Objectives and Context of Software Measurement, Analysis, and Control

Michael A. Cusumano

Massachusetts Institute of Technology

Sloan School WP#3471-92/BPS

October 8, 1992

Objectives and Context of Software Measurement, Analysis and Control

Michael A. Cusumano

Sloan School of Management
Massachusetts Institute of Technology
Cambridge, Massachusetts USA 02142

Abstract This paper focuses on the *what* and *why* of measurement in software development, and the impact of the *context* of the development organization or of specific projects on approaches to measurement, analysis, and control. It also presents observations on contextual issues based on studies of the major Japanese software producers.

1 Introduction

This paper focuses on the *what* and *why* of measurement in software development, and the impact of the *context* of the development organization or of specific projects on approaches to measurement, analysis, and control. It begins by discussing three basic questions: (1) Why do we measure? (2) What is it we want to know? And (3) how does the context affect the ability to measure? The second section presents observations on contextual issues based on my ongoing study of the Japanese [3]. This paper also relies heavily on extensive interviews conducted at Hitachi, Fujitsu, NEC, and Toshiba during May, August, and September 1992.

2 Basic Questions

Why we measure aspects of the software development process and conduct analytical exercises or experiments, as well as what we try to do with this information, depends very much on who is doing or requesting the measuring and analysis. There already exists a broad literature on technical aspects software measurement and the relationship to process management, which will not be reviewed here [1, 2, 5, 6]. In general, firms as well as academic researchers measure software activities and outputs across the development life-cycle to facilitate the following: (1) development planning and control, for particular projects; (2) learning and improvement, about products, the development process, planning and control methods, customer preferences, or to validate what is measured; and (3) bench-marking, consisting of comparisons of products, projects, or organizational performance. Managers generally use bench-marks to establish baselines for control or improvement, or for competitive analysis.

These categories of planning and control, learning, and bench-marking may seem obviously essential to every organization. Yet most industrial software projects operate under severe delivery time and budget constraints, suffer from shortages of good technical or managerial personnel, or have to cope with top managers or customers who do not fully understand the problems inherent in software development. For example, all but the very simplest planning, measurement, analysis, experimentation, and feedback activities require specific allocations of time and people. These allocations will not happen spontaneously within firms because they drain resources from current projects and primarily provide benefits only over the course of multiple projects. It is useful, then, to think about what we measure in software development in terms of who is measuring and what functions these groups normally perform. There are at least five relevant functions to consider:

(1) top management control and resource allocation;
(2) project management and internal project QA;
(3) independent inspection and QA;
(4) marketing and customer QA;
(5) process R&D.

2.1 Top Management Control and Resource Allocation

Top management in a firm -- that is, management above the project level, such as division-level managers or senior executives -- usually requests a set of relatively coarse data that indicate the status of major projects and how company divisions are allocating resources. These kinds of measures are too coarse to be useful for direct learning about processes or products, and do not help very much in managing projects more effectively.

Nonetheless, managers request such data for a variety of reasons. These include high-level purposes of control (such as management of multiple projects), performance comparisons (projects with other projects, divisions with other divisions, yourself with industry standards or competitors), or identification of major trends (such as increases in system size or changes in the mix of products). This information is useful, therefore, for creating a high-level picture of how an organization is utilizing its resources, and how resource needs are evolving. Top managers then need to adjust to changes or make policies and major resource-allocation decisions that could eventually affect software planning and control as well as learning activities.

2.2 Project Management and Internal Project QA

Project managers, including sub-project leaders or managers in software organizations organized by product departments and functions rather than by projects only, measure the development process primarily to know how long it takes to build, review, and test particular kinds of systems with teams of given skill levels. More specifically, project managers need enough information to formulate estimations, make plans, and then monitor current projects. They need to be able to recognize problems before projects are finished, and need sufficient information to make adjustments in the project team, in tools or processes, in product requirements, or in the schedule and budget, or they may want to alert senior management (and sometimes customers) in extreme cases of problems.

The kinds of data collected at organizations with good reputations for process and quality control can be quite simple. Table 1 presents an example of data items from NEC, this time from the Basic Software Division. The items listed are comparable to what Hitachi, Fujitsu, and Toshiba collect in their better development divisions [3], as well as to what the Software Engineering Institute [5] recommends (which is based heavily on IBM practices in its better divisions). Some companies, particularly for applications projects, add items such as percentage of design structures or lines of code reused (including minor changes). Hitachi also includes more data on personnel backgrounds and skill levels.

Table 2 contains another example of more detailed process data that relates to progress and comparisons of the quality assurance process, and shows how these data are used in NEC's Switching Systems Division. Of particular note is that NEC tries to measure quality as well as the progress of reviews quantitatively even in a phase (specifications) where progress is difficult to quantify. Management also has used historical data on comparable projects to establish baseline or "model values" for the various quality indicators as well as for "control boundaries." If actual values fall below or beyond the model values, this requires a specific manager to authorize or take a specific action.

2.3 Independent Inspection and QA

Why and to what degree independent inspection departments (including QA or software-engineering departments responsible for quality control and process issues) measure characteristics of the software development process or of software products depends on the scope of their responsibilities and their authority relative to project managers. Many software organizations have no independent inspection or third-party reviews, no independent QA or

Table 1: Process-Control Data Items in NEC Basic Software Division

1. Requirements Analysis/Definition Process

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

2. Functional Design Process

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

3. Detailed Design Process

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

4. Coding Process

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

5. Unit Test/Function Test Process

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

6. System Test Process

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

Source: [3] p. 304, taken from [8] pp. 3-9

Table 2: Design Quality Indicators Model in NEC Switching Systems Division

Specifications Phase

Indicator	Specifications Volume (# of Spec Sheets/ Total Est. LOC)	Review Coverage Rate (# of Pages Reviewed/ # of Spec. Sheets)
Model Value	15 pp/KL	90%
Control Boundaries	- 40%	- 20%
Control Method/Tool	Spec Sheets	Review Record Chart
Action Items	Intensive review of described items, gaps, content check	Continue review of uncovered portion
Decision Level	section manager	group leader

Indicator	Review Manpower Rate (Review Work-Hours/ # of Spec. Work-Hours	Bug Count (# of Serious Items in Review Comments/# of pages reviewed)
Model Value	15%	0.3/page
Control Boundaries	+/- 30%	+/- 40%
Control Method/Tool	Review Record Chart Man-Hours Total	Review Record Chart Bug Estimate Curve
Action Items	If +30%, then reviews are inefficient; correct method, more review items focus If -30%, reviews are insufficient; check bug count, and review again if too few	If +30%, then analyze comments & recheck specs. If -40%, check review points, and revise review check list
Decision Level	group leader	group leader

Table 2 continued

Summary of Other Phase Indicators, Model Values, and Control Boundaries

Quality Indicator	*Model Value*	*Control Boundary*
<u>Design Phase</u>		
Design Sheets Volume (# of Design Sheets/Total estimated LOC)	50pp/KL	-30%
Review Coverage Rate (# of pages reviewed/Total # of design sheets)	70%	-20%
Review Manpower Rate (Review Work-Hours/# of Design Work-Hours)	10%	+/- 30%
Bug Count (# of serious comments in review items/ # of pages reviewed)	0.4/page	+/- 40%
<u>Coding Phase</u>		
Review Coverage (# of source lines reviewed/total est. LOC)	30%	-20%
Review Manpower Rate (Review Work-Hours/Coding Work Hours)	10%	+/- 20%
Bug Count (Detected bugs/total est. LOC)	7/KL	+/- 40%
<u>Unit Test Phase</u>		
Test Items Selection Rate (Test items/total est. LOC)	30/KL	+/- 30%
Trace Comprehensiveness Rate (Items Traced in Desk-Top Debugging/ Estimated items to be found)	100%	-20%
Review Manpower Rate (Review Work Hours/Unit Test work hours)	10%	+/- 20%
Bug Count (# of detected bugs/total est. LOC)	8/KL	+/- 40%

Integration Test		
Test Items Selection Rate (Test items/total est. LOC)	8/KL	+/- 50%
Bug Count (Bugs detected/total est. LOC)	4/KL	+/- 30%
System Test		
Test Items Selection Rate (# of test items)	30	-30%
Bug count (detected bugs/total est. LOC)	1/KL	+/- 30%

Source: [7]

engineering departments; project members (alone or with customers) make all decisions on methods and tools, and conduct, if there is time, their own final reviews, final testing, and other QA activities. In other organizations, QA, inspection, or engineering staff departments exist to help or make sure projects do what they are supposed to do, but even they exist in a spectrum from "weak" to "strong."

There are ways to measure how centralized are independent inspection or QA activities, or, alternatively, how time-consuming are process-control and QA activities. Possible measures include (1) percentage of project manpower or time planned for project management as well as for quality-assurance activities (the latter may include reviews, configuration management, test management, post-release quality assurance, or problem-solving meetings); (2) to what extent independent personnel from a QA, inspection, or engineering department actually become involved in monitoring and improving the quality of the development process, and (3) to what extent an independent QA or inspection department can and does hold up the shipment of major projects if the manager of this department feels the quality of the final product is inadequate.

In organizations with little process-control or QA activities, there usually are only a few staff hours dedicated to outside reviews and independent inspection or QA tasks. Inspection and QA personnel may also not be software experts or even engineers. Their primary role is to collect numbers, such as numbers of defects found in testing or reported by customers, or review documents, to give project management and upper management no more than a coarse indication of how the development organization is doing and how projects are proceeding. If top management instructs project managers to devise tactics and policies to improve quality on the dimensions measured, QA or engineering departments might get involved in planning quality-improvement activities. Weak QA or inspection departments, however, generally do not have the technical expertise, respect from development personnel, or the data to become important mechanisms for learning and then transferring knowledge about quality and process improvement within or across projects.

Japanese organizations with "strong" centralized process-control and QA functions include the basic software divisions of Hitachi, Fujitsu, and NEC, and the applications divisions of Hitachi and Toshiba. These companies generally devote from 3% to 10% of project manpower to QA and inspection activities, depending on the type of system and the project characteristics. Independent staff for inspection or QA, or QA sections within software engineering departments, are thus relatively large. They also have broad responsibilities, extending from monitoring the quality of the development process, to validation of metrics, to evaluating the quality of the final product and reserving the rights of final approval for shipment.

Managers of large QA or inspection departments and their staff are a combination of experienced engineers and specialists in testing, process management, and quality assurance. In Hitachi, for example, personnel in these strong departments may become integral parts of projects and participate actively in reviews, perform checks throughout the development process, classify problems, help project members with examples from other projects, become heavily involved in training personnel to spread expertise and process knowledge, and even take charge of final testing for some kinds of systems or do product testing to mimic customer situations. The Hitachi QA departments are also responsible for interfacing with customers and assuring follow-up solutions to quality problems ranging from major faults that cause system crashes to more subtle design issues such as those affecting ease of use or man-machine interfaces (Table 3). These broad responsibilities obviously require various kinds of data on the development process and on the software products themselves. In the Japanese firms, these data again usually take the form of numerous simple but quantified and complementary metrics, as in Tables 1 and 2. The Japanese also use subjective or "soft" data, such as comments in review meetings.

Major questions that need to be answered are how necessary and cost-effective are independent QA, inspection, or testing activities. When, and to what extent, can project members manage themselves and "build-in" quality from the very beginning of the development process, with no one looking over their shoulders or checking their work independently? There is another way to look at these kinds of measures: Are large, strong QA departments or large percentages of time devoted to QA activities signs of a good process or a poorly controlled or perhaps an immature process? The software industry needs more data as well as more organizational case studies to answer these questions.

2.4 Marketing and Customer QA

Some marketing departments in companies, and independent market research organizations, including industry organizations and trade publications, collect detailed data about items such as prices or, more usefully, customer

Table 3: Activities of a "Strong" Independent QA Department

-- Managed by Respected Software Development, Testing & QA Experts

-- Responsible for Collecting & Analyzing Quality Data from Multiple Projects & Customers

-- Active Participation in Reviews Through Checklist Analysis & Discussion of Problems in Other Projects

-- Independent Testing of Final Products

-- Formal Approval Over Shipment

-- Participation in Defining Methods, Tools, Procedures

-- Participation in Training

-- Responsible for Meeting with Customers on Quality Problems to Ensure Follow-Up

-- Classification of Problems From All Phases of Development

-- Dissemination of this Knowledge Through Reports, Training, Review Checklists & Discussions, Project Procedures, New Tools & Methods, etc.

Sources: Interviews with managers in Hitachi's QA Departments for Basic Software and Business Application Systems reactions to software products and to company performance in various aspects of the software development process. Table 4 contains an example of such data collected by *Nikkei Computer*, Japan's leading computer-industry journal.

The immediate purpose of this data-gathering for company marketing departments is internal bench-marking, which provides a source of information to improve products and operations. Public compilations of these kinds of data are also for bench-marking, but for comparing performance across multiple firms or products, and usually for the benefit of consumers, who then have more information that they can use to choose among vendors. Whatever the source of the data, analyses of customer satisfaction can provide important information to marketing and QA departments, top management, and development departments on whether the company is doing well compared to competitors and meeting the needs of customers, as opposed to performing well or not on more technical dimensions of the software-development process, such as project progress and estimation accuracy, bug detection and correction, testing coverage, and the like (which may and often are related to customer satisfaction). For these data to be most useful, they should relate characteristics of the software products or services to the development or systems engineering process, although few firms have yet to do such sophisticated analyses.

2.5 Process R&D

Increasing numbers of software organizations now have process analysis or auditing activities as well as a process R&D function. The latter consists of a group of people who do not have project responsibilities but whose primary job is to learn and transfer knowledge, if possible. They analyze project performance and needs, develop and evaluate new metrics, tools, and methods, they may help introduce these technologies into projects, or study recurring process or quality problems and try to propose solutions, which may take the form of particular metrics, tools, or techniques. These groups may also be involved in validation exercises as well as comparative bench-marking.

Process analysis and R&D activities of this sort can take place within large projects or systems engineering departments through special assignments, or they may take place within functional departments such as for software engineering methods and tools, production management, quality assurance, common technology development, or R&D. Outside consultants can also undertake process analysis and R&D functions for clients. The major distinction with project activities, however, is that process R&D groups are not concerned with planning and controlling the development process or outcomes of particular projects. Process analysis and R&D groups within firms are generally interested in more fundamental questions that come close to what academic researchers in universities study, though with usually with more of an applied focus that combines concerns for quality, costs, and practicality.

3 The Development Context

It is relatively easy for researchers and industry experts, quality-assurance specialists, or project managers to define a set of ideal measurements that they believe would be useful for planning, control, and learning. Even if these measurements can be satisfactorily validated as being effective for management and process feedback, the *context* of the project and the development organization very much affect what kinds of measurements firms or projects actually use, and what they do with the information that results. Since contextual issues themselves constitute a very broad topic, this section focuses on some examples taken from Japanese software development organizations, and discusses what appear to be particularly important contextual parameters:

(1) optimal process for the product;
(2) business characteristics of the project;
(3) development-organization culture.

3.1 Optimal Process for the Product

In *Japan's Software Factories*, I argued that it was possible to identify a spectrum of software product or system

Table 4: Nikkei Computer Customer Satisfaction Survey Questions

Sampling of Questions asked by *Nikkei Computer* during 1988-89 in National Surveys, with ratings on a 1 (dissatisfied) to 10 (satisfied) scale.

Systems Systems-Engineering Service

1. Explanation of new products
2. Systems software version-up support.
3. New system configuration support
4. Proposals for solving problems related to improving system efficiency
5. Technical support for software development
6. Technical support for machine security
7. Promptness in responding to requests to fix defects
8. Technical support in communications areas
9. Offering of broad information from a neutral standpoint
10. Businessman-engineer morality
11. Total satisfaction.

Applications Systems-Engineering Service

1. Proposals for system planning and design
2. Understanding of business strategy
3. Knowledge regarding the application
4. Knowledge of industry trends
5. Ease of understanding product documentation
6. Communication ability
7. Application-system development methodology
8. Technical support in communication areas
9. Businessman-engineer morality
10. Total satisfaction

Factors Influencing System Selection

1. Price-Performance
2. Upward compatibility
3. Reliability and fault tolerance
4. Available software
5. Company policy and business contacts
6. Systems engineering support
7. Same industry operating results
8. Maintenance service
9. Technology excellence
10. Salesman enthusiasm
11. Installation conditions
12. Reputation
13. Other

Source: [3] pp. 58, 462-463

characteristics for any type of application as well as a corresponding set of process options. These options range from a craft or job-shop approach, such as a unique, complex system that developers must fully customize or invent for a single customer; to an application-oriented project for a broad set of general users, as in a personal-computer applications package; to a more "factory-like" approach. What I have described as a software-factory approach in many ways overlaps with the concept of a structured, well-defined software-development process, where project managers, developers, quality-assurance staff, and process R&D staff utilize quantitative measurements and qualitative information, including a store of historical data for baselines as well as current project data, for planning, control, reuse-promotion, and learning. A problem with this factory-like approach, however, is that it requires historical data and applies best in cases where current projects are more-or-less similar to past projects, i.e. where the problem domain and development personnel are relatively stable, and where projects are not trying to write the equivalent of the first version of a best-selling book.

For systems on the high-end of the spectrum ("high" in the sense of relative system cost and innovative or inventive technical requirements), historical data, like reusable components, are scant or do not exist, because the requirements or architectures are almost completely new. In these jobs, managers may decide to measure various aspects of the process to begin creating historical data for future project control or process improvement. But unless measurement is highly automated, then the resources required to collect data only make sense if management expects to build similar systems in the future. Since most organizations do build series of similar systems (estimates of redundancies in system development in Japan and the United States range from 60% to 90%), measurement usually makes sense as an investment for the future, even if there are no historical baselines. Yet, as will be discussed below, many company managers and project managers under schedule, budget, and manpower constraints often do not have or, at least, they often do not allocate resources to tasks such as measurement that represent an investment which pays off only in the future.

On the bottom end of the spectrum shown in Table 5 managers of application-oriented projects aimed at producing best-seller packages might also not want to impose a strict measured process on their developers. If the task is to invent for a broad range of users, and the system can be electronically replicated for thousands or millions of users, then the development costs are trivial, and it may not be possible or advisable to attempt to plan and control the cost of development. On the other hand, measurements that affect defects or other forms of quality are critical, since they affect sales of the system and other products of the same company, and it is costly to replace products for such a large user base.

Data from various firms indicate that the middle of this spectrum might account for the majority of all systems built, as well as subsequent versions of full-custom systems or low-end packages. For development organizations building multiple versions or generations of similar systems, a development process emphasizing measurement, historical baselines, feedback mechanisms for control and learning, or systematic reusability, make the most sense technically and economically.

Another implication of this framework is that large systems cannot be built, or cannot be built very well, in an ad hoc manner, because these require some division, coordination, and then integration of work. Division of work, as well as coordination and integration, require some degree of standardization in interfaces and probably in design methods, tools, management practices, and other areas. Thus even producers of high-end custom systems or of large packages may find that they have to move toward a more standardized process simply to survive, and this requires measurement and control; and the more continuity or redundancy there is in the tasks they undertake, the more logic there is in pursuing a measured, controlled, and continually improved process.

These different process options can also exist within a single organization. For example, Hitachi, Fujitsu, NEC, and Toshiba have "elite" in-house development organizations that build large and complex, but relatively routine, systems, in a factory-like process, with extensive subcontracting of detailed design, coding, and unit-test work to subsidiaries and subcontractors for the more routine applications. They also use special projects, laboratories, or specialized subsidiaries for unique or one-of-a-kind systems, and employ software houses for routine work or small-scale projects that are not economical to do in-house.

It is also important to realize that software and hardware technology are still dynamic. Firms may evolve from a craft-like approach to a factory-like approach for many of their systems, but then find that technical changes force them back to a partial craft-like or invention mode at least in the sense that reuse of existing designs or of historical baseline data is no longer helpful. For example, Hitachi has an historical database of 150 projects from the mid-1980s that managers use to estimate defects in large-scale basic software and applications systems that run on Hitachi mainframes. The database, estimation algorithms, and procedures are called the Software Quality Estimation Systems (SQE) ([3] pp. 188-191). For basic software and for many large-scale applications systems,

Table 5: Product-Process Strategies for Software Development

Product Type	Process Strategy	Organization Type
HIGH END:		
Unique Designs (Full Custom, "Invention")	Meet Customer Requirements & Functionality	
High-Priced Premium Products	Hire Skilled Workers To Design, Build Needed Tools & Methods	**CRAFT-ORIENTED JOB SHOP**
Small To Medium-Size Systems	No Organizational Skills To Perform A Series Of Similar Jobs Or Do Large Jobs Systematically	
MIDDLE:		
Partly Unique Designs (Semi-Custom)	Balance Customer Needs & Functionality With Production Cost, Quality	
Medium-Priced Products	Skilled Workers Mainly In Design, Standard Development Process	**SOFTWARE FACTORY**
Small To Large-Sized Systems	Organizational Skills Cultivated To Build Large Systems And Reuse Parts, Methods, Tools, And People Systematically	
LOW END:		
Unique, Mass-Replicated Designs (Scale Economies)	Maximize Application Functionality For Average User Needs	
Low-Priced Products (Packages)	Hire Highly-Skilled Workers Knowledgeable In Application	**APPLICATION-ORIENTED PROJECT**
Small to Medium-Sized Systems	No Organizational Skills To Develop Large Products Or A Series Of Similar Products Systematically	

Source: [3] p. 15

an independent quality assurance department needs to give its approval for shipment, and this is far from automatic. The QA Department's decisions are driven primarily by historical data: whether or not the number of detected defects in testing equals the projected number, which is based on historical data and quality objectives for the system being built (Figure 1). However, Hitachi customers are gradually demanding more basic software and applications for distributed networks of work stations and smaller computers. While many functional requirements remain the same, the architectures of the small-scale distributed systems are sufficiently different that Hitachi management has decided not to use the SQE system for these projects until they accumulate a sufficient store of projects to generate new baseline estimates. For project management (cost, schedule, and manpower estimation), Hitachi has also had to accumulate separate sets of data and create a version of its project management system for work-station software. In addition, reusable components that made it relatively simple to build "semi-customized" versions of systems that run on mainframes are also not completely transferable to distributed work-station-based systems, hence, estimates of productivity and quality affected in the past by high reuse rates are no longer accurate for these new systems. Fujitsu, NEC, IBM, and other mainframe software producers around the world have undergone similar transitions, although it is primarily a matter of time before they update their historical databases.

3.2 Business Characteristics of the Project

Table 5 refers to management decisions to adopt variations in process standardization, control, reusability, divisions of work, and other process elements because of basic product characteristics. In practice, however, for economic and contractual reasons, not all of the projects where measurement and control through historical baselines are technically possible will be measured and controlled with the same degree of intensity. In other words, even the elite Japanese programming organizations will select, for business reasons, which projects receive the "best-practice" process and which do not. Part of the rationale for such variations is that, at least in Japanese organizations, much of the data-collection and analysis process is not automated, and thus requires considerable work-hours. In addition, management requires the best-practice projects to devote considerable time to reviews, testing, problem-solving, and other quality-assurance activities; for economic, technical, or scheduling reasons, certain types of projects may curtail these activities and the measurement or analysis work they require.

For example, in applications development, Hitachi managers identify two types of projects: One type are customer in-house or "private" applications, used internally only by the customer, such as systems for inventory management or hospital information management. A second type are "public" applications where there might be one customer but the system has many end-users or a high possibility of affecting many end users. These include systems such as for stock exchanges, on-line banking, and reservations (Table 6).

Hitachi managers apply a higher level of control for public systems. Members of the QA Department join these projects from the beginning, and work closely with project members to assure quality of the development process through reviews and the quality of the end product by reviewing test plans and performing some independent tests. Hitachi manages these high-priority public systems in the same way as basic software, and allows the QA Department to take on some of the responsibility for quality assurance. The goal is essentially to deliver systems with zero defects, and management chooses to organize and invest in an independent QA or inspection department to assist projects.

The private, internal-use systems require a lower level of inspection because they do not need to be zero-defect. These are projects where the impact of a software failure is minimal on the customer or on society, and where there is only a small number of users. Hitachi will curtail some control and inspection activities to reduce short-term development costs, while also believing that, in the long term, the small number of users will mean relatively few bugs will be found in the future. Hitachi will deliver a system to the quality (reliability) specifications required by the customer, which is usually decided along with the cost of the total system, since testing time is a major component of reducing defects. For customers without specific quality requirements from the customer, Hitachi delivers a level of quality that the project manager, with the agreement of the QA Department, feels is appropriate given the price the customer has agreed to pay for a system. The QA Department monitors negotiations between project managers and customers on price and quality issues, although project managers have de facto control over these kinds of projects. Part of the reason for this behavior is that, in contrast to basic software or public systems, for the private in-house systems, ownership rights rest with the customer, and customers generally fix their own bugs after a set period.

NEC, Fujitsu, and Toshiba make similar kinds of distinctions in their applications projects, and vary accordingly what they measure and how closely they attempt to review and control the development process and the quality of final products. For example, Toshiba's Fuchu Works, which builds a variety of process-control systems, classifies its projects into three categories -- A, B, and C -- depending on several contextual factors, as quoted in Table 8. The "A-category" systems are comparable to Hitachi's public systems, which have many end-users, like an on-line banking system, or where a major failure in a software control system (such as in a nuclear power plant or chemical plant) can have disastrous social consequences.

There are other examples of areas where Hitachi makes different decisions depending on the project categorization. Again, Toshiba, as well as NEC and Fujitsu, make similar kinds of decisions.

One area is the extent of independent system testing. The QA Department in Hitachi's applications development division operates its own testing facility called the SST (System Simulation Test) Center. This has a staff of approximately 100 engineers, which develop and conduct tests and build testing tools. The facility is used by Basic Software Division also in that they test operating systems with applications programs on Hitachi hardware. While the SST is a special phase of testing that comes after system test done by project systems engineers, Hitachi puts only about 10% of the applications systems it develops through the SST Center, because this is expensive. The QA Department selects which 10% of systems to put through SST. The 10% are usually totally "new" software products, such as new packages or new private/internal software, or public systems with high reliability requirements.

Another area is the extent of total QA activities. Hitachi allocates from 3% to 10% of project manpower to QA activities, which includes the cost of QA Department staff working with particular projects (development plan audits, reviews, quality audits, customer service, and special testing) as well as the quality activities of project personnel such as participation in reviews. These percentages do not include system and integration test work done by the systems engineers. This does include SST work. In basic software, the common figure for QA activity allocations is 10% of project manpower, including more extensive product testing done by the QA Department. In applications, the 10% allocation occurs mainly for the large-scale public systems, and 3% for the internal-use systems. Percentages are affected by the number (and position) of people who attend reviews and the amount of time they spend, as well as the amount of time spent by the QA Department in their regular activities of test planning, special testing, problem analysis, and customer support.

A third example is the extent of independent QA or inspection department authority. For internal/private as well as public systems, Hitachi project managers have access to and are required to use data from past projects for scheduling as well as for quality targets, although quality targets are adjusted to customer specifications and prices. The QA Department is aware of the customer requirements and contract price for the system, and is supposed to take these factors into consideration, rather than seek absolute quality levels such as zero defects for these kinds of systems.

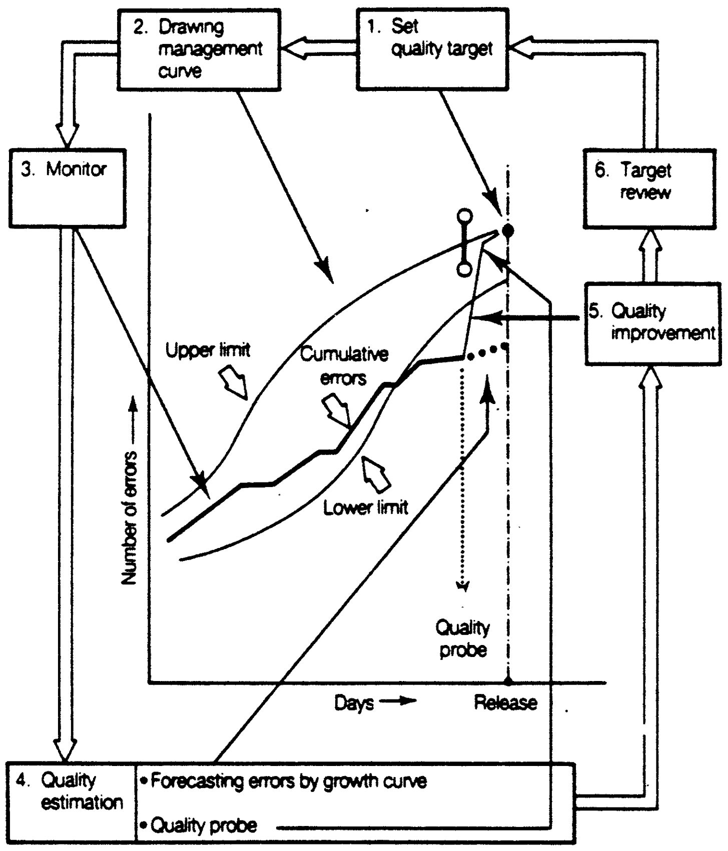
In basic software, the QA Department has final authority on determining shipment of a product. There are few disagreements with project managers in the cases of systems where there is extensive historical data, primarily number of bugs detected and fixed in system test versus projected and targeted numbers. For new systems or systems with many new features, the historical projections may be inaccurate, and the quality targets require considerable negotiation among division management, project management, and the QA department.

In applications projects, formally, the QA Department has the authority to stop shipment on all systems, and does occasionally do this even for in-house private systems if QA people are not satisfied with the quality of the system. If QA people are not satisfied but the project manager argues that a system has met the customer requirements and the customer wants it, the QA manager will check directly with customers that this is the case before approving shipment. For all but the newest systems, the QA Department has historical data that allows it to estimate how many bugs remain in a system (see Figure 1). If the customer accepts this situation, then the department will agree to ship. The contract will determine who pays for fixing bugs over what period of time.

In the case of public systems, as in basic software, quality targets are driven by historical data and experience. The QA Department collects data that comprise the guidelines, and project managers make the actual decisions on targets at particular stages, as with in-house/private systems. However, the guidelines are relatively clear, and project managers cannot set goals far from the guidelines. Their plans are also closely checked by the QA Department. If QA people feel a project manager has set quality objectives that are too low and the project manager does not want to change them, the QA manager will go to the department manager or the factory manager. The QA Department also has much clearer and absolute authority on shipment for the public systems.

There appear to be few cases where project managers conflict seriously with the QA Department on public systems, where goals are pretty much data driven. There is more opportunity for disagreements with the

Figure 1: Hitachi Quality Target Management



Source: [10] p. 198.

Table 6: Hitachi Applications Categorization

(1)	Customer in-house ("private") applications, used internally only by the customer.	
	Comments:	Software failures are not critical for the customer or for society, and the small user base means few errors are likely to be found in the future, so less intensive reviews, testing, and independent inspection is needed.
	Examples:	inventory management, hospital information systems.
(2)	Customer with many end-user ("public") applications or used by the customer to service end users.	
	Comments:	Software failures are potentially critical for the customer or for society, and the large user base means many errors are likely to be found in the future, so intensive reviews, testing, and independent inspection is needed.
	Examples:	stock exchange, on-line banking, reservations systems.

Source: Personal interviews at Hitachi Information Systems Development Center, 4 August 1992, with Yasuo Yoshida, Dept. Mgr., Quality Assurance and Inspection Dept.; Katsuyuki Yasuda, Dept. Mgr., Software Engineering Department; and Takamasa Nara, Deputy Dept. Mgr., Quality Assurance and Inspection Dept.

Table 7: Toshiba Applications Categorization

1) The level of the system's influence on society, our customers, and on our own business;

2) The level of risk involved, depending on:
   - a) our technical experience in the system to be developed;
   - b) the number of new items that must be developed;
   - c) the amount of modification to the existing system that is to be renovated;
   - d) the size and complexity of the system;
   - e) the urgency of the product's delivery date and system acceptance date;
   - f) the tightness of available project funding.

In the beginning of each project, we assign a management-level ranking that has been determined using the above criteria. There are three ranks: A, B, and C. Compulsory attendance by each level of managers and specialists at design reviews and internal acceptance tests are defined based on this rank classification.

Source: [9] p. 383

internal/private systems, where project managers are given more authority, and QA plays more of a role of setting guidelines. However, QA people recognize that public systems have different reliability requirements than private systems or non-critical systems, and that business decisions have to be made sometimes that are compromises with quality, but only for one-user, non-critical internal/private systems.

3.3 Development-Organization Culture

In addition to the product, process, and project characteristics described above, there is a hierarchy of cultural influences that affect to what extent companies attempt to use measurement for planning, control, and learning:

(1) national culture (weak influence);
(2) company culture (strong influence);
(3) product division culture (stronger influence).

Discussions of Japanese software development practices might indicate to some people that characteristics of the Japanese people, including how they are educated in general, or how they expect managers, workers, and customers to behave, make it easier to introduce and utilize measurements or even a factory-like process in software development. This may be true. Japanese customers demand highly reliable products; in response, companies have developed elaborate mechanisms to root out errors and causes of errors, using quantitative and qualitative methods. Even Japanese high schools teach basic statistics, and even blue-collar workers in most manufacturing industries are required to record, analyze, and monitor performance, and then take corrective action if necessary and as indicated by simple statistical data. It also appears that the culture of software programming in Japan is somewhat different than in the United States or Europe, with many managers and engineers viewing this more as a production activity, rather than as an art or craft.

Yet there are variations among Japanese managers, engineers, and firms, especially when comparing the large computer manufacturers, who also develop most of the basic software and large amounts of applications programs in Japan, to smaller software houses that may have no defined process of their own. In addition, there are many non-Japanese firms that measure intensively and use quantitative data for management control as well as for process improvement. This suggests that company culture, as well as the cultures of particular product divisions, play an even more important role in determining the context of the project than the national culture.

Variations within firms sometimes exist as the result of a rational selection process, such as technical differences in reliability requirements, system complexity, or customer contracts. Other times, however, variations may be the result simply of chance, history, timing -- such as different stages of maturity -- or bad management. For example, Hitachi and Toshiba have long histories as electrical equipment and heavy machinery manufacturers, and they brought into their software development operations in the 1960s and 1970s managers with hardware backgrounds or backgrounds in inspection who believed firmly in process and product measurement as well as the collection of historical data and active use of this data. Fujitsu and NEC followed the lead of Hitachi as well as IBM, attempted to meet rigorous quality standards set by NTT for switching systems, and, under the influence of insightful managers, introduced rigorous standards into their basic software operations, which then influenced gradually how other parts of the company developed software.

But while Hitachi more-or-less applied the same practices started in basic software to applications (both areas were managed within the same software factory organization from 1969 until 1985), NEC and Fujitsu exhibit more variations. Applications in general are controlled less than basic software, and neither company utilizes an independent quality assurance or inspection department to determine product shipment. Both companies have independent QA or inspection departments that collect data and make this available to project managers for reference, but managers in Fujitsu and NEC both complain that the performance of their applications projects in terms of cost, schedule, or quality management is not as predictable and thus not as well controlled as in Hitachi. The NEC and Fujitsu approaches may be lower in cost during the short run but higher in cost over the long run or for unusually complex projects.

4 Conclusions

The discussion above reflects an unfortunate reality. It is naive to expect a consensus to emerge easily within even a single organization, let alone within an entire industry or among a set of different actors or observers, regarding what to measure and control in software development. The perspective of senior managers, project managers, QA or inspection personnel, marketing and customer service groups, and researchers from companies or academics can be very different. Different types of systems and customers, as well as different company and division cultures, also may have an enormous impact on how projects use data for planning, control, and learning.

Top management, for example, may not have the foresight to act for the long term and may react to high software costs by pressuring project managers even more on current projects; this, in turn, makes it more difficult for project managers to escape the pressures of short-term schedules and budgets. Top management may also not allocate the resources needed for effective metrics, QA, or process R&D programs. Within projects, engineers may not want to be measured or to collect data that is needed for effective planning, control, and improvement, or for bench-marking. Hitachi, for example, tests its programmers and records their scores for project planning and estimation purposes. But many software personnel do not like to be treated in this manner, even in Japan, and academic researchers as well as company researchers may not be interested in metrics or technologies that project managers and engineers see as practical or immediately beneficial at a reasonable cost. Improved tools, techniques, and metrics, and more automated or unobtrusive data collection and analysis, will solve some of these difficulties in time. But it is also important to try to create a culture of measurement where projects collect, analyze, and use data primarily to improve quality and overall efficiency. This will require identifying and assisting weak project members or weak projects, which will provoke some resistance; and there is only a subtle distinction between using measurement for improvement and using it for control. But companies need to make these distinctions because there is a larger danger: that managers and engineers can become paralyzed by these debates on resource allocations or the search for "perfect" metrics to the point where projects measure inconsistently or measure nothing at all. Either option makes it almost impossible to learn because companies must establish a quantitative baseline from which to improve.

Part of the resistance to measurement relates to an old quality-management debate that centers around the issue of whether a company can "build-in" rather than "inspect-in" quality. In industries such as automobiles, Japanese managers in the 1950s and 1960s made all production workers serve as inspectors of the previous person's work, and taught workers how to use simplified statistical control techniques. Companies thereby were able to eliminate large inspection or QA departments, eliminate large amounts of rework, and improve productivity as well as quality. Japanese firms also extended the concept of quality circles and quality-improvement activities to engineering departments, research departments, and all other white-collar jobs [4]. But, in software development, to what extent can we build-in quality while eliminating or drastically reducing time spent in outside reviews, independent testing, and other forms of inspection? Can software developers measure and control themselves, or is software development a different kind of activity from what we see in other industries? Or is this combination technical and business decision, reflecting the nature of the system, the specific user requirements, the developer's general process, the likely amount of profit from the project, and other factors?

The Japanese, who are innovators and experts in quality management, seem to be playing it both ways technically and to let business as well as technical decisions determine what they actually do in a given project. Companies allocate many hours to personnel training in common development methods, to problem-solving meetings and reports, to reviews, and other activities that project personnel plan and do to assurance a high level of quality in their work. But Hitachi and Toshiba, in both basic software and applications, as well as NEC and Fujitsu at least for basic software, also believe that short-term pressures of schedules and budgets make it difficult for project managers and developers to take adequate care in designing, testing, documenting, reviewing, and doing other tasks not directly related to building and delivering code. Therefore, managers in these companies, for their most critical projects -- basic software and large "public" systems -- insist on having independent departments measure as well as monitor the quality of the development process and the quality of the final product. For less critical projects, they will vary how carefully they manage. The ideal approach is not to have to manage very much and to build-in quality every time, but no one has yet reached this ideal. In the interim, projects must continue to measure, control, inspect, and test to the extent that this is economically and technically practical, as well as important to the user. At the same time, and with the help of process researchers, companies must strive to reduce the amount of time and effort they spend in measurement and control activities.

References

1. J. Arthur: Measuring Programmer Productivity and Software Quality. New York: Wiley 1985

2. S.D. Conte, H.E. Dunsmore, and V.Y. Shen: Software Engineering Metrics and Models. Menlo Park, CA: Benjamin/Cummings 1986

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

4. M.A. Cusumano: The Japanese Automobile Industry: Technology and Management at Nissan and Toyota. Cambridge, MA: Harvard University Press 1985

5. W.S. Humphrey: Managing the Software Process. Reading, MA: Addison-Wesley 1989

6. C. Jones: Applied Software Measurement: Assuring Productivity and Quality. New York: McGraw-Hill 1991

7. NEC Corporation: Discussion on the Switching Software Development Process. Switching Software Engineering Division, Abiko Works, 31 August 1992

8. NEC Corporation: QA System in NEC: Scientific Control of Production and Quality in NEC -- Basic Software. Unpublished internal document, September 8, 1987

9. K. Yamashita and O. Sasaki: Computer Application Systems Engineering Center -- A Software Factory. Fuchu Works, Toshiba Corporation 1992

10. K. Yasuda: "Software Quality Assurance Activities in Japan," in Y. Matumoto and Y. Ohno, ed., Japanese Perspectives in Software Engineering. Reading, MA: Addison-Wesley 1989