Beyond the Waterfall: Software Development at Microsoft

Michael A. Cusumano* and Stanley Smith**
*MIT Sloan School of Management
**International Business Machines

Working Paper #3844-BPS-95
Draft: August 16, 1995

Section 1: INTRODUCTION

This paper analyses the approach to software development followed at the Microsoft Corporation, the world's largest personal-computer (PC) software company. Microsoft builds operating systems such as MS-DOS, Windows, and Windows NT; applications such as Microsoft Excel and Word (now usually packaged together in Microsoft Office); and new multimedia products and on-line systems such as the Microsoft Network. Inherent in our analysis of this company is a comparison to older software producers that have built operating systems and applications for mainframes, minicomputers, and workstations. Many of these older organizations have followed what has been called a sequential "waterfall" type of development process, which has both advantages (it can be relatively structured) and disadvantages (it is not very flexible to accommodate specification and design changes during a project). The objective for doing this comparison was to examine the effects on Microsoft of developing large-scale software products for the rapidly evolving PC software market. This challenge might have prompted Microsoft to become less unstructured or "hacker-like" and more like older software producers; on the other hand, it might also have prompted Microsoft to seek yet another development process that is structured but departs from waterfall-like practices in order to introduce more flexibility.[1]

Section 2 of this paper begins with a brief discussion of a stylized waterfall process, which includes such typical phases as requirements specification, design, implementation, testing, and product release. The waterfall model is widely considered to be the first well-defined development methodology and the base upon which most current software-development processes have been formed, particularly for formal project planning activities. The approach was probably first utilized on a large-scale project by IBM while developing the System/360 operating system in the 1960s. It represents a structured process and organization created for the purpose of developing large-scale software systems. But, even though many firms have refined this process over many years, companies have continued to encounter numerous problems in software development, due both to a lack of effective process management and to inherent deficiencies in the waterfall process itself. As a result, many companies, including parts of IBM, have been moving away from certain elements of the waterfall process.

Section 3 presents the analysis of Microsoft's development approach, which we have labeled the "synch and stabilize" process. Since 1988-1989, the company has gradually been introducing techniques that do add structure to software product development but which also depart from the waterfall model. Many aspects of what Microsoft does resemble concurrent engineering and incremental development practices found in other software companies and in companies in other industries. We begin by describing the development life-cycle steps, using the same terminology as in the waterfall model. Beyond this basic conceptualization of the development steps, however, we also attempt to describe more fundamental characteristics of the Microsoft process, such as approaches to project control, metrics, configuration management (how to manage the evolution of pieces of the product and various versions), process ownership, and process improvement initiatives.

Section 4 summarizes the similarities and differences in Microsoft compared to waterfall-type producers. In general, as its products have grown larger and more complex, Microsoft has

moved closer to the approaches used by older companies with more formalized development processes. Nonetheless, there are important differences. In particular, Microsoft's synch-and-stabilize process allows the product specification to evolve during development. It also makes it possible for large teams to work like small teams through techniques that enable frequent synchronizations among the developers and periodic stabilizations of the code without relying on one large integration and test phase at the end of a project. These techniques help Microsoft compete on the basis of new product features, whose details may change during a project. These techniques have also helped Microsoft build increasingly large and complex software systems, which it must continue to do as it expands into interactive video systems, video-on-demand, on-line network services, and other types of software products and services associated with the information highway.

Section 2: SOFTWARE DEVELOPMENT PROCESSES AND PROBLEMS

The Classic "Waterfall" Process

The waterfall process originated during the 1960s in firms building large-scale software systems for the U.S. defense and space industry as well as for commercial applications. These companies worked on multi-year projects and designed software for large computer (mainframe and minicomputer) systems that evolved relatively slowly. They modeled the waterfall process after hardware design projects, where engineers could more easily (though not always completely) predict how pieces of a system would interact.[2]

The classic waterfall model views the optimal process for software development as a linear or sequential series of phases that take developers from initial high-level requirements through system testing and product shipment (Figure 1). Designers begin by trying to write a specification that is as complete as possible. Next, they divide the specification into pieces or modules in a more detailed design phase, and then assign different individuals or teams to build these pieces in parallel. Each team tests and debugs (finds and fixes errors) in its pieces. Only in the last phase of the project, which could range from a few months to a few years for a large system, do the designers, developers, and testers try to put the pieces together and test the entire system. Usually, this process of integration and system testing requires reworking the modules and writing new code to correct problems in the operation or interactions of the pieces due to unforeseen problems as well as mistakes, miscommunications, or changes that have crept into the design of the parts during the project. If this integration work goes well, the team will ship the product when there are no serious bugs (errors or defects) remaining.

Coordinating the work of a large group of people building interdependent components that are continually changing requires a constant high level of coordination and synchronization that departs from the simple sequence of activities prescribed in the waterfall model. How to enforce this coordination but still allow programmers the freedom to be creative is perhaps the central dilemma that managers of software development face.

The waterfall model works reasonably well in stable problem domains where product development consists mainly of adding incremental changes to an existing core of functionality. It

is also suitable for projects where producers can control changes in design details and proceed to the end of projects or incorporate unplanned rework with little or no interference from customers or competitors. But the waterfall model is not a good framework to control the development process for products that have so much new content or so many uncertainties to resolve that changes in the design are inevitable and often desirable. In the PC software market, for example, both hardware technologies and customer requirements change very rapidly. In these types of cases, designers cannot write a specification at the beginning of the project that accurately captures what the optimal product design should be. Accordingly, product development cannot proceed in an orderly sequential fashion from design to coding and then testing at the end of the project.

In cases of uncertain requirements or fast moving markets, if designers try to create detailed specifications for the product and its pieces too early in the development cycle, the project team will end up building a product that does not meet customer needs very well or that is out of date before it even ships. If developers try to make changes in parts of the product as they go along, for example, due to interim feedback from customers or evolution in particular hardware or software technologies, or even just to add a feature that a competitor has just introduced, then the project may end up with pieces that no longer fit together. The integration effort and system testing fail. The project team then has to rework the pieces extensively -- even though they thought they had finished coding. They may even have to throw much of the code away. With these types of difficulties, it is no surprise that so many software projects end up being late, over budget, and riddled with bugs, due to errors in the pieces as well as in how the pieces interact.

Researchers and managers have proposed alternatives to the waterfall since the 1970s in the form of more "iterative" approaches to product development. These include notions of "iterative enhancement" as well as the "spiral model" of software development.[3] These alternatives see developers moving around in phases, going back and forth between designing, coding, and testing as they move forward in a project. This type of iteration is, in fact, a relatively accurate description of what many, if not most, software developers experience in projects -- but usually in an unplanned manner. Some companies refer to these iterations as "incremental builds," although they do not always incorporate these into formal project plans and checkpoints. Nor is it always clear in the iterative models when to stop the iterations or how to break up work into manageable pieces and phases.

Because most software projects require extensive iterations among specification, development, and testing activities, many companies have been moving away from the classic waterfall model in practice. Some companies also try to build software around reusable modules or objects, which means that developers need to begin with a detailed overview of the system objectives and then borrow from or build a library of reusable components as part of the design and development process. This type of activity requires a departure from the linear waterfall steps, although many companies known for pushing reusability tend to end with a single integration phase at the end of the project.[4] In short, we still find companies using elements of a waterfall process to plan and control software development, as well as other types of product-development. At least until very recently, in cases of procurement for the U.S. Department of Defense, information system providers have had to follow, as well as document that they follow, a precisely specified waterfall process referred to as the development life-cycle.[5]

Waterfall Process Implications and Problems

The major advantage of the waterfall or lifecycle model is that it provides a structure for organizing and controlling a software development project. The single most important methodological need in this approach, however, is to identify user requirements accurately.[6] Most software projects pay inadequate attention to fulfilling this need.

Disadvantages associated with the waterfall include problems that occur if there is no iteration and feedback among phases, beginning with the need to define the system requirements specifications accurately. Unless there is iteration and feedback, there may be no way to improve initial imperfections in one of the later phases. Realistic life-cycle development processes are, therefore, iterative and interactive, and a company's software development process needs to accommodate this fact.[7]

The largely linear nature of activities defined in the classic waterfall model thus have important consequences. For example, waterfall projects generally include a certification step at the end of each phase to mark the completion of one phase and the beginning of the next. Projects can accomplish this through some form of verification and validation to ensure that the output of the phase is consistent with its input, and that the output of the phase is consistent with the overall requirements of the system.[8] The goal of each phase is generally to produce a certifiable output. Many firms have found that reviews (formal meetings to uncover deficiencies in a product) are useful for certain key phases, such as requirements, design, and coding.

In a waterfall process, documents or code are the normal outputs of a phase, and outputs from one phase become the inputs for the next phase. Team members are not supposed to change outputs that the project has already certified.[9] In reality, however, requirements changes almost always happen. Software development projects thus need some mechanism for configuration control to ensure that team members make modifications in a controlled manner after evaluating the effect of each change on the product and progress of the project.

Companies have introduced many refinements to the waterfall process as well as introduced new programming tools and techniques to aid the software development process. Nonetheless, firms continue to report problems, especially in building large software systems for the first time. A recent list of common problems focuses on ten root causes of "runaway" projects and "missed objectives." Runaway projects are characterized by significant overruns of schedule, resources, or funding, while missed objectives consist of projects that do not meet customer expectations in some significant way:

1. Inadequate requirements statements.
2. Lack of specific and measurable goals.
3. Architecture design flaws and changes.
4. Inadequate change control systems.
5. Inadequate project status reviews and reporting.
6. Inadequate project metrics.
7. Lack of open project communications.
8. Lack of clear project milestones.

9. Overly optimistic estimations of project feasibility.
10. Various management difficulties.[10]

This list is similar to other lists compiled in the 1980s, 1970s, and 1960s, suggesting that the field of software engineering management has not made much progress.[11] Nonetheless, many firms have recently begun to depart from the waterfall style of management and to make other changes to improve their ability to control software projects, often in response to one or more disastrous projects.

Microsoft is one such company. It had a number of product recalls and extremely late projects during the 1980s and 1990s. A particularly important runaway project was the first version of Word for Windows. Microsoft started this in 1984 and ended up being off 500% in its original scheduling estimate.[12] Microsoft projects also typically generated hundreds and even thousands of bugs during the development process, too many of which the company did not catch prior to delivering products. By 1988-89, many applications products were also hundreds of thousands of lines of code, while the Windows operating system had grown to be more than a million lines of code. Microsoft could no longer build these products with small groups of programmers using ad hoc practices. As its customer base expanded to corporate users who demanded more reliable products, quality and delivery problems became unacceptable to Bill Gates and other top Microsoft managers. In addition, both retail customers and original equipment suppliers that ship computers with Microsoft operating systems and applications packages began to express concern that Microsoft was not utilizing quality processes to create its products, which had often become "mission-critical" for individuals and organizations.

The challenge Microsoft managers and developers began facing in the late 1980s was to introduce more structure and predictability into their loosely organized development process but still retain elements of the flexible and creative PC "hacker" culture. They viewed the traditional waterfall model as inadequate because it required too much structure: It demanded that teams determine and "freeze" a product's requirements at the beginning of a project, and that they build components that precisely followed this specification. Microsoft needed a different process that allowed team members to evolve product designs incrementally, responding to customer inputs, prototypes, competitors' products, and changes in the fast-moving PC hardware industry. We will now describe the process that Microsoft groups have been using since 1988-1989 to build products such as Excel, Word, and Windows NT. Other product groups have also adopted versions of this approach, which we refer to as the "synch and stabilize process," during the 1990s.

Section 3: MICROSOFT CASE STUDY[13]

The Company

Bill Gates and Paul Allen (who retired from Microsoft in 1983 but remains on the board of directors) founded Microsoft in 1975; they company went public in 1986. Microsoft started by building programming languages, and now develops, markets, and supports a wide range of microcomputer software for business, professional, and home use. Its software products include operating systems, programming languages, application programs, communications programs, and an on-line network. Microsoft also develops and markets microcomputer-oriented books, hardware, and multimedia CD-ROM products.

Corporate headquarters are in Redmond, Washington. Total worldwide employment was about 17,800 in 1995, including 13,300 people in the United States. Research and development is also based in the Redmond complex, with small satellite operations in Tokyo, Japan, and Vancouver, Canada. Diskette manufacturing takes place in Washington, Ireland, and Puerto Rico. Microsoft also has direct and indirect marketing operations in 30 countries.

Microsoft is the leading PC software vendor based on revenue with 1995 sales of $5.9 billion. Microsoft is also one of the most profitable companies in the world. The company's revenues and profits have been 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 international operations to new areas.

Leadership and Organization

Microsoft has a "communal" leadership first instituted in 1992: the Office of the President. After a July 1995 reorganization this includes, in addition to Bill Gates as Microsoft's chairman and chief executive officer, five senior managers who direct Microsoft four operating groups: Group vice presidents Nathan Myhrvold (formerly head of advanced technology) and Pete Higgins (formerly head of desktop applications) jointly preside over the new Applications and Content Group. Group vice president Paul Maritz (formerly responsible for product and technology strategy) heads the new Platforms Group. These two groups build Microsoft's products and conduct research and development. Executive vice president Steve Ballmer is in charge of the Sales and Support Group, while Robert Herbold is head of the Operations Group and also serves as chief operating officer. Reporting to these group executives are division vice presidents and general managers. Below them are product unit managers, followed by functional team managers and then team leads in the product groups.

The Applications and Content Group has four divisions: desktop applications, consumer systems, on-line systems, and research. The Platforms Group also has four divisions: personal operating systems, business systems, developer and database systems, and advanced consumer systems. Most of these divisions contain their own marketing departments staffed by product planners and share a centralized usability lab (staffed by about 35 people) to test features and product prototypes. The Sales and Support Group has separate divisions for worldwide OEM sales, product support services (abbreviated as PSS), international operations (mainly Asia), advanced

technology sales, strategic enterprise systems (special sales and consulting to large firms), North American sales, and European sales. The Operations Group includes finance, diskette production, manuals and book publishing (Microsoft Press), information systems, and human resource management.

Within the Platforms Group, the personal operating systems division produces Windows and MS-DOS. The business systems division produces Windows NT and Object Linking and Embedding (OLE), with a separate product unit for workgroup applications (electronic mail and PC server systems). The developer and database systems division builds programming languages such as Visual Basic, programming support tools, and database products such as Access and FoxPro. The advanced consumer systems division contains groups for interactive TV systems and broadband communications and multimedia technologies. Within the Applications and Content Group, the desktop applications division contains the Office product unit. This supervises the Word and Excel product units and works closely with the Graphics (PowerPoint) product unit to make sure that these three products function together properly in the Office applications suite. The division also builds Project, a popular project-management tool. The consumer systems division includes the "Microsoft Home" product groups, which build multimedia applications for home entertainment and education, and a combination word processor, spreadsheet and database product for novices called Works. The on-line systems division develops and manages the new Microsoft Network. Research explores new product and programming technologies, and works closely with various product groups.

Within the product organizations, there are five major functions that can each have an individual manager for the larger products or can be combined for smaller products. These functions are program management, development, testing, marketing (product management), and user education. The functional groups in each product area define their own development processes as well as continually inject new ideas and improvements.

Program managers are responsible for consulting with developers and then writing down specifications for the features of a new product. They also manage the product through all stages of development. In addition, program managers act as the liaison to other dependent groups, and manage independent software vendor relationships for applications bundled with the product they manage.

Developers are responsible for feature design and coding. For each product, they form feature teams that work with one or more program managers. Recalc, charting, printing, and macros are examples of the eight feature teams on a former Excel project. Each feature team has a team leader along with several team members (usually 5 to 8).

Testers are responsible for working with developers and testing products from the perspective of average users. They also organize by feature teams that match up with the development feature teams and individual developers. Testers start their work early in the development cycle by reviewing specifications as they evolve, and they continue all the way through final testing.

Marketing, called product management in Microsoft, consists of marketing specialists as well as product planners. Marketing specialists study competing products, research customer needs, and prepare for the sale of new products. Product planners work with program managers

to define a vision statement for new products. This statement outlines and prioritizes basic feature ideas, as well as clarifies the objectives and target audience of the new product. They also use focus studies and other mechanisms to get product input that is combined with input from the sales organization during creation of the product requirements.

User education is responsible for producing manuals, electronic help files found in products, and other documentation for customers. These people also determine what user training new products might require..

Beyond the formal organization and reporting structure, Microsoft also has an informal "brain trust." This consists of more than a dozen senior managers and technical people spread throughout the organization. Bill Gates and other top executives call upon them for advice or to take charge of particular projects and research efforts. In addition, Microsoft has company-wide directors for functions such as product development, software development, and testing, to help groups share good practices, learn from their experiences, and adopt common methodologies and standards.

Microsoft's Sales and Support Group, which receives about 20,000 phone calls per day, contributes significantly to product development and improvement. The more than 2,000 people handling these phone calls log each customer inquiry, noting what product the call concerned and what type of problem the user encountered. Support staff then generate a report summarizing information from these calls as well as customer suggestions, and send this to the development groups as well as to top executives (who avidly read the reports) each week. In addition, the support people have one team in place for each product, and a person on the team maintains direct contact with the developers, working with them on a consistent basis. For example, the product support team discusses customer calls as well as reads new product specifications and begins preparing for customer support during beta testing of a new product.

Culture

Microsoft's culture is evident in two of the company's most important goals: hire the best people possible, and give them private offices as well as good tools to do their jobs. Hiring the best people has been the focus within the company throughout its history. Microsoft recruits graduates from a variety of universities with backgrounds in computer science and other technical fields, as well as experienced PC software developers. These new hires join product teams and usually stay with the same team for at least two development cycles (three to four years for applications, and longer for systems products). This is true for program management, development, testing, marketing, and the other functional groups. Staying with the product gives people a long-term investment in the product, ensures their familiarity with it, and helps them understand process liabilities and benefits as well as learn from prior project experiences.

Bill Gates' presence and personality continue to have a significant influence in Microsoft. Many observers believe that the key reason for Microsoft's remarkable success is actually Gates himself. He represents a technical visionary who is also the leader of the company and a skilled manager who knows what to delegate and what to control. His involvement extends to reviews and input on the specifications for major products and their long-term development plans. The chairman and the people he has hired over the years are also fiercely competitive, driven to

achieving technical excellence, making and meeting aggressive commitments, and doing whatever it takes to ship products. Due to the aggressive schedules that are frequent in the company, significant stress, turnover, and "burnout" are common among employees. On the other hand, the work atmosphere is one of flexible hours, flexible dress, and open, honest relationships.

Changes in development methods provide some evidence of cultural changes through the years. The early development culture in Microsoft was one of small teams, ad hoc methods, and 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, did not like the way the product worked, so he started from scratch and completely rewrote it. He still did not like the product, so he sat down and started from scratch one more time.[14] The process involved his own vision of how the product should work and how the internals should be designed and coded.

When Microsoft moved from doing OEM work to developing products for the retail market, the culture changed with the addition of formal specification, testing, marketing, and support groups. IBM also significantly influenced testing at Microsoft 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 with the growing size and complexity of its products, developers changed the culture by adopting practices such as informal code reviews and more formal design and planning methods. The final significant influence has been the evolution of PC software to become "mission critical" applications for many companies and other organizations. Purchasers now demand that PC software suppliers have high-quality, repeatable processes in place to develop and support their products. As more systematization has become necessary in Microsoft, the company has increasingly tried to combine more structure in its development processes with enough flexibility to maintain the individual creativity and freedom of action needed to create leading-edge products.

Product Description

Microsoft now has about 200 separate products, although most are either systems or applications software. The company also sells some hardware products, such as a mouse and trackball pointing devices, and a keyboard, but these are a small part of the business.

Systems and language products generate about one-third of Microsoft's revenues. The Windows operating system is the major product offered in this area. Windows is a graphical user interface and operating system shell written for Intel-based PCs that works on top of the older MS-DOS operating system. Windows is easy to use, allows convenient data sharing, provides support for organizing and managing files created by applications programs, and allows switching between different application programs. It also lets programmers write larger applications than MS-DOS does. Estimates are that about 70 million users have adopted Windows since its introduction in the mid-1980s, with a majority adopting this after the 1990 introduction of version 3.0 and the 1992 introduction of version 3.1. Microsoft recently

introduced another version, called Windows 95, which is widely expected to become the next standard for desktop PCs.

MS-DOS was the base operating system for the first IBM PC and has continued to be a standard for character-based PCs. The initial MS-DOS version came out in 1981 and updates have continued through MS-DOS 6.0 in 1993. This product still brings in a significant amount of revenue. Windows NT is the advanced product in the operating systems group, first introduced commercially in July 1993. This is a 32-bit operating system designed for networking PCs and workstations at corporations and other sophisticated users. It also functions as a server for interactive television, video-on-demand, and on-line services provided through the Microsoft Network.

Applications products generate about 60% of Microsoft's revenues. These include an extensive range of products for Intel-based PCs and Apple Macintosh computers. Microsoft Excel, the company's spreadsheet application, competes with Lotus 1-2-3 for leadership in this category on Intel-based PCs and is the clear leader for the Macintosh. Microsoft Word, the company's word processing application, competes with WordPerfect for leadership in this category on Intel-based PCs and is the clear leader for the Macintosh. Microsoft also offers Word and Excel in an integrated application suite called Microsoft Office, which costs about the same or less than just one of these programs cost when purchased individually only a few years ago. Office now accounts for about 70% of suite sales in the industry and more than half of Word and Excel sales. The standard Office suite includes PowerPoint for business graphics, while a professional version contains an electronic mail program as well as the Access database management program.

Review and Planning Cycle

The review and planning cycle is a logical point to began our analysis of software development in Microsoft. The company splits the cycle into two portions occurring in October and April. The result of the cycle is agreement among company executives on product roll-outs and funding for the divisions.

The October review centers on presentations of three-year product plans. The product groups define the number of releases they are planning, explain why they are doing a release, and discuss 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 Gates during this phase. Gates also interacts extensively with major product groups during the development process, usually through electronic mail and some personal visits and meetings.

After completing the October review, the marketing organizations create sales forecasts based on the product plans. The divisions then plan their budgets based on product sales forecasts. Managers look at the sales versus budget mix to determine how it compares with the profit model for the company. Based on this analysis, managers determine head-count for the fiscal year that begins in June. To our knowledge, Microsoft has never hit a point where managers limited the personnel needs of its divisions due to head-count or budget restrictions.

Release Structure and Strategy

Product unit managers determine and gain approval for release plans for the individual products during the October review. In earlier years, product releases were more function-driven, based on the key features that product managers, program managers, and developers wanted to see in the next version. This has changed through the years to where the delivery date is now most important, except in the cases of operating systems (such as Windows 95), where product stability and reliability (quality) is most important. In applications and systems, however, groups make tradeoffs in functionality to reach the target delivery date. 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 during 1986-1988 and came after a long history of missing dates, a practice no longer considered acceptable by customers or Microsoft managers.

Product groups change a lot of code for each new release of a product, which makes it even more difficult to predict when a product will be ready to ship. Estimates are that groups change 50% of the existing product code for each new release. In addition, groups tend to add another 30% to a product in new code for new functions in the release. The result is that code in Microsoft has an average half-life of only 1.5 years. For this reason, extensive automated regression tests are critical to development at Microsoft. Without them, groups could never test new products in time to make Microsoft's aggressive update schedules, which often call for a new product release every 12 or 18 months.

Development Process Overview

Some Microsoft product groups are further along in areas such as usage of metrics and adherence to review steps, but nearly all follow a relatively consistent high-level methodology for software development. Microsoft managers and developers first discussed elements of this process in a May 1989 retreat, where about 30 people in the company gathered to discuss ways to produce software with less defects. A memo from this retreat, dated from June 1989, dealt with the subject of "zero defect code" and the strategy of building products daily. A 1989 "Scheduling and Methodology Document" of about 40 pages, drawn up by the old Office Business Unit, also discussed elements of the new Microsoft process. Neither of these documents are widely circulated in the company, in part because of a general dislike within Microsoft to document processes in much detail since this may prevent change and improvement. Each group is also free to define the details of their process. Nonetheless, most groups have adopted similar processes, and variations are relatively minor.

In general, Microsoft's development process has three main characteristics that differ from a more traditional waterfall process once used commonly at mainframe and minicomputer software producers. First, Microsoft divides the development cycle into three or four milestones, with each milestone containing its own coding, testing, and stabilization (debugging and integration) phases. Groupings of features determine the milestones. In general, projects try to do the most difficult and important features first, in case they run out of time later on. This milestone process contrasts with conventional life-cycle development, where projects try to write up as complete a specification as possible, then break up the work into parallel teams, followed

by one large integration, system test, and stabilization phase after development. In fact, Microsoft (and many other firms) have found it difficult to specify and then build all the pieces of a complex system and then try to put them together only at the end of the project. As a result, Microsoft puts pieces of a large system together three or four times during the development cycle, with refinements of the specifications as well as development, testing, debugging, and integration activities all done in parallel during each milestone. Microsoft's process thus resembles concurrent engineering as well as incremental styles of product development used in other industries.

Second, Microsoft projects assume specifications will change during development, so they do not try to write a complete specification and detailed design document at the start of a project. They write a "vision statement" to guide developers and a functional specification from the user's point of view, but produce detailed specifications only for well-understood features, and allow project members to change the specification as they deem necessary.

Third, Microsoft projects create a "build" of the product everyday. (A build consists of checking pieces of code into a master file, and then putting the pieces together to see what functions work and which do not.) Not every developer has to check in code every day, but any developer who checks in code that conflicts with other features that have changed since the last build has to revise his or her code. The result is that developers try to check in their code as frequently as possible, usually about twice a week. This daily build process also helps developers evolve their features incrementally as well as coordinate or synchronize their changes with the work of other developers, whose features are often interdependent.

To develop products, Microsoft utilizes empowered teams that are responsible for all stages and the decisions required to get their product to market. The groups attempt to keep the teams small or arrange larger teams by product features to keep the small-team atmosphere. A full team that includes people from the five functional areas is in place for all products.

From a high-level viewpoint, the development teams are responsible for the following things: (a) Producing a vision for the product which states what quality means for this product (bugs, performance, reliability, function). (b) Producing specifications, designs, code, tests, and validations of the final packaged product. (c) Product improvement with input from marketing, program management, Bill Gates, and anyone else with an opinion. (d) Process improvement through usage of post-mortem reviews along with mid-project changes needed to get delayed or problematic products back on track. And (e) customer awareness via ties to the product support organization, monthly 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, although most groups use a minimum of three in the product cycle: *schedule complete* (the functional specification is complete and approved), *code complete*, and *release to manufacturing*. The development team commits to the set of features or 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 writing the actual product code. Groups also have other internal checkpoints and interdependency plans. Microsoft people do not see

themselves as having significantly unique process concepts, but instead, they feel that they utilize some new ways of putting good development concepts together.

Investments within Microsoft for development have tended to follow, in order, people, specifications, tools, design and test plans, and code test cases. When problems arise during development, managers go through these investments in reverse order in an attempt to fix the project. For example, they act starting from the bottom, making people changes only as a last resort. Microsoft managers have found that people changes are the most destructive in the long run and should be avoided if at all possible. Recognizing this as a decision model appears to have been effective for negotiations and efficient problem solving within the company.

Requirements Phase

The product marketing team in each product unit creates a Vision Statement for a new product or version that defines its general direction (Figure 2). The statement describes the overall focus of the product, how to market it, the purpose of the next release, and the basic areas that the next release will address. Statements like, "Fix the top 20 problems reported to the product support organization and add functions 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. Managers approve schedules during that review, and groups either proceed or change their general direction 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, especially the developers, who best know the code and what is feasible technically. Program managers utilize inputs to create a list of what to include in the product release.

Program managers write specifications from a user viewpoint. They show menus, commands, and dialogues that users will see, as well as error messages that can come up. They do not specify "how" to solve a functional requirement at the level of coding, which developers will do during the design stage. Even though they are incomplete during the specification stage, specs evolve during development and can be quite lengthy due to the amount of graphical presentation in them. For Excel, the spec is usually 300 to 500 pages, although it reached over a thousand pages in one recent version. A team from program management, marketing, development, testing, and user education do continuous reviews of the spec before holding the final review. Development groups generally do not use a formal review process during this stage, but most try to do a complete review of the specification that exists prior to starting development.

Development and testing groups are responsible for refining the spec during the development process. Developers flesh out details surrounding the functions, estimate the amount of work in person months, and estimate the schedule for their individual pieces of the project. Testers provide early input on whether features seem testable or not, estimate the amount of work in person months for their part of the schedule, and define what they need from development to test and support the product.

We noted earlier that Bill Gates still plays an important role in the specification process. Program managers are responsible for figuring out how to get his input for their product. They need to obtain this during the specification stage and get Gates to "buy in" to the spec. Each major product will have at least one formal review with him, and key products may have multiple meetings. During the meetings, Gates will set some key goals for the product that may relate to quality, cost, or function. Before a major product can move on to the implementation stage, it must have formal approval from Gates; this constitutes the schedule complete checkpoint. In the past, he personally reviewed every spec in detail, but has since hired a full-time assistant to help review the specs and follow up with projects as well as monitor competitors and their products.

An important aspect of the specification stage is the use of prototyping, done mainly using a Microsoft tool, Visual Basic. Program managers always build prototypes during the specification stage that include menus, commands, and dialogues and serve as inputs to the spec. In some cases, the prototype may become the spec and program managers will use this for the final meeting to get approval before starting implementation.

Development Phase

We noted earlier that Microsoft groups generally break up a project into three or four milestones (Figure 3). Product managers, with program managers, write up a Vision Statement that outlines and prioritizes features for the new product. Each feature has multiple functions and may require more than one milestone to complete. In general, however, product teams try to complete, for example, the first third of the most important features (or the most important functions in particular features) in the first milestone, the second third in the second milestone, and so on. Code complete is the final step after the last milestone indicating that the team has finished design and coding work and the product is ready for final testing. Individual developers and groups determine the process and checkpoints necessary to meet the functional and schedule commitments.

Design: Developers need to do enough preliminary design work or analysis during the specification stage to make a solid estimate of the amount of effort and time required to complete each feature. Developers make individual estimates and view these as personal commitments, which encourages them to do a reliable job on the estimates and the early design work.

Microsoft does not have a formal set of design stages. It is up to the development team to determine what to detail during this step. Developers do much of this determination based on their experiences from prior releases. They deal with module structure, dependencies on other functions, input/output details, and other normal design stage considerations during this period. Developers may also hold design reviews for their work. They do not use any special specification languages or code generators.

Coding: PC and Macintosh products utilize much of the same code. Only about 10 to 15% of the code is unique for these two different platforms. On the other hand, reused code between products amounts to only about 5 to 10% of all product code, although this is now changing.

Most reused code in the past has been for user interfaces, which have many standard elements. Microsoft has not usually developed code with reuse as the objective. Most reuse used to happen through the general developer approach of "stealing what I can." As more Microsoft products came to contain common features, like graphing and spreadsheet functions in Word, Excel, and other products, Microsoft has begun to design these features as large "objects" that one group will write once and then link and embed in numerous products. Microsoft calls this technology Object Linking and Embedding (OLE), and has made this available as a commercial product. Groups do not widely use object-oriented (OO) programming languages like C++ for major products. New projects, however, including an object-oriented version of Windows NT,[15] are doing more work with C++ as a basic programming language. Parts of newly released products, such as some of the communications and network portions of Windows NT, are written in C++.

There is great allowance for individual coding styles, although most groups use a naming convention called "Hungarian," invented by Microsoft developer Charles Simonyi. This helps people read each other's code. At least one group, Windows NT, also has a coding manual, which serves as a rough style guideline.

Another important feature of development in Microsoft that corresponds to the idea of building prototypes and testing work under development is the utilization of internal "usability labs." Developers and some program managers use these labs to test how easy a particular feature or presentation of a feature is for the average person to understand. Microsoft internally has several rooms set aside as usability labs. A test consists of 10 people brought in from "off the street" to try a feature under development. The lab staff videotapes the session and tracks the number of people that get the feature right on the first try. Most developers make very extensive use of the labs to prototype and test their features during development.

Code reviews have increasingly become part of the standard process at Microsoft. Various groups tried the reviews and found them so beneficial that most development teams decided to use them. But, unlike at most companies that conduct code reviews in relatively large formal meetings, Microsoft code reviews usually have only one or two reviewers. Reviewers go through another person's code independently, usually in chunks of 2,000 to 5,000 lines at a time. Strong competition exists to find both defects and design mistakes during this stage.

The coding phase focuses on the code complete checkpoint after the last milestone. Developers estimate this date and all activities center around achieving it, even though it is not usually clear that a project has reached code complete until a month or more after this point, when it becomes certain that the code (and features) are indeed stable. The development manager polls each developer to determine whether they consider themselves finished. When all are ready, the team declares code complete and testing can begin. After reaching the code complete target, the only code changes allowed are approved bug fixes.

Before code complete, four other targets are part of the coding stage. (1) *Private releases* 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. (2) *Visual freeze* 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 to 30% change occurs after the freeze. (3) *Functional freeze* is utilized to lock the text information used for

documentation, although not all products use this checkpoint. The user education department drives this and negotiates the date with development. Typically, 20 to 30% change occurs after the freeze. (4) *Beta test release* signals confidence that the code is good enough to send to selected user sites for actual usage testing.

During the coding stage, developers continually tell testers what sections of the code are complete and which are incomplete. This communication allows targeted functional testing to begin as soon as possible. As a final step in development of the new code, developers run a mandatory suite of tests as internal checks 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.

Daily Builds and Integration Testing: Developers keep all code modules 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 with their PC. When developers finish making changes, they run a set of unit regression tests to validate the new function they have added. In addition, they must run a suite of preliminary integration tests to validate that base functions are not affected by the changed code. These tests are called quick tests, synch tests, or smoke tests, depending on the group. If all tests are successful, developers can do a "check in" to put the new version into the master library.

Most projects do daily builds of the master code. The builders then run build tests to ensure the products will operate. Problems found must be immediately resolved and everyone stops work until the guilty developer fixes the problem. Since teams do builds daily, tracing back to find the change that caused a problem is reasonably easy to do. Daily builds ensure that the product will function at all times and control the amount of churn in the system, which helps stability. For large integrated products that consist of separate products or components from different groups, or products that rely on different systems, such as Office or the Microsoft Network, Microsoft groups will do daily builds of the components and weekly builds of the entire system.

Testing Phase

Unlike organizations that rely heavily on specification and design reviews to find defects as early as possible, Microsoft relies heavily on daily builds, automated testing, and manual testing by testers that work in a one-to-one ratio with developers. Automated suites of tests available for developers to run prior to integrating their code are extensive and widely used. Test tools for developers and testers to test new functions are also available and very useful.

Each of these items is helpful, but the most significant difference in Microsoft's approach from other firms is in the relationship between the testing and development groups. Testing is a functional group within the product development organization. There is no independent quality assurance organization, although the testing managers report directly to the product unit general managers, not to the development managers. Testers also have a very close relationship with developers. Like the developers, they are involved with the product over multiple releases, and

they are organized in feature testing teams that work in parallel with the feature development teams. Involvement starts at the spec stage and continues through the rest of the cycle.

Each developer also creates what Microsoft calls a "private release" for the tester assigned to work with him or her. 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 test cases while the developer can get bugs discovered early and re-code as necessary. This coordination assists developers during development tests and assists testers for their final tests.

Microsoft carefully plans testing phases. Testing personnel do their own estimates of resources and schedules during the spec stage and commit to meeting the plan. They create formal test plans and review test cases. Developers participate in 70 to 80% of the test case reviews. Testers add automated tests from prior releases to the plan so that they can understand the total test coverage.

Final test is the main verification step that the testing organization runs. Microsoft tests products through customer-like usage and tracks results closely against the test plan. Testing includes documentation, tutorials, set-up, hardware configurations, primary functions, and supporting utilities. Automated test cases are key to validating existing functions; testers use them extensively. They also measure performance against goals set for the product. Results from the final test are the most critical input to the ship decision.

Most groups use three types of beta tests to stimulate awareness and excitement for a new product or feature (marketing reasons), and to get feedback and remove bugs (technical reasons). The three types of tests 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 of the product to employees to get results similar to wide tests. Beta tests tend to get a very low response rate of 5 to 6% of users giving feedback to development.

Developers have a set of scheduled checkpoints during the test phase where they attempt to get the number of outstanding severe bugs down to zero. "Zero bug releases" are one set of checkpoints where development consciously attempts to drive down to the target of zero known severe bugs. Product groups tend 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 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 a solid product.

Projects make ship decisions after final test. The senior (group) program manager organizes a "committee" that includes himself or hersel as well as the development manager, the test manager, the product marketing manager, and a representative from the customer support organization. This committee recommends whether or not to ship the product, although the product unit manager is ultimately responsible for the ship decision.

Product Support

Separate support teams exist for each product. These teams are part of the Sales and Support organization and not part of development. Their main responsibilities are to handle customer calls for the product and channel information from customer calls into the business units to guide decisions on features or fixes for subsequent releases. When problems come in, the support organization logs them and creates problem reports that come to the development group on a weekly basis. Program managers, developers, and testers all carefully follow these problem reports and arrange for solutions. The development staff on the current product also handles all maintenance work such as fixing bugs from the current release, and all or part of the team will work on fixing problems when they come in.

Process Usage and Compliance

Each product group is responsible for choosing the development process they will use, although nearly all groups utilize a version of the synch-and-stabilize process described in this paper. The process we describe originated primarily within the Excel group during 1989-1990, although other groups used aspects of it before this time. There are also some differences in how groups building applications products as opposed to systems products utilize elements of the process. Nonetheless, the principles of synch-and-stabilize have gradually spread throughout the company's development groups, as people have moved and as managers such as Dave Moore, the Director of Development, and Chris Peters, Vice President of the Office Product Unit, have encouraged groups to adopt "best practices" that have been proven to work.

As a result of these conscious efforts to "evangelize" best practices from the Excel group, over the last few years, Microsoft has rapidly progressed in usage of a more definable and repeatable process. Developers have recognized that these practices have become necessary as project sizes and products have grown enormously in size and complexity. Excel and Word each have at least 10 program managers, 30 developers, and 30 testers, and are both a million or so lines of executable C code. The Office group overall (including Word and Excel as well as PowerPoint and a group developing common components) has about 100 developers and an equal number of testers, and the total product is several million lines of code. Windows NT and Windows 95 each have teams of approximately 200 developers, 200 testers, and 30 to 50 program managers; their core products are between 4 and 5 million lines of code.

Customer demands have also been a key factor in accelerating the adoption of more solid and verifiable processes. As PC applications have become more central to organizations, customers have demanded in-process metrics and other indicators of quality before they will install new versions of products.

Microsoft does not handle process compliance via formal mechanisms, however. The pressure of the daily builds, milestone integrations, and program reviews are the main drivers of compliance. Another mechanism is internal audits done by Dave Moore and other functional directors, which now include Moore's boss, Chris Williams, the Director of Product Development, and Roger Sherman, the Director of Testing. Senior managers also occasionally ask the functional directors to work with different groups by analyzing problems and the current status of projects, and make recommendations for improvement. Managers use a formal "audit" to change things quickly. They use a "review" 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

In addition to writing down specifications with the help of developers, program managers keep track of schedules and coordinate with other groups who may be providing components to a particular project. Their job is difficult because they must rely on developers to write code, but they do not have direct authority over developers, who report to their own development team leads and a development manager. Nonetheless, program mangers work closely with developers, with one program manager usually assigned to work with each feature team.

Beyond constant contact with the groups creating the product, there are two other mechanisms that are critical to project management: First, each of the functional groups, and individuals in those groups, determine their own schedules. This means that people doing the actual work do all their own estimating. By having this relationship between estimates and work, individuals become very committed to meeting the schedules they set. One problem with this approach has been that developers are usually overly optimistic about how much time a job will take, leading to badly mis-scheduled projects or developer "burnout" as people try to catch up with a schedule that was unrealistic to begin with. New personnel also do not have much experience to create estimates. As projects accumulate historical data on past work, however, they are improving in their ability to make realistic estimates. The development leads also give assignments and schedules to new developers for the first several months after they join a project. In addition, teams now debate each member's estimates informally to improve accuracy.

Second, product groups utilize project reviews throughout the development process. Program managers schedule and run these reviews either weekly or monthly. Managers review everything associated with the project with each group as they report their status. Monthly status reports also come in from each functional area. Major program reviews on project status are also held with Bill Gates and other senior executives. Timing of these program reviews vary depending on the strategic importance of the product.

Change Control and Configuration Management

Microsoft has network servers to store source directories accessible by everyone in the company. Groups use password control to limit access to some of the source directory servers. Groups use network-based configuration control on everything associated with the products under development. Items in source directories include project documents such as specifications, 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 information related to a project.

Groups allow changes to requirements, specifications, and code during the development process. After checkpoints such as Schedule Complete and Code Complete, the program managers take control of changes to specifications and code respectively. By allowing approved changes, they let innovation continue to happen during phases such as coding and testing. When decisions are required for necessary changes, many groups use an informal decision model to

determine the action necessary. The model, from highest priority to lowest, is: (1) schedule and resources; (2) components, functions, or features of the product; (3) future extensibility and maintenance (these are bad for the long run but may be necessary); (4) product performance; (5) product reliability and quality (this is definitely only done when no other options exist. The changes may be "not fixing" something that was previously planned).

Tools on the system manage code changes. 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 a function in the network control tool that compares changes to ensure the same lines have not been altered. Before developers can check code back in, they must run "synch" tests that make sure the code does not degrade the system. Nearly all projects do daily builds on the total product, then run synch tests. Any problems discovered holding up development are resolved by the developer making the faulty change. Daily builds allow the product to be usable everyday. In addition to the "check out" procedure, the project's senior managers must approve changes to code after Code Complete.

Microsoft groups manage defects through a set of bug tracking tools that run on the server. Team members enter bug reports into a database along with a description of how the problem can be recreated. Severity levels running from 1 (most critical because they cause a system to crash) to 4 (not critical and may simply indicate a new function request) are assigned by the discoverer of the bug, although managers usually debate these levels for remaining bugs before moving on to another milestone or shipping a product. Development managers continuously monitor the database so that they can assign the problems to someone on the team when they are reported. Testing and program management also closely track the defects.

At the end of the development process, the change control process takes on an additional level of formality. The committee of four (one member each from development, testing, program management, and product support) meets daily to review all remaining problems and determine which to fix. Internal testing and beta tests both generate problem reports. Utilizing the committee review helps ensure that groups make decisions based on data rather than emotions or pressures to ship. Approval requirements, plus the tracking capability, also provide a level of change management for bugs.

Metrics

Data and metrics are important to resolve conflicts and make decisions on actions to take. Many company people told us that "Microsoft is data driven." Top management supports the usage of metrics since experience suggests that these help projects ship on time. The most watched and used metrics involve bugs. Tools are also in place and available to generate metrics and data. Some common metrics are bugs to date, bug severity mix, open versus fixed bugs to date, bugs found versus bugs fixed, clusters of defects, code churn, code test coverage, and customer problem calls versus units sold.

The bug metrics described above are very important during the development process. Groups also generate standardized queries and reports for management at defined intervals. Some project teams collect and use historical data as well, although Microsoft does not have a central

company-wide database for project metrics and data. If internal data does not exist for a product, groups may share data or use applicable external data. Some projects frequently use data that indicates how many bugs are likely to be in a product and how many should have been removed through each of the development stages.

Process Improvement

New process ideas come largely from the teams themselves. Managers encourage teams to find "best practice" solutions to process problems, try them out, and talk about their experiences to other groups. The best practices information has come from the efforts of key project managers, as well as the functional directors, to identify what works well in Microsoft's product units as well as what has worked well in other companies. Groups adopt improvements by trying new ideas and spreading information on the results; functional directors and other managers generally do not mandate that groups use particular processes or tools.

About two-thirds of all projects also write postmortem review reports, which can range to a hundred pages or more. Usually, the manager of each functional area (program management, development, testing, user education, product management) consults with team members and takes responsibility for writing up the portion of the postmortem that relates to their part of the process. Each section generally contains three parts: what went well in the last project, what well poorly, and what should the project do next time. Groups debate and make process changes and introduce them for the next release of the product. Since the teams tend to stay together for several years, the postmortem analysis is very effective and helps with process learning.

Tools

Development environments consist of personal computers and a few work stations in offices connected to the local area network (LAN) server. Developers pick the hardware systems they wish to use; many have multiple systems in their offices since developers generally work simultaneously on both Windows and Macintosh versions of their products. 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 manages at least 600 servers in one building along with a worldwide network.

A good suite of specialized tools is available for automated testing. Microsoft groups have used these tools for several years and they have now progressed to event recorders and playback tools that make it possible to analyze all of the keystrokes and pointer movements of a user trying to accomplish a particular task. Automated test tools also run in multiple environments.

Developers and testers will run automated tests "hundreds of times" during development. Testers continually add to this set of tests. Developers use "quick tests" before all check-ins and after all daily or weekly 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

Microsoft offers orientation classes that describe their development cycle but the company does not have detailed formal process education classes. Most education is done within the team. Major product units have 2 to 4-page documents that describe their products. Testing groups also have a series of brief documents that serve as checklists of job responsibilities. In addition, managers assign mentors to each new hire on their team; these help introduce the new hire to processes used in the company.

Managers generally expect technical personnel to undergo about two weeks of training each year. People use a combination of in-house courses, university seminars, and corporate or conference seminars to meet the objective. In-house training is available for corporate training on management skills, and product group training is available for technical skills.

4: DISCUSSION

Table 1 presents key characteristics of the classic "waterfall" approach to software development as compared to Microsoft's "synch-and-stabilize" process. The latter relies primarily on daily builds for frequent synchronizations and incremental milestones for periodic stabilizations of the products under development. We have based this stylized description of the waterfall process on our 1992-1993 analysis of IBM, Fujitsu, and Hewlett-Packard divisions building systems and applications software for mainframes, minicomputers, and technical workstations.[16] Some groups in these and other companies now build software using processes that have significantly evolved from the base waterfall model and have incorporated steps such as more frequent builds and incremental development, which are similar to steps Microsoft utilizes. We also believe, however, that Microsoft stands out for how it has institutionalized this style of software product development. More importantly, we believe that Microsoft's process is similar but more structured and repeatable than approaches used at other PC software developers in the United States such as Lotus, Borland, Novell-WordPerfect, and IBM's OS/2 group.

At a very high level of abstraction, the development life cycles appear similar across firms using a waterfall process versus a synch-and-stabilize process. Each utilizes common phases of requirements, design, coding, testing, delivery, and maintenance. Each spends significant effort on process support activities such as release management, change management, metrics, and process improvement, although, in some respects, the development process seems less formalized at Microsoft. When we look in more detail, however, we see that Microsoft's development process has some important differences from the conventional waterfall model.

In comparing a small sample of companies, an important difference is from where the products have evolved and how quickly they continue to evolve. IBM, Fujitsu, and Hewlett-Packard, for example, all develop relatively stable operating systems for mainframes, minicomputers, and technical workstations that have multiple user groups on a single hardware platform. These development organizations have been in place producing this type of software for many years. In contrast, Microsoft and other PC software companies develop products for a dynamic and relatively new set of markets. These companies themselves are relatively new and need to accommodate markets and hardware platforms that are rapidly evolving. As a result, one

can argue that the PC software market requires more flexibility and creativity than the mainframe, minicomputer, or technical workstation software markets, even though some of these markets are merging to some degree, and PC software producers are introducing many of the same techniques and controls as their predecessors.

The basic problem that Microsoft has tried to address is that most of its projects now consist of teams with twenty to two hundred developers, and the larger teams are usually building components that are interdependent and difficult to define accurately in the early stages of development. In this situation, the teams must find a way to proceed that structures and coordinates what the individual members do while allowing them enough flexibility to define and change the product's details in stages as the project progresses.

As we have discussed, Microsoft's solution is to have development teams begin by outlining the product in sufficient depth to set priorities in terms of product features that they want to create but without trying to decide all the details of each feature, as in a more conventional waterfall process. In other words, they do not lock the project into a set of features and details of features that they cannot later revise as they learn more about what should be in the product. The project managers then divide the product and the project into parts (features and small feature teams), and divide the project schedule into three or four milestone junctures (sub-projects) that represent completion points for major portions of the product. All the teams go through a cycle of development, testing, and fixing problems in each milestone phase. Moreover, throughout the project, the team members synchronize their work by building the product, and by finding and fixing errors, on a daily and weekly basis. When most serious problems are fixed, they stabilize (agree not to change) the most important pieces of the product and proceed to the next milestone and, eventually, to the ship date.

In many ways, the synch-and-stabilize approach resembles prototype-driven product development processes that use a series of incremental design, build, and test cycles.[17] It also has elements of concurrent engineering to the extent that Microsoft groups refine specifications, start building the product (coding), and do testing, debugging, and integration of components *in parallel*, rather than in distinct sequential phases. Many if not most software producers end up following this type of process as they run into problems during the first attempt at integration, but they often proceed in an ad hoc manner. What Microsoft has done is introduced a concurrent, incremental, and iterative but structured approach to product development that offers several benefits to the development organization:

- It *breaks down large products into manageable chunks* (a few product features that small feature teams can create in a few months).
- It *enables projects to proceed systematically even when they cannot determine a complete and stable product design* at the project's beginning.
- It *allows large teams to work like small teams* by dividing work into pieces, proceeding in parallel but synchronizing continuously, stabilizing in increments, and continuously finding and fixing problems.
- It *facilitates competition on customer input, product features, and short development times* by providing a mechanism to incorporate customer inputs, set priorities, complete the most important parts first, and change or cut less important features.

There are also some caveats with the Microsoft process that firms need to be aware of. First, when launching a new product, firms need to design the product architecture so that it can accommodate adding or subtracting features in future versions of the product. Periodically, firms may want to go back and redo the product architecture, and this will require extensive planning in the beginning of the development process. Thus, the synch-and-stabilize process is primarily well-suited for "N+1" versions of a product, like the second or third versions of the Excel spreadsheet or Windows NT operating system, rather than the very first versions or product versions that are almost completely new.

For example, Windows NT (which was about 9 months late on a four-year development cycle) and Windows 95 (which was about 18 months late on a three-year cycle), adopted this process only in the last year or two of development, after the projects had completed a set of features and functions on which to create daily builds. The Microsoft Network also went through more than a year of experimental planning, design, and development work before moving to frequent builds and milestone integrations.

Operating systems and network communications software also tend to have many interrelated functions that designers must analyze and plan before they start writing functional specifications and code, because these features and functions cannot be easily changed or cut late in a project. In addition, systems and communications software have to test nearly infinite numbers of user scenarios involving thousands of combinations of hardware and applications software; teams often require 6 months to a year or more to test these products in the field to make sure that users will not run into a major bug (such as the one that plagued Intel's Pentium microprocessor). Windows 95, for example, was available in beta copies since June 1994, but Microsoft delayed the official commercial release until August 1995 in order to take extra time to test the product with 400,000 users. This time was necessary because Microsoft had trouble perfecting two new and highly complex architectural components: plug-and-play, which is supposed to detect and set up hardware peripherals automatically, and multitasking within certain memory limit targets, which allows the computer to run several applications programs simultaneously.

Second, projects can proceed with an evolving specification, but they can run into problems if they are dependent on components built by other projects, or build components for other projects to use. If these interdependencies exist, then projects must have coordinated schedules, and they need to establish standards for designing components and stable interfaces for integrating components, and add these to the specification process at the beginning of the project, with minimal changes thereafter. Microsoft managers have been encouraging projects to design components for other groups to share, and this has led to some delays when all projects were not tightly managed. For example, a recent version of Office was late because of delays in building OLE, which all the Office products use to share components, as well as Visual Basic, which serves as a macro language in the Excel product. Microsoft also had to coordinate changes and delays in Windows 95 as it built the Microsoft Network software.

Third, firms using a Microsoft-type synch-and-stabilize process need to commit extensive resources to testing the product as they build it, which also makes the process especially well-suited to N+1 versions where there is some stable core of features with which testers can test new

features. On the other hand, the project should be able to reduce the total amount of resources needed for rework as well as for system testing and integration at the end of a project, because developers and testers have been designing, testing, fixing, and integrating features throughout the project.

The synch-and-stabilize also supports competition based on product features and incremental innovations, rather than product invention, which depends more on how well a company manages its research organization. Microsoft began to invest heavily in research only since around 1991; hence, it has not invented many new product technologies, and has trailed competitors in introducing innovative new products to market. For example, Intuit (which Microsoft wanted to acquire but did not after opposition from the U.S. Department of Justice on antitrust grounds) has been the leader in personal finance software with Quicken, Lotus has been the leader in office groupware with Notes. Novell has been the leader in corporate networking operating systems with NetWare. Several companies, including CompuServe and America Online, have led in the introduction of on-line information highway networks.

In all these areas, however, Microsoft has designed competing products in its research and development organization, and then used the process we have described in this paper to evolve these products incrementally by introducing new versions every year or two. Since this process makes it possible to synchronize the efforts of a large number of individuals and teams, we think it is well suited to building the complex software systems of the future. Microsoft groups need to understand, however, that complex new operating systems and network communications systems require more advanced planning and architectural design work than the company's desktop applications products..

Finally, we should cite one additional area of concern for Microsoft as a company. It has now entered nearly every PC software market, both for home consumers and for corporate customers. It is unlikely that Bill Gates and other Microsoft managers, as well as Microsoft's development teams, can pay the same level of attention to 200 products as they could to two or three products, as in the early days of the company. Even a highly strategic project such as Windows 95 does not seem to have been well-managed in its early stages; announced shipping dates have been extremely optimistic, and the development team has significantly underestimated the complexity of the tasks required to deliver this new product. Nonetheless, Gates has cultivated a talented "brain trust" to help him manage the company. Microsoft has also recently hired hundreds of experienced managers and researchers from universities as well as other companies, and acquired a dozen or more firms with a variety of new skills, such as in multimedia software and communications technologies. These new people, as well as Microsoft's extensive financial resources and existing pool of technical experts, should help Gates and Microsoft compete effectively in desktop software as well as in the world of the information highway.

Figure 1: CONVENTIONAL WATERFALL DEVELOPMENT PROCESS

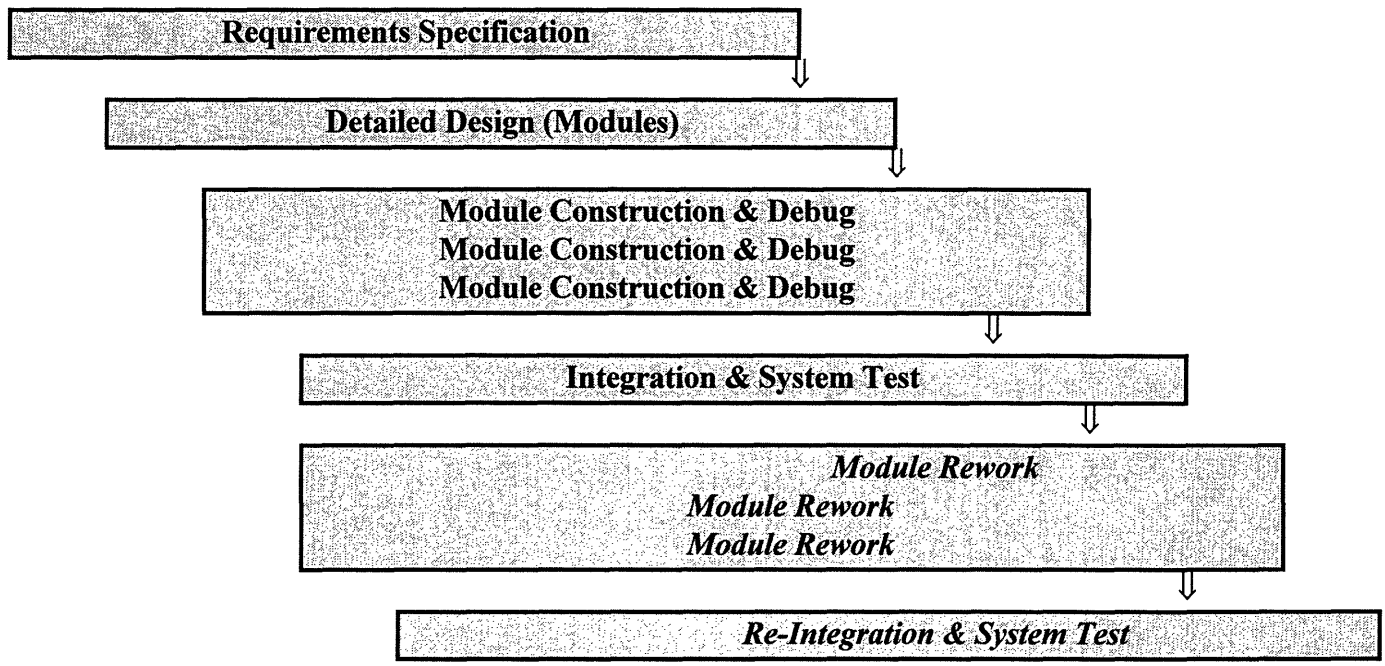

Figure 2: MICROSOFT'S "SYNCH-&-STABILIZE" DEVELOPMENT PROCESS

Time: Usually 12- or 24-month Cycles

Planning Phase:

VISION STATEMENT
E.g. 15 Features and Prioritization
Done by Product (& Program) Management

⇓

OUTLINE & WORKING SPECIFICATION
Done by Program Managers with Developers.
Define Feature Functionality, Architectural Issues & Component Interdependencies

DEVELOPMENT SCHEDULE & FEATURE TEAM FORMATION
A big feature team will have 1 Program Manager, 5 Developers, 5 Testers

⇓

Development Phase:

FEATURE DEVELOPMENT in 3 or 4 MILESTONES
Program Managers: Evolve the Spec
Developers: Design, Code, Debug
Testers: Test, Paired with Developers

⇓

Stabilization Phase:

Feature Complete
CODE COMPLETE
ALPHA & BETA TEST, FINAL STABILIZATION & SHIP
Program Managers: Monitor OEMs, ISVs, Customer Feedback
Developers: Final Debug, Code Stabilization
Testers: Recreate and Isolate Errors

Figure 3: MICROSOFT'S "SYNCH-&-STABILIZE" MILESTONE BREAKDOWNS

Time: Usually 2 to 4 months per Milestone

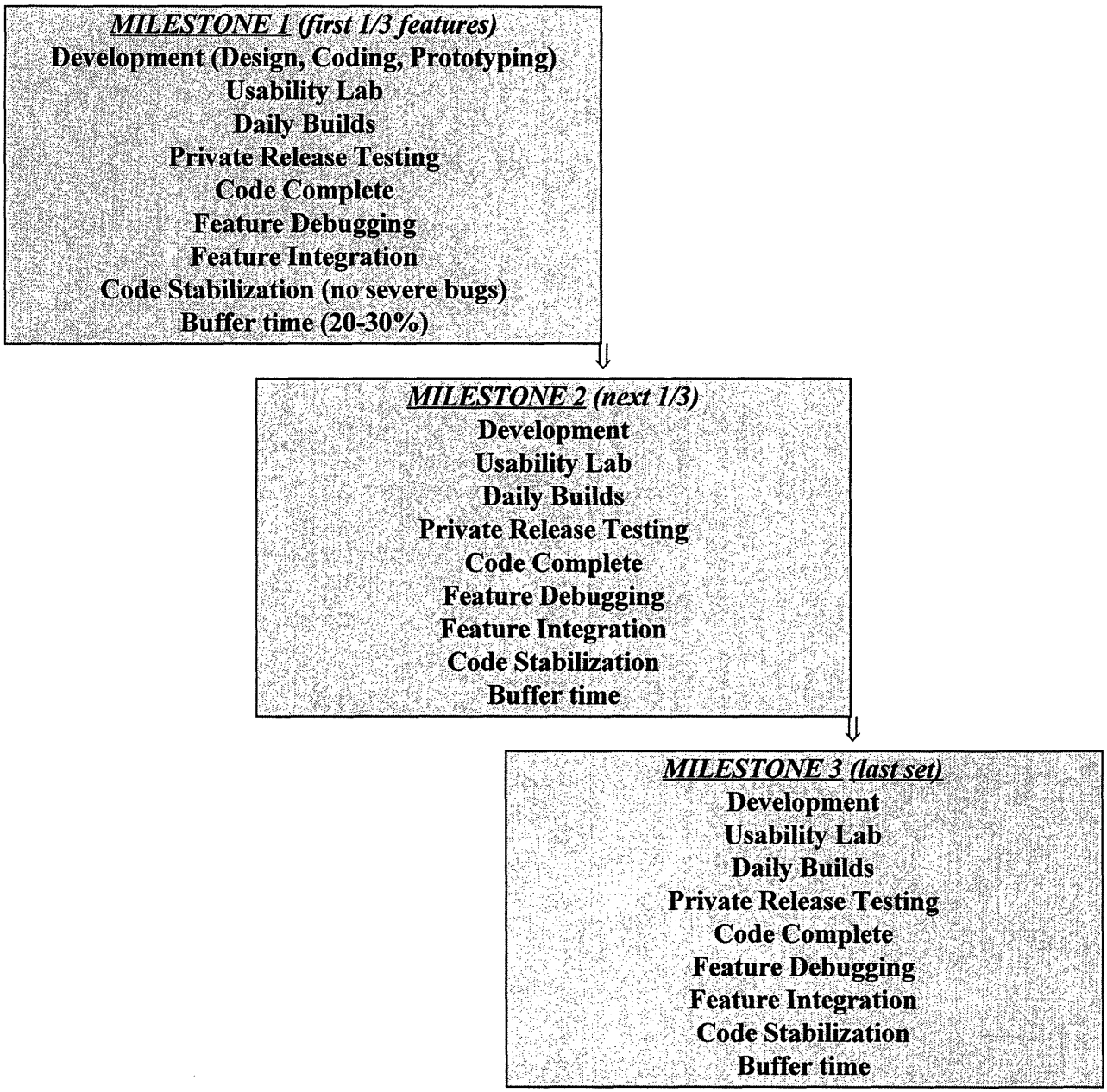

Table 1: Comparison of "Waterfall" and Microsoft Development Processes

Activity	*Classic "Waterfall" Process*	*Microsoft "Synch-and-Stabilize"*
Release Structure & 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.
Planning Process	All organizations utilize formal processes for well-defined development phases that proceed more or less sequentially, with a large integration phase at the end.	Development proceeds in broader phases. Projects have 3 or 4 milestones or sub-projects, each with a full set of development, testing, and stabilization phases.
Require-ments	Requirements and development phases are driven by product management groups with executives having final approval of contents and schedules. Organizations strive to meet the needs of diverse customer bases while expanding to new markets. Projects try to write as complete a specification as possible before proceeding to detailed design and coding.	Requirements and schedules are determined by a very small group that maintains control over the product as it matures. Projects do not try to write complete specifications up front, because they know these will change. Instead, they allow requirements to evolve through prototypes and continual input from developers, program managers, and users on what should be included in products. There is also has a formal process to fix the top customer complaint areas in each release.
Design & Coding	Design and coding is manually done. 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 (though OO usage is rising). They are adopting design and code reviews due to positive early results.
Testing	In-house test groups are generally independent from the developers. The in-house groups are strong, with ratios ranging from 5-10 developers per tester. Beta and other customer tests are used by each for technical and marketing reasons. Few	In-house groups work more closely with developers than in the classic 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. Personnel use their product's latest

	companies use software being developed in day-to-day operations.	code levels on a continuous basis to get additional testing of the code.
Process Usage & Compliance	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 to predict schedules and quality. A strong use of metrics aids the checking of compliance.	There is more independence surrounding process choices. Fewer formal compliance measures are in place. A standard process is being introduced due to the need to predict schedules and error control for the growing systems.
Release Management	This is a significant activity due to large size of product and number of people usually involved. Some companies 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.	Lead Program Managers are 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 and testing groups, and all outside groups (support, manufacturing, subcontractors, etc.).
Change Management	This is done throughout the development phases with increased formality during coding and testing. Some companies 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	Theses 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	This is extensively pursued. Causal Analysis and Defect Prevention processes are used to remove sources of error injection.	This is used by the most mature organizations in the company. Post-mortems are now a common and high-profile activity. Group continuity appears to be necessary for this to be effective. Projects are working to get more sharing and learning across groups.

Tools	Most companies have a well-established tool set to support change management, coding, and product builds. Most make investments in automated tools for testing.	Work station-based networks are standard. Source code management tools are utilized with a variety of languages and linkers employed. The company attempts to use its product while it is going through development. Object-oriented programming is gradually being introduced for parts of 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 process are tightly linked. Groups are finding the need to add structure to their development processes and have been continually doing that.

End Notes

[1] This article is based on a longer study that included an additional IBM (now Loral) site, the Federal Systems Company in Houston, Texas, which makes space shuttle software; Hewlett-Packard, which makes work-station operating systems; Fujitsu's mainframe operating systems development site at Numazu, Japan; and Lotus Development, which makes a variety of PC applications. For the full set of case studies, see Stanley A. Smith and Michael A. Cusumano, "Beyond the Software Factory: A Comparison of 'Classic' and 'PC' Software Developers," Cambridge, MA, Sloan WP#3607-93/BPS, September 1993. For a full treatment of the Microsoft story, see Michael A. Cusumano and Richard W. Selby, Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People (New York, The Free Press/Simon & Schuster, 1995).

[2] The first description of the waterfall model was by Winston W. Royce, "Managing the Development of Large Software Systems," Proceedings of IEEE Wescon, August 1970.

[3] See Victor R. Basili and Albert J. Turner, "Iterative Enhancement: A Practical Technique for Software Development," IEEE Transactions on Software Engineering, Vol. SE-1, No. 4 (December 1975); and Barry W. Boehm, "A Spiral Model of Software Development and Enhancement," IEEE Computer, May 1988. Also, B. Blum, "The Life Cycle -- A Debate Over Alternative Models," ACM Software Engineering Notes, Vol. 7, No. 4, October 1982; and B. Blum, "Three Paradigms for Developing Information Systems," Seventh IEEE International Conference on Software Engineering Proceedings, March 1984, p. 534. Discussions of iterative approaches to product development can also be found in marketing and management of technological innovation literature, although mainly in the sense of probing customer needs in the initial conceptualization phase of product design. See, for example, Glen L. Urban and John R. Hauser, Design and Marketing of New Products (Englewood Cliffs, NJ, Prentice Hall, 1980), or Eric von Hippel, The Sources of Innovation (New York, Oxford University Press, 1987).

[4] For conventional approaches to reuse, see, for example, the discussions of Toshiba and NEC in Michael A. Cusumano, Japan's Software Factories: A Challenge to U.S. Management (New York, Oxford University Press, 1991). For an alternative approach that focused on building a data architecture first and then objects, see "Brooklyn Union Gas: OOPS on Big Iron" (Boston, Harvard Business School Case# 9-192-144, 1992).

[5] See, for example, U.S. Department of Defense, DOD-STD-2167-A (Defense System Software Development Standard), February 1988.

[6] P. Sage and James D. Palmer, Software Systems Engineering (New York: John Wiley & Sons, 1990).

[7] Sage and Palmer.

[8] Pankaj Jalote, An Integrated Approach to Software Engineering (New York: Springer-Verlag, 1991).

[9] Jalote.

[10] Richard A. Sulack, "Advanced Software Engineering Management Core Competencies," Presentation at Spring 1993 COMMON Meeting.

[11] See Peter Naur and Brian Randell, eds., Software Engineering: Report on a Conference Sponsored by the NATO Science Committee (Brussels, NATO Scientific Affairs Division, January 1969); Barry W. Boehm, "Software Engineering," IEEE Transactions on Computers C-25, 12, December 1976; Richard Thayer, "Modeling a Software

Engineering Project Management System," Ph.D. dissertation, University of California at Santa Barbara, 1979; C.V. Ramamoorthy et al., "Software Engineering: Problems and Perspectives," Computer, October 1984.

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

[13] This section is based primarily on information obtained through an interview on March 15, 1993 with David Moore, Director of Development, Testing, and Quality Assurance for Microsoft. Some information was obtained through materials received and interviews done during April and August 1993, and September 1994.

[14] Gill.

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

[16] See Smith and Cusumano 1993.

[17] See, for example, Steven C. Wheelwright and Kim B. Clark, Revolutionizing Product Development (New York, Free Press, 1992).