

Essence of Toshiba Software Factory

MATSUMOTO, Yoshihiro, Dr.Eng.

Presented as a Guest Professor, Institute for
Informatics, University of Stuttgart

(IEEE Life Fellow)

Prologue

- **The author set up the Toshiba Software Factory in 1977 that accommodate approximately 2,300 software developers and workers.**
- **The factory was specialized in the production of real-time industrial application software systems for the domains such as: electrical power generation, power transmission and distribution, steel rolling-mill, factory automation, robotics, traffic control, building management/control, etc.**

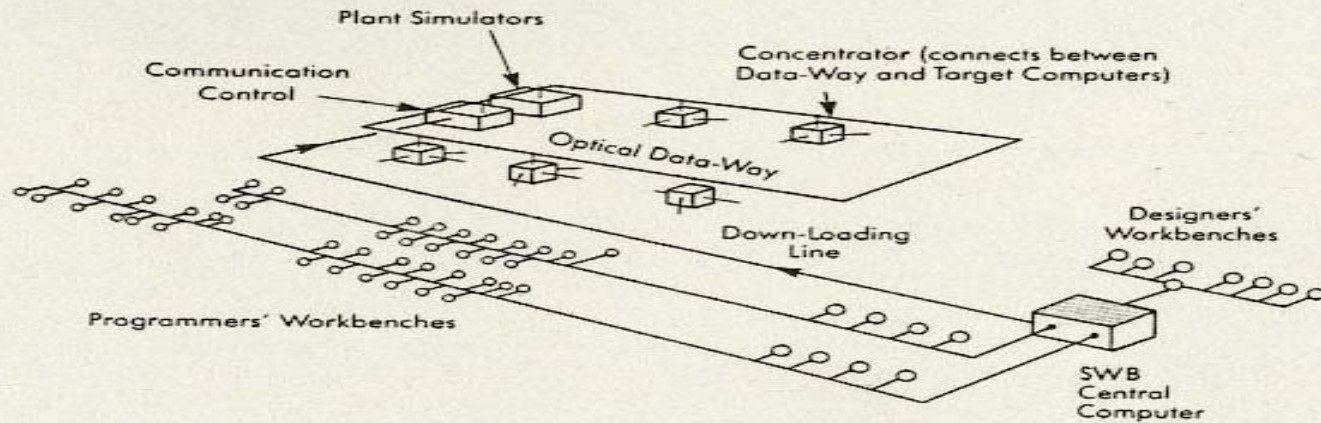
Prologue (continued)

- **From 1954 to 1988, Matsumoto was in Toshiba Corporation, where he was responsible for the development of the Toshiba Fuchu Software Factory.**
- **In January, 1989, he moved to Kyoto University, as a Professor, and then to several other universities.**
- **In this slides:**
 - **fundamental concepts underlying the software engineering management actually practiced in this factory are presented.**

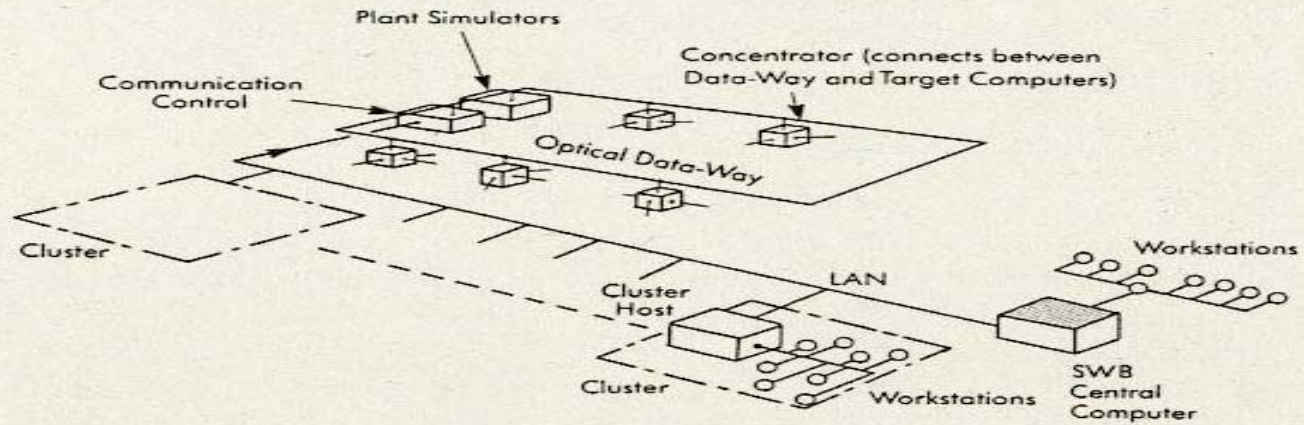
Factory layout

256

THE FACTORY APPROACH



Hardware Configuration of the Centralized SWB System



Hardware Configuration of the Dispersed SWB System

Content

- **organizational management**
 - **policy management**
 - **personnel management**
 - **communication management**
- **process/project management**
 - **initiation**
 - **planning**
 - **review and evaluation**
 - **closure**
- **software engineering measurement**
 - **goals of measurement programs**
 - **measurement selection**
 - **collection of data**
- **evaluation**

Policy management

(from the aspect of general organizational management)

- **Top organizational objectives**
 - **to cope with the software crisis**
 - **to make the “software” visible for everybody in the organization in order to improve manageability**
 - **to improve software productivity toward some target value (e.g. in the order of 10% increase per year)**
 - **to improve software quality toward some target value (e.g. decrease faults/KLOC to get the value less than a half)**

To make the “software” visible for everybody

- The principles that had been applied in hardware production management were also applied to software production management regarding planning at the tactical and operational level, organizational culture and behavior, and functional enterprise management in terms of procurement, cost management, progress management, configuration management, quality management and productivity management.
- Software development and hardware (excluding computer) development were put in the same organizational responsibility.
- Unit Workload Order Sheet (UWOS) was one of the typical examples of our efforts for the implementation of visibility.

Unit Workload Order Sheet (UWOS)

- **Each hardware component in the factory usually was accompanied by a sheet of slip.**
- **UWOS is a sheet of slip attached to a software component or a set of assembled components to be built under the responsibility of one specific person. UWOS was an analogy from the hardware slip.**
- **The UWOS format includes precondition, mission, post-condition, communication needs, the traces from the requirements specification, and constraints for timings, cost, reuse-rate, productivity, standards, and legal requirements, etc.**

UWOS

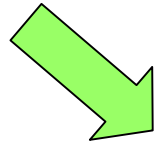
UNIT WORKLOAD ORDER SHEET

UW No.	Name of responsible person		
UW Name	Purpose		
Precondition	Background		
UW Denotation	Objectives		
Import Specification	Quantity		
Export Specification	Quality level		
Parameters	Cost		
Process Denotation	Productivity		
	Reusing		
	Constraints		
	Size		
	Performance		
	Time		
	Standards, Methodologies		
	Formalisms		
Postcondition			

Figure 4. An UWOS form.

How UWOS looks like?

The UWOS assigned to him is displayed.



It is connected with the project management system.

Unit workload

- **A unit workload (UW) is defined as:**
 - **A unit of work assignment given to one individual responsible person. The nature and size of each UW varies depending on the skill level of each responsible person.**
 - **Unit workloads are defined phase by phase at the beginning of each phase.**
 - **The objectives and constraints to be given to each workload are derived through the so-called UW planning meetings. UW's are sometimes modified or added/deleted in the meetings during the course of the project progress. The objectives and constraints of each UW are decided under the consensus of all responsible members. But, the project manager has to make the final decision.**

UWOS has pointers to:

- **Each UWOS also has pointers to:**
 - **product**
 - **given resources such as global data, interface signatures, to-be-imported resources, input/output format, and programs or specifications to be produced**
 - **process**
 - **recipe, construction steps, solution steps**
 - **examples**
 - **existing similar UWOS formerly produced**
 - **text books, data books, and other documents to be referred to**

Extended use of UW

- **Configuration control can be managed more easily (UW structure is tightly related with software architecture).**
- **Standardized (patternized) software processes can be made visible using UW networks.**
- **Every element in the knowledge management database (product/process patterns, frequently asked questions, etc.) may have a pointer to some particular UW's. Using these pointers, actual past UW's can be made referable from each knowledge element.**
- **CPM* charts underplayed by UW networks increases manageability.** *critical path method

Personnel management

■ Career development

■ Clarification of the skill levels

■ Career development (consultation) program (CDP) based on some typical referential career-path examples was applied to all participants.

■ CDP sheet for each individual includes short term objectives, long term objectives, and the record of achievement at each evaluation milestone.

■ Consultation for each individual was made twice a year by the responsible manager.

Typical career course

Skill levels	Course 1	Course 2
General management	General manager	Principal engineer
Organizational management, Project management	Organizational Manager, Project Manager	Specialist
Design	System designer/program designer (at least 5 years)	
Programming	Programmer/testing engineer/maintenance engineer (at least 7 years)	

Communication management

- **Project data for management, such as progress, cost, quality, productivity, and reusing rate were kept transparent throughout the project with using database management system, and shared by all organization members.**
- **Examples of the formal communication: weekly project meeting, design/inspection meeting at each base-line, quality circle, organizational quality conference, and award system.**
- **Project portfolio management**
 - **Results from the analysis of failures-at-site per project were used to prioritize the effort of multiple projects within the organization.**
- **Procurement management**

Process/project management (1)

■ Initiation

■ Project initiation and scope definition for both hardware and software were tightly collaborated.

■ Model exploration for finding a solution for the given requirements using existing domain-based meta-models were made through applying spiral processes.

■ Meta-models were generally described in the style of functional block diagram, a metaphor from the hardware block diagram, in order to enable hardware-based members to understand easily.

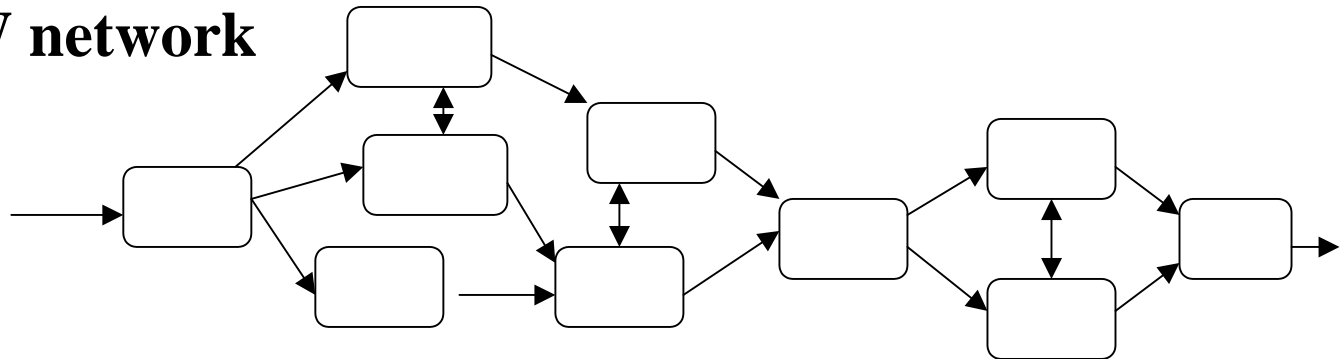
■ Each existing domain-based model has its specific traces to existing architecture style/patterns, and to the existing UW network.

■ Determination and negotiation of requirements, and feasibility analysis were made using those models, traces, and style/patterns.

Process/project management (2)

■ Planning

■ UW network



- Each UW is accompanied by a UWOS.
- If precondition is satisfied, UW can start.
- Communication paths between UW's are shown.
- UW network can be described in hierarchical style .
- The traces from requirements specification to each UW are specified so that adaptation to the changes of the requirements can be coped with efficiently.

Process/project management (3)

■ Review and evaluation

■ Design review/inspection teams were organized at each project baseline. Delegates from many different organizations participated.

■ Configuration management (first introduced to the management of ROM patterns) was applied to all configurations.

■ PERT charts were used to visualize and analyze schedules, progresses, and critical paths.

■ Gantt charts were used to visualize scheduled tasks, and allocation of responsibilities.

Process/project management (4)

■ Closure

- **Accept visitors from Reusable Components Management Team.**

- **Extract candidates for the reusable components from specifications, design descriptions, architecture, codes, and processes.**

- **Analyze economic benefit of developing reusable components.**

- **Develop reusable components and put into the asset archives.**

- **Make plan for the needs of maintenance and evolution.**

Software engineering measurement (1)

■ Goals of measurement programs

- Should be based on the top management objectives

- The goal values for the software process improvement were derived from the cause-analysis of the failure reports taken from the operational (customers') site.

■ Measurement selection

- All the artifacts through upstream documents to downstream codes were measured.

- Features such as size, structure, included items, personnel (including customer), time, quality, meeting frequency, fault, and failure were measured. Especially the measurements that were related to the top management objectives were thoroughly made.

Software engineering measurement (2)

- **Collection of data**

- **Automatic collection of data that were input on daily/weekly basis by project members**

- **Input data were automatically converted to progress reports, productivity data, cost accumulation, PERT charts, and Gantt charts, per project or per individual, based on the UW network.**

Software engineering measurement (3)

- **Productivity measure**
 - **EASL (equivalent -assembler source lines of code) per a number of total factory members**
 - including reused codes
 - excluding reused codes
- **Quality measure**
 - **Numbers of faults**
 - per 1,000 specification lines
 - per 1,000 EASL
- **The cost for measurement was constrained.**

Software engineering measurement (4)

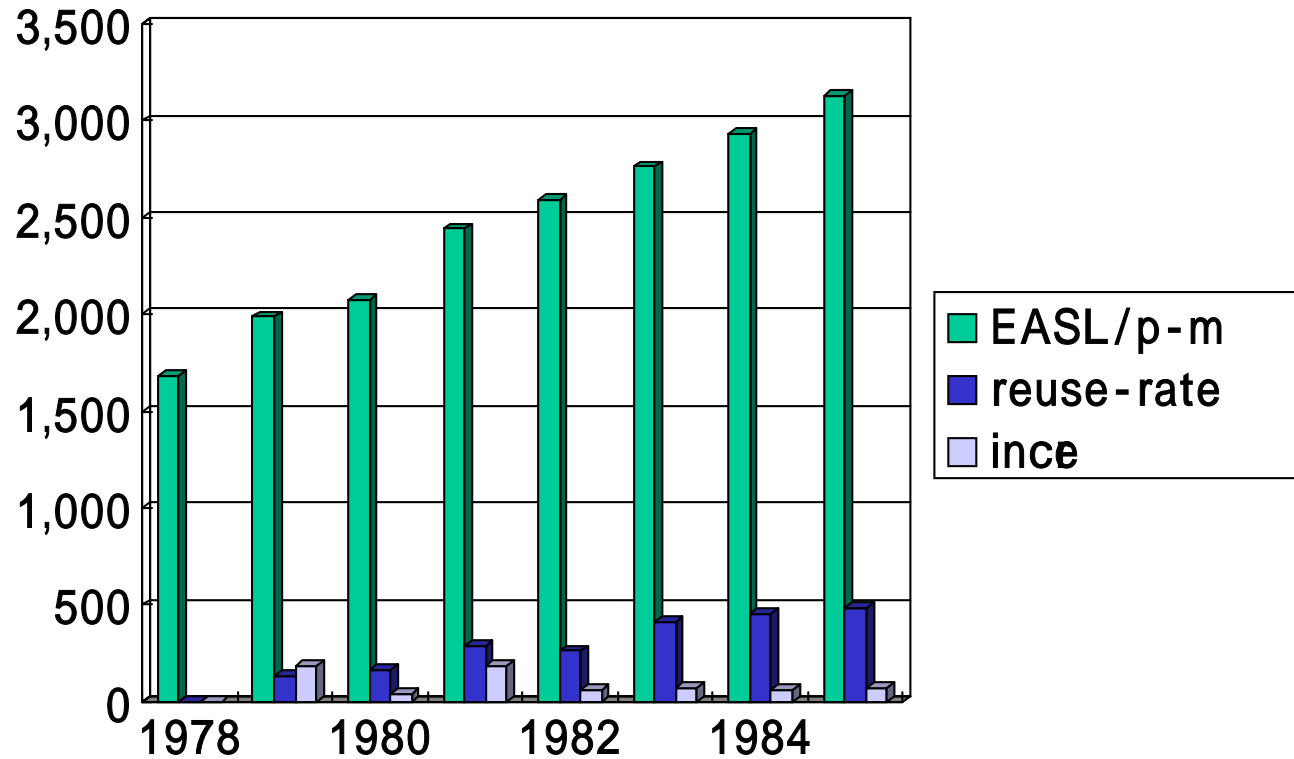
■ year	EASL/person-month	reuse rate	faults/KSLOC
■ 1978	1,684 (base)		7-20
■ 1979	1,988 (+18%)	13%	
■ 1980	2,072 (+ 4%)	16%	
■ 1981	2,443 (+18%)	29%	
■ 1982	2,595 (+ 6%)	26%	
■ 1983	2,763 (+ 7%)	41%	
■ 1984	2,931 (+ 6%)	45%	
■ 1985	3,130 (+ 7%)	48%	0.2-0.05

■ **Note1:** The number of all factory members (total-members), not only engineering members are counted to estimate EASL/person-month. The ratio between non-engineering-members and total-members is about 0.25.

■ **Note2:** About 20% of codes were written in Fortran(real-time). Remaining codes were written in assembler languages.

Software engineering measurement (5)

- reuse-rate and ince (rate of productivity increase) in % is one tenth of the values shown in the graph.



Software engineering measurement (6)

$$\text{Total-members}=2,300=0.75E+0.25NE$$

$$\text{therefore: } E(\text{no.of engineers})=1,725$$

$$\text{Fortran-E}=345, \text{ Assembler-E}=1,380$$

We assume that both Fortran and Assembler Engineers have the same productivity level.

$$\text{Total-EASL-per-month}=3,130 \times 2,300=7,199,000$$

$$\text{Total-new-EASL-per-month}=7,199,000 \times 0.52=3,743,480$$

We assume that the ratio between Fortran source lines and the EASL is 0.3, the ratio between Assembler source lines and EASL is 1.0, and 20% is the codes for Fortran.

After doing several simple calculations, it brings:

$$\text{New-EASL-per-month-per-Engineer}=1,486$$

Software engineering measurement (7)

- **Comparing productivities for writing new source codes**
 - **In 1985**
 - **From the report shown in the previous pages**
 - **New-EASL-per-month-per-Engineer=1,486**
 - **This number include data declaration lines.**
 - **In 2002**
 - **I had a report from a software factory which tells that 1,100-1,300 source lines of COBOL code are usually produced per programmer per month excluding data/comment lines.**

Software engineering measurement (8)

■ Major customers required us to guarantee (at customer site):

■ MTBF 8,000 hours

■ availability 99.99%

■ To meet those requirements, system tests at the factory were conducted with using logistic curves until:

■ the estimated number of remaining faults 1~0.5 faults/KSLOC

KSLOC: 1,000 × source lines of code

Evaluation (1) – plus factors

- **Management principles were made clear and shared by all factory members.**
- **Software artifacts and processes were made visible and manageable.**
- **Baselines and baseline activities (design-review/inspection, configuration management) were clearly defined and enforced.**
- **Improvement of working spaces and in-house development of the tool environments were enforced.**
- **Domain-based familiarization of software products was strongly promoted.**
- **Development of the program generator (and 4th generation language) for each domain-based product family was enforced.**

Evaluation (2) – plus factors

- **Archives for the reusable components and legacies were in everyday use.**
- **Quality circles were most successful voluntary activities.**
- **CDP, especially the development for the specialist career courses was promoted successfully.**
- **Project and individual evaluations accompanied by awarding resulted good efficacy.**

Evaluation (3) – minus factors

- **Adheres too much to the waterfall life-cycle model sometimes made it difficult to realize spiral-type life-cycle model.**
- **Too much importance was attached to the product families (or product lines) and domain orientation.**
- **Process families (or process lines) were also attached importance.**
- **However, supports by the tools and environments were not sufficient.**

Future Improvement

Following needs should be further observed.

- **Needs to coping with rapid changes in:**

- **the underlying technology, such as LSI, network, and common information platform; and**

- **the scope and nature of the application, such as ubiquitous computing, web application, etc., are required.**

- **Needs for meeting with ISO9000 and CMMI certification levels are required.**

Conclusive Note

■ Now, Matsumoto is trying to implement the software factory concept, that he developed and applied through the Toshiba Software Factory, using the Microsoft's Visual Studio in corporation with Microsoft Co. Ltd. In Japan.