ACHIEVING QUALITY IN THE DEVELOPMENT PROCESS

Caryl L. Pettijohn

Caryl L. Pettijohn *is director of the Signal Processor Laboratory at AT&T Bell Laboratories in Whippany, New Jersey. He is responsible for development of the Enhanced Modular Signal Processor, a standard computer for the Navy, along with all the support computer programs needed for naval applications. His earlier assignments included developing and conducting the AT&T Bell Laboratories Software Project Management Workshop. Mr. Pettijohn joined the company in 1956 after receiving a B.S.E.E. degree from the University of Texas that same year. In 1983 he was named a Bell Labs Fellow.*

Quality as applied to a product, either software or hardware, is fitness for use. Products that are fit for use generate satisfaction for the customer and profits for the producer. For the producer, the cost of quality is the sum of the costs to prevent, appraise, and correct defects. Quality must therefore be addressed from the early development stages—specification, design, implementation, and test—through manufacture and distribution. The most important step in preventing defects is to define the right product in the first place. As design proceeds, appraisal techniques further reduce defects, so that in the final stages defect correction is minimal. This article describes the techniques and tools for achieving quality and thus improving productivity and profits.

What Is Quality?

Quality has traditionally been addressed primarily in the factory. Only recently have we begun to focus on quality in the development process. This article discusses tools and techniques to improve the quality of hardware and software designs. But before looking at specific tools and techniques, it is useful to address the following issues: What is quality? Why is it important? Who is responsible? Where should it be applied?

Quality has been variously defined as the absence of defects, conformance to requirements, fitness for use, and customer satisfaction. The preferred definition is *fitness for use*. This definition includes many attributes overlooked in the more traditional definitions. Products that are fit for use

- Cost what the customer can afford
- Have the features most desired by the customer
- Are easily installed and operated
- Perform to customer expectations
- Are reliable

- Are durable
- Are well documented
- Are easy to fix and maintain.

In short, products that are fit for use generate satisfaction for the customer and sales and profits for the producer.

Quality is important because of its direct impact on bottom-line profitability. A financial metric that doesn't appear on corporate balance sheets is cost of quality, or C_q. In equation form, cost of quality equals the sum of the costs to *prevent, appraise,* and *correct* defects.

$$C_q = C_p + C_a + C_c$$

Obviously, the better the processes for preventing errors, the lower the costs of appraising and correcting errors, and the lower the cost of quality (see Figure 1). Let's take an example. James Olson, AT&T president and chief operating officer, said in 1985,[1] "Conservative estimates are that activities associated with quality-control—inspection, testing, re-work and so forth—account for at least 25 percent of our [cost of] sales. . . . We believe it's possible to get that down in the 10 percent range, and still maintain product quality." That would be annual savings of one to two billion dollars. Where will these savings go? The answer is bottom-line profits. That is why improved quality is so important. And that is why the quality-improvement methods described in this issue are gaining rapid acceptance.

As to who is responsible for quality, the traditional answer is "Quality Control," and the traditional role for the Quality Control organization is on the factory floor appraising products for defects. Today, this mode of operation is obsolete. Today, the job is twofold:

1. Design products with minimum design defects.
2. Manufacture products with minimum latent defects.

This is not a job that the Quality Control and Quality Assurance organizations can do alone. Quality Control's primary role continues to be appraisal. This includes evaluation of product quality, separating the good from the bad, collecting and analyzing fitness-for-use data, and feeding the results of the data to the development and manufacturing organizations so that their processes can be improved. Quality Assurance's primary role is prevention. This includes measuring the effectiveness of development and manufacturing processes, developing improved processes, and teaching effective prevention, appraisal, and correction techniques and processes to everyone. Development and manufacturing organizations must also put effective prevention, appraisal, and correction processes in place, and must work endlessly to improve these processes. So the real answer to "Who is responsible for quality?" is "Everyone."

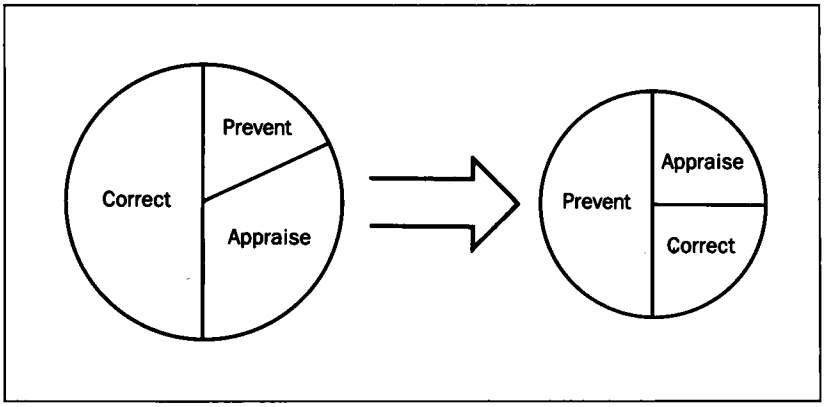

Figure 1. Cost of quality includes the costs associated with preventing, appraising, and correcting defects. When quality measures are not applied early, correction cost is high. When quality is built into the design and development processes, the cost of prevention goes up modestly, while the cost of correction goes down dramatically, and the savings turn into profits.

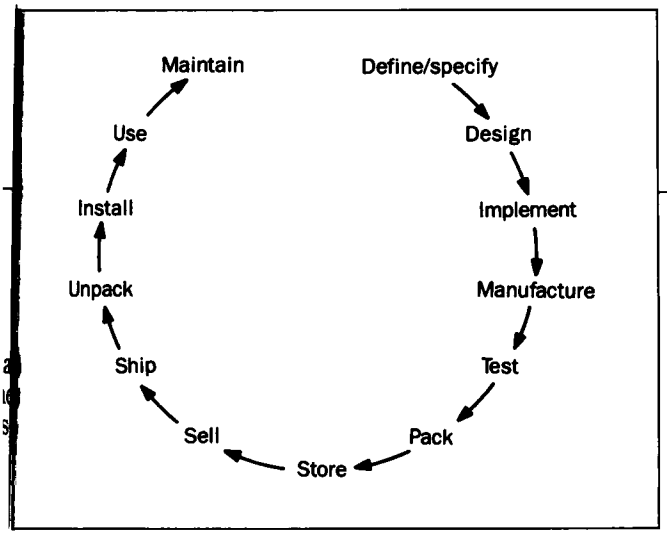

Figure 2. Fitness for use must be considered at each step in the development, manufacture, and distribution process. If the requirements specification is inaccurate or incomplete, it isn't fit for the designer to follow. If the design and implementation are defective, the product isn't fit to manufacture and test. Defects in any step adversely affect fitness for use in the steps that follow, up through maintenance and use by the customer.

Since quality is everyone's job, it should be applied everywhere. Specifically, quality processes should be applied to the specification, design, implementation, and test phases of every hardware and software development project, and at every step of product manufacture and distribution (Figure 2).

The remainder of this article focuses on tools and techniques for achieving a quality design by prevention, appraisal, and correction.

Design Defect Prevention

The key issues in design defect prevention are these:

- Define the right product.
- Specify it correctly and completely.
- Design it correctly and completely.
- Implement it correctly and completely.

Defining the Right Product. Of all the steps in the development process, none is more important nor receives less attention than defining the right product. To define the right product successfully, it is essential to understand existing systems and their deficiencies, and to understand the market for and economics of any new system that corrects some or all existing system deficiencies. Without such understanding, the end result is a large commitment of resources to the development of a system or product with substantial fit-for-use deficiencies.

There is a simple set of rules known to be effective in helping to define the right product:

- Draw a diagram of the existing system, showing all workstations, inputs and outputs, and the flow of work through the system. This will usually identify bottlenecks and inefficiencies worth correcting.
- Evaluate the product(s) against the list of fit-for-use considerations given in the left-hand panel on the next page. This will usually identify major fit-for-use deficiencies.
- Armed with the above analysis, talk to potential users/customers to find out what is important to them.
- Armed with the above data,
  - — estimate the technical feasibility of correcting identified deficiencies;
  - — estimate the cost and worth of the new system or product;
  - — estimate the market for the new system or product.

Following the above outline will usually provide sufficient data to make an informed decision to invest or not invest in the new system or product.

Specifying the Product Completely and Accurately. Another set of rules leads to requirements specifications that are fit for use, that is, complete, clear, and accurate:

- Establish a standard, user-oriented format that includes all the important fit-for-use requirements.
- Make a model or prototype to fill in the gaps.
- Use a small number of experienced people, and give them enough time.
- Require economic justification for each major feature.
- Inspect the document.

The third and fourth rules require no further explanation; the other three are expanded below.

Format. Aside from introductory and wrap-up boilerplate, the format of a requirements document should

Is it fit for use?

A first step in designing a new product or system is to look at an existing one for possible deficiencies in fitness for use, particularly in the following areas.

- Features
- Cost
  - — Purchase
  - — Operating
- Human Interface
- Performance
  - — Capacity
  - — Response Time
- Reliability
  - — Mean Time to Failure
  - — Mean Time to Recover
  - — Mean Time to Repair
- Maintainability
- Size and Weight
- Power Consumption
- Installability
- Documentation

Feature Description

Part of the design process involves specifying the feature requirements. A standard feature description, like the one outlined below, helps to ensure that the requirements will be complete and will meet the fitness-for-use test.

Name – Short name of feature
Identifier – Unique and traceable
Description – Processing in equation or specification definition language
Stimulus – What initiates the feature
Processing – Subfeatures
Input – What the user does
Output – What the user gets
Bounds – Ranges and limits
Bounds Exceeded Response – Error conditions
Frequency of Use – How much, how often
Performance – Capacity and response time
Test Approach – How to test (essential)
Expected Volatility – High, medium, or low (high volatility = high risk)

contain two sections. The first should specify general requirements, including environmental, physical, performance, reliability, and cost. The second section should list specific requirements, that is, features, organized in a hierarchical or layered structure. Each feature should be described in a standard format, as shown in the panel on the right.

Modeling or Prototyping. The specific (feature) requirements are frequently underspecified, that is, important features are missing, because the system or product is not understood well enough. The remedy is modeling or prototyping. A model is a simulation of algorithmic activities or processes. A prototype is a "quick-and-dirty" version of some part or parts of a proposed system or product. The primary issue is to decide when and what to model or prototype. These techniques are helpful when there is lack of clarity about algorithmic or processing requirements, about design alternatives, or about interfaces (both user and system). Because modeling and prototyping aren't free, use should be limited to those areas where payoff is high. It may make no sense to model or prototype parts of the system for which complete and accurate requirements exist—unless, of course, there are system-level issues such as fuzzy system interfaces that require a full model or prototype.

Inspection. This topic is covered in the section on Design Quality Appraisal. But one example will show the effectiveness of the inspection process as applied to requirements documents. In this case the process simply involved a small group (eight to ten people) sitting around a table while the requirements document was read aloud, line by line. Typos and minor defects were corrected on the spot. Major rewrite was done off line and reinspected later. The 2200 pages of requirements were inspected at an expense of 2500 staff hours. On the average, the process detected 7.7 defects per page. The inspection rate was just over one staff hour per page or 0.15 staff hour per defect.

Designing the Product Right. Assume that the previous steps (definition and specification) have been done right. The designer now knows *why* the system or product is needed and *what* must be designed. Next the designer needs to know *how* to do it and *when* it is needed. For this, each project should produce a design handbook to tell *how* and a development plan to tell *when*. (Planning, while critical to successful development, is not the topic of this article; it is covered elsewhere in this issue.[2])

The design handbook should describe three kinds of design standards, all aimed at doing the job right the first time:

- Design methodology, which defines the design process and answers such questions as how and when to conduct reviews and inspections, how and when to obtain design approval, and how to change the design.
- Design rules, which specify practices for electrical, physical, and software designers to follow. It is basically a list of "do's" and "don'ts" with examples. Usually every project group will develop its own unique set of design rules.
- Design tools, which list electrical, physical, and software design tools.

After the proper design processes, rules, and tools are in place, one might expect a quality design to emerge automatically. But it is not that easy. Some additional design phase cost-of-quality drivers are discussed below.

Painstaking attention to detail is essential to a quality design. This means detailed functional and timing analysis and simulation over the full set of design margins.

Choice of prototype medium is another important cost-of-quality decision. The key issue is how to do design validation—via software simulation or hardware breadboard. On the surface, this looks about even, trading the cost of computer time against the cost of a real model. But it's not that simple. The hardware prototype is usually more costly. It introduces an extra manufacturing cycle, and its design errors are more difficult and more time-consuming to find and fix. With the simulator, there are better debugging tools. All one has to do is correct the logic error and run again, whereas with the breadboard, one has to correct the logic and change the physical hardware before running again. In addition, the physical presence of a hardware prototype tends to inhibit change for the better, because it's costly to change what already exists physically.

Electrical designers need both functional and timing simulators that will accommodate device, circuit card, and module level simulation. The goal is zero-defect logic designs that cut circuit debug times dramatically. In addition, electrical designers need one-page circuit card electrical schematics, not the multipage schematics (sometimes as many as 40 pages) often produced by today's tools. Physical designers need automatic circuit card layout, routing and audit tools, as well as enclosure assembly and parts-detailing tools that are easy to use and that accommodate changes quickly.

Reuse of existing architectures, designs, and products reduces cost of quality. As an example, a software operations support system was built using tools and code from another system. Of the total software product, 60 percent was reused from a different project. Productivity (based on lines of delivered code per staff month) was three times greater.

Products that are designed to be easy to manufacture and transfer will have fewer defects. A few simple rules ensure this quality:

- Require joint design and manufacturing participation in product design reviews, production line design reviews, selection and use of CAD (computer-aided design) and CAM (computer-aided manufacturing) systems, and regular design transfer meetings.
- Require transfer of people from the manufacturer to the design organization when design begins, and from the design organization to the manufacturer when the design is transferred.
- Require overlap of product design and production line design so that the production line designers can influence the product design.
- Require electronic design transfer. Paper takes too much time and may not accurately reflect the design residing in the CAD system.

Implement the Product Right. Whereas hardware products are implemented by the manufacturer, software products are implemented by the designer. In the software design process, the key ingredients that ensure successful implementation are these:

- *High-level languages*, which are easier to write, less error prone, easier to debug, and easier to maintain.

- *Structured programming,* which is also easier to write, less error prone, easier to debug, and easier to maintain.
- *Reuse,* which means using something that already exists, so the code doesn't have to be debugged or rewritten at the risk of introducing errors. Usually, someone else maintains it, so that task is also eliminated.
- *Code inspections,* which are perhaps the most effective method known to force painstaking attention to detail.
- *Machine simulators,* which provide the only available stable debugging environment when writing code for a newly designed machine.

Design Quality Appraisal

The cost of appraisal covers both product and process. The two basic product quality appraisal mechanisms are *test* and *inspect,* and the guiding rule is to test and inspect at every step in the design, manufacture, and distribution processes. The products of the design process are definition, specification, and design documents. The productive way to "test" these documents is to inspect them. This allows testing to begin long before the final system or product exists.

The Inspection Process. The code inspection process is widely used throughout the AT&T Bell Laboratories software community and elsewhere.[3,4] The inspection process is also being applied to specification, design, and test documents. A modified version is used with hardware design at AT&T Bell Laboratories at Merrimack Valley. The primary purpose for inspecting documents is early detection of design defects. Data show that it is about an order of magnitude less costly to correct defects by inspection than in the laboratory.

Test Purpose and Tools. While inspections are effective at reducing the amount of testing required, testing will still be needed to detect remaining defects and to exercise all the previously specified fit-for-use features.

A *content check* is required to check that all advertised features are present and accounted for.

Regression tests are useful to verify that changes have not introduced errors in previously working features.

Functional tests exercise all normal and error paths and algorithms of the system. Typically this requires a model or simulation of the real environment that the system or product will experience. To ensure that all features are tested, it is useful to have a test coverage matrix, that is, a list of all features with the identifier(s) of all tests that exercise each feature. Path counters are useful for determining test coverage of software. This technique requires that the compiler embed a counter in every path of a program. The counters are then incremented during testing every time each path is executed. After test execution is completed, the path counters are analyzed. This information shows where additional tests are necessary to cover all existing paths through the program.

Fault insertion is another strategy for determining test effectiveness. This technique introduces a number of known bugs into a system before testing begins. From the ratio of known bugs to unknown bugs found during testing, one can infer the total number of bugs programmed into the system.

Interface tests exercise all internal and external interfaces.

Performance tests measure capacity and response time. They require a performance simulator to predict performance, a performance monitor to measure performance, and an environment simulator to load the system. The usual procedure is to simulate the performance of the proposed system architecture to identify bottlenecks so the architecture can be tuned for better performance before it is committed to design. After the system is built, the monitor is connected, a simulated load is applied, and performance data are collected. Analysis will usually reveal unpredicted bottlenecks. Further architecture simulation is used to predict the effect of changes to correct bottlenecks. High-payoff changes are then incorporated into the system, and performance data are again collected to confirm the effectiveness of the change. This cycle (predict, change, measure, tune) continues until performance is acceptable. Obviously, this process is costly. Changes (especially hardware changes) after the system is built are sometimes not feasible. This puts a premium on

perfecting the architecture (and corresponding architecture simulation) before the system is built.

Stress tests assess system responses when subjected to abnormal use. Abnormal use includes too much input (overload), wrong (illegal) input, loss of input (data or messages), and loss of hardware resources. System responses to such conditions include error control and recovery and hardware reconfiguration. Stress testing is difficult because it requires the creation of complex inputs and the monitoring of complex responses.

Reliability/maintainability tests measure mean time to failure, mean time to recover, and mean time to repair. Like performance testing, reliability/maintainability testing involves both predictive models and measurement. Once the real system is available, actual mean time to failure can be measured by meticulously keeping track of failures. If stress testing was successful at finding ways of inserting error conditions, then a variety of ways of causing recovery and reconfiguration sequences should be available. One must then measure the recovery and reconfiguration times. Finally, one must simulate a number of hardware faults and measure the mean time to repair. This includes fault detection, fault location (diagnostics), faulty card replacement, and rerunning diagnostics to confirm repair.

Usability tests assess the adequacy of the user interfaces, error responses, "help" features, and documentation. The best techniques include thoughtful human factors design and evaluation, and letting real users use the system.

Quality Metrics. Collection of quality metrics is an important part of the cost of appraisal. Without appropriate metrics one does not know how good or how bad the product is, nor can one measure quality improvement. A standard set of corporate quality metrics for software has been established at Bell Laboratories.[5] A similar set of corporate quality metrics is needed for hardware.

There are two primary sources of quality data appropriate to the development process: inspection data and maintenance (trouble) reports (MRs). From these data, one can derive the important quality metrics. For each step in the development process, one needs the following data:

- Cumulative defects detected
- Cumulative defect density
- Where (in development process) the defect was injected
- Where (in development process) the defect was detected
- Cause(s) of defects.

The first two are easy to get from inspection and MR data. The last three require analysis, and are typically more difficult to obtain. Requiring that the last three data items be present on MRs before the MRs can be processed usually proves satisfactory. It is also helpful to include a collection of typical causes of defects. Examples of typical design and coding errors from a large software project, along with frequency of occurrence, are shown in Tables I and II. These tables illustrate the importance of having "Cause of Defect" on the MR. In each case there are several dominant causes of defects. People can now place more emphasis on these dominant causes of defects during design and inspections, thus reducing the number of errors committed.

Process Quality Appraisal. Just as we have to appraise the quality of our product (test and inspect), we must also appraise the quality of our processes. The two parts of process quality appraisal are fit-for-use analysis and corrective action; see Figure 3. Fit-for-use analysis identifies dominant causes of defects, and corrective action identifies changes to the design process that will reduce defects in the future.

Defect Correction

The final part of the cost-of-quality equation is the cost of correcting defects. This is the price paid for not preventing defects. It includes scrap, rework, and warranty costs, and is usually much larger than either cost of prevention or cost of appraisal. Warranty costs are those costs associated with customer returns and complaints against products still under warranty.

Diagnosis can be very difficult, especially for defects reported by the customer or user. Users may very well use the system in ways the designers never intended.

Table I. Types of Design Errors*

Percent of Total	Description
7	Communication Error
20	Forgotten Cases or Steps
10	Timing Problems
2	Initialization Error
12	Inadequate Checking
17	Extreme Conditions Neglected
8	Sequencing Error
12	Misunderstanding of Problem Specifications
12	Other

Table II. Types of Coding Errors*

Percent of Total	Description
14	Misunderstanding of Design
4	Initialization Error
29	Inadequate or Forgotten Checking
4	Case Selection Error
20	Inconsistent Use of Variables or Data
7	Sequencing Error
4	Loop Control Error
7	Language Usage Problems
4	Incorrect Subroutine Usage
7	Other

*In a project that included 458K lines of noncommentary source code.

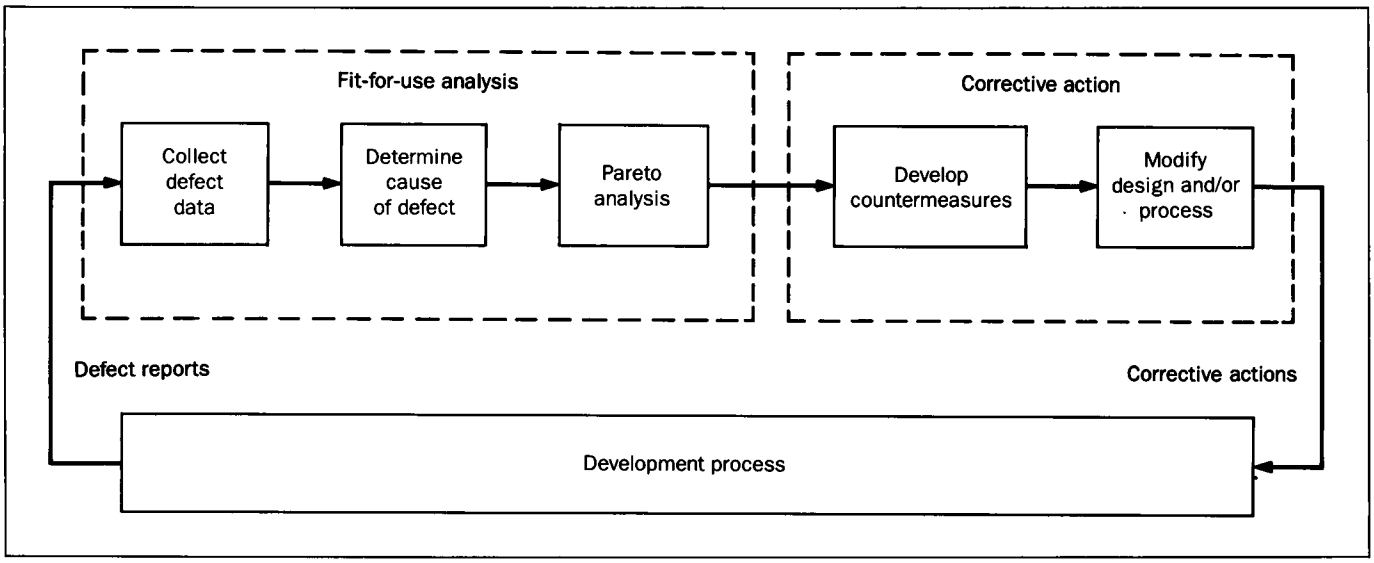

Figure 3. Fit-for-use analysis collects data on all system defects. The cause of each defect is then determined, and the data are analyzed to determine the dominant cause(s) of design defects. All that remains is to develop effective countermeasures for each dominant cause of defects, and to modify the design and/or process so that fewer such errors will be committed in the future.

Designers should first try to recreate the problem in the laboratory, using available test tools. But many of these problems are difficult to recreate in the laboratory because of small or incomplete data bases or an inability to duplicate the load placed on the system by the user. Remote access to the customer's system (when permitted) is helpful. Otherwise, a field trip may be necessary to diagnose the problem correctly. Needless to say, all of this is very expensive.

Once problems are correctly diagnosed, a design fix must be generated. For hardware, a field kit must be devised that can be installed in the field. For software, one can sometimes generate a field fix (patch). Other times a new load module must be generated. Whatever the nature of the fix, the system must be retested to ensure that the fix works, and that the fix didn't "unfix" something else. Then once the system is fixed, the fixes have to be distributed to all users—another expensive process.

Correcting defects is *always* expensive. Expenses can be minimized by applying the quality criterion, fitness for use, at all stages of development. The cost-of-quality metric, defined herein, allows designers to measure what quality costs, and to measure quality improvement.

References

1. *Management Focus*, Vol. II, No. 1, February 4, 1985, p. 4.
2. L. B. Robertson and G. A. Secor, "Effective Management of Software Development," *AT&T Technical Journal*, Vol. 65, No. 2, March/April 1986, pp. 94-101.
3. P. J. Fowler, "In-process Inspections of Workproducts at AT&T," *AT&T Technical Journal*, Vol. 65, No. 2, March/April 1986, pp. 102-112.
4. M. E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal*, Vol. 15, No. 3, 1976, pp. 182-211.
5. J. Inglis, "Standard Software Quality Metrics," *AT&T Technical Journal*, Vol. 65, No. 2, March/April 1986, pp. 113-118.

(Manuscript received November 14, 1985)

MARCH/APRIL 1986 • VOLUME 65 • ISSUE 2