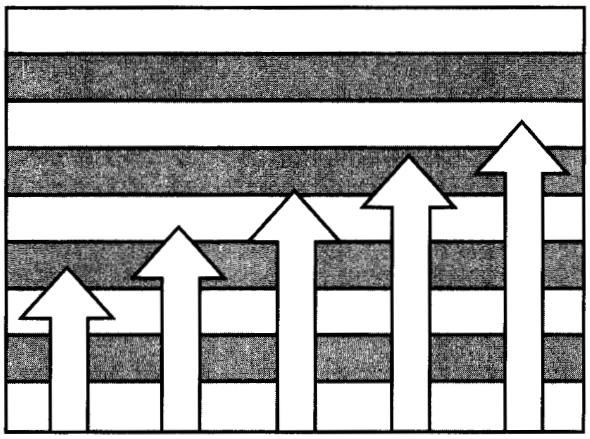
Technical forum



Programming quality improvement in IBM

A management system was established in IBM to improve the quality of its software products. It represents a nontraditional approach to quality improvement. The approach is based on empowering programming development teams, guided by a set of principles that were defined by the programming community and driven by aggressive goals established to improve quality and enhance customer satisfaction. In turn the experiences of the newly empowered teams led to a set of good programming practices that were shared across the programming community in IBM. The result has not only been a dramatic improvement in the quality of IBM's program products, but also the fostering of a work environment based on creativity and excellence that engenders pride of ownership for work performed.

Not unique to IBM, problems in software development, and the approaches taken to address them, existed throughout the data processing industry in its earlier days. Traditionally, programming development used a system of centralized control,[1] but it was not very adaptive, it contained few feedback mechanisms, and it did not account for developments that occurred outside of the company. Under this system the environment, or work culture, tended to stifle creativity, cause product cost and development cycle time to suffer, and fail to reward risk-taking. Programmers in many areas in IBM had to rigidly follow rules and procedures for development, documented in a guidebook.

Through the years, the number of program defects in IBM products declined, but by 1988 the rate had only improved by a factor of two from what it was 10 years before, and that was on a relative basis.[2] Because of continuous growth in the number of lines of code in many installations, a customer was apt to see more defects in spite of the improvement in the relative defect rate.

During the latter part of the 1980s several programming projects in IBM achieved high levels of quality and productivity by focusing on requirements to attain them. Then in 1990 a program instituted in response to IBM's commitment to market-driven quality (MDQ) established goals to improve defect quality throughout all of the software development organizations. To reach these goals, a system was set up to obtain commitments and measure results without dictating the implementation, which led to the establishment of principles to guide the IBM programming community.

The principles. A set of principles evolved based on benchmarking, consultant studies, and long-used internal processes and practices. Although some studies had been done about programming development in IBM, two studies were the primary source of the principles. The first, called the "best of breed" study, was performed internally and completed in April 1989. It looked at 35 projects from 19 different IBM laboratories. Characteristics of these projects were short cycle

times, high productivity, large amounts of code reuse or porting, excellent reliability or performance, improved usability, etc.

The second was a consultant study, completed in June 1989, that looked at development processes, methods, and tools of 11 other companies.[3] The conclusions of this study were essentially identical to those of the first one. A consultant study of six Japanese computer manufacturers completed later confirmed the results of the first two studies,[4] as did a number of "benchmarks" performed by IBM programming teams.

From these findings a list of 12 "principles" was published in August 1990 and serves as the primary guidance for programming development in IBM. Programming teams are empowered to apply these principles in ways that are best suited for their own use.

The principles represent nothing new. Most of them have been employed by all programmers at one time or another. However, taken collectively they provide an effective framework for programming development. Each is briefly described below.

A rigorous product development process. The process used by each development group must be defined and documented. It must be amenable to inspection to the point where definite checkpoints are established and the status of a project can be measured against these checkpoints. Status reporting and measurements must be an integral part of the process. The process must not be so rigid that it does not allow changes at the end of each product cycle or as the needs of the laboratory or product change, or both. Kickoff meetings for each stage of the process introduce the most recent modifications to the developers. Postmortems can determine what should be added to or deleted from the process for the next cycle.

Total customer satisfaction. The foundation of this principle is based on understanding and addressing the wants and needs of customers. This principle has to become an integral part of all software development processes.

Verification by simulation, analysis, and early prototyping and continuous validation. Products under development must be continuously refined to ensure that they meet customers' wants and needs. Customers and members of the marketing and service organization should be part of this activity. In addition, experts or consultants should be used where appropriate. Verification can take such forms as: customer advisory councils, making prototypes available for customer use, early support programs, and beta tests. This principle must be applied much earlier in the product development cycle than at the point of the traditional test period and early shipments to some customers.

Early software manufacturing involvement for all products. The distribution and support plan must be at the forefront of the product development and planning processes. Distribution and service of products are fast becoming key inhibitors to growth. We no longer live in the era of the host environment in which all program products were installed on a single CPU. Personal computers and workstations, along with distributed systems and client/server computing, have shown the importance and complexity of distribution and service. In addition, new ways for distributing products and providing service are now being used, i.e., CD-ROMs, electronic documentation and help, electronic publications, hypertext, etc.

Benchmarking. For products to attain world-class status, they have to be compared against their best competitors, externally and internally. A comparison implies some form of benchmarking for process, results, tools, techniques, etc. In addition, developers should be aware of technical activities within the internal programming community and participate in on-line bulletin boards and internal and external technical conferences on process and quality.

Dependency management. This principle applies to all dependencies—those upon which the developing group depends as well as those that are dependent on the group. In every case studied, successful projects exhibited strong dependency management characteristics. Ownership and commitment were present from the beginning and so was a clear definition of the expected deliverables. Closely related is good communication of the true status of dependencies to avoid surprises or late decommitments.

In-process measurements and ratings. This principle is primarily for products under develop-

ment. To predict the quality of a product when it ships, it is necessary to do some amount of defect tracking throughout the development cycle. Tracking should start as early in the cycle as possible, and historical records should be kept for cycle-to-cycle comparisons. In addition, some formal methodology (conforming to standards for the Malcolm Baldrige National Quality Award, ISO 9000, the Carnegie Mellon Software Engineering Institute's Capability Maturity Model, etc.) should be used by an organization and process, respectively, so that there is a consistent basis for comparisons and improvement. The methodology can provide a set of key measures to predict what the quality of a product will be when it is shipped.

Defect causal analysis and prevention. A very powerful technique,[5] this principle systematically eliminates defects and continuously improves products and processes. It can be applied to all aspects of development and not just to code or test. Basically defects are analyzed until their root cause has been identified. Then it is necessary to determine what changes must be made in the processes to prevent these types of defects from happening again.

Robust change control and problem management. All successful projects exhibited this attribute. Managing requirements and the associated change process are important. If not managed, they lead to constant change in plans, long development cycles, and low productivity. A related key success factor keeps data up to date so that real-time decisions on resource allocations, priorities, etc., can be made with confidence.

Tools, use of personal computers and workstations, technical education. Programmers should have the latest in both CASE (computer-aided systems engineering) tools technology (external as well as internal) and the hardware platforms necessary to run them at their full potential. High-speed connectivity is also needed to permit local and remote teams to work together efficiently. Programmers should know how to best use their development environment and also have an understanding of the latest advances in software engineering and process technology. Continuing technical education is required to best achieve that end.

Reuse of code or design and also of knowledge and experience. The intent here is to develop a means for sharing as much as possible across the programming community and as a result to achieve improvements in productivity, quality, and cycle time. Common libraries for code, on-line bulletin boards, and technical conferences aimed at gaining knowledge or experience are also part of this means. Design, test, and documentation should also be included to gain further leverage.

Linkage of customer value analysis to product content and investment. It is important to establish a clear linkage between customer satisfaction for a product, as determined by surveys and other customer interactions, and the investment in a product. From this linkage we determine what value new products or enhancements have for customers. By analyzing the effects that product offerings will have on customers' costs, productivity, and overall business effectiveness, we can deliver those solutions that are of greatest value.

We have standardized a methodology for measuring customer satisfaction so that we can discuss it on a community-wide basis and better target our improvement efforts. The methodology is based on customer satisfaction surveys (many different types of surveys are used depending on the product, target audience, etc.) in which we ask customers to rate our products and services on eight different attributes, and to give an overall rating. Additionally they are asked to identify the attribute that is most important to them and the one that most needs improvement. The methodology is called CUPRIMDSO, a term composed of the initial letters from each of the following words: capability, usability, performance, reliability, installability, maintainability, documentation, service, and overall satisfaction.

We use surveys with a five-point scale for answers—very satisfied, satisfied, neutral, dissatisfied, and very dissatisfied. We also focus on the percentage of nonsatisfied responses to our surveys—the number of those that are neutral, plus dissatisfied, plus very dissatisfied, divided by the total number of responses. Previously, neutral respondents were included with the satisfied and very satisfied customers to produce a satisfied index. However, the approach used today understands neutral customers as not really satisfied; accordingly they should be treated the same as dissatisfied or very dissatisfied customers.

Although this methodology is connected to the principles of customer satisfaction and the linkage of customer value analysis, it is also connected to good programming practices.

Good programming practices. In late 1990 the IBM programming community began to develop a set of good programming practices (GPPs) to go with the principles. The focus was on process items and techniques that had demonstrated improved MDQ results. They were generic in nature, rather than laboratory- or product-specific, and could be used to augment any development process. In essence they were specific implementations of the principles. Three of the more widely used GPPs are: Six Sigma Module Analysis, Statistical Process Control for Software,[6,7] and Orthogonal Defect Classification.[8] Defect causal analysis[5] is at the root of several of the GPPs.

Six Sigma Module Analysis is a technique that can be used to determine "error-prone" modules. All the modules in a product are classified according to the number of defects found in customer use, and are grouped into categories based on the results. Within categories some may require complete redesign and recoding, others may be candidates for more intense inspection and testing, and still others may require that the programmers responsible for them receive more education. As a refinement of this technique, Statistical Process Control for Software (SPCS) tries to identify error-prone modules before they are delivered to customers. Essentially, profiles are developed on several characteristics of modules that have exhibited a high number of defects after being shipped. Some sample characteristics are module size, number of unique operands referenced, number of unique operators executed, number of defects found in component test, and number of defects found in system test. From this information, control charts can be developed to track modules while they are under development. In this way, when a module exceeds one of the control limits it can be checked before it is released, thus avoiding potential problems for customers. With such use SPCS is a proactive technique, whereas Six Sigma Module Analysis is a reactive technique.

A still further refinement is Orthogonal Defect Classification (ODC). This technique[8] looks at all the defects and categorizes them as to type and when they should be found in the development process. Profiles can then be constructed on the strengths and weaknesses of the processes being used to develop the product. For example, with ODC it is possible to find out whether the initial specifications are sufficiently detailed. ODC allows a proactive posture to be taken on the development processes while products are being developed. It permits the fine-tuning of processes while they are being used rather than having to wait for defect feedback from customers.

The purpose of all the programming practices is to increase the product manager's knowledge of precise customer "wants and needs" so that subsequent releases of a product can overcome specific areas of weakness. Although some of these practices were followed in the past, they were not done with this level of thoroughness and rigor.

Results to date. The three primary measures that the IBM programming community has focused on during the past four years as part of IBM's MDQ efforts are: product defects, service defects, and customer satisfaction. In general, the following results have been achieved up to the present time:[2]

For over 25 years the IBM programming community has measured product defects by taking the number of defects found in a product after it ships and dividing it by the size of the product in terms of the number of lines of code. The only variable has been the product life span. Until 1988 we used a four-year life span, but since 1989 we have used a three-year life span because our shorter cycle times make previous product releases obsolete on a faster time scale. The base year is 1989, and our initial goal for product defects was to be 10 times better by 1991. By the end of 1991 we had achieved an improvement of approximately 4 to 5 times across the board, and by the end of 1992 we had become 10 to 12 times better. Although we were about a year behind our original goal, we greatly accelerated the pace of improvement. By far the most important result from a customer perspective is a reduction in the absolute number of defects in products. The number increased from 1989 to 1990, but it was flat from 1990 to 1991, and it declined slightly from 1991 to 1992.

Service defects are measured by the absolute number of defective fixes shipped to customers.

In 1988 15 to 20 percent of our fixes were defective. Today the number is a fraction of 1 percent, and in 1992 the majority of our product teams shipped "zero" defective fixes. By the end of 1993 a defective fix was expected to be the exception rather than the norm for all but a very small number of products. The improvement was accomplished by focusing attention on basic process work, on defect causal analysis, and especially on communicating and sharing ideas across all laboratories.

Customer satisfaction has proved to be our biggest challenge. Our latest product releases have scored about 10 to 15 percent better than their predecessors. Although encouraging, this rating is difficult to improve because customer expectations are increasing each year.

Summary. The transition of the IBM programming community from an environment of classical mechanistic management to one based on empowerment provided programming developers with the opportunity to effectively achieve IBM's MDQ goals in software products.[9] A framework based on proven principles was developed, and it led to a number of good programming practices being shared across the community. As a result, there has been a renewed spirit of pride in ownership as well as a marked improvement in key measurements of product quality. The papers in this issue of the *IBM Systems Journal* describe some of the ways and the projects in which the new environment has fostered improved program quality.

Cited references and note

1. W. S. Humphrey, *Managing the Software Process*, Addison-Wesley Publishing Co., Reading, MA (1989).
2. A. G. Ganek, presentation to GUIDE Board of Directors, Anaheim, CA (March 9, 1993).
3. IBM-funded study by SRI International of 11 U.S. data processing companies completed June 1989. Results not published externally.
4. IBM-funded study by SRI International of six Japanese data processing companies, completed October 1990. Results not published externally.
5. R. G. Mays, C. L. Jones, G. J. Holloway, and D. P. Studinski, "Experiences with Defect Prevention," *IBM Systems Journal* **29**, No. 1, 4–32 (1990).
6. K. Tang and J. B. Lo, "Determination of the Optimal Process Mean When Inspection Is Based on a Correlated Variable," *IIE Transactions* **25**, No. 3, 66–72 (May 1993).
7. J. B. Lo, "A Quantitative Approach for Software Development Process Control," presented at *34th Joint National Conference of ORSA/TIMS* (November 1–4, 1992).
8. R. Chillarege, I. Bhandari, J. Chaar, M. Halliday, D. Moebus, B. Ray, and W. Wong, "Orthogonal Defect Classification—A Concept for In-Process Measurements," *IEEE Transactions on Software Engineering* **18**, No. 11, 943–956 (November 1992).
9. More extensive information is available from the author.

D. L. Bencher
Software Solutions Division
Somers
New York