objects for describing program structures: forms (graphic models of the program), tables, nodes (graphic symbols specifying particular attributes of the program), edges (symbols representing connections between nodes), inclusions (symbols representing relationships between nodes and edges), and text fields (inputs of data into any of the objects). The NS (Notation Specification) Compiler took specifications written by program developers in the specified format, and produced machine-readable notations. These became inputs to the Software CAD editor for modification, and finished portions were deposited in the Software CAD Document Database.

The Software CAD Editor allowed the user to access the Document Database and modify any of the objects, relying on menus or templates, as well as a "mouse" interface and a "multi-window environment." The Software CAD Transformer used transformation rules specified by the tool user to transform the documents stored in the database into either executable code or textual documentation (Figure 7.8). The notation system, according to Fujitsu, was relatively easy to learn, requiring a few hours for an experienced programmer and a few days for a new employee. Describing the transformation rules was not simple, although Fujitsu claimed that programs developed with Software CAD still came .out faster than comparable systems done manually.[113]

FIGURE : OUTLINE OF SOFTWARE CAD



Ongoing R&D in Fujitsu attempted to integrate artificial intelligence techniques with Software CAD and apply AI in other support tools. Fujitsu's approach was to use AI techniques to process design documents using a "knowledge data base" for specific applications, a model of the desired design, and a "knowledge transformation" algorithm. For example, based on inferences made in processing the documentation, a tool would produce documents for a subsequent phase -- planning inputs would produce a detailed system design, which would be processed into a program design, which would be transformed (compiled) into source code. Fujitsu engineers were especially interested in applications to requirement specification and lexical (design-language) editors, since AI appeared useful to help designers understand constraints in hardware, the operating system, the computer language, or specific applications, and to verify designs. Inference-search methods were being applied to identifying software components for reuse. Other expected AI applications were in testing support and maintenance, such as problem diagnosis.

Factory applications of AI consisted mainly of automatic translation systems. In 1982, both Kamata and Numazu introduced ATLAS to translate manuals and other technical documents from Japanese to English and vice-versa (English to Japanese became available in 1986).[114] In addition, an experimental AI system developed in Fujitsu Laboratories supported switching systems software development. This was a "knowledge-based system" in the sense that it withdrew information from flow charts and deposited this in a data base, in the form of general and detailed state-transition diagram elements. The tool then used inference rules to produce I/O transformations automatically and generate computer code.[115] Other tools in trial use in Fujitsu included an expert system for system-engineering consultation.[116]

Performance Improvement

Although productivity and quality were difficult to measure, Fujitsu data, and summaries on unpublished in-house studies, indicated that these efforts resulted in significant improvements. According to a 1981 article, adoption of the SDEM standards alone brought roughly a 20% gain in productivity (lines of code per month) over projects not using the new methodology. These gains came mainly from the "absence of rework," achieved through better work scheduling, early discovery of problems, standardization of work and documentation, and use of formal planning and reviews to reduce differences between user requirements and finished programs.[117]

With SDEM in place and tool and methodology development continuing, productivity, according to Murakami, doubled between 1980 and 1984, and after 1984 has continued to rise about 15% annually, due to increased use of automated program generation and design, as well as automated testing tools. ACS alone doubled productivity, while ACS/APG and PARADIGM doubled productivity again. Directly connected with nominal measures of productivity was reusability, which these tools supported. According to Murakami, programming efforts using PARADIGM and related tools generally achieved a reuse rate of about 40%, compared to about 10% in projects that did not.[118] Specific cases reported in internal Fujitsu literature on ACS/APG cited reuse rates between 63% and 86% for program parts such as graphics control software and Japanese language processing access routines.[119]

	Manual	ACS	ACS-APG/PARADIGM
Productivity Index:	1	2	3-4
Estimated Reuse%:	0	10%	40%

On a factory basis, productivity by the mid-1980s was approximately 1500 to 2000 lines of COBOL code per month, including comments (about 10%). These numbers, however, excluded system engineering, and included extensive overtime. According to a former programmer, male employees often worked 70 or 80 hours per week, although women worked less because of legal restrictions. In addition, managers did not emphasize the writing of short, "elegant" programs, but stressed schedule deadlines and correctness. As a result of these practices, programs tended to be long and lines-of-code productivity high. On the other hand, Fujitsu hardware was so powerful that customers rarely noticed if programs ran somewhat slow.[120]

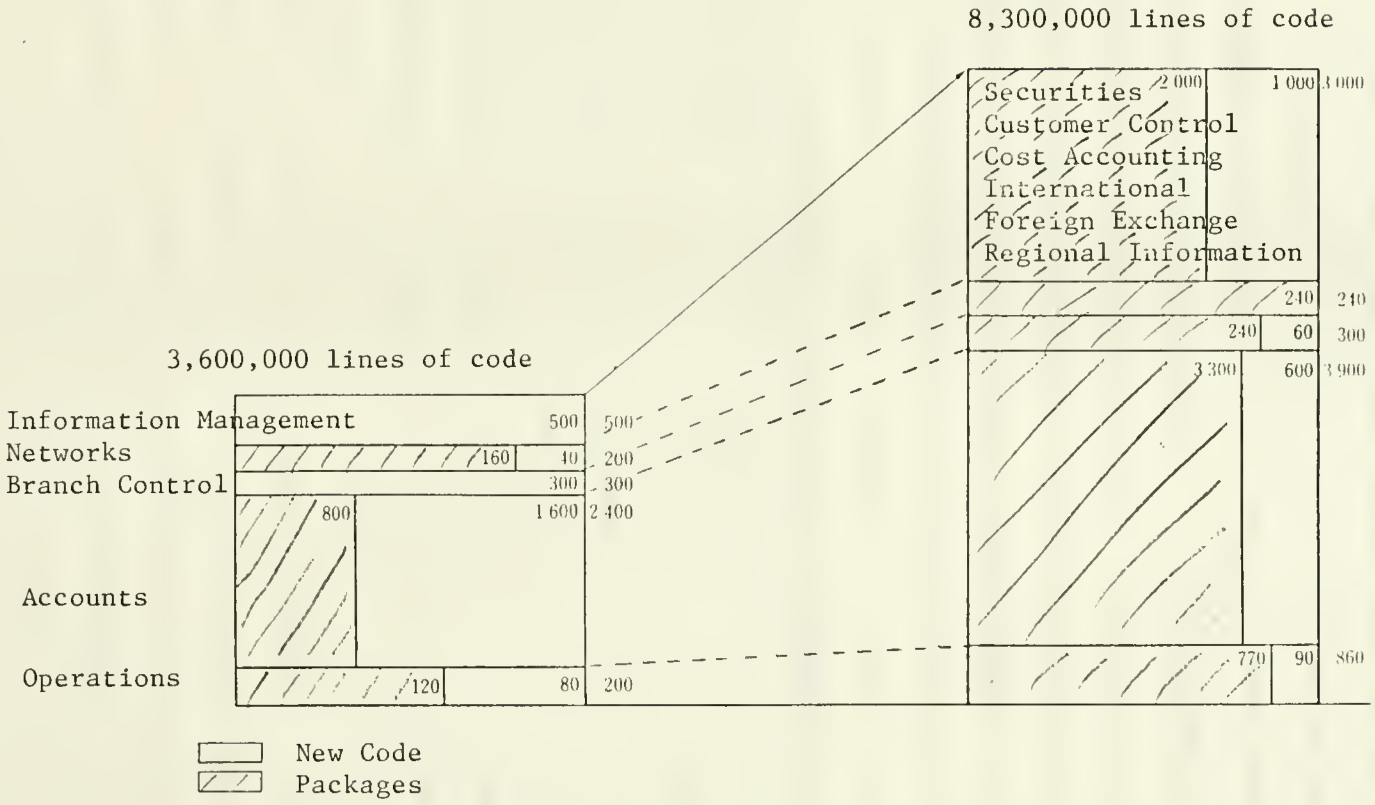
Fujitsu provided one specific example of the effect SDAS tools and packages had on productivity. For a particular banking customer, Fujitsu replaced an old software system that consisted of 3.6 million lines of code. Existing packages had covered 30% of the old system, and man-months required for the complete system was 8000. Thus, there was a gross productivity rate of 450 lines of code per man-month, including an average of 315 per month for new code, including time to integrate the existing programs. The replacement system was more than twice as large -- 8.3 million lines of code. But Fujitsu was able to reuse 79% through SDAS packages for securities processing, customer control, cost accounting, international transactions, foreign exchange dealings, regional information control, and a general data base, as well as for the outside network, store operations, accounting, and general operations. For the other 21% that had to be newly developed, programmers used the YPS and BAGLES tools,. which automatically generated code. The resulting gross productivity was 902 lines of code per man-month, although only 190 if one only looks at the amount of new code developed (Table 7.10 and Figure 7.9).

It is possible that reusing so many packages resulted in a total system longer than would have occurred had the software been written from scratch. Accordingly, a project with high reuse might produce higher lines-of-code productivity than a fully customized program. This appeared to account for some of the high productivity figures reported in Japanese firms, although some tools appeared to reduce length. Fujitsu claimed, for example, that software produced through BAGLES was actually much shorter than similar programs written without the tool -- on average, merely 1/14th the size. This was because, in developing BAGLES, engineers identified a large number of repetitious functions that could be handled through subroutines, which could be cited repeatedly in the program without being rewritten.[121]

Table 7.10: BANKING SYSTEM DEVELOPMENT PRODUCTIVITY[122]

	Old System	New System
System Length	3,600,000 LOC	8,300,000 LOC
Reused Code	1,080,000 LOC (30%)	6,550,000 LOC (79%)
New Code	2,520,000 LOC	1,750,000 LOC
Total Productivity	450 LOC/Man-Month	902 LOC/Man-Month
New-Code Productivity	315 LOC/Man-Month	190 LOC/Man-Month

FIGURE: BANKING SYSTEM PRODUCTIVTY COMPARISON



Fujitsu also discovered, as did Toshiba and Hitachi, that there were limits to the benefits of reusing designs and code, depending on how much of the component or subsystem had to be modified. In systems software, Fujitsu data indicated there was a clear improvement in productivity as long as 70% to 80% of a particular module or program part was reused without significant changes. In a sample of 47 projects, median productivity, including man-power devoted to maintenance, was approximately 60% higher with 20% new code and an 80% reuse rate. In projects where there were attempts to reuse code but reuse without significant changes proved to be less than 70%, the impact on productivity tended to be negative.[123] Studies in applications system programming confirmed this tendency, showing a "break-even" point of about 60% for a given piece of software; if more than 60% had to be newly written, then the impact on overall productivity of attempting to reuse code was slightly negative.[124]

Software CAD appeared to improve productivity without necessarily recycling existing code or designs, but through partially automating the transformation of design specifications into code. The present tool was most easily used for specific tasks, but results reported by Fujitsu in limited trials were impressive. For example, Software CAD cut the time needed to convert a job-control diagram to job-control language from 6 months to 1 month, and total manpower from 9.3 man-months to 0.5 man-months. Programmers using the tool also transformed a database logic structure diagram into programming language (ADL) in one-sixth the time (8.4 man-months to 0.5 man-months). Other efforts showed similar improvements, usually cutting development time one-fourth or one-third, with even larger gross productivity gains, compared to conventional design and programming (Table 7.11).[125]

Table 7.11: USE OF SOFTWARE CAD FOR TOOL DEVELOPMENT[126]

Key: K-LOC = 1000 lines of cod
MM = Man-Months
MH = Man-Hours
mo.= month

Task	Conventional	Software CAD	Development Time Improvement
Job-Flow Diagram to Job-Control Language (7.5 K-LOC in C)	9.3 MM/6 mo.	0.5 MM/1 mo.	6-fold
Database Structure Diagram to ADL (6.7 K-LOC in C)	8.4 MM/6 mo.	0.5 MM/1 mo.	6-fold
Screen Format to Definition Language (100 screens)	2.3 MM	0.6 MM	4-fold
Screen Transformation to Programming Language (10/screen)	8.0 MM	2.0 MM	4-fold
Database Structure Diagram to ADL	6.0 MH	2.0 MH	3-fold

SUMMARY

While Fujitsu did not emphasize software process analysis, standardization, or factory-type organization as early as Hitachi, NEC, or Toshiba, by the early 1980s it had clearly adopted factory-type practices in both systems and applications software development. Huge demand for customized programs also led Fujitsu to create Japan's largest network of software subsidiaries, as well as to develop an impressive collection of reusable code and automated tools that facilitated customization. Table 7.12 summarizes of Fujitsu's activities in areas common to other software-factory efforts.

Table 7.12: FUJITSU SUMMARY

FACTORY CONCEPT	IMPLEMENTATION
Strategic Integration	Direction in systems software from the inspection department and software engineering departments, and in applications software from the Development Planning Office
Product-Process Focus	Tool and methodology development and standardization centered on two categories, systems and applications
Scale and Scope	Systems software centralized in Numazu Works, applications software in Kamata Software Factory and subsidiaries
Improvement, Not Innovation	Focus on producing IBM-type operating systems efficiently, and simplifying the development of common applications programs through tool worker specialization, tool support, and reuse of packages
Process Analysis/Control	Systematic data collection, standardization efforts, development planning report system, and BSMS data base from mid-1970s
Quality Analysis/Control	Systematic data collection from the late 1970s; quality circle activities; Design of Experiment techniques
Central Tool Support	System software tool development centered in Numazu Works and applications tools in Kamata, with assistance from central labs
Training	Laboratory for training established in 1970; SDEM training program from 1978
Reusability	Extensive investment in package libraries and tools accessing reusable designs and code, such as PARADIGM, ACS/APG, CASET, and BAGLES
Automated Customization	Reuse-support tools, as well as new systems such as Software CAD

REFERENCES

1. Fujitsu Limited, Annual Report, March 1987.

2. Nikkei Computer, 13 October 1986, p. 75.

3. Fujitsu, "Numazu kojo" (Numazu Works), undated brochure.

4. Nikkei Computer, 13 October 1986, pp. 70, 79.

5. Shimoda Hirotsugu, Sofutouea kojo (Software factories), Tokyo, Toyo Keizai Shimposha, 1986, p. 82; interview with Yamaji Katsuro, Deputy General Manager, Software Division, Fujitsu Ltd., 7/31/86.

6. Shimoda, pp. 81-82.

7. Fujitsu, "Kaihatsu taisei: Densanki Jigyo Honbu, Sofutouea Jigyobu" (Organizational structure, Computer Group, Software Division), unpublished company document, 1986.

8. Interviews with Yoshida Tadashi, Deputy General Manager, Quality Assurance Department, Software Division (Numazu Works), Fujitsu, 7/31/86 and 9/7/87.

9. Interviews with Murakami Noritoshi, Manager, Software Development Planning Office, Fujitsu Limited, 9/1/87 and 3/22/88.

10. Nikkei Computer, 13 October 1986, pp. 70, 79, and Shimoda, p. 82.

11. Murakami interviews.

12. Nikkei Computer, 13 October 1986, p. 80.

13. Fujitsu Ltd., Nyusha no shiori (Guidebook for company entrance), Education and Training Department, February 1986, pp. 30-32; and Kiriu Hiroshi, Sofutouea sangyo no jitsuzo (The state of the software industry), Tokyo, Nikkan Shobo, 1986, pp. 78-81.

14. Major sources for the corporate history are Usui Kenji, "FACOM kaihatsu shi" (History of FACOM development), Computopia, April and May 1975; Iwabuchi Akio; Fujitsu Kabushiki Kaisha, Fujitsu Kabushiki Kaisha shashi II (History of Fujitsu, Ltd.), 1976, reprinted in Nihon Shashi Zenshu Kankokai, Nihon shashi zenshu: Fujitsu shashi, Tokyo, 1977; "FACOM no ayumi" (FACOM history), FACOM janaru, Vol. 11, No. 1 (1985), pp. 20-47.

15. Minamisawa Noburo, Nihon konpyuta hattatsu-shi (History of the development of Japanese computers), Tokyo, Nihon Keizai Shimbunsha, 1978, p. 145.

16. Iwabuchi Akio, Fujitsu no chosen (Fujitsu's challenge), Tokyo, Yamate Shobo, 1984, pp. 34-42, 128-129, 198-202; Ogino Yuji et al., "Fujitsu-Hitachi no shin-konpyuta 'M shirizu' no senryaku o tettei kyumei" (A close look at the strategy of the new Fujitsu-Hitachi 'M-series' computers), Computopia, February 1975, p. 17-18.

17. Iwabuchi, p. 93; Nikkei Computer, 13 October 1986, p. 81.

18. Usui, Computopia, May 1975, pp. 17-18; Japan Economic Journal, 29 January 1985, p. 10.

19. "FACOM no ayumi" (FACOM development), FACOM janaru, Vol. 11, No. 1 (1985), pp. 20-47; and Fujitsu Kabushiki Kaisha, "FACOM seino ichiran" (Overview of FACOM performance) in Ikeda kinen robun shu (Anthology of articles in memory of Ikeda), Tokyo, 1978, pp. 254-267.

20. Ogino, pp. 30-31.

21. Datamation, 1 June 1985, p. 60; The Wall Street Journal, 12 November 1985, p. 18; Japan Economic Journal, 7 June 1986, p. 15.

22. Interview with Fujitsu Managing Director Miyoshi Mamoru in "Sofutouea bijinesu no mirai" (The future of the software business), Computopia, April 1986, pp. 85-87.

23. Fujitsu Limited, Annual Report 1983 and Annual Report 1984.

24. Fujitsu, Information Processing Group, No. 1 Software Division, "Sofutouea kaihatsu: hinshitsu-seisansei kojo ni tsuite" (Software development: quality and productivity improvement), unpublished company document, received 9/24/85, p. 3.

25. "Sofutouea kaihatsu," pp. 40-41.

26. Nihon Keizai Shimbun, 5 August 1987, p. 9.

27. David Lammers, "Sigma: Japan's Effort to Close the Software Productivity Gap," Electronic Engineering Times, 21 July 1986, p. 9.

28. Murakami interviews.

29. Kawaguchi Izawa, "Shoki no shisutemu sofutouea: FONTAC 8040 MONITOR no kaihatsu made o chushin to shite" (Early systems software: focus on period through the FONTAC 8040 MONITOR development), Joho shori, Vol. 24, No. 3, March 1983, pp. 225-237; Usui (May 1975), pp. 20-21.

30. Okamoto Akira, "MONITOR-V shisutemu no omoide" (Recollections of the MONITOR-V system), in Kyoto Daigaku Ogata Kensanki Senta 10-nen-shi, pp. 224-229; Fujitsu shashi II, p. 104; Usui (May 1975), p. 25.

31. Iwabuchi, pp. 44, 221-222.

32. Okamoto, pp. 224-229.

33. Yamamoto Takuma, "Opereteingu shisutemu kaihatsu no omoide" (Recollections on operating system development), in Kyoto Daigaku Ogata Keisanki Senta, ed., Kyoto Daigaku Ogata Kensanki Senta 10-nen-shi (A 10-year history of the Kyoto University Ogata Computer Center), Kyoto, Kyoto University, 1980, pp. 217-219.

34. Uemura Mitsuo, "Sofutouea kaihatsu no omoide" (Recollections of software development), in Kyoto Daigaku Ogata Kensanki Senta 10-nen-shi, pp. 230-236.

35. Ogino, pp. 30-31.

36. Tabata Akira, "Kihon sofutouea no kaihatsu" (Basic software development), FACOM janaru, Vol. 11, No. 1 (1985), p. 54.

37. Shimoda, p.73.

38. Shimoda, pp. 73-76.

39. This discussion is based on Yoshida interview, 9/24/85, and "Kensa-bu no rekishi" (History of the inspection department), internal Fujitsu memo, prepared by Yoshida.

40. Fujitsu Ltd., "Kensa-bu no rekishi" (History of the inspection department), internal Fujitsu memo.

41. "Sofutouea kaihatsu"; and Yoshida interviews.

42. Yoshida interviews.

43. Morita Shoken and Kanda Shigeru (Fujitsu, Ltd., No. 1 Software Division, Inspection Department), "Sofutouea hinshitsu no toraekata" (Ways of looking at software quality), **Hinshitsu**, Vol. 14, No. 2 (April 1984), p. 91.

44. Tadashi Yoshida, "Attaining Higher Quality in Software Development--Evaluation in Practice," **Fujitsu Scientific and Technical Journal**, Vol. 21, No. 3 (July 1985), p. 306.

45. "Sofutouea kaihatsu," especially pp. 7-10.

46. Yoshida interviews.

47. Fujitsu, Computer Group, Software Division, "Sofutouea no hinshitsu kanri" (Software quality control), unpublished company document, 11 September 1985, p. 6; and "Sofutouea kaihatsu," p. 6.

48. This discussion is based on Nihon Noritsu Kyokai (Japan Management Association), **Fujitsu no ko-shinraisei undo** (Fujitsu's High-Reliability Movement), Tokyo, Nihon Noritsu Kyokai, 1985, especially pp. 144-176.

49. Yoshida, "Attaining Higher Quality in Software Development -- Evaluation in Practice," p. 305.

50. Yoshida, "Attaining Higher Quality in Software Development -- Evaluation in Practice," p. 305.

51. Yoshida, "Attaining Higher Quality in Software Development -- Evaluation in Practice," pp. 308-309.

52. "Sofutouea kaihatsu," p. 15.

53. Keizo Tatsumi (Quality Assurance Department, Computer Software Development Group, Fujitsu Limited), "Test Case Design Support System," International Conference on Quality Control, Tokyo, October 1987, pp. 1-2.

54. Also, citing Taguchi's 1976 book in Japanese, is a more detailed article on these methods used in Fujitsu. See Sato Shinobu and Shimokawa Haruki, "Jikken keikau-ho o mochiita sofutouea no tesuto komoku settei-ho" (Software test-case selection method based on experimental design methodology), Sofutouea Seisan ni okeru Hinshitsu Kanri Shinpojium (Symposium on quality control in software), No. 4, ca. 1983.

55. Yoshida, "Attaining Higher Quality in Software Development -- Evaluation in Practice," pp. 314-315.

56. Murakami interviews.

57. "Sofutouea kaihatsu," pp. 37, 44.

58. Yoshida interviews.

59. Fujitsu Kabushiki Kaisha, "FACOM M-shirizu OSIV/X8 FSP," pp. 171-173.

60. "SDAS: Application Software Development Made Easy," Electronics News from Fujitsu, July 1987, p. 2; Yoshida interviews; and "Sofutouea no hinshitsu kanri," p. 16; and Yoshida's 2/8/87 written response to 1/20/87 Cusumano questionnaire on "Large-Scale Systems Software." Yoshida believed that a major reason U.S. firms have not refined flow chart systems to this extent was because, for native speakers of English or similar languages, writing in a high-level computer language is close to their native language. But, for Japanese this was not the case, and YACII, PAD, or SPD diagrams could be written in Japanese, making them easier for Japanese programmers to use.

61. Yoshida response to questionnaire.

62. Yoshida Tadashi, "Sofutouea no keiryo-ka" (Software quantification), Joho shori (Information Processing), Vol. 26, No. 1 (January 1985), p. 49; and Yoshida interviews.

63. "Sofutouea kaihatsu," p. 29.

64. Yoshida interviews.

65. "Sofutouea kaihatsu," p. 30.

66. Yoshida interviews.

67. "Fujitsu," Computopia, June 1975, p. 92.

68. Usui, Computopia, May 1975, pp. 25-26.

69. Fujitsu Kabushiki Kaisha shashi II, pp. 102-103.

70. Nikkei Computer, 13 October 1986, p. 82.

71. Nikkei Computer, 13 October 1986, pp. 81-82.

72. Sources: for chart, Nikkei Computer, 13 October 1986, p. 79; for 1988 employee totals, author estimates based on 1986 data and 1988 Murakami interview.

73. Murakami interviews.

74. Shimoda, p. 74.

75. Murakami interviews.

76. Murakami interviews.

77. Interview with Sakurai Hiromi, former programmer (1981-1983), Distribution Systems Group, Software Factory Department, Fujitsu Limited, 5/4/88.

78. Murakami and Sakurai interviews.

79. Murakami and Sakurai interviews.

80. Sakurai interview.

81. Sakurai and Murakami interviews.

82. Noritoshi Murakami, Isao Miyanari, and Kazuo Yabuta, "SDEM and SDSS: Overall Approach to Improvement of the Software Development Process," in H. Hunke, ed., Software Engineering Environments, Amsterdam, North-Holland, 1981, p. 288.

83. Sakurai interview.

84. Murakami interviews.

85. Murakami and Sakurai interviews.

86. Nikkei Computer, 13 October 1986, pp. 80-81.

87. Yoshiro Nakamura, Ryuzo Miyahara, and Hideshi Takeuchi, "Complementary Approach to the Effective Software Development," Proceedings of Computer Software and Applications Conference (COMPSAC78), New York, Institute of Electrical and Electronics Engineers (IEEE), November 1978, p. 236.

88. Nakamura et al., p. 236.

89. Nakamura et al., pp. 235-236.

90. Murakami et al., pp. 284-286.

91. Murakami interviews.

92. Nagata Yuzuru, Mori Kuniaki, and Takahashi Tomio, "Shisutemu keikaku giho EPGII to C-NAPII" (System Planning Methodologies EPGII and NAPII), Fujitsu, Vol. 39, No. 1 (January 1988), pp. 13-20.

93. Murakami et al., pp. 286-287; Watanuki Hisashi, Hatta Makoto, and Okudaira Hajime, "Apurikeshon puroguramu kaihatsu toki no hinshitsu kanri" (Quality Control for Application Program Development, Fujitsu, Vol. 34, No. 6 (1983), pp. 857-865.

94. Murakami et al., pp. 289-293.

95. Noritoshi Murakami and Tomio Takahashi, responses to Cusumano surveys on "Large-Scale Applications Software" (1/19/87) and "Software Facilities" (9/87).

96. Akiyama Masayuki and Nishimura Shuji, "SDAS ni okeru pakkeji kaihatsu to sono tekiyo gijutsu" (SDAS Improving Package Development and Its Application Methods), Fujitsu, Vol. 39, No. 1 (January 1988), pp. 36-41.

97. Akiyama and Nishimura.

98. Murakami interviews.

99. Nikkei Computer, 13 October 1986, pp. 81-82.

100. Akiyama and Nishimura.

101. "SDAS: Application Software Development Made Easy," Electronics News from Fujitsu, July 1987, p. 3.

102. "SDAS: Application Software Development Made Easy," pp. 1-2.

103. Fukuta Zenichi, Yoshihara Tadao, and Maruyama Takeshi, "SDAS sogo kaihatsu shisutemu no teisho" (SDAS Systems Development Architecture and Support Facilities), Fujitsu, Vol. 39, No. 1 (January 1988), p. 3.

104. Murakami and Takahashi survey responses; "SDAS: Application Software Development Made Easy," p. 2; Murakami interviews.

105. Murakami interviews.

106. Fujitsu Kabushiki Kaisha, "FACOM M-shirizu OSIV/X8 FSP," pp. 193-194.

107. "Sofutouea kaihatsu," p. 42.

108. "FACOM M-shirizu OSIV/X8 FSP," pp. 193-194.

109. Murakami interviews.

110. Fujitsu Limited, "BAGLES kaihatsu no keiro" (The process of BAGLES development), unpublished Fujitsu document, 20 March 1986.

111. Ueki Sadahiro, Miyauchi Kazuto, and Nakada Haruyoshi, "Koseisansei tsuru CASET" (High-productivity tool CASET), Fujitsu, Vol. 39, No. 1 (January 1988), pp. 21-28.

112. "Fujitsu Software: Banking to Realtime," Electronic Engineering Times, 11 February 1985, pp. 63-64.

113. Kazuo Yabuta, Akihiro Yoshioka, and Noritoshi Murakami, "'Software CAD': A Generalized Environment for Graphical Software Development Techniques," COMPSAC'87, pp. 1-8.

114. Murakami interviews.

115. Fujitsu, "System for Supporting Software Development (for Switching Systems Software)," internal document, 1977; Murakami interviews.

116. Murakami interviews.

117. Murakami et al., pp. 287-288.

118. Murakami interviews.

119. "Sofutouea kaihatsu," p. 43.

120. Sakurai interview.

121. "Fujitsu Software: Banking to Realtime," pp. 63-64.

122. Fukuta Zenichi, Yoshihara Tadao, and Maruyama Takeshi, "SDAS sogo kaihatsu shisutemu no teisho" (SDAS Systems Development Architecture and Support Facilities), Fujitsu, Vol. 39, No. 1 (January 1988), p. 11.

123. Yoshida, "Sofutouea no keiryo-ka," pp. 48-51.

124. Akiyama and Nishimura, p. 38.

125. Yabuta, Yoshioka, and Murakami, pp. 1-7.

126. Yabuta, Yoshioka, and Murakami, p. 7.