CONTROLLING SOFTWARE QUALITY IS A MAJOR CHALLENGE IN MODERN SOFTWARE DEVELOPMENT PROJECTS. THIS ARTICLE LOOKS AT SOFTWARE DEVELOPMENT PRACTICES AT THE TWO LARGEST PRODUCERS OF SOFTWARE, IBM AND MICROSOFT, AND COMPARES THEM AGAINST THE SOFTWARE ENGINEERING INSTITUTE'S (SEI) CAPABILITY MATURITY MODEL (CMM). IT CONCLUDES THAT RIGOROUS QUALITY CONTROLS AND THE CMM'S BASIC PREMISE FOR REACHING HIGHER MATURITY ARE STILL THE BEST WAYS TO PRODUCE QUALITY SOFTWARE.

THE PURPOSE OF THIS ARTICLE IS TO STUDY SOFTWARE QUALITY MANAGEMENT STRATEGIES AND PROCESSES THAT ARE USED IN DEVELOPMENT PROJECTS AT IBM AND MICROSOFT, AND TO ASSESS THEIR LEVELS OF CAPABILITY AND MATURITY. THESE COMPANIES WERE SELECTED FOR THIS STUDY BECAUSE OF THEIR LEADERSHIP IN THE FIELD AND THEIR GREAT INFLUENCE ON THE SOFTWARE QUALITY CONTROL PROCESSES OF THOUSANDS OF THEIR SOFTWARE PARTNERS. THE ARTICLE PROVIDES SOME LESSONS LEARNED AND CRITICAL SUCCESS FACTORS THAT MAY PROVE BENEFICIAL TO OTHER COMPANIES.

INTRODUCTION

DESPITE CONTINUOUS EFFORTS TO IMPROVE THE SOFTWARE DEVELOPMENT PROCESS, MANAGING SOFTWARE QUALITY REMAIN DIFFICULT IN TODAY'S SOFTWARE DEVELOPMENT ENVIRONMENT. RECENT REVIEWS OF VARIOUS INNOVATIONS IN SOFTWARE ENGINEERING TECHNOLOGY AND PROCESS IN THE PAST DECADES BY FREDERICK BROOKS (1995) AND ROBERT GLASS (1999) FOUND THAT THESE INNOVATIONS PROVIDE ONLY MODEST GAINS. THE BEST KNOWN WORK CONCERNED WITH PROCESS IMPROVEMENT TODAY IS THE SOFTWARE ENGINEERING INSTITUTE'S CAPABILITY MATURITY MODEL (CMM).

THE BASIC PREMISE OF THIS MODEL IS THAT HIGHER MATURITY PROCESS WOULD LEAD TO INCREASED PRODUCTIVITY AND REDUCED CYCLE TIME AND DEFECTS. THIS ARTICLE ATTEMPTS TO MEASURE THE CAPABILITY AND MATURITY IN QUALITY PROCESSES OF TWO LEADING COMPANIES USING CMM AND TO DISCUSS LESSONS LEARNED AND CRITICAL SUCCESS FACTORS.

MEASURING CAPABILITY AND MATURITY

IN RECENT YEARS, THERE HAS BEEN SIGNIFICANT EMPHASIS ON PROCESS MATURITY AND TOTAL QUALITY MANAGEMENT (TQM) IN SOFTWARE DEVELOPMENT PROCESSES. THE SOFTWARE ENGINEERING INSTITUTE (SEI) HAS DEVELOPED A CAPABILITY MATURITY MODEL (CMM) PREDICATED ON A SET OF SOFTWARE ENGINEERING CAPABILITIES THAT FOCUS ON PROCESS QUALITY MANAGEMENT (15). THE CMM HAS FIVE LEVELS OF MATURITY WITH ASSOCIATED KEY PROCESS AREAS (KPA) AS PRESENTED IN EXHIBIT 1.

Since the completion of the CMM in 1993, many studies have been made of the impact of the model's key processes on development productivity and quality. A study by Larry Putnam in 1994 of organizations that moved from level 1 to level 3 of the model reported schedule time reduced by a factor of 1.7, peak staff needs reduced by a factor of 3.2, and efforts reduced by a factor of 5.7. Another study of organizations that moved from level 1 to level 3 of the model found productivity gains of 6 to 9 percent per year, and a defect detection gain of 6 to 25 percent per year (8). A third study focusing on organizations that moved to level 3 from a lower level found

- A 5 to 1 return on investment at Hughes Aircraft
- A 7.7 to 1 return on investment, a 75 percent decrease in rework costs, a decrease in cost of quality from 41 percent to 21 percent of project costs, a 190 percent increase in productivity, and a decrease in problem report rates per thousand lines of code (KLOC) from 17.2 to 4.0 at Raytheon Corp.
### EXHIBIT 1: SEI’s Capability Maturity Model

<table>
<thead>
<tr>
<th>Maturity levels</th>
<th>Focus</th>
<th>Key Process Areas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1: Initial</td>
<td>People</td>
<td>Competent people</td>
</tr>
<tr>
<td>Level 2: Repeatable</td>
<td>Project Management</td>
<td>Software configuration management, Software quality assurance, Subcontract management, Project tracking and oversight, Project planning, Requirements management</td>
</tr>
<tr>
<td></td>
<td>Processes</td>
<td></td>
</tr>
<tr>
<td>Level 3: Defined</td>
<td>Engineering processes</td>
<td>Peer reviews, Intergroup coordination, Software product engineering, Integrated software management, Training program, Organization process definition, Organization process focus</td>
</tr>
<tr>
<td></td>
<td>and organizational</td>
<td></td>
</tr>
<tr>
<td></td>
<td>support</td>
<td></td>
</tr>
<tr>
<td>Level 4: Quantitatively</td>
<td>Product and</td>
<td>Software quality management, Quantitative process management</td>
</tr>
<tr>
<td>Managed</td>
<td>process quality</td>
<td></td>
</tr>
<tr>
<td>Level 5: Optimizing</td>
<td>Continuous process</td>
<td>Defect prevention, Technology change management, Process change management</td>
</tr>
<tr>
<td></td>
<td>improvement</td>
<td></td>
</tr>
</tbody>
</table>

- A 7.5 return on investment, a 90 percent reduction in defect rate, and a 26 percent reduction in average cost of maintenance activity at the Oklahoma Air Logistic Center (3)
- For their part, in 1993, K.H. Moller and D.J. Paulish argued that The predominant measure of software quality is the counting of known faults at various points in the software development life cycle (14).

**MEASURING SOFTWARE QUALITY**

The major task in improving software quality is to minimize the number of faults or defects that exist during and even after development. Because of their negative impact on the overall quality of the software, these faults or defects must be prevented or eliminated.

Numerous efforts to define and measure software quality have been made in the past decade. The generally accepted definition of software reliability is the probability of failure-free operation of a software component or system in a specified environment for a specified time.

Other terms that are often used with similar meaning as fault include defect, bug, and error. Common measurements of software quality are:

- Mean time between failures
- Mean time to repair a defect (MTTR)
- Defect rate by hour, day, week, or month
- Defect rate per unit of software size, such as function point or thousand lines of code
- Size of the defect backlog
- Number of clean lines of code or system components that passed quality assurance on the first attempt
- Cumulative defects per version
- Timeliness of response to defect or time to fix defect
- Customer level of satisfaction with software and quality of work done to fix defect
• Rate of defects per unit of software size discovered in the first year after delivery (9, 16)

In practice, quality is often measured by a combination of the foregoing common measurements. This is to avoid focusing on a limited aspect of quality perception during and after development (9). Industry standards in quality control include those in the Malcolm Baldrige National Quality Award and the ISO 9000 series. The Baldrige award requires that candidate organizations use software quality metrics and sophisticated quality measurement systems. ISO 9000 series quality standard requirements for software development include the definition of management responsibility in quality control, document control, implementation of process control by inspection and testing and verification of test results, performing corrective actions when appropriate, internal quality audits, personnel training, and after-delivery servicing and statistical analysis.

Defect Removal Model and Estimation of Latent Errors at Delivery Software defects can be modeled to predict the trends and patterns of software reliability during and after development. The two common ways that software reliability can be quantified and modeled during development and testing activities are (1) by error detection and removal at each development stage and (2) by determining the rate of defects discovered per unit of time. These reliability models help to estimate and plan for quality assurance efforts, as well as to measure the effectiveness of the software quality control tools and methodology.

However, because building quantitative quality models requires historical estimation of defect entry and defect removal rate, the models are not commonly used by many software makers (12).

SOFTWARE QUALITY PROCESSES AND STRATEGIES AT MICROSOFT CORP.

Michael Cusumano and Richard Selby studied the project management and quality processes at Microsoft, Inc., in the development of Windows 95, Windows NT, and Microsoft Office products. Since the 1980s, Microsoft had recognized the importance of quality processes in its development efforts. Most of the projects at Microsoft are very large.

For example, Windows 95 contains more than 11 million lines of code and required a development team of more than 200 programmers. Microsoft's Development Framework Rather than following the common software engineering methodology used in the industry, Microsoft tried to scale up a loosely structured small-team (hacker) style of product development (11, 2). The Microsoft solutions framework (MSF) is a flexible, interrelated series of models that guide the company through assembling the necessary project resources to ensure that technology infrastructure and solutions meet business objectives. Great emphasis is placed on the relationship between business objectives and technology implementations in this framework, which allows Microsoft to cope better with change in technology and environment.

The Team Model

Microsoft's team model is defined as a team of peers working in interdependent and cooperating roles. Each team member has a well-defined role on the project and is focused on a specific mission. This approach encourages ownership and ultimately results in a better product. The leaders of each team are responsible for management, guidance, and coordination, while the team members focus on carrying out their missions. In order to effectively empower quality at the team member level, Microsoft gives all members of the team ownership of the quality of the
component before a product or solution is put into use.

The team model defines roles and responsibilities but does not define the management structure of the team. The team may report to a business unit manager, or it may be formed for a particular project from several organizations. The organization chart determines who is in charge. The team model defines how the work is divided and who is responsible for seeing that the work gets done. As a consequence, it gives each team member a stake in the success of the product or service and improves accountability for all roles.

The Quality Process Model

Microsoft's process model is a milestone-based model that provides guidelines on planning and controlling results-oriented projects based on their scope, the resources available, and the schedule. According to the official Microsoft Solutions Framework process model, the software development cycle consists of four phases: envisioning, planning, developing, and stabilization (however, Cusumano and Selby combined the envisioning and planning phases into the planning phase (2)). Each phase culminates in a major milestone.

The Envisioning Phase. In this phase, product and program management uses extensive customer input to identify and prioritize product requirements. Once a new product (or in the case of infrastructure deployment, a new service) gains interest and approval, a project team is assembled to define the product. While a vision statement articulates the ultimate goals for the product or service and provides clear direction, the project scope defines the limits of the product or service at a particular version. Further improvement of the software may be added later in future versions.

The Planning Phase. The project plan contains the functional specification: the combined plans of each team member and a schedule. The functional specification provides the project team with enough detail to identify resource and quality requirements within schedule and budget. Based on the vision statement, program management and development groups define feature functionality, architecture, and component interdependencies. Then project management coordinates the schedule and arranges development teams; each contains approximately one manager, three to eight developers, and three to eight testers (2).

The Developing Phase. In this phase, program managers coordinate all design, code, and testing tasks in the project. An approved functional specification and associated project plan provide the baseline for focused development to begin. The development team sets a number of interim delivery milestones, each of which involves a full test/debug/fix cycle. The customer's involvement includes an assessment of the product's functionality and verification of the support plan availability. Customers also verify that deferred functionality is documented for the next release.

The Stabilization Phase. In this phase, comprehensive internal and external tests are performed before delivery. To reduce cycle time, testing activities are set to be performed concurrently with code development. During the stabilizing phase, bug finding and fixing become the primary focus. At this milestone, the product is formally turned over to the operations and support groups. Typically, the project team members either begin work on the next release or are reassigned to other development projects.

Microsoft's Quality Strategies and Processes

Doing everything in parallel with frequent synchronization and periodic stabilization highlights Microsoft's main strategy in software development. Because there are many teams working in parallel who are given some freedom to introduce new concepts and technology, they must synchronize their innovation frequently, so that product components all work together before the integration test stage. Teams and individuals periodically synchronize and stabilize the product in increments as the project proceeds, rather than at the integration testing stage. This technique is also known as "synch-and-stabilize," "milestone," "daily build," or "nightly build."

Daily Build Process
Source codes in development are stored in the project repository, where programmers check out a copy to work on their own as a private copy. If the private copy works correctly, the master source code and the private copy are synchronized.

The programmers use the "diff" tool to find the differences in the master source code since the time they checked out, to make sure that they use the most recent centralized version of the master source code before the deadline time each day -- say, 3:00 P.M. Each project has its own daily deadline time. The source library manager (SLM) tool then updates the private copies automatically to incorporate changes from all developers made to the source code. This update process is called merging. Inconsistencies or conflicts in the merged code detected by the SLM tool must be resolved by the programmers.

The build and test of merged code are performed daily. After the Unit test and the "quick test" (a highly automated test which performs to make sure the new features do not conflict with other features), the private copy is checked in to the master source code in the repository. Here, the private copy is integrated and tested with the entire system. In order to prevent overriding changes that were made by other developers on the same source code during the day, synchronization with changes performed by other developers is performed again at check-in time. A developer who is designated as the "build-master" for the day then generates the complete build of the product after the check-in deadline each day. After the build, the build-master executes a series of automated tests.

The daily build and synchronization help developers to maintain high quality of the new code developed by many interrelated teams. The daily build process also provides a code control mechanism that requires all project members to work as an integrated team in order to produce a daily version of the product that works.

One of the major problems that teams often encounter is in "breaking the build" -- an architectural error caused by incompatibility among various functions when the build progresses overnight. If no one discovers the breakdown until the next morning, the team loses valuable time in waiting for the fixes. In order to encourage developers to carefully test and synchronize their private copies, whoever breaks the build will become the next buildmaster and must wear a funny hat during his or her build-master duty.

Other major quality strategies include:

- Growing rather than designing and building software
- Changing or replacing approximately 50 percent of the code after each release
- Justifying changes, whereby features must be at least twice as good
- Building multiple versions simultaneously
- Single-site development
- Continuously testing the products as they are built

Growing Rather Than Designing and Building. In addition to parallel development, Microsoft uses the iterative development process to grow existing software, rather than building it from scratch. This innovation in the software engineering process has long been promoted by Frederick Brooks, Jr; however, as a software maker, Microsoft has pioneered growing a great portion of its software from the previous version. For example, much of the file system driver software of Windows NT 3.0 was grown from the old driver software.

Changing or Replacing Approximately 50 Percent of Code after Each Release. This is the "short half-life of code" strategy used by Microsoft. It is estimated that Microsoft teams change or replace 50 percent of code in their products every 18 months or so. Because Microsoft releases a new version of a particular product every 12 to 24 months, this suggests that 50 percent of code in a product changes or is upgraded between releases. In order to comply with the short half-life strategy, at least 50 percent of the code in the new version must be new or upgraded. As a result of rapid changes in technology and the highly competitive software market, frequent change in source code is necessary for Microsoft to catch up and prevent competitors from making inroads into Microsoft products.
Justifying Change by Features That Are at Least Twice as Good. Learned from the OS/2 development project in which the benefits of change were less than the damage inflicted to the integrity of the software architecture, Microsoft implemented this strategy to tightly control changes. It does not allow changes that bring only a little improvement but require lots of effort and cause inconsistencies in system software.

Building Multiple Versions Simultaneously. Because of the global diversified market, it is critical for Microsoft to have versions of each product simultaneously ready all over the world for different hardware platforms. Microsoft Office 2000, for example, was built in numerous versions to accommodate many different languages in different operating platforms (Windows and Macintosh), and in both beta and production versions. Internally, Microsoft also had numerous in-debug versions that were used internally. Only versions that are stable enough are released to the market.

Single Site Development. The biblical story of the Tower of Babel is one of the earliest major construction projects that failed. The main reason of this fiasco was poor communications. In the past, Microsoft had a bad experience when it codeveloped OS/2 software with IBM in a multisite development environment. It attributed the failure of the OS/2 product to poor communications. To help developers adopt a common language, tools, processes, and conventions, and use face-to-face communications, Microsoft develops most of its products at the company headquarters in Seattle, Washington. The only exception to this strategy occurs when Microsoft acquires another software maker outside.

Continuously Testing the Products as the Software Grows. There are several tests used at Microsoft for developers during development: structured test scripts (scenario tests), gorilla testing to break the product, bug bashes for everyone to find bugs, and usability testing. However, as the schedule pressure increases, there is less thorough testing. Code that is being debugged is also tested but is not included in the product code. As the project approaches the shipping date, developers do weekly releases of the production version. Because there are several versions and releases of the software on hand, if the latest version or release is delayed, the next best version is used for shipping. Once the product is available, it is put to use by Microsoft employees before shipping to customers.

Results With the new quality processes and strategies in place, Microsoft has made substantial progress in quality control, especially compared with the 1980s when Microsoft products were defect-ridden. Today, Microsoft claims its MSF model produces better decisions, less rework, higher morale, and a higher quality product. It also claims that Microsoft subcontractors and software partners have used its model with good results. Strategies such as frequent synch-and-stabilize, multiple versions, growing rather than building, and short-half-life, have allowed Microsoft to continue innovations and exploit first-mover advantage in the market, while preventing competitors from making inroads into Microsoft products. However, the lack of ability to predict latent errors through a quantitative defect model caused Microsoft to ship "buggy" products to the market, as exemplified by DOS 6.0, MS Word 6.0, and Windows 98.

QUALITY PROCESSES AND STRATEGIES AT IBM

There have been several studies on IBM's quality management during the decade from 1986 to 1995 (see 16, 18, 10, and 19). Phan et al. studied the quality control processes at IBM on OS/400 development and its subsequent efforts to earn the Malcolm Baldrige award in 1990. Kaplan et al. studied IBM quality strategies at several IBM sites during the period 1988-1993, and Sawyer and Guinan studied the relationship of software development processes and team performance in 1997.

IBM Rochester

The OS/400 R.1 development project was a very large-scale development project at IBM. During the 26-month development period, more than 50 million lines of high-level code, including scaffolding, were developed in Rochester, Minnesota. The code was then refined to more than 3.6 million lines of
programming code, from which 1.2 million lines were reused.

Throughout the OS/400 development life cycle, written code had to go through rigorous inspections, which included three reviews and four tests. Many of these reviews and tests were performed jointly with customers and end users. User involvement and a good feedback process helped in detecting and fixing problems at early stages.

- **Rigorous reviews.** The three rigorous reviews used in the development methodology for this project included: a high-level design review, a low-level design review, and a code review. The high-level design review examined the software's functions, interfaces, control blocks, and data structures. The low-level design review focused on the design's logic flow, pseudocode, and error conditions. The code reviews involved a program walk-through by experienced programmers. During the design and code reviews, problems and defects were logged. All defects discovered had to be resolved before proceeding to the next step.

- **Customer involvement and early feedback.** In addition to the internal project reviews, customer involvement and feedback to the development process also helped to ensure that requirements and specifications were met. IBM Rochester business executives and project team members set up regular meetings with users to discuss what users disliked about the system. Customers were invited to review and validate project requirements and test results to make sure that the delivered products would be trouble-free at customer information systems sites.

- **Software tests.** Experience, tools, and processes carried over from many prior systems development projects helped developers build test plans and test cases early; even before coding started. The software test stages in this project included: unit test, component (integration) test, system test, and beta test. In all test phases, defects were recorded into the database for online query and monitoring.

- **Version control.** By the end of the integration testing stage, there were a few versions of the OS/400 in beta testing at customer sites. Under the just-in-time development philosophy, new releases were delivered to beta test customers as frequently as every 30 days. During this time, IBM Rochester continued its defect-removal activities until each version met or exceeded its own quality goals as well as the expectations of its beta test customers. The availability of multiple versions gave IBM customers the freedom to choose the best version for delivery. With many beta versions available, customers could try new features at their own risk until they were satisfied with what they got.

### Development and Quality Control Tools

Much attention was paid to computer-aided software engineering (CASE) and development support tools. For example, the integrated development support system (IDSS) played a major role in development activities and in maintaining the inventory of millions of lines of good-quality reusable code.

The following three tools were used to control and monitor software quality:

- The design change request (DCR) tool was used to record and track design changes and to control the implementation of new functions.

- Whenever there was a need to create or change capabilities, features, or design, a DCR feasibility review was conducted to seek approval for implementation.

- The problem tracking report (PTR) system recorded and tracked suspected problems and their fixes. It provided a way to monitor progress in defect-removal efforts, identify error-prone components, judge the workload, and certify quality of the software. It also provided an online documentation of problems. Whenever a defect was discovered, a PTR entry was logged into the system. The system then
notified the developer responsible for providing answers and fixes.

- Guidelines were established for the allocation of time to solve a problem. Developers were allocated time to respond, ranging from one to seven days, depending on the severity level of the problem.
- Error models included a defect injection and removal model, and a weekly error arrival model. Although these models are conventional, their accuracy was high. The first model predicted the rates of error injection and removal per thousand lines of code (KLOC) at each development stage. These were then compared with actual defect rates. The second model predicted the number of errors discovered, which had to be fixed each week during development. IBM had these software quality control models to track software quality many years before the OS/400 development. Projected quality data produced from IBM's own models were communicated regularly to senior management to fend off the possible temptation to reduce critical quality assurance efforts.

**Assessing IBM's Quality Models**

As illustrated in Exhibit 2, IBM's model predicted the rate of defect at each stage with great accuracy. Thanks to the defect injection removal model, IBM was able to predict the latent errors in its software, and make sure that its product met user quality expectations. For more information about the prediction accuracy of these models, refer to Phan et al (16).

**Quality Results**

Based on the data gathered on customer service calls, the number of customer requests per user per month for software assistance on documentation, administration, operations, and software bugs decreased by an average of 50 percent between the period before (September 1988) and after (September 1989) system delivery. The cost of software service calls on average was 20 percent to 40 percent less than those for System 36 and System 38. Systems delivery and installation time was reduced from an average of six weeks before 1985 to two weeks in 1989.

**Exhibit 2 IBM's Modeled versus Actual Defect Rates**

The number of errors in documentation found by users was reduced by 50 percent over that for the earlier projects, and overall satisfaction level with user manuals, gathered from user surveys, was higher than was the case for manuals for Systems 36 and 38. With the use of the electronic customer support system to download software fixes to customer sites, the average waiting time after diagnosis was reduced from 24 hours in 1988 to 30 minutes in 1990. Complaints per thousand installed systems were reduced by 10 percent between 1988 and 1989 (see Phan et al. (16)).

**More Studies at IBM**

Kaplan, Clark, and Tang at IBM went further, studying quality control strategies and processes in many of IBM's other development sites in addition to IBM Rochester and noted 40 innovations in quality management at IBM. Thanks to these innovations, during the period 1989-1993, IBM Santa Teresa labs' field defects for six leading products decreased by 46 percent, and lab service costs decreased by 20 percent. During the same period, revenues per employee at this lab increased by 58 percent, and customer satisfaction increased by 14 percent. Detailed discussions of these quality innovations are described in the book Secrets of...
Software Quality: 40 Innovations from IBM, New York: McGraw-Hill, 1995. IBM also experimented with the new Cleanroom process at the Santa Teresa lab in 1990. This process was originally developed at IBM Federal Systems Division in the 1980s by Harlan Mills et al. (13). In the Cleanroom process, software developers use formal verification to find all errors prior to testing. Unlike the traditional waterfall method, there is no unit test, and testing is entirely statistical -- that is, randomized. Despite the great potential of this technique, however, Kaplan et al. reported that the experiment using the Cleanroom process at Santa Teresa lab yielded mixed results. Of the 11 teams that completed the training for the Cleanroom technique, four teams went on to develop and test at least one increment of an actual product; two teams dropped it after using the techniques; one team voted not to use it; and the remaining teams did not use it seriously (10).

Sawyer and Guinan performed another study at an undisclosed IBM development site on software development processes and teams (19). The processes used include rapid application development (RAD), Cleanroom, object-oriented, joint application development (JAD), and formal methodologies. The study found that diverse software teams create variability, while production methods and processes reduce variance. The findings also suggest that factors such as social actions or individual level differences probably account for large variations in team performance. However, the authors reported no hard data on quality improvement.

LESSONS LEARNED

This section discusses and compares the differences and similarities between IBM and Microsoft quality processes and strategies. The maturity of their processes are then measured against the five levels of CMM and the key processes.

Development Processes

Microsoft's processes reflect novel ways to develop software in an industry that is filled with innovations and market niches. It differs from the traditional development models that IBM followed in the following ways:

- A project at Microsoft focuses on vision and scope rather than on requirements. At IBM, after the project proposal is approved, the project focuses on requirements.
- Project milestones at Microsoft are customer oriented, rather than development oriented. Each milestone is a synchronization point at which the product is recalibrated to meet the changes in customer expectations. Trade-offs of resources, features, and schedule are allowed at each milestone. At IBM, once the schedule and milestones are set, development teams are in charge, and trade-offs of resources, features, and schedule at milestones are less frequent.
- The "half-life" product that requires Microsoft teams to replace approximately 50 percent of its code for each version or every 18 months, in order to respond quickly to innovations, has prevented Microsoft from taking advantage of reuse. By contrast, reuse plays a major role in productivity and quality improvements at IBM.
- Microsoft's team structure, daily build, synch-and-stabilize, and single site development approaches demand great team communications, empowerment, and socialization. In comparison with IBM's processes, Microsoft's processes give teams greater freedom in planning and developing phases, and thus facilitate greater creativity, innovation, and team collaboration.

Microsoft and IBM development processes are similar in the use of rigorous inspections, reviews, and version control. Features and functions in each release are versioned, rather than all being included in the first version. Versioned releases allow Microsoft and IBM to respond to emerging user needs that may not have been foreseen. Other similarities include product development and testing that are done in parallel and continuous customer feedback in the development process.

The differences and similarities in development processes and strategies of IBM and Microsoft can be attributed to their unique organizational cultures and history. For its part,
EXHIBIT 3  Software Quality Processes and Strategies at IBM and Microsoft Measured Against SEI's CMM

<table>
<thead>
<tr>
<th>Software Engineering Quality Processes/ KPAs</th>
<th>CMM (level 5)</th>
<th>IBM</th>
<th>Microsoft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peer reviews</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Inter-group coordination</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Software product engineering</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Integrated software management</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Training program</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Organization process definition</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Organization process focus</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Tracking of errors</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Team empowerment</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Version control</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Frequent synchronization (daily or weekly)</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Customer involvement</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Software quality management</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Quantitative process management</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
</tr>
<tr>
<td>Process change management</td>
<td>Y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Technology change management</td>
<td>Y</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Defect Prevention</td>
<td>Y</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Legend: Y = Yes;  - = Not defined or required

IBM is a well-established software maker with a traditional management style. By contrast, Microsoft is a relatively young organization that offers developers more freedom in development processes than IBM does. Microsoft evolved its organizational structure and development processes through various stages and through trial and error.

While IBM processes required separation of test groups from development groups, Microsoft was very reluctant to do this until after 1984. The most influential individual in forming Microsoft corporate culture is Bill Gates, but his vice-president, Steve Ballmer, played a major role in bringing openness and freedom to Microsoft's development culture. Since the early days at Microsoft, Ballmer preferred a relatively unstructured environment and lots of freedom for developers and program managers to innovate, select their own development process, test their own products without separate testing groups, and make changes in the product late in the development cycle.

In 1988, after many serious quality problems, Bill Gates recruited Mike Maples, a former executive at IBM, to join Microsoft to oversee applications and systems development. At Microsoft, Maples implemented many quality concepts and processes that were successfully implemented at IBM, such as "zero defects," "do it right the first time," automated tests, and rigorous code inspections and reviews. It was the melding of management styles from key executives such as Bill Gates, Steve Ballmer, and Mike Maples throughout many years that built Microsoft's unique development culture and processes of today.

Process Maturity Measured by CMM

In this section the capability and maturity of IBM and Microsoft are measured against...
EXHIBIT 4  Summary of Benefits Resulting from Use of New Technology (4)

<table>
<thead>
<tr>
<th>Technology</th>
<th>Payoff Range</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>4GL</td>
<td>-90% to 500%</td>
<td>Enormous variability</td>
</tr>
<tr>
<td>CASE</td>
<td>9% to 128%</td>
<td>Probably 10% on average</td>
</tr>
<tr>
<td>Formal Methods</td>
<td>9%</td>
<td>Very few studies</td>
</tr>
<tr>
<td>Cleanroom</td>
<td>0% to 70%</td>
<td>Results depend on interpretation</td>
</tr>
<tr>
<td>CMM</td>
<td>6% to 570%</td>
<td>Wide Variability- more studies needed</td>
</tr>
<tr>
<td>Object Orientation (OO)</td>
<td>No hard numbers</td>
<td>“Most influential methodology”</td>
</tr>
</tbody>
</table>

CMM’s focuses and key process areas at all five levels, as listed below:

- **Level 1**: People -- competent workers
- **Level 2**: Project management -- software configuration management, software quality assurance, subcontract management, project tracking and oversight, project planning, and requirements management
- **Level 3**: Engineering process and organizational support – peer reviews, inter-group coordination, software product engineering, integrated software management, a training program, organization process definition, and organization process focus
- **Level 4**: Product and process quality -- software quality management and quantitative process management
- **Level 5**: Continuous process improvement defect prevention, technology change management, and process change management

A comparison of quality maturity and capability in processes based on CMM at IBM and Microsoft is presented in Exhibit 3. While neither IBM nor Microsoft officially adopted CMM, their great strength in key process areas that focus on people, project management processes, engineering processes, and organizational support (Exhibit 1) show that these two software makers' capability and maturity passed CMM level 3. However, only IBM quantitatively managed product and process quality, and reached level 4 of the CMM. This is because IBM achieved the two required KPAs to reach level 4 of CMM: quantitative process management and software quality management.

Like IBM, Microsoft collects product defect data, and tracks it for quality control. However, it lacks a robust defect projection and removal model that is used consistently throughout the entire corporation in order to quantitatively managed its product and process quality. This is the key difference in the capability and maturity between the two software makers. Because it does not have a defect prediction model in its quality management and decision-making process, the number of latent errors in products after delivery cannot be accurately predicted.

Microsoft requires that the number of fatal bugs be near zero before delivery, without considering the number of latent errors that remain in the product. Because near-zero errors is not a well-quantified concept, and the number of latent errors remains unknown, Microsoft has on many occasions delivered products to the market without adequate testing (Ref. 2, p. 296).

By contrast, IBM management relies on defect removal models and knows when its software product passes the required quality level at delivery. Quantitative models in the quality control process help IBM to closely predict the number of errors injected, removed, and remaining at the end of each stage and at delivery.

The results were also compared with competitors' benchmarks and those of early projects to measure the effectiveness of new quality processes and techniques. Rigorous inspections, a formal test methodology, and consistent models allow software to be tested and measured the same way with uniform results.

Finally, despite both companies' claims to some accomplishments in defect prevention, technology change management, and process...
change management (in italics in Exhibit 3) -- the required KPAs of level 5 --their claims have not been validated.

Newly Emerging Software Processes and "Silver Bullets"

In 1986, Brooks reviewed new software technologies and processes and concluded that there is "no silver bullet" for problems in software development (1). Thirteen years later, Glass reviewed several new software technologies and found they still show only modest benefits (Exhibit 4), with productivity and quality improvements averaging in the vicinity of 20 to 25 percent (4).

The most spectacular improvement in software quality improvement resulted from use of the Cleanroom process at NASA/Goddard Space Flight Center (6). It is reported that the Cleanroom process removed 91 percent of errors from the product in this study before the first test case was run and reduced time spent fixing errors in rework by 95 percent. Glass noted that it was the static error removal, a "rigorous inspection" technique, that removed 91 percent of errors before testing, not the Cleanroom process itself as claimed. The best process model that provides reliable data in his review was CMM, which showed a defect detection gain of 6 to 25 percent per year.

However, the gains in productivity and quality from these new processes are not much better than the gains that IBM achieved in the past decades using traditional methods and processes. In short, despite great progress in software technology for the past decade, as noted by Brooks Jr., there is still "no silver bullet" in software quality management.

Why Cleanroom Is Not Widely Used

Despite great initial success reported by Mills et al., Cleanroom techniques did not gain popularity in software engineering. Some reasons why the Cleanroom process has not gained widespread usage in the software industry today are as follows (7, 5):

- It is too theoretical, mathematical, and radical for use in a real development environment.
- It abandoned unit testing by developers, but instead replaced it with correctness verification and statistical testing. This is a radical departure from the traditional waterfall process that many organizations are using.
- It uses statistical (randomized) testing and thus
  - Cannot effectively test anticipated weak spots in the software product;
  - Prevents testers from controlling test inputs and anticipating test results. As a consequence, the costs of test result analysis increase.
- It requires a high level of maturity in the software development industry that not many companies have reached. As Henderson stated: "The use of cleanroom processes requires rigorous application of defined processes in all life cycle phases. Since most of the industry is still operating at the ad hoc level (as defined by the Software Engineering Institute Capability Maturity Model), the industry has not been ready to apply those techniques" (7).

Key Success Factors

Key success factors resulting from the study in this article include

- Feedback mechanisms from end users must be provided. End users should be involved in the entire development process. Problems encountered should be reported and recorded as soon as possible.
- Teams should be given ownership of the quality of the components being built.
- Organizations should test early and test frequently, especially in terms of beta tests. The availability of beta test versions allows customers to try versions at their own risk to best fit their needs and environment.
- If possible, good reusable artifacts should be utilized, to help reduce development time.
• Components that are built in parallel need frequent synchronization.

• Project management techniques and quality control standards should be enforced through rigorous reviews, inspections, unit tests, and a synchronization process. Changes and discovered defects should be closely monitored.

• Test cases and quality models should be built in advance, for better planning and control of quality assurance efforts. The test plan must be comprehensive and should be planned early.

• If possible, defect removal models should be built to predict the number of latent errors before delivery. Removal of all fatal errors before delivery is not good enough. Development quality should be quantitatively managed.

• Software versions must be well managed to minimize the impacts of design changes during development. Multiple versions also give customers the freedom to choose the version that gives them the best satisfaction. Today, many commercial software makers such as IBM, Netscape, and Microsoft make many beta versions of their software available for customers' use before delivery.

• A defect prevention process and the management of changes in technology and process should be implemented to optimize an organization's quality process and help the organization reach the highest level of maturity and capability, in software development.

CONCLUSION

With the dramatic increases in worldwide software expenditures, software developers cannot afford to make software quality an afterthought. This study provides some general strategies and processes to allow organizations to perform self-evaluations in the management of quality in development projects using CMM and processes at IBM and Microsoft as the benchmarks. While the software development environment today remains without a "silver bullet," the traditional, rigorous reviews and verifications that IBM and Microsoft use in their quality control processes are proving to be highly effective in today's development projects. The results also confirm CMM's basic premise that higher maturity capabilities and processes lead to better control of defects, shown by the improved better quality of software products at IBM.

As exemplified by the studies of IBM and Microsoft, the quality assurance management and processes discussed here can serve as a menu of strategies and techniques. Developers in various organizations with different cultures can choose from these strategies and techniques in order to improve their software quality processes.

References

Dien D. Phan is the Nicole Marie Stata MIS professor at the School of Business Administration of the University of Vermont in Burlington, and professor of BCIS at St. Cloud State University, Minnesota. Previously, Dr. Phan worked at IBM Corp. in the development and architecture areas.