SOFTWARE DEVELOPMENT IN ESTABLISHED AND NEW ENTRANT COMPANIES: CASE STUDIES OF LEADING SOFTWARE PRODUCERS

by
STANLEY A. SMITH

B.S., Computer Science
Iowa State University
(1982)

Submitted to the Alfred P. Sloan School of Management
and the School of Engineering
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE
in Management of Technology

at the
Massachusetts Institute of Technology
June 1993

Signature of Author: [illegible]
Alfred P. Sloan School of Management
May 7, 1993

Certified by: [illegible]
Michael A. Cusumano
Thesis Supervisor

Certified by: [illegible]
Chris Kemerer
Thesis Reader

Accepted by: ______________________
Rochelle Weichman
Director, Management of Technology Program

SOFTWARE DEVELOPMENT IN ESTABLISHED AND NEW ENTRANT COMPANIES: CASE STUDIES OF LEADING SOFTWARE PRODUCERS

by

STANLEY A. SMITH

Submitted to the Alfred P. Sloan School of Management
and the School of Engineering on May 7, 1993
in partial fulfillment of the requirements for the Degree of
Master of Science in Management of Technology

ABSTRACT

Software has become a dominant part of both high and low scale technology products. Much has been written about the different software development processes to be considered when developing products. There is also a constant flow of new processes and tools that are being touted as the next wave of software development.

This study concentrates on the software development processes employed by leading "established" and "new entrant" software producers in the United States and Japan. Key objectives were to understand the processes used and see whether differences exist between the groups. For the study, process encompasses the formal lifecycle stages of requirements through maintenance, project management, metrics, configuration management, process ownership, and process improvement. "Established" producers turned out to be IBM Federal Systems Company, IBM Application Business Systems, Fujitsu, and Hewlett-Packard Commercial Systems and Open Systems Software Divisions. "New Entrant" producers turned out to be Microsoft and Lotus.

The results show significant commonality between the two groups, with an evolution occurring within each. "Established" producers have well-defined and relatively predictable processes, but they are seeking ways to foster increased individual creativity. "New Entrants" are moving from an environment of extreme creativity and are implementing process steps to help development achieve more predictable results. Each appears to be seeking a middle ground that benefits from both structure and creativity.

Thesis Supervisor:	Michael A. Cusumano
Title:	Associate Professor of Management

Table of Contents

Introduction

Introduction

Writing software to operate commercially available programmable computers began in the 1950's. Over the last 40 years, software has become a dominant part of both high and low scale technology products. As a result, software development has become a very significant worldwide business. It has been estimated that expenditures for software development and maintenance in the U.S. were $70 billion in 1985[1]. Projections put software costs growing to over $225 billion in the U.S. and $450 billion worldwide by 1995[2].

Developing reliable and usable software that is delivered on time and within budget is a difficult endeavor for many software organizations[3]. The creation of software has both intrigued and plagued engineers, managers, and customers since the beginning of the computer era[4]. The standard development process consists of requirements specification, design, implementation, testing, installation, and maintenance. Even though these phases are expected to be sequential they frequently become more iterative in nature. They are also unpredictable in time and costs, because the productivity of individual programmers tends to vary enormously and depend on elements difficult for management to control, such as personal talent and experience. The projects that succeed generally come about through the heroic efforts of a dedicated individual or team, rather than through repeating the proven methods of an organization with a mature software process. In the absence of an organization-wide software process, repeating results depends entirely on having the same individuals available for the next project[5].

Software producers may thus encounter budget and schedule overruns as the rule rather than the exception, especially when attempting to build large complex systems with many components for the first time[6]. The sheer difficulty of software design and programming, exacerbated by a demand for programs rising faster than the supply of software engineers, led to a situation referred to as the "software crisis" as long ago as 1969[7]. As software projects continue to increase in size and importance, these problems have become magnified. Products that are late, over budget, or that don't work as expected cause significant problems for the software organization's customers.

As the need for software has increased, there has been much study of ways to improve the speed in which software is developed, the costs associated with developing new software, and the quality of the resulting software. Significant advances have occurred in the areas of tools for software development, project management techniques, design techniques, software development techniques, and testing methods. Despite all these advances, problems remain and the industry is continuing to search for new ways to develop software more productively, with higher levels of quality and with higher user satisfaction. After two decades of unfulfilled promises about productivity and quality gains from applying new software methods and technologies, industry and government organizations are realizing that their fundamental problem is the inability to manage the software process[8].

Throughout much of the early industry years, and still today for many people, software development was considered to be more of an art than a science. This school of thought tends to discourage any efforts to standardize the process of software development, and in fact claims that it cannot be standardized without greatly diminishing its value. "Software artists" should give free reign to their creativity and not stifle it with disciplined procedures, corporate policies, or standardized methods, according to this view. But another view, which is gaining in popularity as demand for software increases and as historical results have shown continued fundamental problems, states that there are some elements of science in the software development process. From this perspective, software development can be improved upon using techniques and practices borrowed from manufacturing science[9].

The increased amount of competition in the software industry today, combined with the shortened life cycle of individual products, places considerable time-to-market pressure on software developers. With a shortage of software engineers in the marketplace, companies are being forced to improve development productivity in order to win the time-to-market race. At the same time, the complexity of software has grown enormously, with some of the most complex programs now totalling several million lines of code. This greater complexity increases the number of man-years required to produce the code. Again, the only way to compensate for increased demands on programmers' time is to improve their level of productivity[10]. The organization must provide the infrastructure and support necessary to help projects avoid these problems.

Thesis Structure

My thesis study has concentrated on the software development process variations being used by a set of the leading software producers today. The initial focus of my thesis research objective was to understand variations in the software development lifecycle steps employed by the different companies. I was interested in the combination of requirements, design, implementation, and testing stages along with the details of how these phases were actually carried out.

The study of lifecycle phases has remained a part of my work and defines the base used to look at software development in the companies. But in the end, it became only a portion of the analysis of how development was carried out. Previous research and writings on the subject of the software development lifecycles showed a shift in focus away from just the lifecycle stages. Process support and the business management activities utilized appear to be the sources of differentiation between the companies. Looking at tools like the SEI Capability Maturity Model for Software along with many of the software quality and productivity writings, "development support" process steps have become the roadmap recommended for process improvement. Process characteristics like process control, metrics, configuration management, process ownership, and process improvement were the focus and have come about due to problems not solved within the steps of the software lifecycles.

Chapter 1 concentrates on the basic phases used in some variation for all software development. The "waterfall" process defined by Royce in 1970 is considered to be the first well-defined development process and is the base most follow-on processes have been formed on. I describe it so that there is a base process to start from. The process is generally considered to have been first utilized on a large scale project by IBM while developing the System/360 operating system in the 1960's. It represents a structured process and organization created for the purpose of developing software.

Even though these processes have been defined and available for many years, there are still a significant set of problems that exist either due to lack of process usage or due to deficiencies in the processes utilized. Chapter 2 describes what are considered to be the key problems associated with software development and the processes utilized to develop it.

After the overview of the processes and the problems of software development, I move to the case studies of a set of the leading worldwide software development organizations. The first group of these is classified as "established software producers". These organizations are part of companies with long histories of large scale software development. IBM organizations were chosen for the study due to their wide recognition as the originator of the structured development process and the significant amount of work that has been done within the company to further refine the classic structured development process. The group includes:

Chapter 3 - IBM Federal Systems Company in Houston, Texas that is responsible for development of software for NASA space programs. This particular IBM group was chosen due to their recognition as having the best software development process in the world.

Chapter 4 - IBM Applications Business Systems in Rochester, Minnesota that is responsible for development of the OS/400 Operating System and program products for the IBM AS/400 computer. This IBM group was chosen because they are developing non-mainframe products, they have done significant work to adapt the structured development process to a rapidly changing commercial product development environment, and they are recognized for outstanding quality.

Chapter 5 - Fujitsu Computer Systems Group - Numazu Works Software Division in Numazu, Japan that is responsible for development of operating systems, compilers, data base systems, and image systems for Fujitsu computers. This group was chosen due to their position as the leading manufacturer of hardware designed to match IBM capability, the fact they widely followed the IBM structured development process, and due to the factory like processes they have evolved to, as described in Michael

Cusumano's book, "Japan's Software Factories: A Challenge to US Management".

The next group of companies is classified as "new entrant software producers". These companies entered the business of large scale software development later than the set of "established" companies. The "waterfall process" had been defined by the time they entered and many lessons had been learned in the industry. The group includes:

Chapter 6 - Hewlett-Packard Commercial Systems Division and Open Systems Software Division that are responsible for the MPE/iX and HP-UX operating systems, respectively. This group was chosen for the study because of the success of their products in a wide set of commercial computer markets, their recognition for high quality products, and the extensive work they have done on metrics and process improvement.

Chapter 7 - Microsoft which has produced MS-DOS, Microsoft Windows, Microsoft Excel, Microsoft Word, and a substantial additional set of system and application software products for PCs. The company was chosen because of their position as the leading producer of PC software that covers both systems and applications software. Their reputation for innovative software solutions was also a factor leading to their inclusion in the study.

Chapter 8 - Lotus which has produced leading PC software products such as Lotus 1-2-3 and Lotus Notes. This company was chosen because of their position as another of the PC software leaders and because of their reputation of being a "more marketing oriented PC software company".

The conclusion looks at whether there are any real differences in the processes employed by "established" and "new entrant" software providers. Process discoveries worth noting again are also highlighted in the final section.

The process and problem sections are intended to provide a framework with which to look at the case studies. It is the actual case studies that are the focus on my thesis. By looking at what leading companies are doing in software development, we can see what is actually happening versus guessing the processes leaders are employing. This knowledge can be helpful as companies implement new processes or decide on process improvement actions to undertake.

Chapter 1: Software Development Processes

The Life Cycle Model of Software Development[11]

A development process is a set of activities, together with an ordering relationship between activities, which if performed in a manner that satisfies the ordering relationship will produce the desired product. A process model is an abstract representation of a development process.

In a software development effort, the goal is to produce high quality software. The development process is, therefore, the sequence of activities that will produce such software. We have seen that the basic phases are requirements analysis, design, coding, and testing, which are usually further broken down into distinct activities. A software development process model specifies how these activities are organized in the entire software development effort.

The purpose of specifying a development process model is to suggest an overall process for developing software. This could be different from the actual process that is employed by any one company or group of people. Existing development processes can also be modeled, and such an exercise is useful for determining the weaknesses in existing practices.

Problem solving in software must consist of these activities -- understanding and defining the specific nature of the problem, creating a technical and development plan for a solution, coding the planned solution, and finally testing the actual program. For any software system of a non-trivial nature, each of the four activities for problem solving should be done formally. For large systems, each activity can be extremely complex, and methodologies and procedures are needed to perform it efficiently and correctly. Each of these activities is a major task for large software projects. Furthermore, each of the basic activities itself may be so large that it cannot be handled in a single step and must be broken into smaller steps. The basic Life Cycle Model activities or phases to be performed for developing a software system are:

Requirements Specification
Design
Implementation
Testing

The steps following development are:

Installation
Maintenance

Most software development today follows the Life Cycle Model. The sequence of stages required to complete a software project have become standard, although a wide range of options exists within each stage. The stages and purpose of each within the Life Cycle Model is described below.

Requirements Specification: Requirements analysis is done in order to understand the problem which the software system is to solve. The overall software requirements are specified and analyzed with the technical objective of establishing a complete and consistent set of requirements to perform software design. Customers and hardware designers are the groups that most frequently define the end product requirements to the individuals responsible for turning them into the set of requirements needed for the software design step. The definition of capabilities should include both functional (i.e. testable) and non-functional (qualitative) requirements. The requirements specification describes what the system will do, but not necessarily how it will be done. After the requirements specification has been written, software specialists will be able to decide whether or not the system is technically feasible.

Design: The purpose of the design phase is to plan a solution for the problem specified by the requirements document. Software designers determine how the system can be structured to meet the requirements outlined in the requirements specification. The design of a system breaks down each functional requirement into modules and submodules, describing each module and detailing its interactions with the rest of the system. In this stage, software designers strive to create a design that will maximize efficiency of the hardware, maintainability of the system, and ease of implementation (coding). In addition, they generally try to minimize complexity in the design. Sufficient detail is required to allow the coding of the actual programs to begin.

Implementation: During the implementation stage, each module must be coded. Coding involves writing the instructions necessary to achieve the program design in accordance with the design specification. The language used for coding should be determined up-front, and will depend upon the functionality and complexity of the system. Languages and tools used to complete this process are often standard for the organization to ensure programs work with each other in the final system. During this phase, the developer will often do a level of testing of the system as a verification of the program logic. Testing may also be carried out between small groups of modules before they are included in the full new system.

Testing: The testing phase is considered a formal verification of the system being delivered. Two objectives tend to be part of this phase: ensuring that all of the programs work together without error, and ensuring that the overall system requirements are met. The testing stage should uncover and correct any bugs that exist in the system due to requirement, design, or coding errors in the programs. Testing is generally performed at each base level of functionality during the implementation stage (i.e. as each module is finished), and at the system level. To ensure requirements are met, some level of customer involvement is often utilized as a step (usually a final one) of the test process. As in all testing for product errors, there is a trade-off between the user's need for assurance of quality and the producer's need to economize on time and cost.

Installation: This is the phase when the new system is formally combined with the existing system. The best mental picture of this is the delivery of a new set of functions to a customer with a base system already installed. During this phase, a set of regression tests is often run to ensure existing functions are not affected by the new system, and to ensure the new system runs correctly when combined with the existing system. It is during this phase that the end customer will determine and document that the final product was delivered and that it satisfied the customer requirements.

Maintenance: This is the phase in which any problems encountered by the customer are handled. Errors that can occur are: software errors not discovered during the system test, failures in other parts of the system that can be fixed by programming changes, and changing requirements for use of the system. Fixes due to mistakes made by the programmers are usually made as updates to the existing code and provided to the customers. Fixes to handle failures in another part of the system and changes due to changed requirements usually involve a formal process of approving the change requested along with a method of delivering these more extensive software changes.

The Life Cycle Model is widely used as a guide for software development. Depending upon the complexity of the system, the time-to-market pressures, and other situational factors, development processes may combine several stages or even skip certain ones. For example, if a simple program is to be written under tight deadlines, it might be easier to start with low-level design and coding and skip the requirements specification. In environments where system quality and accuracy are essential, testing may be performed at every stage, beginning with the requirements document. In companies where there is a shortage of programmers, it might make sense to combine design and coding into a single process. Prototyping is another response to rapidly changing requirements and the need to continuously demonstrate and validate products with customers. Evolutionary software prototype processes are coming into place that use various stages of prototypes through the development stages of requirements, specifications, design, and finally coding. There are other variations on the model that have proven effective in different environments.

Cost Breakdown of Life Cycle Phases[12]

For software, the cost of development is the cost incurred during requirements specification, design, implementation and testing which all occur before product delivery. The cost of maintenance is the cost of modifying the software due to defects in the software or updates of the software. This cost is spread over the operational years of the software.

It is generally accepted that the cost of maintenance is likely to be higher than the development cost. The ratio of development cost to maintenance cost has been estimated to be in the range of 30 - 40% development and 60 - 70% maintenance. This is significant considering that during most development projects the development organization is most concerned with reducing the development cost, and are not as concerned with maintenance cost. Since maintenance critically depends on the software characteristics that are decided during development, maintenance costs can be reduced if maintenance concerns are carefully addressed during development.

Of the development cost, a typical distribution of effort within the different phases is:

Requirements	10%
Design	20%
Implementation	20%
Testing	50%

The exact cost ratios will vary by organization and type of product being developed. They are reasonable enough to use as a base for making observations on the lifecycle stages. One observation is that maintenance and testing dominate the resource usage for a project. Efforts to improve requirement, design, and implementation work can provide significant benefits to testing and maintenance. Knowledge of the significant costs of fixing requirement and design defects late in the development process suggests the need for well planned and executed work in these up-front phases.

These ratios show likely improvement over what the situation would be without some form of structured process model for the lifecycle. Lifecycles that concentrate on doing more structured work early in the development process phases help avoid the need to test in quality and fix significant amounts of defects after the software has gone out the door. Structured lifecycles appear to make economic sense for companies developing software.

Waterfall Life Cycle Model for Software Development[13]

Credit for the introduction of a lifecycle for use in software development is generally attributed to Royce[14]. The model introduced by Royce was termed the Waterfall model and embodied the systems engineering approach to building large-scale systems that had already been in existence for some time. The waterfall image is meant to signify a linear flow from the beginning phase of the process until the final destination of the operations phase. Since its initial use by Royce, many modifications and iterations have been made to the model. The use of the conventional (waterfall) lifecycle has demonstrated that better software will result from the careful use of a software development lifecycle.

In a very ideal situation, each of the lifecycle phases could be carried out individually to completion and in sequence. When all the phases have been executed, the software product is delivered and works as intended by the original user or client, in accordance with the initial need or user requirements for the system. Actually, this rarely happens. What usually happens is that one or more phases are often repeated, after deficiencies are found, until the system is correct or nearly so.

Iteration and interaction are the key factors to be attended to in order to produce a successful software product. It is better to "do it right the first time" though, since it is often very expensive to accomplish major iteration, reiteration, and interaction due to much recycling through the lifecycle phases.

The initial waterfall lifecycle model has undergone numerous refinements. Many versions of the software development lifecycle models appear in the literature, and almost all of these define five to seven-phases for systems engineering of the software development process. These later models have attempted to decrease the rigid boundaries of the phases, increase overall flexibility to account for iteration, and to introduce automation tool usage. Each of the models are generally based on the phases introduced with the waterfall model.

In the Waterfall model [See Figure 1], Royce defines the 6 phases as: System Requirements, Software Requirements, Preliminary Program Design, Program Design, Coding, Testing, Operations. In addition, he includes an Analysis step between Preliminary Program Design and Program Design. What follows is a brief description of each phase along with the documentation that is generated during the phase. Royce was very clear in his position that documentation is absolutely critical.

System Requirements: In implementing this phase, it is assumed that the system user and the systems analyst are sufficiently informed about what the new (or modified) system is intended to achieve. This knowledge is used to develop the system-level requirements to an acceptable level such that they can be defined in sufficiently complete detail for preliminary design to be initiated. All of this must be done before detailed design and coding can be initiated.

Software Requirements: The development of the software requirements phase focuses on the outcomes of the system requirements identification carried out in phase 1 of the waterfall model of the software development lifecycle. It is concerned with the nature and style of the software to be developed, the data and information that will be required and the associated structural framework, the required functionality, performance, and various interfaces. Requirements for both the system and the software are reviewed for consistency and then reviewed by the user to be certain that the software requirements faithfully interpret and produce the system requirements. A Software Requirements Document is produced to document the results of this phase. It becomes a technology and management guideline that is used throughout all subsequent development phases.

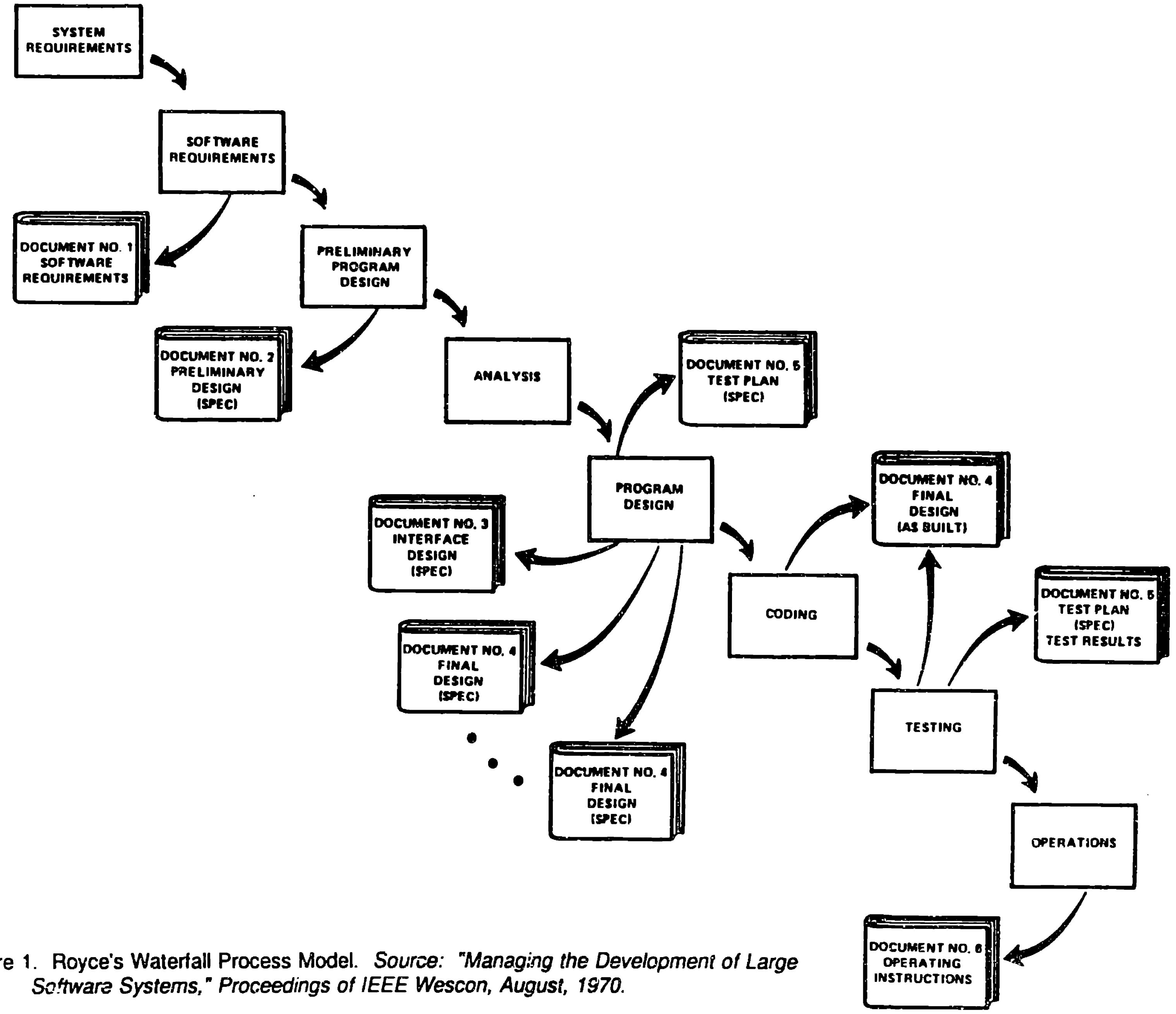
SYSTEM REQUIREMENTS
SOFTWARE REQUIREMENTS
DOCUMENT NO. 1 SOFTWARE REQUIREMENTS
PRELIMINARY PROGRAM DESIGN
DOCUMENT NO. 2 PRELIMINARY DESIGN (SPEC)
ANALYSIS
DOCUMENT NO. 5 TEST PLAN (SPEC)
PROGRAM DESIGN
DOCUMENT NO. 3 INTERFACE DESIGN (SPEC)
DOCUMENT NO. 4 FINAL DESIGN (SPEC)
DOCUMENT NO. 4 FINAL DESIGN (SPEC)
CODING
DOCUMENT NO. 4 FINAL DESIGN (AS BUILT)
TESTING
DOCUMENT NO. 5 TEST PLAN (SPEC) TEST RESULTS
OPERATIONS
DOCUMENT NO. 6 OPERATING INSTRUCTIONS

Figure 1. Royce's Waterfall Process Model. *Source: "Managing the Development of Large Software Systems," Proceedings of IEEE Wescon, August, 1970.*

Preliminary Program Design: The software requirements are converted into a preliminary software product design which is aimed primarily at the further interpretation of the software requirements in terms of software system-level architecture. The product of this phase is an identification and micro-level definition of the data structure, software architecture, and procedural activities that must be carried out in the next phase. Preliminary software design involves representing the functions of each software system in a way that these may be readily converted to a detailed design. The results provide insight as to how the system is intended to work at a structural level and satisfy the technological system requirements. The design is documented in a Preliminary Design Specification.

Program Design: This phase involves definition of the program modules and intermodular interfaces that are necessary in preparation for the writing of code. Specific reference is made to data formats, detailed description of algorithms, and tracing of all inputs to and outputs from the detailed design modules. Two design specifications are generated during this phase. The Interface Design Specification documents the user interfaces and program interfaces for each program module. The Final Design Specification is the key output of this phase and may undergo multiple reviews and revisions depending on the number of design steps in this phase. During this phase, the testing group begins work on the Test Plan Specification.

Coding: During the coding phase, the detailed design is translated into machine-readable form. This is either a manual activity by a programmer or can involve some machine generated code based on the structure of the detailed design. Initial testing by the developer happens during this stage and generally bugs are discovered Debugging and recoding takes place to validate the integrity of the overall coding operations of this phase.

Testing: The individual units or programs are integrated and tested as a complete system to ensure that the requirements specifications are met. Testing procedures center on the logical function of the software. They assure that all statements have been tested, and that all inputs and outputs operate properly. After system testing, the software is frequently operated under controlled conditions to verify and validate the entire package in terms of satisfying the identified system specifications and software specifications.

Operations: This phase is often the longest from the perspective of the entire useful life of the software product. The system is installed at the user location, tested, and then used under actual operating conditions to the maximum extent possible. Maintenance commences immediately upon detection of any errors that were not found during the earlier phases. Maintenance is primarily the process of improving the system to accommodate new or different requirements as defined by the user.

Waterfall Process Implications

Across the phases, there are important consequences of the linear nature of activities. Certification should be done at the end of each phase to mark the completion of one phase and beginning of the next. Certification is accomplished through some form of verification and validation that ensures the output of the phase is consistent with its input, and the output of the phase is consistent with the overall requirements of the system.[15] The goal of each phase is to produce the output that can be certified. Reviews are necessary for certain key phases (requirements, design, and coding). Reviews are formal meetings to uncover deficiencies in a product.

Documents or code are the normal outputs of a phase. Outputs from one phase become the inputs for the next phase. The outputs that have been certified should not be changed or modified, though.[16] Reality says that requirement changes will happen and must be addressed in the process. Due to the changes, configuration control is needed to ensure changes are made in a controlled manner after evaluating the effect of each change on the product.

The major advantages from using the classical waterfall software development lifecycle model, or one of its many modifications, are those of organizing and controlling the software development project. The single most important methodological need associated with use of the waterfall lifecycle model is that of effective identification of user requirements.[17] In the usual lifecycle development, inadequate attention is paid to fulfilling this need. Overall, software development lifecycle models provide management and control over the processes and procedures to be accomplished in software development.

Disadvantages associated with use of a lifecycle model include problems that occur if there is no iteration and feedback among phases. Unless there is iteration and feedback, there may be no way to improve initial imperfections in one of the phases. Realistic lifecycle development processes are iterative and interactive, and the software development lifecycle is one that encourages management to recognize these facts of life in order to be successful in guiding software development projects.[18]

Chapter 2: Software Development Problems

Problems with Software Development

A software crisis was initially discussed 25 years ago. The software industry was relatively new at the time, but already had a track record of software projects being delivered significantly behind schedule, costs well above budgeted amounts, quality that was poor, and maintenance costs already beginning to mount. As more complex systems and applications were found, software developers fell further behind schedule and quality became poorer as they attempted to catch up. Software costs continued to rise at a very steep rate.

Through the years since that period, the industry has continued to be in a level of software crisis. Knowledge and tools have advanced, but the problems persist. The reality even today is that there are many difficulties associated with the production of reliable, cost effective software. In their 1990 book *Software Systems Engineering*, Andrew Sage and James Palmer created a list of attributes associated with software today. The result of a very large number of studies showed that:

Software is expensive
Software capability is less than promised and expected
Software deliveries are often quite late

Software cost over runs often occur and are generally large
Software maintenance is complex and error-prone
Software documentation is inappropriate and inadequate
Software is often cumbersome to use and system design for human interaction is lacking
Software products often cannot be integrated
Software often cannot be transitioned to a new environment or modified to meet evolving needs
Software performance is often unreliable
Software often does not perform according to specifications
System and software requirements often do not adequately capture user needs

Because of this set of central problems, they point out that the following difficulties arise: inconsistent, incomplete, and otherwise imperfect system requirements specifications; system requirements that do not provide for change as user needs evolve over time; and poorly defined management structures for product design and delivery. The products that come from this environment are frequently difficult to use, do not solve the intended problem, operate in an unreliable fashion, are nearly impossible to maintain, and are not used in a number of cases. These same studies generally show that the major problems associated with the production of trustworthy software are more concerned with the organization and management of complexity than with direct technological concerns that affect individual programmer productivity.

The Uniqueness of Software

Frederick P. Brooks, Jr. has done some of the most intriguing and entertaining writing on the subject of software development and the difficulties associated with it. In his book, *The Mythical Man-Month*,[19] Brooks stated his belief that problems associated with software development were interrelated, difficult to isolate, and inherently systematic. Regarding the techniques to estimate and to monitor schedule progress, he felt they were so poorly developed that they were almost "mythical".

It was in Brooks' article, "No Silver Bullet - Essence and Accidents of Software Engineering"[20], that he summarized key properties of software that made it difficult to build. He considered the software entity to be a construct of interlocking concepts: data sets, relationships among data items, algorithms, and invocations of functions. The resulting product is abstract in nature, and such a conceptual construct can be represented in multiple different ways (no single way to implement it). Even though it can be done multiple ways, the software product is nonetheless highly precise and richly detailed. Since the hard part of building software is the specification, design, and testing of this conceptual construct versus the labor of coding and verifying the code, building software will always be hard. The inherent properties of modern software systems that make them irreducibly difficult to build are: complexity, conformity, changeability, and invisibility.

Complexity. Software entities are more complex for their size than perhaps any other human construct because no two parts are alike. A scaling-up of a software entity is not merely a repetition of the same elements in larger sizes, it is an increase in the number of different elements. In most cases, the elements interact with each other in some nonlinear fashion, and the complexity of the whole increases much more than linearly. The complexity of software is an essential property, not an accidental one. Many of the classic problems of developing software products derive from this essential complexity and its nonlinear increases with size. Not only technical problems, but management problems as well come from the complexity.

Conformity. Much complexity comes from conforming to other interfaces; this complexity cannot be simplified out by any redesign of the software alone. The complexity is arbitrary complexity, forced on the software developer without rhyme or reason by the systems which his/her interfaces must conform. These interfaces differ because they were designed by different people.

Changeability. The software entity is constantly subject to pressures for change. The software product is embedded in a cultural matrix of:

applications:	it embodies the function of the system and feels the pressure for change;
users:	people try to use it in different ways beyond the original domain;
laws:	easier to change software than hardware;
machine vehicles:	successful software survives beyond the normal life of the machine it was first written for.

These all change continually, and their changes force change upon the software product.

Invisibility. In spite of progress in restricting and simplifying the structures of software, they remain inherently unvisualizable, and thus do not permit the mind to use some of its most powerful conceptual tools. This lack not only impedes the process of design within one mind, it severely hinders communication among minds.

Software Problems are Not "New News"

The difficulties associated with software development and the limitations of the software products created is not a new discovery. As mentioned earlier, it was 25 years ago that the "software crisis" was first identified. One of the most comprehensive reports on the state of software engineering is a 1968 NATO Science Committee conference report. Conference participants agreed that software difficulties started with the determination of system requirements, which initiated a cycle of uncertainty that rippled through the development of the product. Cost and schedule estimates invariably were changed. Programming modifications made to account for changed requirements affected configuration, testing, documentation, and maintenance. When this involved large projects, things like programming productivity variations, poor communications within projects, lack of tools, poor measurement standards, rapid growth in demand and size of programs, and few reusable components resulted in tremendous maintenance costs when the systems invariably required changes.

The resulting "NATO Report on Software Engineering Problems" that came out of the conference is included in a form presented in Michael Cusumano's book "Japan's Software Factories". It provides an important reference in understanding where the software industry has progressed from.

NATO Report on Software Engineering Problems (1968)

Lack of understanding of system requirements on the part of customers and designers.
Large gaps between estimates of costs and time with actual expenditures due to poor estimating techniques, failure to allow time for changes in requirements, and division of programming tasks into blocks before the divisions of the system are well-enough understood to do this properly.
Large variations, as much as 26:1 in one study, in programmers' productivity levels.
Difficulty of dividing labor between design and production (coding), since design-type decisions must still be made in coding.
Difficulty in monitoring progress in a software project, since "program construction is not always a simple progression in which each act of assembly represents a distinct forward step."
Rapid growth in size of software systems.
Poor communication among groups working on the same project, exacerbated by too much uncoordinated or unnecessary information, and a lack of automation to handle necessary information.
Large expense of developing on-line production control tools.
Difficulty in measuring key aspects of programmer and system performance.
A tradition among software developers of not writing systems "for practical use," but trying to write new and better systems, so that they are always combining research, development, and production in a single project, which then makes it difficult to predict and manage.
Rapid growth in the need for programmers and insufficient numbers of adequately trained and skilled programmers.
Difficulty of achieving sufficient reliability (reduced errors and error tolerance) in large software systems.
Dependence of software on hardware, which makes standardization of software difficult across different machines.
Lack of inventories of reusable software components to aid in the building of new programs.
Software maintenance costs often exceeding the cost of the original system development.

Source: Michael Cusumano, "Japan's Software Factories" (New York: Oxford University Press, 1991).

The Situation in the 1970's

From that point in 1968, the industry grew at a very rapid rate. In 1979, Richard Thayer published the results of another snapshot taken of the industry at that time. He combined the survey results of a large group of industry practitioners along with the results of 60 software projects from the aerospace industry. The result was a smaller list of 7 key issues managers of software development were most concerned about. Even though the list was shorter than the NATO results, it indicated that the overall position of the industry had not progressed significantly since 1968. The significant issues were virtually the same. The list is:

1) Incomplete, ambiguous, inconsistent, or unmeasurable requirements specifications
2) Poor project planning
3) Poor cost estimation techniques
4) Poor scheduling estimation techniques
5) Lack of decision rules to help in selecting the correct design techniques, equipment, and other design aids
6) Lack of decision rules to help in selecting the correct procedures, strategies, and tools for software testing
7) Lack of standards and techniques for measuring the quality of performance and the quantity of production expected from individuals

To further highlight the state of the industry during this period, an article by Barry W. Boehm listed some of the problems associated with the different phases of the software lifecycle along with the "frontier" solutions he viewed as being viable for the phases. His conclusion at that time was, "There are more opportunities for improving software productivity and quality in the area of management than anywhere else." The following table is a partial reproduction of his results that includes only the "Development Step" and the "General Problems" of each phase:

Development Step	General Problems
Requirements specifications	Ambiguous, errors, ad hoc
Design	Manual, bottom-up, error-prone
Implementation (Coding)	Unstructured code
Testing	Unplanned, much wasted effort
Maintenance	Unsystematic, costly
Project management	Poor planning, control, resource estimation, accountability, and success criteria

Source: Barry W. Boehm, "Software Engineering," IEEE Transactions on Computers C-25, 12, December 1976.

Progress in the 1980's

In 1984, another snapshot of the industry was taken. By this time, nearly a decade had passed since Boehm's work and over 5 years had passed since Thayer's survey results were compiled. C.V. Ramamoorthy led a group that looked at progress that had been made in software engineering since the software crisis was identified in 1968. The group looked at past accomplishments, the current set of significant problems, and future needs in the important aspects of software engineering. The key advances noted were:

Formal specification languages and methodologies to write requirements specifications. This simplified the writing of specifications and made it easier to analyze them for internal consistency and accuracy.

Graphic representations of specifications and executable specifications also came about to a limited extent.

Research had moved forward on specifications languages as well as program generators that automatically produced code from requirements.

Tools that automated aspects of test-data generation, error location, and results verification.

Some improvement in tools that analyzed source code and generated information such as control flow and data flow.

Metrics improved for measurement of phase productivity, cost, and quality. Improvement was also seen in quality measures like correctness, modifiability, reliability, maintainability. For productivity, estimation models had improved.

While advances occurred, there were still a set of problems that remained. These problems continued to be in the general areas noted in the 1968 NATO report. They show evidence of progress in that each problem had progressed to more specific elements of design, metrics, and other categories versus the previous global nature of the problems. The summarized list is:

The design process remained difficult to rationalize. Designers used one set of criteria (such as functionality, dataflow, or data structures) but then had to account for a range of nonfunctional requirements (maximizing maintenance, reliability, speed, reusability, or memory constraints) as they partitioned the resulting procedures into modules. The final step required design of actual data structures and algorithms, and transcription into text or graphic forms. The problem was no design methodologies existed for handling complex control structures, real-time applications, or new distributed processing requirements.

Challenge of how to select the fewest number of cases that would verify a program is logically correct and meets user specifications. Minimizing test items remained critical to reducing the huge amounts of time and labor normally required to evaluate large, complex systems.

Insufficient documentation; inconsistency between documents and code; designs difficult to understand, modify, and test; absence of good records on past maintenance are all problems contributing to high maintenance costs.

Metrics had improved greatly but still needed improvement for effective use in design and quality control.

Performance of individuals remained difficult to measure.

Prototyping began to be used more but they were not yet efficient to build and tended to be "throw away" when going to actual code.

1990's View from the Case Study Organizations

All previous information on the problems associated with software development deal with broad industry surveys of key problems during multiple different time periods. This information is very useful in understanding the historical progression of the overall industry. During interviews with the companies that make up my study, I had the opportunity to discuss the key problems each company was working to manage/overcome through their development processes today. It can be viewed as a 1993 snapshot from an elite set of software producers.

The IBM Federal Systems Company group responsible for developing the software for NASA space systems faced the problem of creating software that peoples' lives directly depended on. With lives hanging in the balance on each manned space mission, the key problem to overcome was code defects. Zero defects became the only acceptable criteria. To deal with this situation, the whole development process is centered on eliminating situations where defects are injected into the code along with removing any defects that may have come from previous stages. Dealing with the creative (and human) act of developing new and leading edge software functions while making sure it is perfect is the main problem faced by this group.

The IBM Application Business Systems group is dealing with the problem of responding to widely varied and quickly changing customer requirements while making sure delivery commitments are met. In 1978, a committed delivery date was missed and became a defining moment for Rochester. Since that time, processes and metrics have been continually improving to be able to predict schedules and quality early in the product development process. On top of that, the pace of customer requirements has rapidly increased to the point where existing processes of translating customer requirements to development plans are seriously strained. Since changing requirements ripple through the development process and are exceedingly expensive, the group is faced with making improvements in this process step. Solving this problem is critical to the on-going need of reducing customer delivered defects and reducing development costs.

The Fujitsu Computer Systems Group has been dealing with the need to put methods/processes in place to create quality software using people with limited computer science training. Their objective was to break the development process into small repeatable steps that workers could gain significant experience with and become masters of. By mastering each of the steps, the overall process could be the most efficient possible. Consistency and repeatability would also lead to quality improvements. Automation has also been a strong area of investment with the desire to reduce human interaction/unpredictability in the process.

Hewlett-Packard has been a company that possesses a strong culture of engineering excellence. Software developers are referred to as software engineers. The group is very analytical and drives for perfection, but attempts to maintain an atmosphere of freedom to pursue invention. The freedom made it very difficult to implement structured processes. The most significant turning point came when Hewlett-Packard's Commercial Systems Division did an analysis of the causes of the software defects being received from customers. The results indicated that a significant number of errors were being injected during the coding phase and were not being removed with testing. The solution determined by the technical community was that code reviews along with more rigor during the requirements and design stages could eliminate defect injection. The results have been improvements in quality and cycle time during product development. The key problem that Hewlett-Packard faced (and still faces) is maintaining sufficient structure in the development process to hold defects low, while keeping a creative edge.

Microsoft was faced with the need to become more predictable in product schedules. The company was unable to reliably predict and feel comfortable with development schedules. Product delays were common and when the customer base continued to expand, this became unacceptable. On top of this, customers have begun to express the need to feel comfortable that their suppliers (Microsoft for PC software) are utilizing quality processes to create products. The challenge within the company has been to implement more structured development processes while maintaining the creative technical edge that characterizes the company.

Lotus is faced with the delicate balancing act of process versus creativity and flexibility. At one point, they dictated a development process which failed due to the variations needed by individual products and due to programmers not wanting that level of structure. Their major problem is coordination and management (for delivery and quality) when projects grow beyond the small team level. Mechanisms are not in-place to effectively manage development in large groups.

Summary

To summarize the chapter, I look to a list compiled by Dick Sulack on the root causes of "runaway" and "missed objectives" projects. Runaway projects are characterized by significant overruns of schedule, resources, and/or funding, while missed objectives are projects that do not meet customer expectations in some significant way. This list was compiled in 1993 and represents an up-to-date view of the problems that still persist, even for some of the leading software producers. As you read the case studies that follow, you will see process steps that directly deal with many of these problems. The companies recognize the problem areas and work to manage them. In most cases, the steps were put in place due to some significant project mistake that has happened in the company's past.

Root Causes of Runaway & Missed Objectives Projects

1) Inadequate requirements statement
2) Lack of specific and measurable goals
3) Architecture/design flaws and churn
4) Inadequate change control system
5) Inadequate project status reviews and reporting
6) Inadequate project metrics
7) Lack of open project communications
8) Lack of project milestones
9) Optimistic viewpoint of project doability
10) "Management difficulties"

Source: Richard A. Sulack, ASEMTECH, INC., "Advanced Software Engineering Management Core Competencies, presentation at Spring 1993 COMMON Meeting.

Chapter 3: IBM - Federal Systems Company, Houston[21]

The Company

Thomas J. Watson, Sr. is really the beginning of IBM. He was recruited in 1914 to be General Manager of Computing-Tabulating-Recording Company, CTR. It was a company of 1200 employees that manufactured and sold a mix of scales, time clocks, and tabulating machines. He focused the company on tabulating machines and drove forward with the vision. His vision was that clerks and accountants were growing faster than the rest of American business, and that information could be made of interchangeable parts that could be encoded, available, reclassifiable, and printable by a machine. He continually led refinement of the company's tabulating machines. In 1924, he became CEO of the company and renamed it International Business Machines. The introduction of the IBM System/360 mainframe computer in 1964 ushered in the computer industry that IBM rode to its current position as leading computer company in the world.

Corporate headquarters are in Armonk, New York. Total employment by IBM and wholly owned subsidiaries is approximately 300,000 people with operations in most countries throughout the world. Research and Development is concentrated in the United States with additional development locations in Canada, Europe and Japan. Manufacturing is done in the United States, Mexico, Europe, and Japan with each business unit controlling their own manufacturing operations.

IBM's business spans nearly all aspects of the computing world. The company develops and sells hardware that includes: mainframes, minicomputers, personal computers, workstations, network servers, disk drives, microprocessors, memory chips, computer terminals, printers, and other peripheral devices. The company is also the leading total software producer in the world with software that includes: operating systems, systems software like compilers and data bases, communications software, contracted software, and applications software as the main groupings. Finally, services make up an increasingly larger part of the company's business.

IBM Organization

IBM is a federati^n of companies. Each company is working through the process of decentralizing and increasing its competitiveness in diverse, fast-moving industries. Each business is driving to become competitive and sufficiently independent so that the possibility of separate ownership can be evaluated against the tests of business logic and the prospects for attracting and rewarding shareholders. This is a radical shift from the centralized and heavily coordinated corporation IBM was.

IBM's primary operating units include: geographic units that provide marketing, services, and support to their customers; and manufacturing and development businesses that have worldwide responsibility for product development, manufacturing, and delivery to geographic marketing units and to other equipment manufacturers.

The geographic marketing and services units include IBM Asia Pacific, IBM Europe/Middle East/Africa, IBM Latin America, and IBM North America. The business units are: Application Business Systems, Application Solutions, Enterprise Systems, Networking Systems, Pennant Systems Company, Personal Systems, Programming Systems, ADSTAR, and Technology Products. Two additional parts of the company are: Research Division, and IBM Credit Corporation.

Federal Systems Company was formed in 1992, combining Federal Systems Marketing and Federal Systems groups from within IBM. The organization is in the forefront of the move to turn IBM into a network of more competitive businesses. In the 1992 IBM Annual Report, the results of this group were highlighted in their own section. Federal Systems markets specialized products and services to the defense, space, and other federal agencies of the United States and several governments beyond the United States. 1992 revenue was $2.191 billion which was an increase over 1991 revenue. A total of 10,798 employees work for the company.

The Federal Systems Company has an IBM Vice President and CEO that reports to the General Manager of IBM North America. The development group responsible for all space systems software development is located in Houston, Texas. Approximately 300 programmers, analysts, engineers and subcontractors generate the onboard and ground systems software, and assist in testing the product. The Space Shuttle Software organization is divided into three groups: Avionics Software Systems Engineering which contains the group of requirements analysts, Avionics Software Development which contains the programmers, and Avionics Software Verification which contains an independent group responsible for testing.

Culture

The culture of the shuttle software group in FSC is concentrated around the drive to zero defects and usage of a team approach to achieve the goal. In their technical environment, teams form freely to work on specific problems or refine methods and procedures. The group has a strong sense of camaraderie and a feeling that what they are doing is important. They realize that working together is the only way to make sure the IBM software is defect-free. The quality improvement process used is mature and focuses primarily on how to improve particular processes instead of just trying to raise general quality awareness.

Satisfying their customer is another feature of the culture. Development is based on three objectives that are all centered around delivering exactly what the customer desires. The three are: develop software which adheres to the letter of the customer's requirements (both explicit and implicit), ensure that the software performs in accordance with the customer's operational expectations for both nominal and off-nominal conditions, and provide software which is error-free.

Structured development is the final feature of the culture. Development is based on a reliability and quality assurance plan that has two key aspects: a software development process which emphasizes the development of error-free software and the early detection of any errors that do occur; and a series of tests, audits, and inspections which ensure that the software developed conforms to the requirements of the customer.

The group is very proud of their quality record and was chosen as a recipient of the first, and recently a second, NASA Excellence Award for Quality and Productivity for its success in producing very complex software for NASA's Johnson Space Center. The onboard flight software has never had an in-flight error that affected the safety of crew or vehicle, or prevented the successful completion of mission objectives. They have been successful in reaching the goal of zero product errors in the last 3 major releases of onboard flight software systems which have spanned more than 15 flights. Recognition and success reinforces the culture of the organization.

Product Description

IBM has been doing software development under contract to NASA for the space system for decades. With the completion of the Approach and Landing Tests in the late 1970's, focus shifted at IBM Houston to development of the remaining mission critical software. The software that remained was the "fly by wire" control for the space shuttle to support all phases of flight, from pre-launch to roll-out. To accomplish this, the organization was divided into the present groups of analysts, programmers, and independent verification. The development was segmented into three major divisions: system software that contains the operating system; Guidance, Navigation, and Control programs which fly the spacecraft in automatic mode; and Vehicle Cargo System which handles all shuttle operations in orbit apart from the flying of the spacecraft.

Specific functions provided by the software are: operating system to control the full hardware and software system, automatic guidance function, navigation and flight control functions, systems maintenance functions, payload interface functions, vehicle utility and checkout functions, servicing for the various hardware sensors and effectors that control vehicle operations, comprehensive set of overall system health and crew input commands, and system health monitors and alarms.

Since the software is "man rated", which means software errors could result in loss of life, every possible effort is expended to keep its error rate as low as possible with a goal of zero defects. Considering the size to the product delivered, this goal is very significant. The organization has delivered a significant amount of computer code to NASA over the years: total shuttle related software of 9,780,000 source lines of code, and total software across all space programs of 30,000,000 source lines of code.

Release Structure and Strategy

Software is delivered in Operational Increments (OIs) to the base software that is in-place and currently operational. This results in an incremental development and release process. The approach is used due to the size, complexity, and evolutionary nature of the orbiter system requirements. OIs occur on roughly 12 month schedules.

Contents of the OI are determined by NASA with IBM's review and concurrence based on resource availability. A very extensive requirements process, which will be described in detail, is utilized to create the agreed to contents. Each OI is used for several flights and serves as the foundation upon which the next OI is developed.

Development Process Overview

The software development techniques employed include process definition, state-of-the-art software engineering principles, rigorous inspection of work products across the process, independent software verification, sophisticated defect cause analysis, and use of a set of specialized tools to assist development and testing. The overall software life cycle consists of four basic phases: requirements definition; software design, development, and integration; independent verification; and customer support. Within these basic phases are multiple sub-processes that come together to create a very extensive and well-defined software development process.

Programming standards are documented and must be followed. Any deviation from baseline requirements requires approval by an IBM control board.

Adherence to standards is audited by inspection and review of work products from each development phase. Another aspect of the overall process is the emphasis on early error detection. It is stated that errors detected early in the process are cheaper to fix and are not delivered with the product. Each phase has steps in place to minimize "defect escapes".

Requirements Definition Phase

Requirements Definition is the area of the development process that is most significantly different between what IBM Houston does and what nearly all other software development groups likely do. Requirements definition and agreement receives significant attention and the results are very clear requirements. The single customer makes this both feasible and a very good practice. Key points that characterize the approach to requirements definition are: management commitment, educated personnel, well-defined process, belief that special focus on requirements is essential, belief that errors found here are cheapest to fix and that error insertion can be reduced, belief that formalization of requirements benefits all processes, and baseline control and release planning are necessary to protect against a chaotic development environment.

NASA controls the requirements which are driven by project goals, required flignt changes, and available resources. The IBM Requirements Analysis group is made up of programmers and engineers very familiar with the avionics system. They are responsible for assessing the feasibility of implementing the requirements into the hardware and software systems. An IBM Control Board utilizes the Change Requests (CRs) to get costing information from each development group and to determine a recommendation for each. Recommendations from the group include a schedule of when the CR could be delivered, cost in man-power, and cost in system CPU and memory requirements. The delivery schedule is based on how much is currently in the OI and whether the CR would fit also.

Three stages are involved in the requirements definition and are key to determining and achieving the delivery schedule described:

System Requirements define the system level interfaces and requirements to a level of detail to allow top-down software design. NASA originates the candidate list of CRs to be implemented during a scheduled OI. Direct meetings are held with NASA (the customer) and the IBM developers to determine whether the capability changes they want will work and how it would be best to implement them in software. Several formal and informal meetings are utilized to get the necessary level of definition.

Functional Requirements start from the system level view and break it down to the functions for each program module. Applications software requirements are specified during this stage. What results is a map of how the function will be spread between a set of system modules.

Detailed Requirements are created for each module. The requirements cover the formulation of the module, logic, mission data used, and the display format of the data. Requirements are documented in the Functional Subsystem Scftware Requirements Document.

Formal reviews are held during the three stages. Review participants include the requirements analyst, development programmer, verification analyst (tester), moderator, NASA system software development representative, and the CR author if possible.

Design Phase

Design of the software is done in two sub-phases. The requirements are quite detailed which makes the design phase flow reasonably smoothly. The two stages are:

Functional Design concentrates on describing integration of this function with existing software, identification of all affected modules, sequencing of modules, dependencies and interfaces between major areas of the software, structural design at the system and program levels, and program structure in terms of function, storage, timing, and operational sequences. The requirements are documented in Program Design Specifications. Formal design reviews by an internal team of peers are held for significant system software capability updates and for all functional changes.

Detail Design produces the detailed "build to" design for each program. The design describes the functional organization of each module through the module structure, database and I/O processing description, and logic flow. Detailed Design Specifications are created to document the design and to also serve as documentation of the final product. A formal design review is held by an internal team for all design work products. Specific objectives are defined and concentrated on during the inspection.

Coding

Coding is a manual process that concentrates on structured programming methods. The operating system software is programmed in assembler language due to the strict performance and space requirements. Applications utilize a high-level engineering language, HAL/S, provided by NASA to IBM and developed specifically for the space program software development. Coding is performed to meet the defined programming standards and utilizes the Detailed Design Specifications which define the module in very extensive detail. Formal code inspections are held for each code unit. Reviews are done with the same peer group that did the detailed design inspection. The aim is to discover errors in coding, interfaces, and requirements implementation.

Development testing is considered to be part of the coding phase. Two levels of testing are done, unit and functional. Unit testing is done by the developer and consists of validating code unit logic, data handling, and testing of design limits. Functional testing is also done by the developer and consists of validating module interfaces, functional capability, and overall exercising of code units in combination with each other.

Successful testing and review of the results allow the code to be integrated and moved to Independent Verification Testing. The review of the results is through a very formal process called the First Article Configuration Inspection (FACI).

Independent Verification

Software verification is an area independent of the area directly responsible for developing the software. The group maintains an adversarial relationship with the development organization. A basic assumption the group uses to guide their work is that the software has been untested when it arrives. This ensures an extensive test of the function.

Verification Analysts develop an independent, detailed understanding of the requirements specifications. They review the specifications, contact the customer independently, and participate in formal inspections during the requirements, design, and code stages. Testcase development follows a formal development process that involves creation of test procedure documents written to describe individual test sequences in detail. Reviews, consistent with what is done in development, are held with the customer community on the overall plans and the specific testcases.

Testing is done in two sub-phases, Detail Verification Phase and Performance Verification Phase:

Detail Verification Phase is used to verify that the software system meets all specified requirements. The testing is further broken into three different aspects: Detail tests that involve explicit testing of all functional requirements, logic paths, algorithms, decision blocks, and execution rates; Functional tests which involve explicit testing of all data interfaces, data I/O, proper sequencing of events, and the order of module execution; and Implicit tests which involves execution of all detail tests while the entire flight software system is running and also involves simulation of hardware failures to ensure proper execution in failure scenarios.

Performance Verification validates that the software system as a whole performs in non-stress and stress environments. Each new function is tested to ensure it performs as specified. Cases are constructed to selectively stress various elements of the flight software with different focuses used for the system software versus the functional software of the guidance and vehicle systems.

Configuration Inspection is a formal technical review to assure that all end item requirements have been included in the specs and that the end item documentation agrees with the delivered product. This is done after detailed verification has been completed.

Validation

After the software has been accepted by NASA at CI, it enters into the validation testing phase. This validation testing and integration is performed in a separate lab. NASA does testing to validate all subsystems interface correctly and that the system functions properly when the changed subsystems are integrated together.

Successful validation allows the software to be moved into operation. Operations support is provided by IBM for user training, software installation, routine maintenance, aid in development of user procedures and test support, and aid in formal flight support.

Maintenance

Problems found are formally documented by NASA in DRs (Discrepancy Reports). Failures outside the software system that can be fixed via software along with changes to requirements are documented in CRs (Change Requests). DRs and CRs are reviewed and must be approved before the software changes are allowed.

Software changes are required to follow a very formal process to keep defect injection low. Requirements are that: the design, implementation, and verification processes are used; documentation updates are made and incorporated; configuration control procedures are used throughout the process; and quality assurance and programming standards are enforced.

Process Usage and Compliance

Process usage is consistent throughout the organization with all development utilizing this formal process. Compliance is ensured through internal peer reviews, strict change control, and in-process metrics that gauge the results of each development stage. The culture is the other factor that ensures the process is used as defined.

Release Management

Program Managers are in-place for each OI. They are responsible for ensuring the software being designed and developed meets the requirements and objectives defined by NASA. The Program Manager ensures adherence to processes, procedures, and configuration management. Development of the application support systems used in the preparation of flight systems is also managed by this person. Finally, the Program Manager is responsible for certification of all flight systems.

The Program Managers put a project-wide team in place to effectively do this broad job. The "control board" team has the responsibility to: define, negotiate, and baseline project-wide development, build, test, and delivery schedules; control flight software changes (DRs and CRs); and coordinate interproject technical issues.

Change Management / Configuration Management

Configuration management spans the full software development cycle. It begins at requirements definition when the Change Requests are generated, it is required during each phase of design and coding, and it extends to schedules, supporting tools, testing, and documentation. Baselines, which are technical control milestones that are officially agreed to, are also controlled. The baselines define software content, schedules, release formats, deliverable items, verification plans, and support software.

A configuration management database is the crucial part of the system. It allows mapping of requirements to code, holds extensive information about requirements and errors that drive changes, documents schedules and software content, drives the actual software system builds, and is used to generate reports used for development management.

Change Requests (CRs) and Discrepancy Reports (DRs) are used to control all changes. Multiple "control boards" are in-place between NASA and IBM to review and determine whether changes are approved. Approved changes are documented and become part of the formal baseline. The configuration management database is updated to provide tracks regarding the change. Different control boards are used for the different types of change requests. An overall control board is in-place to monitor the amount of churn in the system.

Metrics

Quality measurements are the key metrics utilized. Quality is more than just "bugs". It covers conformance to requirements and performance of a task correctly on the first attempt. Three measurements were chosen in early 1980, with a fourth added in the mid-1980's, that have been used since that time to monitor the quality of software. Separate measurements are tracked for each OI of the flight software.

Total Inserted Error Rate is defined in terms of major inspection errors found, an in-process measurement, plus all valid errors that resulted in documented Discrepancy Reports. This is measured per KSLOC. The measurement is the fourth that was added and is designed to understand the total rate of software defects throughout the development life of the product.

Software Product Error Rate is defined in errors per thousand lines (excluding comments) of source code (KSLOC) and is measured for each release of software. Software errors have declined from 2 errors per KSLOC in 1982 to a rate of .11 per KSLOC in 1985 and has continuously decreased since that time.

Process Error Rate is an in-process measurement of the efficiency of error detection early in the software development process. Process errors are computed prior to software delivery and are measured in errors per KSLOC. This measurement fell from more than 10 errors per KSLOC in 1983 to less than 3 in

1985 and has continuously decreased since that time.

Early Detection Percentages of Software Errors is another in-process measurement that measures the ratio of errors found in inspections of the code to the total number of errors that are known to exist in the code. The total number of errors is estimated based on past history and the quality goals for the release. Major inspection errors are any error found in a design or code inspection which would have resulted in a valid DR if the error had been incorporated into the software. Each phase of the process is measured with models existing for each. The measurement has decreased from 10 errors per KSLOC in 1983 to about 4 in 1992.

Defects found during development are categorized. Analysis utilizing this data is used as another method to drive to higher quality. It has been used to study the process to determine the defect cause and to aid in process improvement actions.

Measurement techniques have been used to increase the quality focus. Product measures have been extended down into the organization including subcontractors, employee participation has been encouraged through tool availability, measurements have been focused on key areas, and data retention has been used to build history that can be referred back to.

Process Improvement

It is heavily communicated and well accepted that process improvement is critical in the drive to zero defects. A formal process of Causal Analysis and Defect Prevention Process is used throughout the organization. Attention is constantly placed on the removal of defect causes. The two processes have reinforced a philosophy of: build quality in versus test errors out, usage of early milestones to identify and measure quality, usage of error data to drive process changes, and for every error found asking how the process can be changed to prevent it next time.

As work products are inspected through the development phases, all defects are recorded and categorized. Analysis is used to study defect trends such as error escapes through the phases. Measurement of the process and the product allow the effect of process changes to be measured to determine impacts on quality.

Tools

Software Development Facility (SDF) is the major tool that is key to the development process used for space shuttle software. SDF is an IBM developed simulation test bed that simulates the flight computer bus interface devices, provides dynamic access and control of the flight computer memory, and supports digital simulation of the hardware and software environments. SDF requirements were defined early by NASA and all new capabilities require NASA approval. Enhancements are developed and tested by a dedicated development and maintenance group that follows the formal process described for the shuttle software development.

SDF also provides program management support to control program data, support to do builds to create a deliverable system, a build facility to map software into the format needed for the system's mass memory units, a simulator, and a documentation and statistical analysis system.

Expert systems have begun to be utilized to analyze test cases to support verification tests. Coverage can be assessed through this mechanism. Extensive automation support is used for test case generation and execution.

Process Education

Documentation on the process is extensive and is available to everyone on-line via their terminals or through printed copy. Mentoring by experienced developers is another effective mechanism for orienting new people to the process. With the level of formality associated with the process, education is quickly gained. Numerous presentations and education packages are available and are given upon request.

Summary – Results

Results have been excellent for IBM Houston in three key areas. The first is the operation at a zero defects level. Three major releases that have spanned more than fifteen missions have been at this level and NASA (the customer) has recognized the group for their outstanding quality in meeting system requirements and having zero defects. The second is through the quality of the overall process used to develop software. In an independent evaluation conducted by NASA headquarters, the IBM Houston group's software development process was given the highest possible rating of "5" on the SEI standard evaluation of process maturity (at a time when most companies are at the "1" level). The third area of success has been in productivity improvement. As quality has improved, the group has also seen productivity improvement in important areas such as software maintenance.

There are limitations that exist within the process when it comes to applying it to other development groups. The process described is very costly in terms of effort expended and cycle time to develop software. For the environment the process is being used for which involves a dedicated customer, a limited problem domain, and cost being less of a consideration than zero defects, the process is an excellent fit. For the wide range of commercial software developers that do not operate in this type of domain, the applicability of the complete process is not feasible and therefore we have process variations and additions described in the other company case studies.

Drawing upon this success in process improvement and quality software delivery, the Federal Systems Company has created a team that now goes out and consults on the software development process. Their success in the implementation of a high quality process is evident and they serve as a good guide of what can be done through a software development process. Different groups may not be able to, or want to, implement the full process but may be able to adopt pieces of what is done. When this is successfully done, the customer and the provider will both be happy with the result.

Chapter 4: IBM - Application Business Systems[22]

The Company

Refer to the IBM Company description for IBM Federal Systems Company in Chapter 3.

IBM Organization

Refer to the IBM Company description for IBM Federal Systems Company in Chapter 3.

Application Business Systems is the business unit that is part of this case study. The unit is headquartered in Somers, New York with the development location in Rochester, Minnesota. Manufacturing operations are in Rochester, Minnesota; Guadalajara, Mexico; and Santa Palomba, Italy. 1992 revenue was $4.5 billion for the unit's hardware and software, and the sales of ABS systems accounted for approximately $14 billion of total IBM product and service revenue. Employment was 5,498 people in 1992.

ABS is managed by a Senior Vice President that reports to the Corporation CEO. The business unit is organized into four separate groups: the development laboratory that is responsible for all hardware and software development, the product marketing group that is responsible for working through direct and indirect sales channels for sales of business unit products, the manufacturing group responsible for manufacture of all computer hardware sold by the business unit, and the finance and planning group responsible for overall financial management of the unit. Within the development laboratory, there are groups responsible for all hardware development, software development, strategy and planning, and support operations (testing, tools, software builds, software distribution, and national language translation).

Culture

Multiple books and articles have been written describing the culture of IBM. Many of these have characterized IBM as either being a marketing company, a company based on key principles laid out by Thomas J. Watson, Sr., a company with vast technology capabilities, or a company that has been working hard to protect the markets it captured in past years. My concentration is on the culture that surrounds the development community for ABS that is located in Rochester, Minnesota.

Rochester is characterized by a very cohesive and productive development group that has been high within IBM in terms of productivity and quality achievement. The location began operation in 1955 and eventually moved into the position of being a single location responsible for developing a family of midrange computers. Most IBM systems are developed in multiple locations, but Rochester has generally operated on its own. This single location situation has lead to a very strong sense of ownership by the development community for the products created and sold.

"Out in the cornfields" has been a label put on Rochester. The situation has helped ABS Rochester move forward on its own without significant intervention from other groups in the company. The System/38 and System/36 were midrange systems that have fueled much of the success. They served as the base for development of the AS/400 computer which has been recognized as a very significant technical and market success.

Working outside the IBM core during the early years, which was mainframes, created a consistent feeling of being "under-recognized and under-funded" for development of products. The feeling of being under-funded created a culture of the site pulling together as a team to overcome the odds. Competitive spirit is alive within the group. The site also benefits from what is often referred to as a "midwest work ethic" that drives people to dedicate themselves to completing the work that needs to get done. In addition, attrition and transfers of personnel are minimal which has helped the site foster a large team attitude of people that know each other and achieve success as a group.

A significant problem in 1978 also helped define the culture in Rochester. Achievement of the committed shipment date for the System/38 computer was missed by nearly a year. Multiple factors lead to the situation, but one of the results of it was establishment of process steps and metrics to make software development more predictable and to understand project status throughout the development cycle. A well-defined process that is adhered to, along with a strong set of in-process and post-ship metrics, are the core of product development.

Quality and customer involvement are the two final important elements of the culture. Market Driven Quality was a specific movement started in 1990. Prior to that, the group had a good record of product quality but the pace of improvement was increased with the MDQ movement. Process improvement became a lab wide drive and the overall site pursued process improvements and assessments that led to the Malcolm Baldrige National Quality Award, and ISO 9000 certification. Customer involvement has been stressed since the days of the System/36 and System/38. Development process phases like requirements, testing, and support have utilized customer involvement. Knowledge of post-ship quality is carried throughout the development group. Efforts to minimize customer problems are enhanced due to relatively close relationships with many customers.

Product Description

Products developed in ABS at Rochester have centered around midrange systems commonly referred to as minicomputers. Significant product families are the System/3, System/34, System/36, System/38, and AS/400. The AS/400 was introduced in 1988 and is the core product now delivered from Rochester. All hardware and software development is managed by the development laboratory with nearly all of it developed in Rochester.

The system hardware spans a low-end set of models appropriate for as few as 2 users and extends up to systems that will attach up to 2,400 terminals. All models of the AS/400 family provide a consistent hardware and software architecture. You can start with a single processor with 4 or 5 terminals, then expand the system into a powerful network, all using the same operating system and applications software. The operating system is proprietary with support for a wide-range of industry standards. An integrated relational database is part of the system. Extensive communications standards such as OSI, TCP/IP, and SNA; a suite of languages such as C and COBOL; co-operative client/server processing with personal computers; and built-in functions like security are significant parts of the operating system and system products.

Release Structure and Strategy

New releases of the OS/400 operating system and program products occur on approximately a yearly basis. The yearly base has become standard due to customer preferences and development lab efficiency. Installing a new release of the software is something most customers prefer only doing on a yearly basis. The grouping of function on a yearly cycle is also efficient from a development management standpoint, a support organization standpoint, and from a marketing support standpoint. For development, the yearly cycle allows concentration of most resources and management attention on one cycle at a time even though the early stages of the upcoming release are beginning during the later stages of the existing release. Support organizations such as testing, software builds, and software distribution manage a single release more efficiently than multiple releases. Finally, the product marketing group is able to concentrate attention more effectively when they can feature one release at a time.

Each release has some critical functions that drive the final date for making the release available. The dates have tended to vary slightly in either direction from the yearly timeframe. Since support for new hardware, such as processors or I/O, is in the software, hardware can have an affect on the dates of a release. Enhancements are periodically made available on schedules separate from the base releases. These enhancements are managed by themselves and tend to be provided to specific requesters of the function versus a broader distribution.

Quality Plan

At the beginning of each release cycle, a system quality plan is established. The plan is where committed quality goals are defined and improvement actions to achieve the goals are described. In order to create an effective plan, a bottoms-up commitment process is used to derive the plan. After distributing an initial draft of the goals and plans, the Development Quality group holds brainstorming sessions with groups of developers to refine the goals and actions. Quality goals and actions are then committed to by each development and support organization. Projections of achieving these goals are then made throughout the development process through use of in-process metrics.

The system quality plan only represents the overall system approach to quality improvement. Development teams and product managers still own overall quality and are responsible for planning, implementation, and outcomes of quality actions. The system development plan, which defines what items will be worked on in the lab, includes development items for the release that are targeted towards quality improvement.

Development Process Overview

The development process is well known in Rochester. Documents exist on-line and in printed form that define the stages, entry and exit checklists, and tips and techniques. These are periodically used for reference, but most adherence to the process comes from experienced development team leaders and team members. The process is well accepted and people believe in the steps as important to delivering quality products on schedule. Refinements to the process continue, but the base lifecycle has not changed. Due to some level of iteration in early stages, most people look at the process as a "modified waterfall" or "incremental development" process.

Requirements Phase

Requirements are formally coordinated by a group of individuals called the "System Strategy" team. Multiple sources input requirements and the flow is continuous versus concentrated on any one period. Sources of requirements include:

Direct input from groups of customers such as the Customer Advisory Council
which meets with the laboratory on regular intervals

Direct input from IBM Marketing and Service groups worldwide

New hardware needs

Technical development input on important/strategic technical items

Executive commitments

Business Partners developing and selling applications for the system

Customer satisfaction calls on software problems

Market research done by system planners

Competitive analysis information

Requirements are written down and input to a database of Plan Content Records (PCRs) that can be submitted by anyone with access to the computer system. Multiple groups can submit PCRs, but the adoption process is tightly controlled by the System Strategy team. A plan commitment process is then used to distill this huge list of requirements down to a set that can be developed with the resources available in the lab. The initial wave of distilling actions occurs with a team of people from System Strategy, the hardware plan management team, the software plan management team, the financial plan management team, and the support organization. It is their job to create a prioritized plan content document that they submit to development and the system plan management team.

From the Prioritized Plan Content document, an iterative process of comparing requirement priorities, development costs, revenue potential, skill requirements, and current staffing options takes place. The team of upper level management responsible for development resources, along with the product marketing management, determine the base of committed projects that development is approved to work on. Formal release plans are generated that contain committed content for each of the planned upcoming releases.

System Level Design

For a select set of development projects, an early design stage is utilized. The System Level Design is scheduled early in the plan cycle. Its purpose is to ensure proper integration of new AS/400 functions into the overall system, consistency of new products with existing products, new functions effectively use existing system support where possible, and modified products continue to be structurally sound.

Functions that are candidates for this type of review are new AS/400 License Program Products, which are functions packaged separately from the base operating system and sold as a separate product. Other candidates are functions that have the potential of becoming LPPs, major enhancements to existing LPPs, and major functional enhancements to the base operating system. Enough design work is required at this stage to describe the product externals, show inter-product dependencies, and show enough of the product design to exhibit design completeness/correctness in areas like user interfaces, interfaces within the product, performance, and compatibility.

Design Phase -- High Level Design

The design phase in IBM was previously characterized by formal detailed specification documents that were tightly controlled within the company. Design Change Requests (DCRs) have replaced specs and are available for on-line access by the full development community. DCRs hold relevant information for the product being developed such as a requirements statement, design specifications, schedules for all development and test phases, tracking information for completed and planned activities, and a small amount of high-level design information.

For the high level design phase, the objective is to define the externals and internals of a function from a component perspective. System components are previously defined areas of system function. External functions, user interfaces, inter-component interfaces, and inter-component data structures are some of the key portions designed during this stage.

Development organizations that will do the ongoing design and coding steps are also responsible for the high level design. The development group works with the system planning organization, a group of system designers called the design control group, and groups responsible for performance, usability, and user information. After the design is completed, any changes for late requirements that are agreed to will require the high level design review to be held again.

Schedules and work efforts are reviewed again at this stage, with changes needing approval. Entry and exit criteria exist for this stage and must be met before progressing to the next phase. Prototype and simulation work are utilized in some cases that involve changes to the user interface. This allows for early feedback and usability testing before the design is completed.

A formal inspection is held and must be satisfied before design work can move to the next phase. Nearly 100% of the groups making functional changes to the system hold high level design reviews. A high level design inspection is expected to be held unless the developer can define the evidence and rationale to support not holding the review. This is considered uncommon and only occurs when very small changes are made.

Design Phase -- Low Level Design

The objective of this phase is to specify the functional breakout of the component into parts, such as individual modules, macros, and includes (macros and includes are common data structures and general code used across multiple modules). The internal design of each new or changed part required for the function is defined during this stage. This level of design work is frequently done by the individual that did the high level design and will do the coding of the module. Schedule and work estimates are reviewed again and changes must be communicated and approved. Groups that participate in work activities outside of development include performance, national language support, user information, usability, and system test.

A formal design review is required before moving from this phase to the next design phase. The review is done by a small team of peers. Low level design reviews are held for over 75% of system changes. General criteria exist that help to define the types of changes that reviews should be held for. Reviews are encouraged but optional for changes that fall outside criteria. Flexibility exists in the review process with low level design reviews sometimes combined with code reviews which gets review coverage of low-level designs above 90%.

Tools exist to assist the design process. They allow a developer to write the design in a language that will allow it to be contained in the actual code. This allows the design to be extracted at any time from the code. Developers are not required to use any specific tools for creating the design. Flexibility is afforded the developer to pick what works best for the module being developed.

Code Phase

This is the formal stage of taking the detailed design and turning it to code. The primary objectives of the stage are to complete documentation in the modules, write code for the modules, and develop the messages, commands, and screens. A majority of the coding is manual with only a small amount of code generated by machine from formal design languages.

A formal code inspection is held to remove defects from the code. Alternatives are used in sequencing code inspections and unit testing of code by developers. The code inspection can be held before, in conjunction with, or after the unit testing. Code reviews of all code is expected and runs very near 100% coverage. Small teams of peers conduct the inspection of the code. Since it has been shown that small changes are especially prone to errors and benefit greatly from code inspections, reviews have become an accepted part of small code changes and this has driven the coverage number near 100%. Formal entry and exit criteria exist for the stage. If changed or late requirements occur during this stage, any approved changes cause reviews to be held again for at least the high level design.

A portion of the code validation is done through unit testing that is executed by the code developer. It is the first test of executable code and precedes integration of the code into the system. The purpose of the test is to validate the detailed code against the design and ensures limits, internal interfaces, and data paths are verified. Developers are expected to have some level of test plan, and track status and coverage based on the plan. Criteria exist for completion of unit test which, upon satisfaction, will allow the module to be integrated and become part of the base system code.

Component Test

This test is run on the integrated system. The integrated system consists of the base system from the prior release along with new code that has been integrated for this release. It will not be a complete system ready for system test until all code from the release has been integrated. Component test phase comes after the developer has tested the module in unit test and integrated it into the system. The purpose of the test is to verify that the portion of the new functions executed by this component work as defined in the requirements. This is the first opportunity to put the new function together with other new functions in the system.

A component test plan is created for the test. The developer may be the only person using the plan, but the more common situation is that others affected by the function will review the plan. Formal reviews of the test plans are recommend and frequently held for more extensive functions. Component test status is tracked for the system with the major comparison being test case attempts versus successful tests. Criteria are in place that must be met before the test phase can be exited.

System Test

System test is the final in-house validation test for the new release of software. There has been a significant evolution of this phase over the last years. Previously, System Test was done totally in-house and had a reasonably large staff of people to do the testing. By moving a more extensive piece of testing to select sets of customers earlier in the development cycle, both the schedule and amount of in-house resource were dramatically reduced without compromising product quality.

During the last 2 years, more validation responsibility has moved to the development groups versus the system test organization. The rationale and approach has been to ensure development process and quality responsibilities were clearer, and to reinforce quality steps being taken while development was in-process versus at the end of the cycle. More comprehensive regression testing and validation of a wider range of function through component testing were two things moved to development. Stricter criteria and enforcement of the number of defects that can be open before the formal start of system test was also put in-place. All actions were designed to move quality work to earlier development process phases, and to solidify the system sooner so that the test cycle time could be reduced.

The system test organization reports through the same management chain as development. All independent quality assurance activities have been eliminated. The system test group has reported to the same management as development for a long period of time, but the elimination of the systems assurance function is new. As the systems assurance group moved from doing a combination of independent test validation along with analysis of development progress to only doing the analysis, they lost much of their value-add to the development of products. During the same time that the move to only analyzing data was happening, development was improving the amount of data analysis being done on quality such that the two efforts were duplicate. At the point where the duplication became abundantly clear, the independent systems assurance group was eliminated.

System Test is called the RAISE test, which addresses the main areas of focus during the test: Reliability, Availability, Installability, Serviceability, and customer-like Environments. The test strategy is to utilize a range of customer-like environments as a base and then vary the physical operating characteristics to stress environments and simulate obscure but complex situations. Emphasis is on stressful, concurrent product interaction to drive each part of the system toward their limits. Artistic tests (versus automated tests) are utilized to stress problem-prone areas or functions that can not be effectively tested tnrough automated tests. Use of automated tests is extensive for providing background work for stress, background work for concurrent product testing, previous release functions for regression testing, and repeatable execution of new functions.

Test planning and two different review stages are used by the group. All test schedule and resource estimates are made by the test teams for each of the significant functions in a release. The estimates are done during the requirements and design stages and are included in the DCR along with development estimates. An aggregate view of the release content is used to define test coverage priorities and to recommend any staffing changes that may be required. The overall RAISE Test Plan defines test coverage plans for each major function in the release, customer-like environments that will be utilized, and network requirements. The test plan is reviewed by development to gain agreement on the adequacy of the test coverage defined. Detailed Test Plans are created that give detailed descriptions of functions tested, test procedures, and validation activities. The detailed plans are reviewed by peer testers and in many cases by the developers.

The RAISE test execution phase continues until the exit criteria have been met. The activity has a very clear end date scheduled, but completion is not controlled by the planned end date but instead by meeting criteria. Status is reported on a weekly basis to upper level development management. Metrics and critical problems are concentrated on during the meetings.

Beta Test and other Early Programs

Multiple programs are in-place due to the increased significance customer based testing has taken on through the last years. Different types of programs are utilized versus a single formal Beta Test. While RAISE test is in progress and before the system is released for the formal Beta Test, a set of users are brought in with the intent to break the system in as many ways as possible. The goal is to find defects while the developers are still heavily involved with the new code.

The initial Beta group is a set of heavy system users that have a set of long-term relationship with IBM Rochester. They get early versions of code and test execution in their normal environment while also doing whatever they can think of to break the code. Problems found during the test are fixed prior to Manufacturing Release and shipment to the full customer base. The next Beta Test phase is used to get a tested version of the system out to a group of customers that have been selected for different purposes. Some selections are due to utilization of a new function, stressing of an area of the system that has been updated in some way, and early exposure to a set of customers for marketing reasons.

All programs are managed carefully to achieve specific objectives. Customer selection is done by a group that looks at the specific customers to understand their work environment and how they use the systems. The initial set of Beta test customers are a well-known group with large, complex configurations or a history of finding a significant number of defects after they have received new releases. Beta Test customers are selected based on requests for new function along with plans that will utilize it early, or due to marketing reasons communicated to the selection group. After selection, the group monitors the tests closely to ensure feedback is received and used. Tight management is critical if the benefits are going to be received from the tests.

Maintenance

Software maintenance is done by the same development groups responsible for developing the next releases of the system. This also includes a support organization that is responsible for handling the phone calls and on-going definition of the problems the customer is encountering. A majority of problems are resolved by the support organization either through answering a question or providing a fix that is available for the problem the customer is facing. Beyond specific individual fixes, fixes are provided on a periodic basis to customers that wish to receive periodic updates that have accumulated.

When problems are unable to be resolved by the support organization, development gets involved. Work goes on between the customer, support organization, and developer to understand the problem in detail. Developers will then work to solve the problem and get a fix to the customer. This whole process is supported by an extensive tracking system that ensures problems are not forgotten, and that code fixes are kept track of when doing future development work.

Process Usage and Compliance

Process compliance is driven through what is considered accepted practice and through the use of metrics. The process is well-defined and communicated to all developers. In addition, it has been in-place for many years and the senior developers have grown up with it as the way to do development. Quality statistics and research are also made available which give quantitative proof of the value different process steps provide. One key part that is important to process usage and compliance is upper management support. The upper managers are advocates of the process and expect developers to use it unless they can provide good rationale for variations.

Metrics are an important guide to how the process is being used. Design review coverage statistics are kept which show whether the reviews are being held and how effective they are. Testing can be measured by test coverage statistics. Defect removal can also be tracked across the phases to determine if additional efforts need to be spent in any of the stages. All of this provides an environment where process usage can be determined and evaluated.

Release Management

An individual is assigned to a specific position called Release Manager for each release of the system that is developed. The Release Manager is a matrix management position that spans development, test, and other support organizations. Release Manager overall responsibilities are to: coordinate, analyze, initiate action, provide alternatives, and assist the development and support organizations in whatever way necessary to deliver the release. The final result of the work is to enable a high quality, well performing product released on the best schedule possible.

More specific actions taken by the Release Manager are setting major checkpoints during the release, escalating problems that are not getting resolved in a timely manner, continually analyzing progress towards completion of development, taking positions on plan changes to the release, and maintaining release status. Release status is handled through constant interaction with development and support groups along with analysis of metrics. Scheduled briefings to the management team and the business unit executives are responsibilities of the Release Manager.

The Release Manager works with a team of functional representatives from each development and support organization. This team will bring problems forward themselves or act on problems discovered through independent investigation by the Release Manager. The Release Manager will use personal experience/judgement and team input to find an owner for the problem while often providing recommendations to the owner on solutions to the problem. Keeping schedules accurate and achievable is another key responsibility. Through usage of thorough reviews at Major Checkpoints determined by the

Release Manager, analysis of each major function is done. From this review, an assessment of the overall release is made along with recommendations on actions that should be taken for individual functions.

Change Management / Configuration Management

Focus in Rochester is better defined as "change control" versus just "change management". With releases that regularly reach about two million lines of new and changed code, changes can get out of control without careful management. Tools and documents are both used to control the change. DCRs (Design Change Requests) are used as an important documentation tool to control change. They contain design information, schedules, tracking of completed and planned activities, and also go through formal phases that involve specified actions that must be completed and reviewed before a project can proceed to the next development phase.

Release content, release schedules, requirements, design, and code are all managed though processes that are designed for proper review and communication of changes. DCRs and documents that track release content are used to communicate changes that are approved. Release content, overall release schedules, and requirements are controlled by the Release Manager and a team of managers from the development, product marketing, and support functions. Design changes are reviewed and approved by a team of technical experts that meets on a weekly basis.

Changes to code are carefully managed by tools that run on the development system. Integrations of new or changed code must be tied to a specific new function or fixes of problems that are formally reported and tracked. Testing must be done before the integrations, with the set of tests determined by the developer. Code Freeze is a formal point at the end of the development process where no further changes can be made to the code without approval. The Release Manager, or a designated person from a development organization, must approve all changes to the system. This process allows for fixes to critical problems but cuts down on code churn from widespread code changes.

Metrics

Metrics are a key part of managing the overall development process. Documented metrics are used at each phase of the development process and are part of both entry and exit criteria for the phases. Usage of metrics and process are for the purpose of identifying how near completion development is, and what the expected quality level will be. The base set of metrics that span the processes are: entry and exit criteria for each phase, requirement/design/code change, in-process bugs/defects, post ship defects, reliability, and performance.

In-process measurements provide the ability to understand progress versus the plan and to implement real-time quality management. The in-process measurements are based on a few key metrics that can be used versus a large set of metrics that are only there for measurement purposes. Metrics are compared to a historical baseline and with the prior release.

Key metrics used during the design stage are time spent in reviews, defects removed by stage, and defect escapes by stage. These three measurements can be used to determine the effectiveness of each stage and whether additional inspection or testing should be done on functions being developed.

Defect tracking during development and test is the other important in-process set of metrics. Metrics used are: backlog, open versus closed defects, closed versus the backlog during a phase, severity mix of defects, and some analysis of clusters of defects. Once the product has shipped, defects continue to be measured. Some metrics tracked are: number opened during a period, backlog, mix of severities, defective fixes, concentration of defects in areas of the system, and releases where the defects were injected. Monthly reviews are held on the post-ship defects to keep management informed of overall status and of critical problems still open.

Process Improvement

Quality has always been important in Rochester, and over the last several years the emphasis has shifted to process improvement activities that improve quality. Causal Analysis and the Defect Prevention Process were introduced two years ago and have been employed throughout the development and support organizations. The goal is to determine the sources of defect injection and remove them. Meetings are held to analyze problems and then individuals go off to determine defect origins and how to eliminate them. Phase kick-offs are an additional step used to start each development process phase. This allows for process learning from previous experience. All of this is centered around the concept of improving the processes.

A Process and Quality Improvement department was put in-place for the laboratory. The group acts as partners and consultants to development with the purpose of bringing in and spreading new process ideas. By being in the development organization, the group is considered part of development and effective team members. The group has a significant role in tracking metrics and interpreting results. Some process changes have come from the data analysis. After seeing significantly high defect rates associated with module interfaces, the high level design and review process was modified so that interface issues are resolved earlier. Movement of quality and metric tools to the developer level to allow wide spread usage was also initiated by this group and has been an aid for the improvement of on-going attention to quality.

Tools

The tool set has evolved through the years. Development is done on mainframe computers and many analysis databases exist on the mainframe. The development language is standard and has been enhanced through the years with function that minimizes the machine instructions required for operations. Compilers and linkage tools are standardized across the system. This has all been done to minimize the integration and linkage time required to pull the full system together. In recent years, personal computers and workstations have been integrated into the development environment.

Metric tools exist on the mainframes, in most cases, with some also running on the AS/400 and on personal computers. Databases of information are available to all users with a set of analysis tools accessible to them. Moving this access to the teams involved with development has improved the accuracy of the data and the usage of tools during the development phases.

Process Education

Education of new developers is accomplished through a variety of sources. Significant process documentation exists and is accessible from every person's terminal on their desk or in printed form. Development team leaders are responsible for ensuring the process is understood and followed by their team members. Education is often handled working together with experienced people. "On demand" education is also available. This has been created and is delivered by process experts and is often scheduled for large groups of developers. The attempt is to time education to when it will be used during the development process.

Phase Review Process

A System Manager is responsible for managing timely delivery of competitive, high quality product offerings that encompass both hardware and software. It is a matrix management role with the functional areas. The common mechanism for carrying out this role is to utilize a team of representatives from each functional area including the marketing and support organizations. A System Checkpoint Process is utilized for each release of the product. Four key business checkpoints are focused on by executives and organizations outside the direct development organization.

The four checkpoints are:

Initial Business Proposal (IBP) : IBP provides an initial analysis of the business case (cost versus incremental sales) and preliminary product definition. A Business Fact Sheet, which highlights a summary of the release, may be provided out of this stage. Exiting the IBP indicates that all functional areas agree to proceed to the Commit Checkpoint, which is next in the process.

Commit Checkpoint: This is the point where plans are examined to determine if announcement and general availability (GA) can be met. The product definition and business cases are completed during this phase prior to the checkpoint. A comprehensive plan is developed that defines the work items and schedule to meet the General Availability date. Exiting the Commit Checkpoint indicates that all functional areas agree to achieve the announcement and general availability dates. The Business Fact Sheet is generated or updated during this phase also.

Announce Readiness Review: This step assesses the program after most of the development activities in the plan have been scheduled. It occurs before the main effort begins to prepare marketing deliverables for announcement. Exiting the Announce Readiness Review indicates that work plans are defined, responsibilities assigned, and schedules are agreed upon.

Announce Checkpoint: Obtain final commitment to announce the product. Exiting the Announce Checkpoint indicates that all requirements are or will be met to support the General Availability.

Summary – Results

Products developed in Rochester have enjoyed a history of high quality recognition and business success. The team is very productive and has successfully adapted the product to the changing market successfully. Quality recognition has come via the Malcolm Baldrige National Quality Award and ISO 9000 registration. Change management and metrics have been incorporated into the development process and are integral aspects now.

The challenge that exists centers around the balance of individual creativity and process compliance. Individuals are recognized as the basic elements that dictate success but it is sometimes difficult for them to feel the freedom to attack an opportunity with the creative approach necessary to solve it the best way possible. Developers recognize the positive aspects of the process, but feel the limits imposed also. Managers continue to struggle with the need to follow the process to improve delivery and quality predictableness while realizing that individual creativity is needed to solve problems that continually become more technically complex.

Chapter 5: Fujitsu[23]

The Company

Fujitsu was established in 1935 when Fuji Electric incorporated its telephone equipment division as a separate company. The company commercialized Japan's first digital calculator, 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 communications and office equipment, computers and computer peripherals, and data processing services. Corporate headquarters are in Tokyo, Japan with major wholly owned subsidiaries operating in the United States and Europe.

Fujitsu's business is concentrated in data-processing related sales, communications systems, and semiconductors. It's stated goal is to become a "fully integrated computer and telecommunications product company". Data-processing services are concentrated on two general areas: computer systems hardware and basic systems software, and large-scale custom applications and applications packages. Computers developed and sold are mainframes, minicomputers, workstations, and personal computers.

Fujitsu is the number one ranked information technology vendor in Japan and the Far East, and is now the second largest in the world, after IBM. For the year ending March 1992, Fujitsu employed 145,000 employees worldwide.

Revenues were $21 billion with a profit of $94 million. To get an idea of how the employee and revenue mix is in the company, the results for the year ending March 1988 are useful. At that time, Fujitsu had over 52,000 employees in the parent corporation, with more than 10,000 involved in software production and staff support. Non-consolidated sales totalled over $13.7 billion with $10 billion in revenue coming from data-processing sales. The revenue from data-processing sales was 72% of total revenue with the other revenue coming from communications systems at 16% and semiconductors at 12%.

Fujitsu Organization

Fujitsu is organized into a set of 14 operating groups, as of 1986. A subset of the key groups focused on hardware include: Information Equipment (disk drives, printers, FAX machines, other peripherals), Transmission Systems, Switching Systems, Telecommunications Systems, and Semiconductors. The groups responsible for sales are: Systems Sales, Office Automation Sales, and NTT Sales. Groups that are involved with the data-processing services part of Fujitsu are: Computer Systems, Printer-Circuit Board Products, Systems Engineering, and Field Service Engineering.

Systems Engineering and Computer Systems are the two groups that handle software development. Systems Engineering is responsible for the development of large-scale custom applications and applications packages sold to multiple users. Computer Systems is responsible for the development of mainframe and minicomputer hardware and software with additional responsibility for factory automation software.

Numazu is the main site for the Computer Group. It is responsible for hardware and basic systems software development for mainframes and minicomputers. Approximately 3000 software personnel work at Numazu. The general organization groups are software engineering, development, inspection (Quality Assurance), and a Field Support Center. It is the Quality Assurance department along with the software engineering departments that continue to have an extremely significant role in software development. Led by the Quality Assurance department, significant development process standardization and improvement has been accomplished along with improvements in testing effectiveness.

Culture

Part of the culture of Fujitsu is common with the other major software producers in Japan (Hitachi, NEC, Toshiba). Their software development is described as "factory-like". Common elements are the centralization of most programming operations at a single facility, and factory-like standards and controls that cover project management and product inspection.

Fujitsu is distinguished from the others by the breadth and depth of its competence in computer hardware and software development. They have relied on talented in-house engineers and a careful study of U.S. technology to gain this competence. This strategy of deliberate, independent development has led Fujitsu's rise to become Japan's largest vendor of computer products for small, medium, and large systems. Japanese customers in 1988 ranked Fujitsu first in Japanese-language processing software among all major firms competing in Japan.

Heavy and continuous in-house training is an additional feature of the Fujitsu culture. New employees generally have no computer engineering or software training. Fujitsu invests heavily in training programmers and system engineers how to use methods, procedures, and tools that are standards in the company. Education is coordinated with career paths, which makes it a central consideration for employees. All of this comes together to create a very tight and controlled bond between the employees and the company, both technically and personally.

Product Description

The core of Fujitsu's computer business is the M-Series of mainframe computers. The M-Series has a range of processors targeted at medium- and large-sized organizations in both the commercial and government sectors. Banking and finance are primary vertical markets. Fujitsu's mainframe product strategy is to manufacture and market IBM-like mainframes offering enhanced price performance and system capacity compared to IBM and other rival vendors. Fujitsu' large systems have been directly sold in Japan or to Japanese-owned companies overseas. Over the last few years, Fujitsu has been gradually expanding its direct mainframe presence in Europe. The Basic Software group develops the full set of system software required for the mainframe machines including: close (but not fully compatible) equivalents to IBM's MVS, VM, and AIX operating systems; a relational database system called RDB II; and communications support such as OSI, SNA, and TCP/IP.

The Fujitsu K-Series is a product line of small- to medium-sized minicomputers intended to compete with IBM's AS/400 Series. Fujitsu recently introduced the K-600 Series, designed to replace older K-Series systems, which feature more innovative and faster hardware along with a new operating system, CSP/FX, which is totally compatible with the operating system used on the older K-Series systems. The Basic Software group is responsible for development of the system software for the K-Series that includes: the CSP/FX operating system; FX/RDB relational database management system; communications support for X.25, OSI, TCP/IP, and ISDN; a range of programming languages including C and COBOL; and support for attachment of personal computers.

Release Structure and Strategy

Release schedules are very consistent with the timing of IBM software releases. Mainframe software releases come on 12 to 18 month intervals and are frequently tied to hardware upgrades. Schedules are most often driven by the availability of the hardware. For the K-Series minicomputer software, release tend to come on 12 month intervals. These releases will often be in support of hardware enhancements but will more frequently involve only software function. Competition differences in the mainframe versus the minicomputer markets are the drivers of the differences in software release cycles. Due to the strategy of providing IBM equivalent function, the Fujitsu functions tend to come out later than the IBM functions but on schedules that are predictable to their customers

Achievement of schedules has significantly improved through the years. In the early 1970's, nearly all projects came in late which meant the Quality Assurance received the product after the scheduled time. The percentage late improved to 40% in the late 1970's and moved to 15% in the 1980's. The improvement came through utilization of product handling and project control measures to improve scheduling accuracy. Remaining delays are mainly from late transitions between functional design and coding that result from skill variations or changes to product specifications and designs.

Development Process Overview

The development process utilized by Fujitsu follows a conventional life-cycle model: basic design, functional and structural design, detailed design and coding, unit and combined test, component and system test, product inspection, delivery and maintenance. Control in the past was solely due to the usage of detailed documentation on the system and module designs; programming reports which included detailed information on the design documentation, review reports, and the test specifications and results; and testing reports which detailed the test specifications and results.

Control shifted beyond these structured documents to making quality assurance a part of the formal organizational and job structure. In each of the development stages, actions to remove defects and to eliminate insertion of new defects became the main objective to drive quality. The description of the phases of the process will highlight the usage of both documentation and inspection steps throughout each phase.

Requirements Phase

Requirements for Basic Software are dominated by the strategy of offering IBM equivalent function. For mainframe software, Fujitsu closely watches the IBM functions and follows the IBM releases with their own set of the equivalent function. Since the Fujitsu systems are primarily sold to Japanese customers and are designed for them, the needs of this set of customers causes some functions to be added specifically for the unique needs of these customers. Fujitsu marketing people establish very close on-going relationships with their customers and the flow of requirements is from the marketing people to the development laboratories. Software releases are clearly defined by this set of IBM functions that will be matched along with a set of functions specific to their Japanese customer set. Available development resources determine the extensiveness of the additional functions that become part of the release.

For minicomputer software, there are three major sources of requirements. First is the function being offered in the IBM AS/400 system that Fujitsu has targeted for functional equivalence. Second is the unique needs of the Japanese customer set that the system is designed for as first priority. The final source of requirements for this market is the increasingly important set of industry standards. Fujitsu has recognized the benefits (and needs) of watching the emerging standards that customers are becoming more aware of and that ISVs (Independent Software Vendors) are developing applications software for. The managers of development decide as a team what the contents of the software releases will be.

Basic Design

This is the first design stage and the objective is to take a new function and determine how to handle it within the system. The design does not go down to the module level but instead concentrates on "what" the function is, "how" the major products that make up the system will be affected by the function, and "what" the end-user interaction should be.

Functional and Structural Design

In this phase, the function is broken down into the module structure within the system. Prior to this level of breakdown, the function is defined by the functions that will be handled by each system component. Upon defining the module structure, the interfaces between the modules are defined. Since interface errors are critical and the most difficult to resolve, they are addressed early in the design of the new function. A testing plan is also required as part of this phase.

Formal design reviews are held for each design. Reviews are held with small peer teams. A "phase completion reporting system" was put in-place to capture the results from reviews. Information includes productivity information regarding the number of defects, who reviewed it, and how much time was spent. Functional Design Documents are created that provide the overall module roadmap.

Detailed Design

Tool support is introduced in this stage to support the structured methods used for design and programming. Yet Another Control Chart (YACII) is a design tool that combines aspects of conventional flow charts and pseudo code. Structured conversational Japanese to define the design is input on work stations or terminals. The YACII system contains an editor, compiler, debugger, and documentation generator.

YACII has a special program that translates the design language inputs into machine readable YACII charts that are the input for the coding stage. With the editing features and the common format, the YACII tool (combined with a code generation feature within the YPS tool set) has significantly increased the amount of automated code generation, as well as design and code reuse at the design-specification level. Reuse at the design specification level has allowed the reuse to spread across different languages and architectures. Poorly structured programs have been virtually eliminated by these tools that support top-down, structured design through graphical and text representation.

The documentation generator produces detailed specifications from the YACII design charts. Design-sheet reviews are held during this stage and prior to code generation. These reviews remove a large percentage of the defects bugs and gradually have become the major method of bug elimination. Results from the reviews are input to the productivity and quality tracking tools to do quantitative analysis.

Coding

YACII Programming System (YPS) is the coding tool used in Fujitsu Basic Software. The tool allows the user to edit YACII diagrams generated from the design stage. After completing any editing required, the YPS tool takes the design language outputs and automatically generates "bug free" code from the machine-readable YACII charts. Executable code is generated in several languages that include C, FORTRAN, COBOL, and LDL. LDL (Logical Design Language) is a C-based language developed by Fujitsu for use with YACII charts.

The tool systems helped Fujitsu achieve a number of development goals. Coding errors are virtually eliminated through use of YPS. The volume of new code programmers have to write is minimized through what is termed "common development": (1) writing program design in a computer language or in flow-chart form (YACII in this case) that can be compiled into different languages and stored in a reusable library; and (2) designing basic programs (and even components such as compilers) for use in more than one of Fujitsu's operating systems. Software reuse is strongly emphasized and has been very prevalent in applications and less prevalent in systems software (between two and three times less). Limits exist with regards to code reuse and Fujitsu has found that clear productivity improvement exists as long as personnel reuse 70% to 80% of a particular module or program part without significant changes.

Formal code reviews are held in the coding stage. The source code is reviewed by a small team of peers that include representatives from the Quality Assurance department. Before the addition of the design-sheet reviews, code reviews were a more significant source of bug removal but this has lessened as design reviews have become more effective. Review results are input into the productivity and quality system.

YPS has an additional feature that is an aid to developers. Reverse generators are part of the tools and allow design-charts to be produced from the source code. Developers can edit the design charts more easily than source code and can store the charts in a data base for later retrieval.

Testing

Fujitsu's objectives in testing are to detect 65% of the bugs in a new program by the end of the coding phase and 95% by the end of system test. They want to remove as many of the remainder as possible by the end of the inspection process following system test. As the complexity of products increased, the combinations of factors and conditions became so large that they were impossible to test completely using old methods of test case generation. To overcome this problem, the company used two approaches. The first was to develop tools that automated as much of the testing as possible, and the second was to create a data base on test-factor analysis methods and conditions to help create test cases for new functions.

Testing is carried out in the development organization and the independent quality assurance department. Automatically generated test cases are used by both groups. The total testing phase is described in the sections that follow.

Unit and Combined Test

Code developers are responsible for execution of unit and combined test during the coding stage. The YPS tool has very extensive testing support used for this and subsequent stages. YPS includes executors, test coverage evaluators, and debuggers that can be used on the YACII charts. Through usage of these tools, developers are able to determine the comprehensiveness of the testcases, collect and analyze data, and debug the code.

Tests cover a very high percentage of the code paths and validate the logic, limits, and boundaries of individual modules. The modules are combined into groupings of new function and tests are run on these groupings. During both these phases, the main concentration is validation of the design and code, not a validation that the requirements were met. Analysis of results is done to determine code path coverage which is one of the criteria for test completion.

Component Test

Before the code is integrated into the base set of code used for system test, component test is executed. This test utilizes large combinations of the new function integrated together. A set of new testcases are run to validate the new functions are executing properly. The other key part of the test is validation of the existing function through a set of regression tests. Once this set of new and regression tests have been successfully executed, the code can be integrated into the base set of code used for system testing.

System Test

This is the phase where validation that the software satisfies the functional requirements occurs. Documentation from the design (YPS generated) along with the requirements statements are used as a guide for the testing. A set of the automatically generated testcases, a set of automated testcases stressing functions from prior releases, and new tests created by the individual validation people are run in combination. Customer-like environments are simulated to provide a representative set of stress situations for testing.

Performance testing and reliability certification are also part of the test. Specs exist for performance in stress and non-stress environments. Reliability is measured in mean time between failures (MTBF) and is measured as the test progresses. Bugs are closely tracked and getting the bug level per module to an acceptable range is one of the criteria for exiting system test.

The Quality Assurance department maintains a powerful and independent voice in deciding on whether the product is ready to ship. Their monitoring of development progress begins during the design phase and continues through system test. Sufficient defect removal is critical to the ship decision and can be analyzed using a combination of the in-process measurements and historical data gathered from prior projects. When quality is considered sufficient (meeting specs that were established at the start of development) and when requirements have been validated, the ship decision is made by the QA department.

Product Inspection

Product Inspection is a formal step in the process that was added in 1971. Fujitsu requires completed software and manuals to pass a formal inspection process before release to the customer. Product-handling procedures require that the source code, manuals, and other documentation must be brought to the Quality Assurance Department together and that they be consistent with the information contained in them. No product can receive a registration number, which is necessary for release to customers, without meeting these requirements and gaining the formal approval of the Quality Assurance Department.

Delivery and Maintenance

The YPS tool facilitates maintenance of the systems. By minimizing coding errors, the amount of maintenance has been reduced by one-third between 1977-79, one-third more between 1979-82, and by 1985 the bug levels for outstanding code were nearly zero. When maintenance is required, the development group is responsible for analyzing the problems and making the changes. The capability within YPS to present the module design versus just the code allows new people to more simply understand problems and also allows experienced people to be quickly updated on the module design and function (instead of having to study a large amount of code to understand the overall design). Through the use of reverse generators, the source code can be used to produce design charts that can be edited instead of the code. Updating of documentation after maintenance changes are made is required. Tools exist to help this get done.

Systems are in-place to report defects and track resolution of them. Customers can report them directly through the ITS product or can report them through Fujitsu marketing and services people.

Process Usage and Compliance

The development process described is standard throughout the Basic Software group that is responsible for all systems software. Variations within a sub-group are rare and do not happen on major projects. Much work is done to keep the process, and tools to support it, state-of-the-art. Usage of a design language also locks people into a structured approach to coding and testing.

Compliance comes through culture and a strong Quality Assurance department. Culture is a factor because people do not question the process and just go out and execute it. Quality Assurance also monitors progress through the development phases and is very active in making sure steps are not skipped or improperly done. Having the final ship voice gives the Quality Assurance department a good power base to operate from.

Project Management

Improved project management was one of the areas Fujitsu invested heavily in to reduce the number of projects coming in late. The central tool for this has been the Basic Software Project Management Support System (BSMS). The tool enforces structured procedures for project initiation, execution, and completion. Support in the tool covers project management, development procedures, schedule estimation, monitoring of the development process, and testing and inspection aid.

Initial project steps are to apply for a product number which is approved by superiors in the organization. A budget must be submitted and development and planning documents must be created that include the following information: market survey, functional design specs, performance design specs, reliability objectives, manpower estimates, and machine time estimates. In total, these give a very clear picture of what is being created and how much it will cost to build.

Development and planning documents are then used to track progress against the plan. Items tracked against the initial estimates are: actual work hours, actual computer time, and phase completions.

Additional project management support is provided through the Generalized Program Editing and Management Facilities (GEM) tool. GEM supports direct measurement of subsets of the project that can be at a group or even individual level. Automatic generation of graphic reports can be done for groups and individuals. Productivity (lines of code developed), progress status (by month and week), and quality (defects per module) are reported through system usage. BSMS is the major tool currently used in Fujitsu, but GEM still has functions of use.

Change Management / Configuration Management

Change management is enforced throughout the development process stages.

Starting with the designs in YACII, library management functions via the GEM tool manage versions of projects under development and existing projects. When the code is generated, the history of the module within GEM control continues to be updated. For system test (and previous) driver builds, the GEM database is used to pull in the integrated modules that make-up the base set of code.

Bugs are closely tracked during development and testing. Tracking ensures that changes made to the system are due to reported defects, and also ensures that all reported defects are resolved. The change management extends to maintenance changes. Modules contain a track of all maintenance changes made. This track is enforced and supported by the GEM tool.

Metrics

Metrics have been a constant part of the process changes introduced by Fujitsu. The Quantified Management System was created as a tool to capture the usage of metrics by management throughout the development process. The company quantified basic measures of performance and then introduced standardized procedures and tools. The goal was to help developers build in quality at each phase rather than rely on inspection at the end of the process. Some highlights of the usage of metrics are:

Quantify and then analyze information on project progress and product quality. Instead of just comparing completed work to initial estimates, they used a two step process: measure quality of work-items by the number of points reviewers gave them in a structured set of review steps, and measurement of how well the results compared to the estimates. This approach allowed managers to define tasks in small realistic segments, and minimize estimation errors.

Utilize methods to predict error occurrence in the programming phase. A management by objectives process was introduced to ensure error correction was completed.

Determine test items through the use of statistical Design of Experiment techniques. Using educated guesses of where problems were likely to occur, tests could by generated to cover quality risk areas.

Quantification of the concept of user friendliness was done. Manual reviews and inspections, along with surveys to validate the accuracy, are done for new functions. The functions are evaluated on basic product characteristics (performance, reliability, quality of information, price, etc), design quality and consistency (degree to which software products and specs met user needs), and sufficiency and attractiveness (extent to which products are acceptable to users).

Design defect and test defect tracking is also done. During the test process, defects are recorded and tools are available to analyze things such as total backlog, open versus closed ratio, arrival rates, and additional statistical measures used to determine the overall quality and the stability of the code. Fujitsu had relied on these testing steps to remove 85% of the defects in the past, but emphasis has been on removal of the defects before the test phase is entered. The test process has remained as rigorous as in the past, but has to deal with a smaller number of potential problems.

Process Improvement

The evolution of the Quality Assurance department work at Fujitsu has been a story in process improvement. Quality Assurance has been a significant driver of controls for product, process, and quality. A Total Quality Control program was instituted in Fujitsu and also extended out to all subsidiaries and subcontractors. There are three key elements of the program, with two additional items also being very important.

Quality Assurance Through Organization: They emphasized quality control as part of the formal organizational and job structure for every employee, and especially those in design and planning. This function is not relegated to the Quality Assurance Department to be done through testing at the end of the process. The Development Planning and Report System is used to monitor quality.

Diffusion of Inspection Ideas Through Horizontal Development: This stresses the utilization of regular meetings between different departments and projects to share data on defects and other quality information. The effort is towards spreading good thinking and practices regarding quality control.

Advancement of Quality Concepts: This involves lectures, workshops, and measures dealing with interpretations of product quality that extends beyond zero defects to characteristics such as product design, function, performance, ease of use, and maintainability.

Quality Groups: They use small groups that meet once or more a month to discuss a range of issues related to productivity and quality as well as work conditions. The small groups are also used to supplement formal training required of new company employees.

Tools: Throughout Fujitsu's development evolution, tools have been an effective way of assisting changes to improve quality. Extensive investments have been made in using tools to help the process versus defining the process around the tools used.

Tools

As is evident in the descriptions of the development process, Fujitsu has made a significant investment in tools for the development process. They made an extensive investment in tools to facilitate the transformation of designs into executable code, and the reuse of designs and code. Investments were also made in tools to support project and library management, product maintenance, and testing. Throughout the process of tool creation, Fujitsu's philosophy has been to use tools to help the process versus defining the process around the tools that were available.

Process Education

Historically, new employees had little or no computer engineering or software training, although this has been changing in recent years as Japanese universities add courses on information technology. The company invests heavily in training and teaches programmers and systems engineers how to use methods, procedures, and tools that are standard. To emphasize the importance the company, and as a result the employees, place on its education program, career paths are coordinated with education. Much of the first year is spent on education, and it continues at a rate of about 8 days of training a year in workshops, with self-study materials and quality-circle activities occurring in addition to the 8 days.

The education is geared to the process steps the individuals are responsible for. Initial education is centered on coding and development testing. Subsequent training tracks the employees progress towards module and structural design, with progression to project management requiring training on estimating and quality assurance. The key to all of this is providing the process education necessary for the activities the employees are responsible for.

Summary -- Results

Process standardization along with the quality improvements described have led to apparent rises in quality, productivity, and schedule control. The percentage of late products dropped from near 100% to 15%. For all code in the field, bugs reported by users dropped one-third from 1977-1979, and another one-third from 1979-1982. Improvement since 1981 has leveled off and remains at about .1 per KLOC which is very low. Productivity, which is difficult to measure and compare between groups, showed continual improvement during this period.

Customers have also viewed Fujitsu's results to be very good as evidenced by the Nikkei survey results that placed them at the top in Japanese-language software. The company provides an excellent study of what can be accomplished with rigorous procedures and tools to support them. Depending on the quality of the requirements stage (ensuring they are producing the correct things), the company can be competitive in what they deliver.

Software Factories has been a label applied to these large software producers in Japan. The Basic Software group at Fujitsu did not directly apply this label to their organization, but the operations of the group did utilize the following set of concepts that define the "software factory" label:

Centralized programming operations at a single facility
Adoption of rigorous standards
Adoption of rigorous controls
Quantification of management variables
De-skilling of testing
Standardization of methods and tools
Reusability of designs and code
Automation of development and management

Chapter 6: Hewlett-Packard[24]

The Company

Hewlett-Packard was founded in 1939 and incorporated in 1947. Corporate headquarters are located in Palo Alto, California. Development locations are located both inside and outside the United States. Marketing organizations are located in 103 countries worldwide. Manufacturing is done at multiple worldwide locations.

Hewlett-Packard manufactures and provides computer products ranging from PC-compatible desktop systems and laptops to super-minicomputers. The company has a strong technical orientation, both in its internal structure and vertical market emphasis. Its reputation for quality and service has earned it a rating of "most respected company" in national polls in the United States. HP is a founding member of the Open Software Foundation (OSF). Its Open Software Environment Service is committed to helping users move from proprietary to open systems and draws on HP's proven strength in providing technical support.

Sales results continue to be positive for the two systems that are part of this case study. Overall demand for RISC-based multiuser systems and workstations was strong in 1991 and 1992. The HP 3000 product line, which runs the proprietary MPE operating system, experienced a 5 percent increase in sales revenues for 1991. The HP 9000 Series 800, which runs the HP UNIX version operating system, HP-UX, had a 51 percent increase in sales revenues for 1991.[25]

Hewlett-Packard Organization

The HP 3000 and HP 9000 systems divisions are both part of the Computer Systems Organization. The HP 3000 is managed by the Commercial Systems Division and the HP 9000 is managed by the General Systems Division. These groups have full profit and loss responsibility. Product marketing functions are also part of these organizations and have overall responsibility for keeping 5 year product plans in place and for managing the product rollout to the worldwide sales units.

Development is managed slightly differently for these systems. Core Technology Organizations have responsibility in HP for developing I/O hardware, I/O software, processors, network software, and the UNIX operating system that can be used on multiple platforms. The organizations contain the developers and designers of the hardware and software. The HP 3000 (CSY) and HP 9000 (GSY) use matrix management with these organizations to develop the support needed for their systems. Since the MPE/iX operating system and the database support are both proprietary for the HP 3000 system, the management of their development is not part of the Core Technology Organizations and instead report directly to CSY management.

Culture

The Hewlett-Packard culture centers around a very strong technical orientation. The base of the company is engineering and the software developers are considered to be "software engineers" versus "programmers". The individuals take a very analytical approach to the work they do. This analytical approach is a critical part of how they manage development projects in the company. As you will see in more detail in the metrics section, the development projects are run by metrics and you can hear the statement made by many people that "the company is run by metrics". The metrics become a guide for the actions that must be taken to deliver on a certain schedule with an acceptable level of quality.

The company also has a perfectionist attitude. The individuals within the company and management strive to do everything "the right way". The attention to perfection has allowed the company to become more structured in their development of software. Quality data was used to show the improvement in software defect rates that could be gained by utilizing more rigor in the design stages and by doing formal code inspections during the coding stage. Because of strong individual commitment to perfection (defined as software quality in this case), a bottoms-up implementation of the formal reviews happened within the development community. Any attempt at dictating these changes and forcing individuals to "comply" with the processes, would have resulted in failure. The phrase that summed this up nicely was, "Commitment is in at HP, compliance is out."

On top of the "analytical approach" and "perfectionist attitudes", there is a strong emphasis on individual creativity. Developers are allowed and encouraged to come up with ideas on how to improve their products and to make the changes (making sure they obtain "buy-in" from the release management team). The analytical and perfectionist characteristics dominate the company, though, and the culture was summed up nicely by one person I interviewed. The statement was, "HP people have an analytical approach and a perfectionist attitude, but they work hard to appear informal".

Product Description

The HP 3000 family is Hewlett-Packard's well-established midrange computer series. The series consists of the Micro 3000 line of products designed for desktop or small workgroup computing, the 5x/7x line of proprietary-CPU systems, and the Series 900 products that are based on HP's new Precision Architecture (PA-RISC). The 3000 series features a tightly integrated hardware/software design and is upwardly compatible at the software level from the entry-level systems through the top-of-the-line systems. Two different versions of the operating system actually run on the Micro 3000 and 5x/7x lines (MPE V/E) and the Series 900 line (MPE/iX), but they remain compatible with applications software developed for the earlier systems and use the same interfaces and peripheral devices. The company has stated that further updates to this product family will be on the PA-RISC platform. The HP 3000 family is typically used in general business applications such as office automation, education, and financial management.

HP 9000 products are UNIX based computers that run Hewlett-Packard's HP-UX version of the UNIX operating system. The series consists of the Series 300 products, the Series 700 and 800 products, and the Apollo 9000 Series 400 systems. These UNIX-based systems are what HP is attempting to expand its user base with. HP-UX offers a variety of programming languages, including C, Pascal, Fortran, and COBOL along with support for functions like sort/merge and file management. As is common with many UNIX-based systems, extensive CASE support for software development is offered via mainly third-party vendors.

Release Structure and Strategy

The development goals are to provide 2 operating system releases per year.

The releases are either required or optional product releases. Required releases come out about 18 months apart and must be installed by all users. The optional releases come out every 6 months between required ones, and customers can choose whether to install them or not. To meet the goal of simultaneous development of 2 releases, HP has gone to parallel development of the releases using the same code base. An extensive revision control system has been developed and installed which allows any developer to change code from the formal code base. This parallel processing of the code for the releases has gone through some refinement and has been the major focus of the tools group over the last years.

The releases are date driven versus content driven. The company uses a concept of a "release train". Function that is ready and can meet the delivery date of the release can get on the train, while functions that can not make the schedule must get off the train. The train runs on a specific schedule which puts the releases under more of a delivery date base than a functional base (i.e. wait until a specific set of functions are complete before shipping the release). Groups are expected to commit to a schedule and make it. Dependencies between functions are generally the key to pull different functions together on the same schedule for a release. There are some independent functions that just come in whenever the release with the appropriate schedule comes by.

Development Process Overview

Each division within Hewlett-Packard owns their development life cycles for the products they develop. They are free to choose different process variations for each of the products. Even with this choice, all processes look remarkably similar across the divisions and products even if everyone does not recognize it as a "standard process". If not considered a "standard process", it can at least be considered a "consistent process".

A formal Life Cycle Document is used across Hewlett-Packard with each division having their own individually created document. The "life cycle" is a key concept in HP and is used by the divisions to manage their businesses. The basic model for the life cycle is the IEEE model of the standard software process. The division's document defines phases and checkpoints. In general, each phase has a description of the work products for the phase, entry and exit criteria, and the outputs of the phase. The overall development process is recognized as basically a waterfall approach to software development with HP making "go-no go" decisions at defined checkpoints.

Key requirements for a release are determined along with a schedule for when the release should come out. The key requirements are determined by an entrepreneurial and contracting process involving functional, strategic, resource, and schedule negotiations. The central point for this work is the Phase Review Process that will be described later in this case study. For purposes of this discussion, the Phase Review participants are the executives from the HP divisions. During this top-down view of projects, key software and hardware functions are brought together on an agreed to schedule. Preparing for the Phase Review includes a bottom-up process of estimating resource and schedule requirements for the functions. Through the use of this information, the executives can make the initial set of top-down release content decisions. Agreements are then reached among the development groups, documented in the Quality Plan, and from that step on the process is driven by metrics. Each of these key steps will be described in detail in the following sections.

Quality Plan

The Quality Plan is the critical development starting point in HP. The plan is created for all major development efforts (each release is considered a major development effort along with some significant cross systems functions such as POSIX support). Each of the development groups involved with delivering function for the development effort participate in writing the plan. Plan creation is overseen by the systems division and owned by the development manager responsible for the overall product. The manager is generally at the systems level, CSY for the HP 3000 and GSY for the HP 9000. Participants in the writing include marketing, laboratory (development), test, and quality assurance.

Corporate standards define what must be in the plan. Measurable objectives for the product are contained within the plan. It defines the expected results of the process steps, checkpoints, and criteria. More specifically, some requirements at the development phases that would be part of the plan are: what must be reviewed in design and code phases, percentage coverage for design inspections, entry and exit criteria for each development phase, MTBF (mean time between failures) quality commitment, path coverage for testing, the amount of automated test coverage, how items that can not be handled by automated testing will be covered, and specific checkoff items for the phase exit. Metrics are an additional part of the plan with a description of how each metric will be reported and who it will be reported to. Metric owners are expected to manage the commitment made for the specific metric.

The usage and content of the plan have been heavily influenced by QFD (Quality Function Deployment) principles. The clear definition of what will be done in each phase, along with defined and measurable entry/exit criteria, are central to QFD and the HP Quality Plan. The final milestone that is defined in the Quality Plan for development is Manufacturing Release. The criteria for Manufacturing Release must be met before the product is allowed to go out for customer usage.

Requirements Phase

The requirement phase for a specific system release occurs during an 18 - 24 month period prior to the release date. Planning is mainly a question of what marketing and the chief development groups wish tc see in the system 24 months from the point in time they are at. The main objective of the planning is to determine the functionality for the major release. Much of the front-end planning involves stabilizing the development plans of the core components: networking, operating system, languages, and data base. Each of these separate groups requires plans that satisfy cross dependencies.

Requirements come from 2 different sources. The first source is the product marketing departments. These departments are focused on different industry segments in the market. They utilize a combination of direct customer input and marketing organization directions to determine the key requirements needed in releases over the next 5 years with particular emphasis on the next 18 - 24 months. The marketing departments involved with this work are part of the development organizations and work with the direct sales force of HP. The second source of requirements is internal directions determined by the technical groups in the company and by executive commitments. Internal directions include things like POSIX compliance, OSI communications support, and RISC-based processors. These relate to strategic directions that are felt to be important to the system along with product deficiencies recognized by the technical groups. All requirements must be formalized before being brought forward. Requirement formalization is done as low in the organization as possible, with determination of the specifics of the requirement generally done by engineers. The desire is to get the engineers more focused on what the customer really wants. The market need is determined and defined, costs to develop the function are estimated, and revenue is estimated. Requirement formalization is required to at least be completed by the Commit Checkpoint which is the point where functions are formally included as part of the release.

The requirements that will be part of a release are decided at a high level in the company. The General Managers of the systems divisions (CSY for HP 3000 and GSY for HP 9000) are the final decision makers. The computer division General Managers are responsible for each of the platforms and approve the content decisions to drive profit and loss goals. Release content is controlled via the negotiation process for resources. The division General Managers provide funding to both the internal division development and the core technology divisions based on release content. Adding content will generally warrant additional development resources.

Design and Architecture Phase

Design and architecture work is done by the people that do the actual software development. HP likes to keep ownership with people all the way through the development of the product to keep knowledge and commitment high. There are some "system designers" who are senior level experienced people in HP that assist in the design phase. Assistance usually is provided for complex cross-system functions, such as POSIX compliance throughout the system.

No formal design methodologies are used. Few specifics are required regarding steps that must be done during the design. Prototyping is used some but is not extensively used at this time. The main mechanism utilized is formal design inspections which are required at the end of the design phase.

Formal design reviews cover external specifications and internal specifications. The external specs detail the user view of the function (user interfaces and application programming interfaces). Internal specs detail some of the major system items such as the system functions used, dependencies, and performance. One major difficulty HP recognizes during design is ensuring that all the "right" groups and people see the design. The most difficult part that comes up during the design phase is dependency management across the functions that are part of the release. Cross scheduling of function delivery must be done to allow testing and integration to progress.

Coding Phase

Coding for operating systems projects at HP is manual, with methods somewhat open to developer discretion. Code generators are not utilized in these development divisions. Standard compilers and linkers are used which enforces a level of consistency within the development groups. A consistent set of development tools is also utilized, which has assisted the standardization of code development.

The most significant change to coding in HP has been the implementation of a formalized inspection process. The impetus for this change was a causal analysis of the problems that were being discovered in the code that went to customers. It was found that coding was a significant problem area due to an abnormally high level of defect injection. Overall usage of the design and code reviews has come about over the last 3 years with it becoming required in the last year. The formal code inspections are now held with a moderator and 3-7 reviewers.

Implementation of the inspection process has been viewed as a success. The effort for reviews is high, but results have made the investment worthwhile. Customer defects have gone down, the number of dependency problems between code modules has been reduced significantly, testing is completed in less time, and the number of integrations required to pull the system together at the end of the process has been significantly reduced from the 50-60 that were previously required. The net has been a reduction in the amount of rework required during the backend of the process.

Development Test

The development test phase involves two types of testing and is executed by groups within the product development organization. The first type of testing done is a functional test. It is a very deterministic test that focuses on boundaries and limit testing of specific code sections. The developers test their own pieces of code and make it work within the system. There is some combining of new pieces during this phase, but it is limited and arranged between developers. For wide scoped functions, developers combine the new code early according to test plans that exist for testing these combined pieces.

The second type of development testing is performed by a separate group within the development organization. This group runs a stress test of the new function. All the new functions are combined together and old (regression) and new tests are run against the system. Workload generating applications are run to load up the system so that the functions are tested on a system under stress versus a clean system running only the one set of functions. Development test is run before the system moves on to the system tests run by the independent verification group.

Independent Verification Testing

Independent Verification testing is the backend of the overall development process. It amounts to a 4 - 6 month process that brings all pieces of the system together. The test cycles utilized are: system integration test, system test, Alpha test, and Beta test. Independent verification testing is done by a group that is separate from the development organization. It has been independent for many years and evolved to that role after initially being part of the development organization. The group continues to be impartial and maintains the power to stop a release if standards are not met. Development organizations respect the group due to the hands-on testing they are responsible for plus their level of analysis of the overall system.

System Integration and Test

System integration is the step where all new code is integrated in with the unchanged base code. The integration includes the new code from the operating system, networking, languages, and data base development groups, along with code from peripheral products like office.

System test consists of two portions that are mainly distinct timeframes in the test: Test Cycle 1 (TC1), and Test Cycle 2 (TC2). Defect and reliability objectives are defined for TC1 and TC2 in the Quality Plan. The group runs a variety of new tests and regression tests. Formal Test Plans are created early in the development cycle that define the tests that will be run against new function, the amount of automated testing that will be done, and how functions not covered by automated testing will be verified. The test plan also defines the regression testing that will be executed.

TC1 is concentrated on stabilizing and validating the core parts of the system such as the operating system itself, networking, languages, and data base. A thorough verification of the new functions along with regression testing of existing function is run until criteria are met for test completion. Criteria are what causes the test to be complete, but there is a clear schedule target for the end of this Test Cycle.

TC2 is concentrated on bringing peripheral functions together with the core parts of the system. Functions like office are introduced into the testing at this point. As with TC1, the test is executed until criteria for test exit are achieved. Criteria also determine when the system can be moved on to Beta Test.

The group is impartial and will stop a release from going to customers if the test standards have not been met. Testing skill along with this power to stop a release causes the Independent Verification group to be well respected in HP.

Alpha Test

For these HP systems groups, Alpha testing is done via internal "self hosting" tests. HP uses their own systems for development of their system code. When a function completes development and early testing, clusters of users will agree to take the new level of code. This allows the function to be used early in the development process when fixes can be made much easier. Alpha Test starts at the tail end of TC1 but before TC2 begins.

Beta Test

Customer Beta Tests have been run by HP for 15 years and are a regular and important step in the development process. HP uses a very conservative approach in selecting customers and in the level of code given to them. The HP Product Support area picks the customers that will be a part of the test. Support works with the Release Team to determine the major functions in the release and then find customers that are interested in that functionality. The customers are signed up on a "case by case" basis dependent on the match.

The code given to customers is at Manufacturing Release level quality. After 4 - 6 weeks of Alpha Test, the system is frozen and Beta Test is started. The code is kept in Beta Test for a specified time period (the period is defined in the

Quality Plan), and then it is moved to mass customer release if Beta Test results are positive.

Maintenance

Maintenance of existing code is done by the development group. Customers can purchase various levels of system maintenance that goes all the way to 7 day/24 hour support. HP does this 7 day/24 hour support by "following the sun" where responsibility for handling calls goes to the area of the world that is in their normal workday. An on-line system gets reported defect information to the necessary development team. The routing of these problems in development is initially handled by a "first line team" for the hottest calls. The team responsibility is to ensure the problem is fixed expediently.

Process Usage and Compliance

Compliance to the process is determined by the metrics during the actual development. In-process metrics are defined in the Quality Plan for the different development phases. Each metric is owned by an individual identified in the Quality Plan. This owner is responsible for managing the metric results. As they track the metric, deviations are determined and the groups responsible for the deviations are expected to put plans in place themselves or within follow-on groups to bring the metric back within criteria. The metric owner is responsible for making sure this happens.

Process differences do exist between development groups. Differences between the process used for MPE/iX and HP-UX have evolved through time. HP-UX started with the MPE/iX process as a base and then determined process areas that needed improvements and focused on change in those areas. These have not been significant for HP-UX. Each organization has a group looking for quality improvements which comes up with ideas that change the process through time.

Release Management

Hewlett-Packard utilizes a designated individual as a Release Manager along with a group called a Release Management Team. A Release Manager is used for each software release and is the person who development and other groups utilize to ensure a decision is made in a timely manner. It is a form of matrix management since the person does not directly manage any of the development organizations but does make decisions that affect each of these groups. The release manager concept started on a smaller basis for subsets of the system (networking, languages, etc) but during the last 2 years has been expanded to overall responsibility for the release. Due to the constant pressure of "fighting fires", the job involves an extremely high level of stress.

Much of the release manager's work is done in conjunction with the release management team. The team is assembled from across the system with one individual from the operating system, networks, languages, and data base development groups. This team is responsible for making some of the overall content, schedule, and coordination decisions across the total system. The group coordinates between the releases under development by ensuring code changes go forward to keep the parallel development across the releases coordinated. This role of making sure code changes go forward is a major effort and responsibility which is done as parts of the system are determined to be stable enough to roll forward to the next release. Trouble shooting from a system perspective is also occasionally a part of the team's work.

Change Management / Configuration Management

HP has a configuration management process in place for the full development process. The formality of the process is dependent on the centrality of the system component, the type of change requested, and the timing of the change. The release management team described in the previous section has a central role in configuration management. Group involvement starts early during the planning phase to influence the planning across the multiple releases being considered. After being involved in the planning stage, the group maintains on-going responsibility for changes throughout the entire release. Requests for changes throughout the process follow the same format. The group wishing to make a change documents it via an electronic memo to the release management team and then the team indicates approval or rejection through the minutes distributed from their meetings. The approval or rejection decision is gained by the team members polling all development groups impacted by a change. Tools are not used to enforce the configuration management, but the process is well-established and changes are not made until agreed upon.

Requirements changes, including requests for new function, must be approved by this group, with the development manager responsible for the overall release becoming involved in these more global changes. Design and code changes are also managed, with code modules not allowed to integrate into the system until they have been formally approved. As mentioned, the formality changes due to timing during the release. In the late stages of development, the system goes under "formal change control". Formal control means the level of management signature required to make a change increases.

Two stages of the process where the more formal process takes place is in "code freeze" and "problem review". Code freeze is used to make people think about the real importance of a code change before going forward for approval. There are blackout phases before component level and system level test where absolutely no changes can be made due to stability requirements for the code.

Outside of that, it is controlled by the level of management signature. The other stage of the process where control is very formal is after the entry into system level test. A problem review team looks at all problems and determines which things should be fixed. This process is very formal with only approved changes allowed into the system.

Though tools are not in-place to enforce configuration management of individual changes to code, there is a level of coordination done via the tools. Revision control is handled by the tools through a check out procedure from the source library. A single version of the code is kept at the source and ensures that developers access the latest level of system code available. The code is "fanned out" to the developers' systems periodically to ensure any testing done on their systems has recent code. This revision control system is not designed to and does not prevent multiple people making changes to the same code module for a release.

Metrics

Metrics are a critical part of the development process at HP and prompt statements such as, "once we have the content defined and agreed to, we let the metrics manage development of the release". In the Quality Plan, each metric is assigned to an owner in the organization who is responsible for meeting expectations set for it. Metrics are compared to standard expectations that have been derived from historical data and are also compared to the results of the previous release. When metrics go outside of the guidelines set, actions must be taken to recover. HP has historically taken those actions.

The common groups of metrics center on defects of various types, code turmoil, and coverage percentages during different development phases. In-process metrics are a clear part of development and post-release metrics are also utilized to determine the quality of code developed. Representative metrics are:

Design inspection coverage (% of code)
Code inspection coverage (% of code)
Test coverage (% of code)
Automated test coverage (% of code)
Test plan coverage
Defects during review stages
Defect backlog, incoming, and resolved during test
Defects by level (3 general groupings of Critical, Serious, and Low)
Code turmoil (how much has been touched during a given period)
Post-release defects

Process Improvement

Process improvement is driven by the individual developers. HP has worked to support continuous process improvement activities as part of the development process. This has fostered internalization of the discovery process for new process steps versus "doing what the Quality and Process group requires". This approach was necessary from two standpoints: quality improvement is more effective when done from within, and the HP culture is adverse to compliance as a method of doing things.

Post mortem analysis is used extensively within the company. Releases are tracked and analyzed to find the root causes of problems that occurred. The results of the postmortem analysis are used to drive changes and are also reported through management. Quality reports are also generated by corporate and group staff organizations and serve as another reporting mechanism that can generate process actions. The final input on quality comes from the Product Support Organization which tracks and reports the results for existing releases. The development lab does not dispute the results reported by the support organization and instead analyzes the results to determine the appropriate acticns to take. Results might indicate pervasive problems in an area of the system that will generate plan items for an upcoming release.

The other sources of process improvement ideas are the CSO (Computer Systems Organization) and Corporate Quality and Technology Groups. Ideas are brokered by the CSO group which introduces them to development groups who might implement them. The CSO group acts as a support organization to these development groups. After seeing the results, the development group and the CSO group will spread information to make other development groups aware of the results. Sources of ideas for the CSO and Corporate organizations are from outside the company as well as other divisions of the company. Assessments are also used to determine change areas or to support suggested changes. Some assessments that have been done recently are SEI Capability Maturity Models, TQC, and internal ISO 9000 assessments.

A final method of bringing process changes to HP is via major projects. In these projects, there is up-front planning of new process variations that will be brought in and used throughout the development organization. These changes are agreed to by the individual development groups and are documented in the Quality Plan. Improved quality goals, shortened cycle time, or improved engineer productivity are the normal drivers causing process changes. This method gives HP a level of planned innovation.

Tools

Software development is done on the systems the software is being developed for. Common compilers and linkers are used on the different platforms. There are no design languages used that lead to automatic generation of code, all coding is done manually by the developers. The HP 9000 and HP 3000 developers all have workstations at their desks. HP utilizes a checkout system where the base level of code is kept in a single location, and individual modules must be checked out by developers wanting to work on them. To keep individual development systems at current levels of code, a fan out mechanism is used to send periodic code updates to the systems.

Through the last years, the HP 3000 tools group has done significant work on a revision control system to allow parallel development of 2 releases. An item they are looking to include in the support is a way of collecting decision criteria to understand why certain code changes were made, which is becoming more critical due to parallel development. Software Configuration Management standards are being used as a guide for improving the revision control system.

Process Education

Formal process training is not required in HP. There are a range of classes available that include inspection training and other leading topic education. Over the last couple years, the HP-UX originated the usage of development templates. The idea has spread to other groups within HP. The templates are available for the overall development process and each of the individual steps. Information on the templates include advice for the step, checklists of recommended activities, and descriptions of pitfalls to watch for. The templates have become extremely popular as an effective device for sharing experiences and suggestions.

Business Life Cycle at the Division Level

As described in the Requirements Stage of the development process, the Hewlett-Packard business is managed at the Division level. The HP 3000 and

HP 9000 both have a management team responsible for managing a portfolio of business costs and choosing between many good ideas presented in the product requirements. The division manager and staff make the final decisions on what big ticket items should be pursued.

Multiple development labs are part of each division and will be affected by the decisions on what projects to pursue. Projects are collections of functions in a release and the management of projects is done at the development management level in the divisions. Multiple investigations must be coordinated to determine how to get the multiple projects out on time for the release.

Stages of the Business Life Cycle represent checkpoints at the project level. The stages of the Life Cycle are:

Proposal Signoff - Occurs after the development proposal is generated. The development proposal is based on customer demands/needs and is written in terms of how the customer demand will be met. The results from this stage is approval to commit resources for Investigation.

Investigation Signoff - During the Investigation Stage, the details of the customer demand and the product response to satisfy it are fleshed out. Elements that are critical in this stage are the time dependency and feasibility assessments that come from the investigation. Costs and schedules are roughed out during investigation. The result of approval in this stage is moving to the Design Stage.

Commit to Development Signoff - During this stage, the design of the technical solution is completed. The internals of the solution are laid out. Items completed during this stage include formal cost estimates, determination and agreement to meet dependencies, and the development schedule. The result of approval in this stage is moving to coding and testing.

MR (Manufacturing Readiness) Signoff - At this point, coding, unit testing, and system testing have completed and the development group is ready to release the product. Signoff indicates that quality goals have been met and that customers can take the product at this point. It then moves to mass production.

Post Review Signoff - This occurs 6 months after the product has been released to the customers. The review is held to determine if the product has met commitments for quality. Any unique problems are also analyzed during this review including performance problems, any major customer concerns. Signoff indicates that the product has met commitments.

Phase Review Process

The Phase Review Process is used at the executive level within HP. Specific executive reviews are held to review the status of phase review checkoff criteria. The criteria are pre-defined and meeting criteria is required before development can proceed to the next phase. The reviews help keep executives aware of progress on major programs and gives them the ability to provide input on big ticket items.

Outcomes beyond proceeding to the next stage can include adjusting priorities of the computer divisions through funding. Funding decisions between divisions may be escalated to the executives through phase reviews, or the executives may use it as an opportunity to adjust funding for multi-year big ticket projects. Programs that are considered major or "big ticket" are programs over $5-10 million in development costs (which covers most major hardware items and most software releases), significant strategic items (most major hardware also falls into this category), and key long-term technologies (RISC architecture).

The phases in the process are:

Phase 0 -- Requirements / Plan -- The objective is to ensure that requirements are consistent with worldwide strategic, annual, and systems business plans. Business and technology with strong leverage potential are identified. The result of approval from this stage is organization resources identified and committed to a plan for studying alternatives for the requirements agreed to.

Phase 1 -- Study / Define -- The objective is to select competitive alternatives that meet worldwide contribution objectives for the systems. Measurable system objectives, release criteria, and verification/validation processes are defined. The result of approval from this stage is organization resources identified and committed to design.

Phase 2 -- Specify / Design -- The objective is to complete the cross-functional plans for implementation and delivery of the specified functions. Organizational resources are identified and committed for system development, release, and support. The result of approval is movement to the development and testing stage.

Phase 3 -- Develop / Test -- During this phase, the development and testing of the functions are completed. Actions taken due to completion of this phase are authorization to publicly announce price, performance, and availability of the product. Qualified customer shipment/access may also be approved.

Phase 4 -- User Test / Ramp Up -- Test results are reviewed during this phase. Approval is given for unrestricted shipment to customers. This denotes Manufacturing Readiness.

Phase 5 -- Enhance / Support -- Product and process strengths are analyzed during this phase to improve results in subsequent systems. Determine enhancements that should be made to the product. Another activity that occurs during this phase is development of a discontinuance plan for the point when the product will not longer be supported.

Phase 6 -- Maturity -- Specialized support options are implemented during this phase, and the system is removed from the corporate price list.

Summary -- Results

Quality results have been consistent through the multiple releases. During these releases, many process steps have changed based on causal analysis and process improvement recommendations. Process improvement activities fit very nicely with Hewlett-Packard's culture of being very analytical and striving for perfection. As the company attempts to balance process structure with individual creativity, process steps like requiring formal code reviews have been added relatively late but have been enthusiastically deployed by the engineers.

Chapter 7: Microsoft[26]

The Company

Microsoft was founded by Bill Gates and Paul Allen in 1975 and went public in 1986. Microsoft develops, markets, and supports a wide range of microcomputer software for business and professional use. The software includes operating systems, languages, communications, and application programs. Microsoft also develops and markets microcomputer-oriented books, hardware, and CD-ROM products.

Corporate headquarters are in Redmond, Washington. Total employment is 12,000 people in 27 countries. Research and Development is based in the Redmond complex with additional centers in Tokyo, Japan and Vancouver, Canada. The new Vancouver R&D center, opened in April 1987, is a workgroup responsible for developing software for the international market. Manufacturing is done in 3 different locations: Washington, Ireland, and Puerto Rico. Direct and indirect marketing operations are located in 30 different countries worldwide.

Microsoft is the leading PC software vendor based on revenue. In 1992, revenue was $2.76 billion and net income was $708 million. These represented increases of 49.7% for revenue and 53% for net income, which came on top of gains of 55.8% and 65.7% respectively in 1991. The company's revenues were positively affected by sales of upgrades, growth in worldwide personal computer sales, the success of the Windows operating system, the rapid release of new products and major new versions of existing products, and expansion of the international operations to new areas.

Microsoft Organization

Microsoft is organized around three areas of strategic focus. Worldwide Product Development is responsible for all software and hardware product development. There are approximately 3200 people in the organization with 1600 in development, 750 in testing, and the rest in market support, program management, and user education. Worldwide Sales and Support is responsible for sales and support (handling calls on problems and questions) of all products. There are approximately 4500 people in the organization with 2000 in support that includes 1100 people answering phone calls from customers. Worldwide Operations is responsible for manufacturing, information systems support, finance, and human resources. The Executive Vice Presidents of the groups report directly to CEO Bill Gates. Removal of a President in the company was done to accommodate future growth by managing the company with a team versus a single individual.

Worldwide Product Development is structured into five product business divisions. The Systems Division has Windows, MS-DOS, OS/2, and Lan Manager along with Windows NT. Desktop Applications is responsible for Excel, Word, Project, and Powerpoint applications for the IBM PC and the Macintosh. Database and Development Tools owns Fox, Access, all languages, and internal tools used in the company. The Consumer Division is responsible for products such as Works, Publisher, Money, Flight Simulator, Profit and multimedia solutions that go to homes and small businesses. Finally, the Workgroup Division is responsible for eMail and other group work products that are in development.

Each product within the Worldwide Product Development group has a product organization managed by a Business Unit Manager. Each Business Unit Manager is responsible for profit and loss management for the product. These tend to be marketing trained individuals that take an overall view of the product and manage future planning, budgets, and headcount. Within the Product Organization, there are five major functions that can each have an individual manager for the larger products or can be combined for smaller products. The five functions are: development, testing, marketing, program management, and user education.

Development is responsible for product design and coding. For each product, the group is organized into feature teams. Recalc, charting, printing, and macros are examples of the eight feature teams on Excel. Each feature team has a team leader along with multiple team members that tend to be on the product for multiple releases.
The testing group is also ordered by feature teams that match up with the development feature teams. The group is responsible for testing that starts early in product development and continues all the way through final test.

User education is the final group that has responsibility for developing material for the product. They write help information on the system, manuals for users, and determine what user training is needed along with creating the materials that will be used.

Marketing is the group responsible for product marketing. Product marketing includes driving the product roll-outs with the sales organizations, competitive analysis, and acting as the main interface to the sales organization. The group uses focus studies and other mechanisms to get product input that is combined with input from the sales organization during creation of the product requirements.

Program Management is the final group in the product team. The group is responsible for all product plans and for managing the product through all stages of development. They act as the liaison to support and all other dependent groups. In addition, they manage ISV (Independent Software Vendor) relationships for applications that are bundled with the Microsoft product they manage.

Beyond the reporting structures described, there are two informal organization structures present within Microsoft. Functional structures exist for the five key disciplines in product development: development, testing, product marketing, program management, and user education. Processes are owned by the functional groups who continually inject new ideas and improvements. The "braintrust" is the other informal structure. It consists of 30+ people spread throughout the development organizations. They are product people that are known and trusted by Bill Gates, and are highly respected by the development organization. These long term contributors of good ideas are called upon to provide input and comments during product planning and review times. Recognized names like Charles Simonyi and Nathan Myhrvold are part of this group that helps keep the company on the leading technical edge.

The Sales and Support organization has some unique aspects that are worth noting. 1100 people handling phone calls is a substantial number that put them on nearly a 1-1 ratio with developers. Bill Gates has stated that his number 1 goal this year is to get the support side of the business under control since it is now the fastest growing area in Microsoft. The support people have a very good knowledge of the products they support. There is one team in place for each product. A person on the team maintains direct contact with the developers, working with them on a consistent basis. The team understands the product specifications and begins involvement during Beta Test to prepare for support of the product.

Culture

Microsoft's culture is evident in their two most important goals: hire the best people, and give them the best tools possible to do their jobs. Hiring the best people has been the focus within the company throughout its history. Cornputer science graduates from major universities and experienced PC-based software developers are hired and brought onto a product team which they stay with for a long period of time. This is true for development, testing, marketing and each of the other functional groups. Staying with the product gives them a long-term investment in the product, ensures they have good familiarity with it, and helps them understand process liabilities and benefits from prior development cycles. The work atmosphere is one of flexible hours, flexible dress, and open, honest relationships. Frank discussions are the norm.

Bill Gates' personality is a significant cultural influence in Microsoft. Individuals and the overall company are fiercely competitive, driven to technical excellence, driven to make and meet aggressive commitments, and willing to do what it takes to get the job done. Due to the hard drive that is the norm, significant stress is part of the developers' lives. Burnout occurs within the company among both developers and non-developers.

Changes in development methods provide some evidence of cultural changes through the years. The early development culture was one of extreme individualism with most products involving only three or four developers. The developers had ultimate control of the way they developed the product. A story which shows the extreme nature of that period is that of a developer who sat down and wrote the code for a new product, didn't like the way the product worked so started from scratch and completely rewrote it, and still did not like the product so sat down and started from scratch one more time to write the product[27]. The process involved his own vision of how the product should work and how the internals were designed and coded.

When the company moved from doing OEM work to developing products for the retail market, the culture changed with the addition of specs, testing, marketing, and support groups. Testing was significantly influenced by IBM through the joint development work for the IBM PC. Microsoft also changed its product quality evaluation systems, project planning, security conditions, and other business processes. As quality and schedule mistakes began to mount in the company, developers changed the culture by adopting practices such as code reviews and more formal design and planning methods. The final significant influence has been the move of PC software into mission critical applications for companies. Purchasers are demanding that their suppliers (Microsoft in this case) have high quality, repeatable processes in place to develop and support their products. As more structure comes into the company, the challenge they are dealing with is the combination of the structure of processes and the individual creativity needed to create leading edge products.

The final cultural element is Bill Gates. Many believe that the key Microsoft difference is Gates. He represents a technical visionary that is also the leader of the company. His involvement extends to reviews and input on each of the products' specifications and long-term development plans. He make a real difference in the products.

Product Description

The major products created by Microsoft can be grouped into systems and applications. Some hardware products such as the Microsoft Mouse and BallPoint pointing devices are sold but are a small part of the business. Systems products generated $1.1 billion of revenue in 1992. Major systems products include:

The Windows operating system is the major product offered by the group. Briefly, Windows is an operating system written for the IBM PC which provides an easy-to-use graphic user interface, allows convenient data sharing, provides support for organizing and managing files created by the applications, and allows switching between different application programs. It also allows programmers to write larger applications than with DOS. Included is a set of general applications and accessories. Estimates are that over 10 million users have adopted Windows since introduction in 1983, with a majority coming since the 1990 introduction of version 3.0.

MS-DOS was the base operating system for the first IBM PC and has continued to be a standard. Windows runs on top of DOS. The initial MS-DOS version came out in 1981 and updates have continued through MS-DOS 5.0 introduction in 1991. It still brings in a significant amount of revenue, accounting for 19% in 1990.

Microsoft's version of OS/2 and the Windows NT operating systems are the advanced products in the systems group. Windows NT is a 32-bit operating system that forms the foundation of Microsoft's new line of operating systems intended to replace both DOS and Windows 3.1. Microsoft sees NT running on a wide-range of hardware platforms in both desktop and server environments.

Applications products generated $1.36 billion of revenue in 1992 which made it the most significant revenue producing group. Applications include an extensive range of products for the IBM PC and the Apple Macintosh computers. Some of the most significant products are:

Microsoft Excel is the company's spreadsheet application for the IBM PC and the Macintosh. It competes with Lotus 1-2-3 for leadership in this category on the IBM PC and is the clear leader for the Macintosh. New versions were introduced in 1992 that have kept the application leading edge in function and performance.

Microsoft Word is the company's word processing application for the IBM PC and the Macintosh. It competes with WordPerfect for leadership in this category on the IBM PC and is the clear leader for the Macintosh. New versions of this product were also introduced in 1992.

Microsoft also competes with PowerPoint for business graphics, Project for support of project management, and Mail for electronic mail networks. The company competes in most product markets with strong products that integrate well with the Windows environment.

Review and Planning Cycle

When looking at development within Microsoft, the review and planning cycle is a logical starting point. The cycle is split into two portions occurring in October and April. The result of the cycle is executive agreement on product rollouts and funding for the divisions.

The October review is centered on presentation of 3 year product focus plans. Each product defines the number of releases, why they are doing a release, and interdependencies they have with other products. Bill Gates sits in on each separate division's dedicated review and on the final review in which all divisions present at once to give everyone a common understanding of the product plans. Each product receives direction from Bill during this phase.

After the October review is completed, the marketing organizations take the output and do sales forecasts based on the product plans. Budget planning is then done based on product sales forecasts. The sales versus budget mix is looked at to determine how it compares with the profit model for the company. Based on this analysis, headcount is determined for the fiscal year that begins in June. Up to this point in time, the company has never hit a limit where needs are limited due to headcount restrictions. Open headcount has been available in all cases and they have hired to fill it.

Release Structure and Strategy

Releases for the individual products are determined by the product's business manager and approved during the October Review. Previously, releases were more function driven based on the key features desired in the next version, but that has changed through the years to where the delivery date is now most important. Tradeoffs of function are made to reach the delivery date that was committed. Developers and the full product team determine the delivery date and commit to it, which raises their drive to make it. The transition from function driven to date driven releases happened in the 1986-1988 timeframe and was due to a long history of missing dates that no longer was considered acceptable by customers or the company.

Changed code is considerable for each release. Estimates are that 50% of the existing product code in a release is changed. On top of that, another 30% of new code is added for the functions introduced in the release. The results are code with an average half-life of only 1.5 years. For this reason, extensive automated regression tests are critical to development at Microsoft. Without them, the product could never be tested in time to make reasonable update schedules.

Development Process Overview

A consistent high-level methodology is now followed throughout the major product groups in Microsoft for software development. Some groups are further along in areas such as usage of metrics and adherence to review steps, but nearly all recognize this general model. Each group has a "Scheduling and Methodology Document" that describes their process model. The document "runs 30-40 pages in length and is mainly verbiage"[28] in that no group pre-defines or adheres to a fixed set product development steps. It does describe the "key" process steps, which are generally based on knowledge gained from prior releases. Groups tend to have some variation in steps. The old "OBU" (Office Business Unit that developed Microsoft Word) process, which was documented following their development experience, has served as the base for most product's processes and is used as the default for any group that has not developed their own.

Microsoft utilizes empowered teams that are responsible for all stages of development and the decisions required to get their product out. The groups attempt to keep the teams small or arrange larger teams by function to keep the small team atmosphere. A full team from the five functional areas is in place for all products. As was mentioned earlier, team members are involved with multiple releases of the product.

From a high-level viewpoint, the development teams are responsible for the following things:

> Producing a quality vision for the product which states what quality means for this product (bugs, performance, reliability, function).
>
> Own specifications, design, code, testing, and validation of the final packaged product.
>
> Product improvement with input from marketing, program management, Bill Gates, and anyone else with an opinion.
>
> Process improvement through usage of post-mortem reviews along with in-stream changes needed to get products back on track.

Customer awareness via ties to the product support organization, monthly flash reports on problems, call logs on problems, and competitive analysis done by the product marketing groups.

Microsoft does not have an extensive set of formal development checkpoints. At a minimum, three checkpoints are used during the product cycle: Schedule Complete (specification is complete and approved), Code Complete, and Release to Manufacturing. The development team commits to the set of functions that will be delivered during the release along with a schedule for the three checkpoints. Internally, they determine what is necessary to meet these three checkpoints. This may involve different combinations of design stages and reviews, along with different approaches to the actual code development. Internal checkpoints and interdependency plans will also be worked through. Microsoft does not see themselves doing significantly unique process concepts, but instead feel they utilize some new ways of putting them together.

Investments within Microsoft for development have tended to follow the following model:

People
Specifications
Tools
Design-Test Plans
Code-Test Cases

When problems hit during development, they go through these investments in reverse order attempting to fix it. Actions are taken starting from the bottom, with people changes only being made as a last resort. They have found that people changes are the most destructive in the long run and should be avoided if at all possible. The recognition of this as a decision model is very effective for negotiation and efficient problem solving in the company.

Requirements Phase

A Vision Statement for the product is created by the product marketing team. This is a general statement defining the direction for the product. It describes the overall focus of the product, how it will be marketed, the purpose of the next release, and the basic areas that will be addressed by the next release. Statements like "Fix the top 20 problems reported to the product support organization and add function XX and YY" characterize statements of the basic areas to address for a release.

This type of input, fleshed out with some specification information, is what goes forward as a part of the April Review input. Schedules are approved during that review and the general direction is blessed or changed as a result of the review.

Specification Phase

The program manager owns and drives the specification for each release of a product. This person is responsible for soliciting inputs from all groups considered important for the product. Inputs are utilized to create a final list of what will be included in the product release.

Specifications are written in a user centric viewpoint. They show menus, commands, dialogue users will see, and error messages that can come up. Concentration is not on "how" to solve the requirement, which will be done during the design stage. Specs can be quite lengthy due to the amount of graphical presentation in them. For Excel, the spec is 300-500 pages. A team of program management, marketing, development, testing, and user education do continuous reviews of the spec before the final review is held. No formal process is used during this stage, but a complete set of specification reviews is just expected from the development group.

Development and testing groups are responsible for refining the spec. Development fleshes out details surrounding the functions, estimates the amount of work in person months, and estimates the schedule for the project. Testing works to understand the spec and then estimates the amount of work in person months, estimates the schedule for the project, and defines what is needed from development to allow the support group to test the product.

Bill Gates also has a role in specs. The Program Manager is responsible for figuring out how to get Bill's input for their product. They need to complete this during the spec stage and have to come out with Bill's buy-in to the spec. Each product will have at least one formal review with him and key products may have multiple meetings. During the meetings, Bill will set some key goals for the product that may be on quality, cost, or function. Before a product can move on to implementation stage, it must have formal approval from Gates which constitutes the Schedule Complete checkpoint. In the past, he personally reviewed every spec in detail but has since brought 2 full-time reviewers onto his staff to review the specs and assist him on inputs.

An aspect of the spec stage that is important is the usage of prototyping. Prototypes are always built during the spec stage. Menu, command, and dialogue will be included in the prototype and serve as inputs to the spec. In some cases, the prototype may be the spec and be used for the final meeting to get approval to go on to implementation.

Implementation Phase (Design and Coding)

There may only be one formal checkpoint used for the Implementation Phase. Code Complete is the final step that indicates design and coding work is complete and the product is ready for final testing. Individual developers and groups determine the process and checkpoints necessary to meet the function and schedule commitments.

Design. Development will do sufficient design during the specification stage to allow for a solid estimate of the amount of effort required and the schedule it can be completed on. Development recognizes the estimate as their firm commitment. The commitment they are required to make drives them to do a reliable job of this early design work.

A formal set of design stages does not exist. It is up to the development team to determine what must be detailed during this step. Much of this determination is done based on process learning from prior releases. Module structure, dependencies on other functions, input/output details and other normal design stage considerations are dealt with during this period. Development does hold a complete design review for their work. This practice has evolved because of the success developers have seen from past usage. Specification languages and code generators are not used.

Coding. PC and Macintosh products utilize a significant amount of common code. About 10 - 15% of the code is unique for the platforms with the rest being common. The system is broken into modules that consist of 8 - 10 functions each with 40 lines of code per function. Reused code between products amounts to only about 5 - 10% of the product code. Most of this is for the user interfaces which have many standard elements in them. Code is not developed with reuse as the objective. The reuse happens through the general developer approach of "stealing what I can".

Code generators are not used with coding continuing to be manual and individually oriented. Object-oriented programming is not currently part of code development for major products, though new projects, including an OO version of Windows[29], are experimenting with it. Formal code reviews have become part of the standard process at Microsoft due to technical push. The reviews were tried by groups and proved to be very beneficial such that all development teams wanted to use them in their processes.

Formal code reviews are done with exactly 2 reviewers per inspection. Reviewers go through the code independently and strong competition exists between them to do the best job. Defects and design mistakes are both found during this stage. Sections of 2 - 5 KLOC of code are reviewed at a time.

The coding phase focuses on one key checkpoint which is Code Complete. This date was estimated by the developers and all activities center around achieving it. The Development Manager polls each developer to determine whether they consider themselves finished. When all are ready, Code Complete is declared and testing can begin. After Code Complete has been declared, the only changes allowed are approved bug fixes.

Before the Code Complete checkpoint, other milestones are part of the coding stage. The major milestones are:

Intermediate functional releases which go to testing or development groups with dependencies on the function. These are agreed to one-on-one between the developers and the individuals needing the code.

Visual Freeze which is utilized for all products to allow screen shots to be taken for user documentation. The user education department drives these and negotiates the date with development. Typically, 20 - 30% change occurs after the freeze.

Functional Freeze which is utilized to lock the text information used for documentation. This checkpoint is used by many products but not all. The user education department drives this and negotiates the date with development.

Typically, 20 - 30% change occurs after the freeze.

Beta Test Release is a statement of confidence in the code versus a testing efficiency statement.

During the coding stage, development continually tells testing what sections of code are complete and which are incomplete. The communication allows targeted functional testing to begin as soon as possible. As a final step in development of the new code, a mandatory suite of development tests are run. The tests are internal checks used by the testing group for assertion testing of the code (assumptions made about conditions that will occur at specific steps which do not need code to directly check for them), and the usage of check routines available through debug menus.

Integration Testing. All modules of code are kept in a Master Library on a central server. The Master Library contains the master version of the code that the product is built from. A library management tool exists on the server that allows developers to "check out" a master version of a module to work on it at their workstation. When the developer completes making changes, they run a set of unit tests to validate the new function they have added. In addition, they must run a suite of preliminary integration tests that validate base functions are not affected by the changed code. If all tests are successful, a "check in" can be done to put the new version into the Master Library.

Nightly builds are done on the master code for all products. Build tests are then run to ensure the product will operate. Problems found must be immediately resolved and everyone stops work until the problem is fixed. Since builds are done nightly, tracing back to find the change that caused the problem is reasonably easy to do. Nightly builds ensure that the product will function at all times and controls the amount of churn in the system, which helps stability.

Testing Phase

Like most organizations, Microsoft's testing strategy is to find defects as early as possible. The company is different in some of the steps they have taken to make this happen. Automated suites of tests available for developers to run prior to integrating their code are extensive and expected to be run. Test tools for developers to develop tests of new function are also available and very functional.

Each of these items is helpful, but the most significant difference is in the relationship between the testing and development groups. Testing is done by a group within the product development organization. No independent quality assurance organization is used in Microsoft. Testers have a very close relationship with developers. Like the developers, they are involved with the product over multiple releases. Most work on a 1-1 or 1-2 ratio with developers. Involvement starts at the spec stage and continues through the rest of the cycle.

Private releases are utilized between the developer and the tester. Developers may pass a private release of code to a tester that contains a new feature that is not fully developed and checked in. The tester will use it to improve and certify testcases while the developer can get bugs discovered early and recode as necessary. This coordination assists the developer during development test and assists the tester for their final test.

Testing phases are very well planned. Testing does their own estimates of resources and schedules during the spec stage and are committed to meet the plan. Formal test plans are created and test case reviews are held. Reviews are held for all testcases and development participates in 70 - 80% of them. Automated tests from prior releases are added to the plan so that total test coverage can be understood.

Final Test is the main verification step run by the testing organization. Products are tested through customer-like usage and results are tracked closely against the test plan. Testing includes documentation, tutorials, set-up, hardware configurations, primary functions, and supporting utilities. Automated testcases are key to validation of existing function and are extensively used. Performance is also measured against the performance goals set for the product. Results from Final Test are the most critical input to the ship decision.

Three types of Beta Tests are also utilized. Tests are done to get awareness and excitement for a new product or function (marketing reasons), and to get feedback and remove bugs (technical reasons). The three types of test are: narrow tests with a select set of customers that will utilize a new function or check compliance against specific goals, wide tests that attempt to catch rare cases not found on typical configurations, and internal distribution to employees to get results similar to wide tests. Beta tests tend to get a very low response rate of 5 - 6% of users giving feedback to development.

Development has a set of scheduled checkpoints during the test phase where they attempt to get the number of outstanding bugs down to zero. "Zero Bug Releases" are used as one set of the checkpoints where development consciously attempts to drive down to the target of 0 known bugs. The organization tends to set multiple checkpoints like this during a test phase. "Release Candidates" are an additional set of checkpoints and involve an attempt to build the final product. While being intended as a verification that the code will fit on the specified number of diskettes and that the build procedures work, this is also an attempt to freeze the code and test on a solid product.

Ship decisions are made after Final Test. Program Management makes the final decision and utilizes the position from the "committee of 4" and in some cases the interests of the Business Unit Manager for the product. The "committee of 4" consists of development management, testing management, product marketing management, and the support organization. In the end, it is the Program Manager that makes the decision.

Maintenance

Separate product support teams exist for each product. These teams are part of the Sales and Support organization and not part of development. Their main responsibility is to handle customer calls for the product. When problems come in, the support organization logs them and creates problem reports that come to the development group. Current development staff handles all product maintenance and all or part of the team will be directed toward fixing problems when they come in.

Product Support needs have grown rapidly in Microsoft. It is the fastest growing group in the company. One of the key goals of 1993 is to get the support "under control".

Process Usage and Compliance

Each product group is responsible for choosing the development process they will use. Experience tends to dictate the process they choose. The process described in this case study applies most to mature products like Excel, and Word. Over the last years, the company has rapidly progressed in usage of more formal processes. Customer requirements are now becoming a factor in accelerating adoption of more solid and verifiable processes. As PC applications become more central to organizations, customers are looking for in-process metrics and other indicators of quality before new versions are installed.

Process compliance is not handled via formal mechanisms. Developer commitment to quality is the main driver of compliance. The other mechanism is internal audits called by the Product Managers. The Director of Development and Quality Assurance is asked to go out and work with the development groups to analyze major problems and current status for projects. A formal "Audit" is used to directly act with the purpose of changing things quickly. A "Review" is used to take a more gentle approach of analyzing the development work (process or current status) and recommending actions to resolve the problems found.

Project Management

Project Managers for the product are the key drivers of the product development. Business Unit Managers give them the authority to make the decisions necessary to meet committed schedules. Beyond constant contact with the groups creating the product, there are two other mechanisms that are critical to project management:

> Schedules are determined by each of the functional groups. All estimating is done by the people doing the actual work. By having this relationship between estimates and work, the accuracy of resource and schedule estimates is very good. In addition, the individuals are very committed to meeting schedules. These factors come together to be major factors in the management of projects.
>
> Project reviews are also utilized throughout the development process. Project Managers schedule and run these with the frequency varying (range of weekly to monthly). Everything associated with the project is reviewed with each group reporting their status. Monthly status reports also come in from each functional

area. Major reviews of project status are held with Bill Gates. Timing of the Major Reviews varies depending on the strategic importance of the product.

Change Management / Configuration Management

Network servers are in place to store source directories that are accessible by everyone in the company. Password control is used to control access to some of the source directory servers. Network based Configuration Control is used on everything associated with the products under development. Items in source directories include: project documents such as specifications, all code, tools, releases (current and previous), plans, and schedules. The parts can be "checked out", changed, and then "checked in" after changes are made. Forcing the parts to be checked out and back in places a level of control on all project related information.

Changes to requirements, specifications, and frozen code are allowed during the development process. After checkpoints such as Schedule Complete and Code Complete, the Program Manager takes control of changes to specifications and code respectively. By allowing approved changes, the Program Manager lets innovation continue to happen during phases such as coding and testing. When decisions are required for necessary changes, a formal decision model is used to determine the action necessary. The model, from highest priority to lowest, is:

Schedule / Resources
Components / Functions / Features of the product
Future Extensibility / Maintenance -- These are bad for the long run but may be necessary
Performance
Reliability / Quality -- This is definitely only done when no other options exist. The

changes may be "not fixing" something that was previously planned.

Changes to code are managed by the tools on the system. Source code must go through the "check out" and "check in" procedures. "Force outs" and "Force ins" allow developers to check out source code when someone has previously done a standard "check out". The forces are managed through function in the network control tool that compares changes to ensure the same lines have not been altered. Before developers are allowed to check code back in, they are required to run Synch Tests which serve to validate the code does not degrade the system. Nightly builds are done on the total product. Synch Tests are then run on the total product with any problems discovered holding up all development until resolved by the developer making the faulty change. Nightly builds allow the product to be usable everyday. In addition to the "check out" procedure, changes to code after Code Complete must be approved by the Program Manager.

Defect management is accomplished through a set of bug tracking tools that run on the server. Bug reports are entered into a database along with a description of how the problem can be recreated. Severity codes running from 1 (critical) to 4 (new function request) are assigned by the discoverer of the bug. Development continuously monitors the database so that they can assign the problems to someone on the team when they are reported. The defects are tracked closely by testing, development, and program management.

At the end of the development process, the change control process takes on an additional level of formality. A "committee of four" (one from development, testing, program management, and product support) meets daily to review all problems and determine which to fix. Problems are generated by internal testing and Beta Tests. Utilizing the committee review helps ensure decisions are made from data versus emotion. Approval requirements, plus the tracking capability, provide a level of change management for bugs.

Metrics

Data is important in resolving conflicts and making decisions on actions to take. It was stated a couple times that "Microsoft is data driven". Top management supports the usage of metrics since they have been shown to help lead to a product being completed on schedule. The most watched and used metrics involve bugs.

Tools are in-place and made available to allow metrics to be generated. Some to the metrics used by various groups are:

Bugs to date
Bug severity mix
Open versus Fixed bugs to date
Bugs found versus bugs fixed
Functional use profiles (not used much yet)
Cluster of defects (helpful to the testing organization)
Code churn
Code coverage of tests
Customer problem calls versus units sold

Bug metrics are described above. They are very important during the development process. Standardized queries and reports for management are generated at defined intervals.

Some historical data is used by the product team. If internal data does not exist for the product, applicable external data is used as a model. Data most frequently used are models on how many bugs are likely to be in a product and how many should have been removed through each of the stages.

Process Improvement

New process ideas come largely from the teams themselves. Creativity and drive for excellence are encouraged. The combination leads to teams finding solutions to process problems, trying them out, and talking about them to other groups. Since 1989, it is clear that Dave Moore, in his role as Director of Development, has individually been responsible for bringing in the concept of spreading "best practices" information throughout the company. The "best practices" information has come from his search of worldwide sources and from his involvement with development groups across the company's projects. Improvements are adopted in this manner of trying new ideas and spreading information on the results, and are not handled through dictating usage.

Post mortem reviews are utilized by nearly all teams. Problems encountered during the last cycle are reviewed and analyzed for improvement possibilities. Process changes are made and used in the next release. Since the teams tend to stay together, the post mortem analysis is very effective and helps with process learning by the team. Program Managers run the post mortem process.

Tools

Development environments consist of personal computers and workstations in offices connected to the LAN server network. Developers pick the hardware systems they wish to use and many have multiple systems in their offices. The LAN has product servers for each product developed and also has network servers which allow access to data throughout Microsoft. A corporate MIS group is in-place to manage 600 servers in one building along with the network that is spread throughout the world.

For testing, a good suite of specialized tools are available for automated testing. These tools have been historically used and have now progressed to event recorders and playback tools that include event editors which allow editing of flows versus re-recording whole sequences. Automated test tools are built to run in multiple environments.

Automated tests are run "hundreds of times" during development. Testers are continually adding to this set of tests. Quick or Synch Tests are used before all check-ins and after all nightly builds. Testers run them frequently during Final Test phase. Development has also supplied a variety of tools to assist in simulation of memory, data structure, system failure, and memory fill errors.

Process Education

A formal process education class does exist. though most education is done within the team. Each product team has 2-4 page documents that describe their product and series of 2-4 page documents that serve as checklists of job responsibilities for each of the major job types (development, testing, support, etc). All major product groups and many smaller ones have their "Scheduling and Methodology" document that describes process information. Mentors are assigned to each new hire on their team and help introduce the new hire to processes used in the company.

Two weeks of training is expected each year for all software engineers. A combination of in-house training, university seminars, and corporate or conference seminars are used to meet the objective. In-house training is available for corporate training on management skills, and product group training is available for technical skills.

Summary -- Results

Microsoft is an outstanding study of the transitions involved in moving away from an immature development process. Early development was very individualistic, dependent on testing to verify quality, and driven by function completion versus recognizing schedule needs. Through the last years, the company has done an outstanding job of introducing a level of structure (maturity) into the development processes while preserving much of the creativity of their developers. The formalization of the process via reviews, metrics, and change management were either introduced or strongly supported by developers. They also moved from a process of developing until a continuously growing set of functions were complete, to a process where they develop towards a scheduled end point and get as much function in as possible while still meeting that date. Demands from a growing customer base, in terms of size and sophistication, have also been factors in the process evolution.

Finally, one of the key items that makes Microsoft unique from the rest is the presence of Bill Gates and his braintrust. Developers drive the organization, but the developers have a keen sense of what the market requires. The combination of technical and market knowledge is the competitive advantage all software providers would desire, and what Microsoft appears to possess.

Chapter 8: Lotus[30]

The Company

Lotus was founded in 1982. The company and its subsidiaries are engaged in the development, manufacturing, marketing and support of applications software and information services. The company sells its products primarily through distributors and resellers. Personal systems and workstations are the main systems the applications are developed for with an additional set being developed for minicomputers and mainframe computers.

Corporate headquarters are in Cambridge, Massachusetts. Total employment is approximately 2800. Product development is concentrated in Cambridge, Massachusetts. Manufacturing and distribution is done in 4 locations worldwide: Cambridge; Dublin, Ireland; Caguas, Puerto Rico; and Singapore. International marketing operations exist in 33 countries with authorized distributors covering an additional 20 countries.

Lotus is the leading DOS spreadsheet vendor and has leading applications in other market sectors for personal systems. In 1991, revenue was $829 million and net income was $43 million. This represented increases of 20% for revenue and 85% for net income. The company's revenues were positively affected by continued growth of the 1-2-3 spreadsheet products along with increases in the sales of the Ami Pro word processing application, cc:Mail electronic mail product, and the Notes workgroup computing product.

International revenues grew 26%, which raised non-US sales to 51% of revenues.

Lotus Organization

Lotus is organized into five major groups: Development, Sales & Marketing, International Operations, Manufacturing, and Finance. Within Development, a Vice President / General Manager is in-place for each of the product families. The major product families are: Advanced Spreadsheet Products, Word Processing Products, Notes, cc:Mail, and Graphics Products. The organization of each product family varies slightly depending on the number of major products in the family. For many major product, there is a Director of Development and a Product Planning Manager.

The Director of Development for the product owns development, but has joint responsibility with Product Planning for content of enhancements to existing products or what will be part of any new product. Once the content has been decided, the Director is responsible for getting the product developed and delivered to the marketing organization. Each Director has a Development Manager responsible for the programmers creating the product, a Quality Assurance Manager responsible for testing and validating the product, a Documentation Manager responsible for on-line and printed documentation and on-line help information, and a Program Manager that is responsible for managing the execution of the release development. Program Manager responsibilities include tracking the schedule, owning and ensuring resolution of issues, working with all outside and inside groups to ensure dependencies are met, working with the international product groups to get the product available worldwide, and making sure the internal development groups are working with each other.

The organization responsible for DOS Spreadsheets consists of nearly 50 people handling programming, Quality Assurance, documentation, and management. The Notes group has approximately 100 people handling the same functions.

Culture

Lotus started out as a "Cambridge company", which was people with diverse backgrounds who were technically oriented and very hard driving towards a specific technical goal. Initially, there were 300 employees focused on 1-2-3 prior to a large set moving on to Symphony which was going to be "bigger than 1-2-3". Frustration was a major problem when Symphony was not very successful, and it was the beginning of a long period where the organization was slowly recovering from the need for products "to be as successful as 1-2-3".

As the company grew after 1-2-3, they continued to hire very talented people. Because of the talent the group possessed, they were still able to make dates by figuring things out as they went. Specs were non-existent and everyone was forced to "figure it out on your own". The Lotus 1-2-3 Release 3 (Godiva) project became the next major cultural influence. Developers took on the task of completely rewriting 1-2-3 in C (a new language for the product), with the capability to run on multiple platforms, and complete it in 1 year. Project dates and functional capabilities were not met.

Over the last years since Godiva, the culture has swung to an environment where people are beginning to work together, code sharing is starting to happen, and groups are actually lending people to other projects (with the people eventually returning to their old groups). The environment has begun to change so that it has become more stable and consistent.

In contrast to staying with a product over a long period of time, Lotus tends to move developers to the next hot project. Product plans do not extend to multiple releases and developers have grown up seeking out the most interesting projects versus staying with a single product. New teams are being built for each project with the group figuring out the existing product before going on to do the new development for the product's next release. "Project hopping" has become a cultural element at Lotus. The Notes group looks to be an exception to the project hopping with the same core development team being in-place since the product began nearly 8 years ago.

Lotus's growth has been a factor in the culture that has developed. Of the 5 product divisions that compose the company, 3 1/2 were acquired. The acquired companies had very little structure and Lotus did not rush in to impose any. As a result, Lotus new has limited structure within the divisions of the company. Some processes are being introduced to increase controls, but they continue to be limited.

Product Description

Lotus 1-2-3 for DOS is the most successful spreadsheet application in history. Measured in dollars, the share in this market rose to 85% in fourth quarter 1991. Upgrade revenues doubled in 1991, driven by new releases of the Windows product and the DOS products. The spreadsheet has a wide set of features that include a powerful relational database, an interactive graphic environment, drawing and editing tools, business graphics, a range of output options, macros that can be programmed, and ability to access and handle data from competitive spreadsheet products.

Lotus 1-2-3 was the first really successful spreadsheet on the PC market. It was oriented towards the IBM PC with Macintosh versions only coming out recently. 1-2-3 products are now available that support OS/2, Windows, and proprietary versions of UNIX. In recent years, the product has also been extended to run on many minicomputers and mainframes. Compatibility has been important throughout these changes with new versions supporting spreadsheets created with the older versions of the product.

Notes is a relatively new product from Lotus that is broadly classified as a workgroup computing product. The product is client-server based with a database existing on the server that can be accessed across the LAN by a set of client workstations. Notes can be considered a document-oriented database that allows users to share any type of unstructured information regardless of the platform or network. Unstructured information can be text, graphics, spreadsheets, reports, and word processing documents. The data is stored on the server which makes it accessible to users in the network, while mail capabilities exist in the product to send a note to inform others of the data availability.

Notes has a simple graphical interface that allows relatively untrained users to access a full set of databases and also set-up their own databases. The mail features are extensive and make including information from the databases very easy to do. Security and network administration functions are provided and are reasonable to use.

Review and Planning Cycle

In some of the case studies I have done, a review and planning cycle was a distinct phase that warranted its own section. Lotus has combined the intent of a review and planning process into their requirements phase for a product. Resource needs are determined by the number of projects that have been approved and are under development. It was not clear what the planning cycles were for resources.

Executives were part of monthly progress reviews for each major product. The reviews address status of the product development along with the outstanding issues the VP should be aware of. The Project Manager schedules and runs the reviews.

Release Structure and Strategy

DOS Spreadsheet products have relatively long release cycles. Historically, new releases have come out in roughly 18 month cycles but the group is moving to 6 - 12 month cycles. More frequent updates are not desired by customers since an installation is required and many users do not upgrade frequently. Updates are also not desired because they cost additional money. Many users are satisfied with what functions they currently have and are not interested in upgrading. The structure is to come out with a new version that has the set of enhancements in it that have been agreed to by the product director and the product marketing person. Fixes to reported problems along with enough new functions to entice upgrades or land new customers are what is included in the release updates. A combination of function and schedule needs determine the contents of the release.

Notes has been running with a relatively long release cycle also. Current releases are being brought to market in 18 month to 2 year cycles. The goal is to bring this to a 1 year cycle, which looks necessary due to the rapid changes occurring in this relatively new software market. The release structure is determined by a combination of date and function. Agreement is reached on a set of functions that are key to the release and a set that are optional. Schedules are then worked out for the key functions and compared with what is desired by the product marketing group. Give and take happens to come-up with the correct combination of function and schedule.

Development Process Overview

Lotus has a published company development process that is used minimally by the development groups who determine the real process they will use for a project. Directors of Development are given the responsibiiity of choosing the development process they will use for a product. This leads to variations among the different products.

Two standards drive development within the company and only one of them is constantly used. The date that is constantly part of a product is the Ship Date. Within Lotus, this is the key date and is most obviously part of each project. Commit Checkpoint is the other major date. The date receives enough attention and is viewed positively enough to be used by most groups.

For DOS Spreadsheets, since the general model does not dictate specific process actions, the Development Director brings the process she wishes to use with her. The process has a level of structure and project milestones in it, which have proven to be necessary in Lotus. It involves the following steps: determination of requirements; planning through the Product, Documentation, Development, and QA plans; design; Commitment Checkpoint; implementation; testing; and maintenance.

Notes has some unique aspects in their development process because of the fact the programmers are in a different location (in a separate company) than the Development Director, QA department, and all other support functions. IRIS is the company responsible for product development. The flow through the stages for a new function in Notes is: generate requirements document, break requirements into pieces, begin functional and design specing the pieces that are ready, implement the product as soon as design completes, and Feature Testing of the pieces when the piece is ready. The net result is multiple independent projects synching up for the release.

Requirements Phase

Lotus has a Product Marketing group for each product family. Individual Product Marketing people may have a single major product or multiple smaller products. The person has the responsibility for gathering requirements for enhancements to existing products or for potential new products. The process of gathering requirements and interacting with the development group vary across the products, as was the case for DOS Spreadsheets and Notes.

For DOS Spreadsheets, the requirements process is initiated by the product marketing person coming forward with an idea on the product features and a desired ship date. The feature ideas tend to come from visits to existing and potential customers along with some analysis of competitive product features. The set of requirements are defined in the Marketing Requirements Document that includes: a definition of the features, the relative priority of each feature, and a desired schedule for shipping the product. The document is used as a base to for development to estimate the cost of the features. Negotiation begins at this point on features, staffing, and delivery date to determine what can be shifted. The Development Director then closes the process by specifying what marketing can get in regards to function, cost, and schedule. Marketing then gives the go-ahead to do the planning and development work necessary to reach Commitment Checkpoint.

Based on the agreed to set of features and schedule, planning begins. For DOS Spreadsheets, four plans are created for the products:

Product Plans describe the what, why, and how for the overall product. The parts of the plan are: the description; functions that are part of the product; staffing estimates; more detailed schedule; feature list for future marketing purposes; a subset of the development, documentation, and QA plans with references to the full plans; and an ongoing list of issues. The level of detail varies with the scope of the product.

Development Plans provide more detail on the features. The parts of the plan include: where they will steal functions from within the Lotus products, key changes to file formats, new technologies that will be introduced such as SmartIcons, the network and tools that will be used for development, and an expanded schedule.

Documentation Plans provide more detail on the printed and on-line product documentation. The parts of the plan include: a description of what the full set of documentation will be, cost of goods, page counts, tools that will be used for on-line documentation and Help text, international issues for translation, and an expanded schedule.

Quality Assurance Plans describe the strategy that will be used for testing. The parts of the plan are: a detailed schedule, description of risks that will be taken, reused tests from prior products, automated versus manual test plans, testing matrix of configurations, and international testing considerations.

Notes is a newer product and the requirements phase is slightly different because of it. Two years ago, it was a small product in a big company which allowed them to operate on their own. As the product has grown in the size of customer base and resulting significance to the company, the planning and requirements process is becoming more organized.

Finalizing the requirements list and laying out a product plan is a combined process. Planning tends to be done by a small group of 2 - 4 people which are the top management in development along with the product marketing manager. Previously, the group, which has significant customer contact, generated the functional requirements and schedules. As the product has grown, more organization of requirements gathering has been introduced. Product Marketing is gathering input from: existing customers; potential customers; other Lotus products that interact with Notes; along with creative input from development, marketing, and executives. Requirements are then categorized and prioritized based on goals defined by the small group that previously was handling the full process. A series of meetings are held where the priorities of the features are finalized and the top items that will be developed are agreed on. The result of this process is the Marketing Requirements Document.

Requirements and specifications for Notes go through an iterative process. Early specifications are very loose due to relatively long development cycles. Throughout the cycle, customer inputs continue and competitive products continue to come out. These changes result in adjustments to the requirements and plans.

Design Phase

Design is not valued at Lotus. In the early days, Mitch Kapor's approach was to "see the product" even in the earliest stages of development. Marketing continues to drive the company with no development people in upper management. Once product features are agreed to, the pressure is on to make the product visible. Functional design work almost has to be hidden since the expectation is that the coding is underway after the requirements are finalized.

DOS Spreadsheets utilizes user interface and functional specifications. Since the look and feel of the products are so important, it is necessary to get some definition early on. Prototypes are becoming prevalent in showing the interfaces, using throw-away code initially and then replacing that with real code as it is completed. The prototypes are used to get feedback from customers acting as "design partners", Lotus marketing groups, and executives looking for updates on the product. Functional design is also used by this group. Past projects have demonstrated the usefulness of laying out the product structure prior to moving to the coding phase, and the Director of Development ensures these are done for all but very short projects. In cases where the formal documents are not feasible or as a supplement to the documentation, the group has begun to use videotapes to capture descriptions of the product, functions, and plans. This is an informal Functional Specification and serves as a good training mechanism.

For the Notes group, informal specifications are generated for major features. The developers write the specifications that provide detail on the general structure of a function and a high-level breakout of the function across program modules. Specifications are available on-line through the Notes product itself and informal reviews are held in some cases. Design work in the group is not significant, with much more focus on getting to the coding.

Both products, along with the rest of Lotus, utilize the formal milestone "Commitment Checkpoint". This checkpoint occurs by the end of the design phase. Plans and product definition are not at the point where all the details are worked out, but enough information exists for major groups within the company to evaluate the project. Each must state whether they can commit to the product responsibilities necessary to take it to market. Major groups involved with the checkpoint are: development, documentation, Quality Assurance, marketing, manufacturing, product support, user education/training, international, and upper management. Key issues are logged and the formal commitments are sought. After Commitment Checkpoint, everyone turns to the heads down implementation stage.

Coding Phase

The Director of Development and the development team determine the network, programming language, compilers, linkers, and test tools that are appropriate for the product they are developing. For enhancements to existing products, many of these choices are dictated by what was previously done, though variations are made. Formal programming standards across the company do not exist, but employees are well educated and utilize structured programming approaches. Coding is a manual process since design languages are not used as part of the design phase.

Code reviews are held with varying levels of coverage and formality. The DOS Spreadsheet group does formal code reviews through a "buddy system" that involves one person going through the code in detail. These were initially threatening to members of the team, but are now viewed positively due to the results that have come out of the review process. For the Notes group, code reviews are informally done at the discretion of the developer. Reviews are currently done by peers in development, and the desire is to find a way to get the product marketing group involved in this step, also.

Development testing is done by the code developers. These tests are at an individual module level or may involve small combinations of new modules. Validation of code paths and limits in the code, along with some validation of functions, is done at this stage. Development's objective is to get the code working to some degree with at least some combined testing before the code is integrated.

Feature Freeze is another milestone used in Lotus that occurs prior to the start of the formal testing phase. This milestone is where everything is in (has been integrated) and things are working to some degree. The product is usually buggy at this point, but all pieces are together and problems can be identified with interfaces and final function. Performance and size are also concentrated on at this point since there is still time to make changes necessary to improve these critical areas.

Testing Phase

Quality Assurance groups are part of each product organization. These groups are responsible for testing the overall product and ensuring requirements have been met. In-house testing, Alpha testing, and multiple phases of Beta testing are used by Lotus. In-house testing is generally referred to as Feature Test and is planned through the Quality Assurance Plan that defines the testcases that will be reused from prior releases along with the new automated and manual testcases that will be used. Customer environments are simulated during the testing with non-stress functional verification and stress testing both used. Quality Assurance is involved throughout the development process and has a good awareness of the product requirements. Test staffing is on roughly a one-to-one ratio with development in Lotus.

Product usage by internal and external customers is an important part of testing. DOS Spreadsheets and Notes use common phases of both Alpha and Beta tests. Alpha tests are done with internal customers and small groups of external customers. The test is done before Feature Freeze so that some validation of the features can be done in time for changes to be worked into the product. An initial Beta Test is done for quality reasons. During this test, the group of customers is expanded and the emphasis is on identifying bugs and getting them resolved before Code Freeze. Some Beta testing is done near the end of the development cycle for mainly public relations reasons. Early exposure to certain valued accounts, along with groups that write about the product, is done through this Beta test phase.

At the end of the test phase, the Code Freeze checkpoint occurs. When Feature Testing is done and the bug count is near zero, Code Freeze is declared. Code can no longer be touched proactively, with approved bug fixes being the only changes allowed to the code. A team from Quality Assurance and development does the review and approval of bugs that will be fixed. After Code Freeze occurs, the Quality Assurance group runs a final regression test that verifies bug fixes and executes a subset of the feature test.

Delivery Phase

Three stages are used to get the product ready to ship to customers. The first is Release Candidate Builds which are built, tested by QA via a subset of the regression test, and then sent to internal and external customers for a 1 week verification. This cycle continues until a build has zero bugs reported against it. "Golds" is the next stage and involves another build of the product (generally, the final Release Candidate Build is used). Minimal testing, that mainly involves a code compare, is done to ensure this version matches the final source code. Once the "Golds" are completed, the product moves to manufacturing which does a final build and compare step before mass production begins.

Maintenance

Support is a separate organization in Lotus. Calls come in to this group which logs the problems and attempts to resolve them over the phone. If support is unable to resolve the problem, it is routed to the development team responsible for the product. Required changes mainly move to the next release of the product. Immediate fixes are not handled cleanly since no formal process exists. Since immediate fixes generally require a full re-build of the product, workarounds are attempted first. If workarounds are not feasible and a fix is necessary, a maintenance release will be built for a specific customer, testing will be done in QA, and the release will be shipped to the customer (and any others with similar problems). When a significant number of fixes are available, products will do a "slipstream" release so that all new shipments include the fixes. Customers with a maintenance contract for Notes (and other products) will be notified of the "slipstream" release, and will be offered an update on request.

Process Usage and Compliance

A few years ago, Senior Management at Lotus led a process improvement effort that created a very structured "Lotus bible" that dictated the development steps that were to be followed. Dictating the process did not work, and Lotus has continued to have groups determine the process that fits for their product. Usage and compliance are managed within the product groups which are generally small enough for all people to be aware of what is being followed for a process. No formal mechanisms exist to ensure compliance.

Project Management

Methods of project management are determined by each separate group. Milestones such as Commit Checkpoint, Feature Freeze, and Code Freeze are used by most groups across their projects to keep the organization coordinated. Some of the significant actions done individually by a group or commonly across the groups are described in the remainder of this section.

The DOS Spreadsheet group does an extensive planning phase for each group (development, QA, documentation) as part of the requirements phase. From there, the groups move to Commitment Checkpoint where they view all aspects of the project and make a formal commitment to meet the ship requirements. During the implementation and testing stage, each of the individual groups continually reevaluates their progress versus their committed schedule. The Director of Development has made it clear that if there is a problem, she wants to take the slippage early. On-going milestones are also set which invclve taking portions of the product to customers for design validation. These occur between the Commitment Checkpoint and Code Freeze to give targets that the development groups can work towards.

An additional step used by the DOS Spreadsheet group to increase project ownership and improve the level of project management is to give ownership of a portion of the function to teams. Teams can include members of development, QA, and documentation. The group is responsible for resolving issues, answering general questions, analyzing progress versus plans, and escalating any problems that require assistance to resolve.

A common step in project management used by these two groups, and the rest of Lotus, is the role of the Program Manager. This person is a part of each product group and reports to the Director of Development. Their responsibilities are totally associated with project management and include: tracking overall project schedules, working inside and outside the project group to coordinate dependencies, own all issues and take responsibility for resolving them or ensuring another group resolves them, preparing and reporting overall project status, and working with the international group and manufacturing to plan the product roll-out.

The Notes development group uses their own product for most project management work. Notes makes database creation and cross-project communications easy. A record exists for each feature that includes information on: product/feature name, development plans, priority of features, ship date, Alpha and Beta Test dates, a detailed product description, and status tables. This is available for everyone to access and text updates can be made to the record. Staffing plans are also kept on the system which allows the management team to monitor resource allocations at all times. Constant communication through eMail and database updates is really the key to project management for the Notes group. The group has had a normal record of success in meeting ship dates, which has involved some slips along the way. The most common factor leading to the schedule slips has been adding functionality during the development cycle.

Change Management / Configuration Management

Change management is another process step that is individually determined by each product group. DOS Spreadsheets uses multiple stages of change management. The following describes some of the stages and measures used:

> The Feature list is frozen at Commitment Checkpoint. A stretch list is created of things that would be nice to add if resources or schedules allow it. The Director of Development must approve any feature changes after the freeze.
>
> A Functional Specification Freeze checkpoint is used early in the coding phase. This stops changes to the user interface which is necessary to allow documentation to be finalized. Some additional freezes are defined to meet the requirements of the international group that is translating the product are: Message Freeze and Help Freeze.
>
> A Code Freeze checkpoint is used to stop the proactive changing of code. All bugs are reviewed and the only changes to code allowed are those necessary to fix an approved bug. This occurs during the testing phase.
>
> Version control/management is utilized for all source code. Developers use a "check-out" and "check-in" procedure when updating code.

For the Notes product development group, some of the procedures vary. The following describes the general approaches used across their development stages:

Requirements and specifications are not under any form of change control. Developers communicate with multiple groups and make decisions on what needs to be fixed. These changes are often not communicated to the QA department or the project management group. A list of key features agreed to at the start of development is maintained and changes from that are known.

Code is not under a form of version control or "check-in" / "check-out" control. Each developer gets a copy of the source code after a build is done and is free to make changes. Communication occurs between the developers but the group has grown so that this does not provide extensive coverage of project information. On a monthly basis, builds are done. During the build stage, the release architect, who is a very senior member of the team, brings together the source code versions one at a time and reviews each change before allowing it into the master code. This process worked good when there were up to 10 developers, but it is showing signs it may not work as the group expands.

Extensive tracking of defects occurs. Each user has access via Notes to the database of problems. Easy to generate reports are available to look at status and the state of fixes.

Code changes that occur within a build are tracked. This information is automatically generated by the build tools and lists what modules were changed between builds. The information has some use when doing problem determination.

An item that I will discuss in more detail in the tools section is a very important step in change management. The full product group for Notes is always running on the latest product build. After going through some verification steps, the build is installed on the servers. This ensures the product is always operational and eliminates most regression errors while providing early feedback on new functions.

Metrics

Metrics currently are not extensively used as part of the development processes at Lotus. The company has gone through phases where they were used more extensively. Formality of metric use depends somewhat on the size of the product, but in most cases they are not used much. Bug/defect count tracking is done by nearly all groups. Find versus fix rate, size of the backlog, and mix of severities are used as in-process metrics during testing and they are also used to monitor post-ship quality. "Progress versus schedule" and "performance" are two measures used by all groups that generally fall into the metrics category.

Process Improvement

With the movement of groups between projects and with each project choosing the process they wish to use, significant process improvement steps do not tend to occur. The combination of movement and no base process is not something that leads to refinement of what has been done in the past. The Director of Development for DOS Spreadsheets does utilize Post Mortem reviews to get some level of learning for her, and hopefully for the individuals that make up the groups. Each individual group (development, QA, documentation) holds individual meetings to understand what was good and what could be improved on. These are not finger-pointing exercises and are designed to improve the next development project.

Tools

Notes uses their own product extensively in all work they do. It provides eMail support, project tracking, problem reporting, and collection of information in databases. The group always is running on the latest version of the product that was built, which exercises the code well before it goes out to any customers. Notes is therefore always operational. This extensive use of a product throughout the development process is very unique and the management team believes it helps the refinement of the product before it goes to customers.

Summary -- Results

Lotus has continued to allow small groups to determine the process they feel is best to get their work done. Requirements are allowed to change to adapt to customer requests and developers are given extensive freedom until the Code Freeze checkpoint is reached. This has led to significant product refinements due to developer freedom, but has made Quality Assurance a very difficult job.

Both products have been successful in their markets. Lotus holds the dominant position in DOS Spreadsheets and continues to come out with competitive refinements of the product. Notes has been very positively received and has a growing customer-base in a market segment that is expected to grow rapidly.

Conclusion

Overall View

Development life cycles are very consistent across the companies. Each utilizes the recognized phases of requirements, design, coding, testing, delivery, and maintenance. Within the phases, variations existed for the number of formal stages within the phase, the amount of pre-defined actions, and the level of formality surrounding the actions.

Significant effort was spent by each company on process support activities such as release management, change management, metrics, and process improvement. It was clear each company recognized that the software development life cycle was only a small part of what is required to develop software products, and they were spending significant effort improving their ability to manage software development. The companies were consistently aware of their problem areas through time and continuously take actions to deal with the problems.

"Established" and "New Entrant" Comparisons

My initial grouping of "established" versus "new entrant" companies didn't prove to be accurate. The most significant grouping is around the type of systems the software must "scale up" to and where the systems have evolved from.

"Established" development groups are IBM Federal Systems Company, IBM Application Business Systems, Fujitsu, and Hewlett-Packard. For the most part, these groups develop system support for mainframe and minicomputer systems that support multiple concurrent user groups on a single system. Each group has been in-place developing this type of support for a long period of time.

"New Entrant" development groups are the personal systems oriented companies developing software for a relatively new market. These companies are reasonably new and are developing products for markets that are still defining themselves. The markets have characteristics that are different from traditional computer software markets, and call for flexibility and creativity to be stressed more heavily. Processes used are evolving at a more rapid pace as the companies try to mix creativity, flexibility, and predictableness. Microsoft and Lotus are the companies that fall into this category.

To highlight some of the similarities and variations among the groups of companies, the following table is a summary of characteristics of each group across the process activities. The first column lists the activity, the second column addresses "Established" organizations, and the third column addresses "New Entrant" companies:

Activity	*"Established"*	*"New Entrant"*
Release Structure and Strategy	Structured to occur on regular intervals with support of hardware being a significant factor driving the content and schedule of a release.	Releases were based on function for initial and immediate follow-on releases. They have moved to a more predictable schedule-driven approach as products have matured.
Require-ments and Planning	Organizations utilize formal processes for these phases. Phases are driven by product management groups with executives having final approval of contents and schedules. Organizations are striving to meet the needs of diverse customer bases while expanding into new markets.	Requirements and schedules are determined by a very small group initially. This small group continues to maintain significant control over the product as it matures. There is continual input from developers and development management on what should be included in products. At Microsoft, there is a formal process to fix the top customer complaint areas in each release.

Design and Coding	Design and coding is manually done for the most part with the exception of Fujitsu. There is minimal usage of specification languages, automatic code generation, and Object Oriented programming. Inspections are formally used by each with very positive results.	There is less formal structure surrounding specific steps that must be carried out during development. These groups also have minimal usage of specification languages, automatic code generation, and Object Oriented programming. They are adopting design and code reviews due to positive early results.
Testing	In-house test groups are independent from the developers for 3 of 4 organizations. The in-house groups are strong for each company with ratios ranging from 5-10 developers per tester. Beta and other customer tests are used by each for technical and marketing reasons (though IBM FSC has a single customer). The HP-UX group uses the software being developed in day-to-day operations.	In-house groups work more closely with developers than in established companies. There is a high ratio of testers to developers (nearly a one-to-one ratio). Beta tests are used for technical and marketing reasons. Both companies use their product's latest code levels on a continuous basis to get additional testing of the code.
Process Usage and Compli-ance	Process usage is dominated by the size of the products and the integration needs. Processes need to be used by all developers. The process is needed for predictableness of schedules and quality. A strong use of metrics aids the checking of compliance.	There is some independence surrounding process choices. Fewer formal compliance measures are in-place. Process is introduced due to the need for predictableness on schedules.

Release Management	This is a significant activity in the organization due to size of product and number of people involved. HP and IBM ABS have gone to an individual focused on fighting fires and ensuring decisions are made on a timely basis. Cross-functional groups are in-place to support the individual.	A Project Manager is part of each product development group. Their focus is strictly on making sure the release gets done. They work closely with the Product Manager, managers of the specific development functional groups (development, testing, etc), and all outside groups (support, manufacturing, etc).
Change Management	It is done throughout the development phases with increased formality during coding and testing. IBM FSC and IBM ABS have moved it all the way up to the requirements stage. Tools are in-place to support change management.	Change management is done during the coding and testing stages to varying degrees. Loose controls are used during requirements and design stages. Change management appears to increase as products mature.
Metrics	They are used extensively to help manage the very large projects. Historical bases are in-place to compare progress results against. Metrics are used to manage schedules and quality. In-process metrics are being used extensively for design, coding, and testing stages.	Use of metrics is dependent on the maturity of the product and the historical base available. Microsoft is now using different metrics extensively for decision support on requirements and on escalations of decisions.

Process Improve-ment	It is extensively used by all four organizations. Causal Analysis and Defect Prevention processes are used to remove sources of error injection. Improvement is a continual process.	It is used by the mature organizations in the companies. Post-mortems are now a common and high-profile activity in Microsoft. Group continuity appears to be necessary for this to be effective. They are working to get more sharing and learning across groups.
Tools	The most extensive investment and usage was at Fujitsu. Each company has a well-established tool set to support change management, coding, and product builds. All make investments in automated tools for testing. HP uses their own product during development work.	Workstation based networks are used by both. Source code management tools are utilized with a variety of languages and linkers employed. Each attempts to use their product while it is going through development. Object Oriented programming is not being used for major projects.
General	The culture of the company and the formality of the development processes are tightly linked.	The culture of the company and the formality of the development processes are tightly linked. Both companies are finding the need to add structure to their development processes and have been continually doing that.

Closing

Even though there are a number of differences between the "established" and "new entrant" companies, there are an even larger number of similarities. Each company is at a different stage of organization evolution, and the processes they use to develop products are tightly linked to the stage in the evolution. Microsoft is a very interesting case to look at due to it being a single case that captures the evolution companies go through when moving from immature to mature processes. The movement is forced by internal and customer demands, and how an organization reacts to the pressure for movement has a significant effect on future company success.

Endnotes

1. Boehm, B. W. and Papaccio, P. N., "Understanding and Controlling Software Costs." *IEEE Transactions on Software Engineering*, Vol. 14, No. 10, (October 1988), 1462-1477.

2. Boehm, B. W., "Improving Software Productivity." *Computer*, (September 1987), 43-50.

3. Weber, C.V., Paulk, M.C., Wise, C.J., and Withey, J.V., "Key Practices of the Capability Maturity Model," Software Engineering Institute, CMU/SEI-91-TR-25, 1991.

4. Cusumano, Michael, *Japan's Software Factories* (New York: Oxford University Press, Inc., 1991).

5. Paulk, M.C., Curtis, B., Chrissis, M.B., Averill, E.L., Bamberger, J., Kasse, T.C., Konrad, M., Perdue, J.R., Weber, C.V., Withey, J.V., "Capability Maturity Model for Software", Software Engineering Institute, SEI-91-TR-24, 1991.

6. Cusumano, Michael, *Japan's Software Factories* (New York: Oxford University Press, Inc., 1991).

7. Ibid

8. Paulk, M.C., Curtis, B., Chrissis, M.B., Averill, E.L., Bamberger, J., Kasse, T.C., Konrad, M., Perdue, J.R., Weber, C.V., Withey, J.V., "Capability Maturity Model for Software", Software Engineering Institute, SEI-91-TR-24, 1991.

9. The description in this paragraph comes from a paper created for a class at MIT Sloan School of Management by Cynthia Schuyler titled "The Software Development Process - A Comparison: Toshiba vs. Digital Equipment", December 11, 1987.

10. Ibid.

11. The flow of the discussion and some details in this chapter are from the book *Japan's Software Factories* by Michael Cusumano. Details in the up-front description of the Life Cycle Model are based on a section of the book *An Integrated Approach to Software Engineering* by Pankaj Jalote.

12. The description in this section is based on a section of the book *An Integrated Approach to Software Engineering* by Pankaj Jalote.

13. The up-front description of the Waterfall Life Cycle is based on a section of the book *Software Systems Engineering* by Andrew P. Sage and James D. Palmer.

14. Royce, W. W., "Managing the Development of Large Software Systems: Concepts and Techniques," *Proceedings of IEEE WESCON*, pp. 1-9, 1970.

15. Jalote, Pankaj, *An Integrated Approach to Software Engineering* (New York: Springer-Verlag, 1991)

16. Ibid.

17. Sage, Andrew P. and Palmer, James D., *Software Systems Engineering* (New York: John Wiley & Sons, Inc., 1990)

18. Ibid.

19. Brooks, Frederick P., *The Mythical Man-Month: Essays on Software Engineering* (Reading, Mass.: Addison-Wesley, 1975).

20. Brooks, Frederick P., "No Silver Bullet: Essence and Accidents of Software Engineering", *Information Processing '86*, 1986.

21. This section is based on information obtained from discussions with Barbara Kolkhorst, IBM Federai Application Development Consultant, in January and April, 1993; externally published articles; internal IBM documents; and the 1992 IBM Annual Report.

22. This section is based on information obtained through interviews primarily with IBM Application Business Systems employees Dick Hedger, Manager of Quality Technology, Dave Amundson, Manager of Development Quality Process Technology, and Steve Kan, a member of Development Quality Process Technology, on March 16, 1993; externally published articles; internal IBM documents; and the 1992 IBM Annual Report.

23. This section is based on information from the book *Japan's Software Factories* by Michael Cusumano and *Datapro* information regarding Fujitsu's products.

24. This section is based on information obtained through interviews with Hewlett-Packard employees Dave Snow, Manager of Engineering Systems for the Commercial Systems Division; MaryAnn Betts, member of Group Process and Technology for Computer Systems Organization; Doug Herda, Manager of MPE/iX Process and Tools Project Team for

the Commercial Systems Division; and Cathrin Callas, Manager of Productivity and Quality for the Open Systems Software Division. The interviews were held January 24, 1993.

25. This data was compiled and published by Competitive Resource Center of the International Data Corporation, Framingham, MA.

26. This section is based on information obtained through an interview on March 15, 1993 with David Moore, Director of Development and current Director of Quality Assurance for Microsoft.

27. Gill, Geoffrey K., "Microsoft Corporation: Office Business Unit", *Harvard Business School Case 9-691-033*, Boston, 1990.

28. Interview with David Moore, Director of Development and current Director of Quality Assurance at Microsoft, March 17, 1993.

29. "Soft Lego", *Scientific American*, January 1993, and "Object Oriented Technology; Where Microsoft meet Berlitz", *InformationWeek*, March 1, 1993.

30. This section is based on information obtained through interviews with Beth Macy, Director of Development for DOS Spreadsheets, on April 16, 1993, and Rich Diephuis, Director of Development for Lotus Notes, on April 14 & 16, 1993.

Bibliography

Brooks, Frederick P., "No Silver Bullet - Essence and Accidents of Software Engineering", *Information Processing '86*, 1986.

Brooks, Frederick P., *The Mythical Man-Month: Essays on Software Engineering* (Reading, Mass.: Addison-Wesley, 1975).

Cusumano, Michael, *Japan's Software Factories* (New York: Oxford University Press, Inc., 1991).

Humphrey, Watts S., *Managing the Software Process* (Reading, MA: Addison-Wesley Publishing Company, Inc., 1989).

Ince, Darrel and Andrews, Derek, *The Software Life Cycle* (London: Butterworth & Co, 1990).

Jalote, Pankaj, *An Integrated Approach to Software Engineering* (New York: Springer-Verlag New York Inc., 1991).

Paulk, M.C., Curtis, B., Chrissis, M.B., Averill, E.L., Bamberger, J., Kasse, T.C., Konrad, M., Perdue, J.R., Weber, C.V., Withey, J.V., "Capability Maturity Model for Software", Software Engineering Institute, SEI-91-TR-24, 1991.

Ramamoorthy, C.V., et al., "Software Engineering: Problems and Perspectives," *Computer*, October 1984, p. 205.

Royce, Winston W., "Managing the Development of Large Software Systems," *Proceedings of IEEE Wescon*, August 1970.

Sage, Andrew P. and Palmer, James D., *Software Systems Engineering* (New York: John Wiley & Sons Inc,, 1990).

Schuyler, Cynthia, "The Software Development Process: A Comparison -- Toshiba vs. Digital Equipment", unpublished paper at M.I.T. Sloan School of Management for the course, "Japanese Technolcgy Management (15.940), December 11, 1987.

Sulack, Richard A., "Advanced Software Engineering Management Core Competencies", presentation at Spring 1993 COMMON Meeting, ASEMTECH, Inc., 1993.

Thayer, Richard, "Modeling a Software Engineering Project Management

System," Ph.D. dissertation, University of California at Santa Barbara, 1979.

Weber, C.V., Paulk, M.C., Wise, C.J., and Withey, J.V., "Key Practices of the Capability Maturity Model," Software Engineering Institute, CMU/SEI-91-TR-25, 1991.