ing diskettes is a trivial manufacturing operation, and we can guarantee that exact duplicates of the software are always created.

Or can we? We need to ensure the tracks are placed on the diskettes within a specified tolerance so that the overwhelming majority of disk drives can read the diskettes. In addition, we need to ensure the magnetic flux for distinguishing a zero from a one is sufficient for read/write heads to detect. The disk duplication machines can, and do, wear and go out of tolerance. So even a "simple" process such as disk duplication may encounter problems due to variation between samples.

KEY POINT

Controlling variation is the key to a high-quality product. In the software context, we strive to control the variation in the process we apply, the resources we expend, and the quality attributes of the end product.

But how does this apply to software work? How might a software development organization need to control variation? From one project to another, we want to minimize the difference between the predicted resources needed to complete a project and the actual resources used, including staff, equipment, and calendar time. In general, we would like to make sure our testing program covers a known percentage of the software, from one release to another. Not only do we want to minimize the number of defects that are released to the field, we'd like to ensure that the variance in the number of bugs is also minimized from one release to another. (Our customers will likely be upset if the third release of a product has ten times as many defects as the previous release.) We would like to minimize the differences in speed and accuracy of our hotline support responses to customer problems. The list goes on and on.

8.1.1 Quality

The *American Heritage Dictionary* defines *quality* as "a characteristic or attribute of something." As an attribute of an item, quality refers to measurable characteristics—things we are able to compare to known standards such as length, color, electrical properties, and malleability. However, software, largely an intellectual entity, is more challenging to characterize than physical objects.

Nevertheless, measures of a program's characteristics do exist. These properties include cyclomatic complexity, cohesion, number of function points, lines of code, and many others, discussed in Chapters 19 and 24. When we examine an item based on its measurable characteristics, two kinds of quality may be encountered: quality of design and quality of conformance.

Quote:

"It takes less time to do a thing right than explain why you did it wrong."
Henry Wadsworth Longfellow

Quality of design refers to the characteristics that designers specify for an item. The grade of materials, tolerances, and performance specifications all contribute to the quality of design. As higher-grade materials are used, tighter tolerances and greater levels of performance are specified, the design quality of a product increases, if the product is manufactured according to specifications.

Quality of conformance is the degree to which the design specifications are followed during manufacturing. Again, the greater the degree of conformance, the higher is the level of quality of conformance.

In software development, quality of design encompasses requirements, specifications, and the design of the system. Quality of conformance is an issue focused

primarily on implementation. If the implementation follows the design and the resulting system meets its requirements and performance goals, conformance quality is high.

But are quality of design and quality of conformance the only issues that software engineers must consider? Robert Glass [GLA98] argues that a more "intuitive" relationship is in order:

User satisfaction = compliant product + good quality + delivery within budget and schedule

At the bottom line, Glass contends that quality is important, but if the user isn't satisfied, nothing else really matters. DeMarco [DEM99] reinforces this view when he states: "A product's quality is a function of how much it changes the world for the better." This view of quality contends that if a software product provides substantial benefit to its end-users, they may be willing to tolerate occasional reliability or performance problems.

8.1.2 Quality Control

What is software quality control?

Variation control may be equated to quality control. But how do we achieve quality control? *Quality control* involves the series of inspections, reviews, and tests used throughout the software process to ensure each work product meets the requirements placed upon it. Quality control includes a feedback loop to the process that created the work product. The combination of measurement and feedback allows us to tune the process when the work products created fail to meet their specifications. This approach views quality control as part of the manufacturing process.

Quality control activities may be fully automated, entirely manual, or a combination of automated tools and human interaction. A key concept of quality control is that all work products have defined, measurable specifications to which we may compare the output of each process. The feedback loop is essential to minimize the defects produced.

8.1.3 Quality Assurance

WebRef

A wide variety of software quality resources can be found at www.qualitytree.com/links/links.htm

Quality assurance consists of the auditing and reporting functions of management. The goal of quality assurance is to provide management with the data necessary to be informed about product quality, thereby gaining insight and confidence that product quality is meeting its goals. Of course, if the data provided through quality assurance identify problems, it is management's responsibility to address the problems and apply the necessary resources to resolve quality issues.

8.1.4 Cost of Quality

The *cost of quality* includes all costs incurred in the pursuit of quality or in performing quality-related activities. Cost of quality studies are conducted to provide a base-

line for the current cost of quality, identify opportunities for reducing the cost of quality, and provide a normalized basis of comparison. The basis of normalization is almost always dollars. Once we have normalized quality costs on a dollar basis, we have the necessary data to evaluate where the opportunities lie to improve our processes. Furthermore, we can evaluate the effect of changes in dollar-based terms.

What are the components of the cost of quality?

Quality costs may be divided into costs associated with prevention, appraisal, and failure. *Prevention costs* include

- quality planning
- formal technical reviews
- test equipment
- training

Appraisal costs include activities to gain insight into product condition the "first time through" each process. Examples of appraisal costs include

- in-process and interprocess inspection
- equipment calibration and maintenance
- testing

ADVICE

Don't be afraid to incur significant prevention costs. Rest assured that your investment will provide an excellent return.

Failure costs are those that would disappear if no defects appeared before shipping a product to customers. Failure costs may be subdivided into internal failure costs and external failure costs. *Internal failure costs* are incurred when we detect a defect in our product prior to shipment. Internal failure costs include

- rework
- repair
- failure mode analysis

External failure costs are associated with defects found after the product has been shipped to the customer. Examples of external failure costs are

- complaint resolution
- product return and replacement
- help line support
- warranty work

As expected, the relative costs to find and repair a defect increase dramatically as we go from prevention to detection to internal failure to external failure costs. Figure 8.1, based on data collected by Boehm [BOE81] and others, illustrates this phenomenon.

Anecdotal data reported by Kaplan, Clark, and Tang [KAP95] reinforces earlier cost statistics and is based on work at IBM's Rochester development facility:

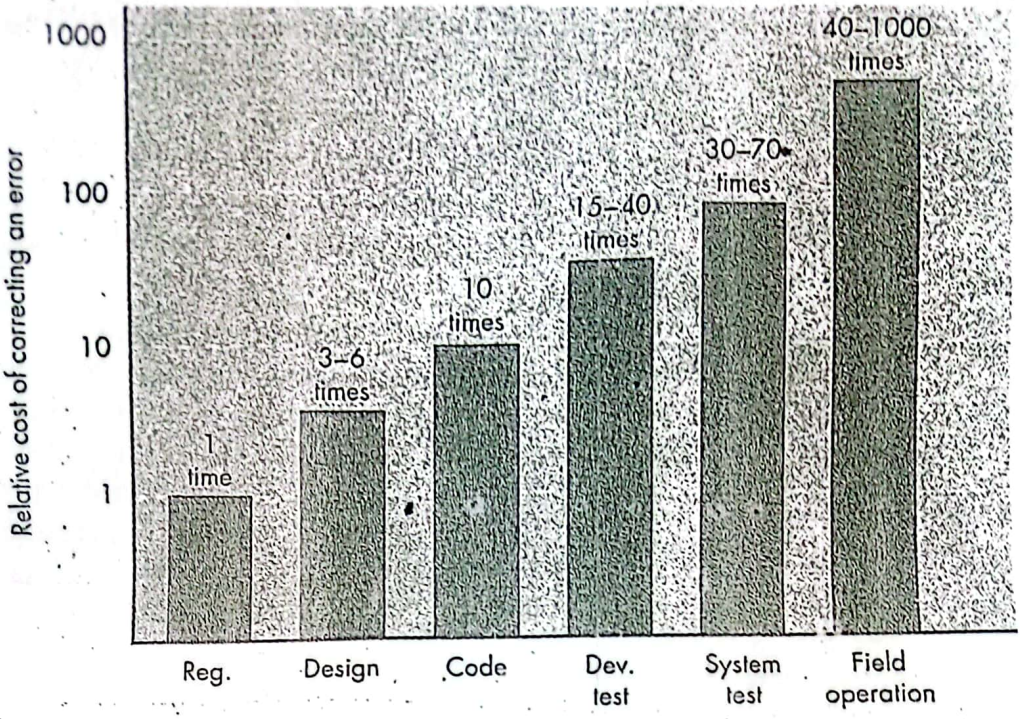

FIGURE 8.1 Relative cost of correcting an error

Testing is necessary, but it's also a very expensive way to find errors. Spend time finding errors early in the process and you may be able to significantly reduce testing and debugging costs.

A total of 7053 hours was spent inspecting 200,000 lines of code with the result that 3112 potential defects were prevented. Assuming a programmer cost of $40.00 per hour, the total cost of preventing 3112 defects was $282,120, or roughly $91.00 per defect.

Compare these numbers to the cost of defect removal once the product has been shipped to the customer. Suppose that there had been no inspections, but that programmers had been extra careful and only one defect per 1000 lines of code [significantly better than industry average] escaped into the shipped product. That would mean that 200 defects would still have to be fixed in the field. At an estimated cost of $25,000 per field fix, the cost would be $5 million, or approximately 18 times more expensive than the total cost of the defect prevention effort.

It is true that IBM produces software that is used by hundreds of thousands of customers and that their costs for field fixes may be higher than those for software organizations that build custom systems. This in no way negates the results just noted. Even if the average software organization has field fix costs that are 25 percent of IBM's (most have no idea what their costs are!), the cost savings associated with quality control and assurance activities are compelling.

8.2 THE QUALITY MOVEMENT

Today, senior managers at companies throughout the industrialized world recognize that high product quality translates to cost savings and an improved bottom line. However, this was not always the case. The quality movement began in the 1940s with the seminal work of W. Edwards Deming [DEM86] and had its first true test in Japan. Using Deming's ideas as a cornerstone, the Japanese developed a systematic

approach to the elimination of the root causes of product defects. Throughout the 1970s and 1980s, their work migrated to the western world and was given names such as "total quality management" (TQM).[2] Although terminology differs across different companies and authors, a basic four step progression is normally encountered and forms the foundation of any good TQM program.

KEY POINT
TQM can be applied to computer software. The TQM approach stresses continuous process improvement.

The first step, called *kaizen*, refers to a system of continuous process improvement. The goal of *kaizen* is to develop a process (in this case, the software process) that is visible, repeatable, and measurable.

The second step, invoked only after *kaizen* has been achieved, is called *atarimae hinshitsu*. This step examines intangibles that affect the process and works to optimize their impact on the process. For example, the software process may be affected by high staff turnover, which itself is caused by constant reorganization within a company. Maybe a stable organizational structure could do much to improve the quality of software. *Atarimae hinshitsu* would lead management to suggest changes in the way reorganization occurs.

WebRef
A wide variety of resources for continuous process improvement and TQM can be found at **deming.eng.clemson.edu/**

While the first two steps focus on the process, the next step, called *kansei* (translated as "the five senses"), concentrates on the user of the product (in this case, software). In essence, by examining the way the user applies the product *kansei* leads to improvement in the product itself and, potentially, to the process that created it.

Finally, a step called *miryokuteki hinshitsu* broadens management concern beyond the immediate product. This is a business-oriented step that looks for opportunity in related areas identified by observing the use of the product in the marketplace. In the software world, *miryokuteki hinshitsu* might be viewed as an attempt to uncover new and profitable products or applications that are an outgrowth from an existing computer-based system.

For most companies *kaizen* should be of immediate concern. Until a mature software process (Chapter 2) has been achieved, there is little point in moving to the next steps.

8.3 SOFTWARE QUALITY ASSURANCE

Even the most jaded software developers will agree that high-quality software is an important goal. But how do we define quality? A wag once said, "Every program does something right, it just may not be the thing that we want it to do."

Many definitions of software quality have been proposed in the literature. For our purposes, *software quality* is defined as

How do we define software quality?

Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.

2 See [ART92] for a comprehensive discussion of TQM and its use in a software context and [KAP95] for a discussion of the use of the Baldrige Award criteria in the software world.

There is little question that this definition could be modified or extended. In fact, a definitive definition of software quality could be debated endlessly. For the purposes of this book, the definition serves to emphasize three important points:

1. Software requirements are the foundation from which quality is measured. Lack of conformance to requirements is lack of quality.
2. Specified standards define a set of development criteria that guide the manner in which software is engineered. If the criteria are not followed, lack of quality will almost surely result.
3. A set of implicit requirements often goes unmentioned (e.g., the desire for ease of use and good maintainability). If software conforms to its explicit requirements but fails to meet implicit requirements, software quality is suspect.

8.3.1 Background Issues

Quote:

"We made too many wrong mistakes."
Yogi Berra

Quality assurance is an essential activity for any business that produces products to be used by others. Prior to the twentieth century, quality assurance was the sole responsibility of the craftsperson who built a product. The first formal quality assurance and control function was introduced at Bell Labs in 1916 and spread rapidly throughout the manufacturing world. During the 1940s, more formal approaches to quality control were suggested. These relied on measurement and continuous process improvement as key elements of quality management.

Today, every company has mechanisms to ensure quality in its products. In fact, explicit statements of a company's concern for quality have become a marketing ploy during the past few decades.

WebRef

An in-depth tutorial and wide-ranging resources for quality management can be found at **www.management.gov.**

The history of quality assurance in software development parallels the history of quality in hardware manufacturing. During the early days of computing (1950s and 1960s), quality was the sole responsibility of the programmer. Standards for quality assurance for software were introduced in military contract software development during the 1970s and have spread rapidly into software development in the commercial world [IEE94]. Extending the definition presented earlier, software quality assurance is a "planned and systematic pattern of actions" [SCH98] that are required to ensure high quality in software. The scope of quality assurance responsibility might best be characterized by paraphrasing a once-popular automobile commercial: "Quality Is Job #1." The implication for software is that many different constituencies have software quality assurance responsibility—software engineers, project managers, customers, salespeople, and the individuals who serve within an SQA group.

The SQA group serves as the customer's in-house representative. That is, the people who perform SQA must look at the software from the customer's point of view. Does the software adequately meet the quality factors noted in Chapter 19? Has soft-

ware development been conducted according to pre-established standards? Have technical disciplines properly performed their roles as part of the SQA activity? The SQA group attempts to answer these and other questions to ensure that software quality is maintained.

8.3.2 SQA Activities

Software quality assurance is composed of a variety of tasks associated with two different constituencies—the software engineers who do technical work and an SQA group that has responsibility for quality assurance planning, oversight, record keeping, analysis, and reporting.

Software engineers address quality (and perform quality assurance and quality control activities) by applying solid technical methods and measures, conducting formal technical reviews, and performing well-planned software testing. Only reviews are discussed in this chapter. Technology topics are discussed in Parts Three through Five of this book.

The charter of the SQA group is to assist the software team in achieving a high-quality end product. The Software Engineering Institute [PAU93] recommends a set of SQA activities that address quality assurance planning, oversight, record keeping, analysis, and reporting. These activities are performed (or facilitated) by an independent SQA group that:

What is the role of an SQA group?

Prepares an SQA plan for a project. The plan is developed during project planning and is reviewed by all interested parties. Quality assurance activities performed by the software engineering team and the SQA group are governed by the plan. The plan identifies

- evaluations to be performed
- audits and reviews to be performed
- standards that are applicable to the project
- procedures for error reporting and tracking
- documents to be produced by the SQA group
- amount of feedback provided to the software project team

Participates in the development of the project's software process description. The software team selects a process for the work to be performed. The SQA group reviews the process description for compliance with organizational policy, internal software standards, externally imposed standards (e.g., ISO-9001), and other parts of the software project plan.

Reviews software engineering activities to verify compliance with the defined software process. The SQA group identifies, documents, and tracks deviations from the process and verifies that corrections have been made.

Audits designated software work products to verify compliance with those defined as part of the software process. The SQA group reviews selected work products; identifies, documents, and tracks deviations; verifies that corrections have been made; and periodically reports the results of its work to the project manager.

Ensures that deviations in software work and work products are documented and handled according to a documented procedure. Deviations may be encountered in the project plan, process description, applicable standards, or technical work products.

Records any noncompliance and reports to senior management. Noncompliance items are tracked until they are resolved.

In addition to these activities, the SQA group coordinates the control and management of change (Chapter 9) and helps to collect and analyze software metrics.

8.4 SOFTWARE REVIEWS

ADVICE

Like water filters, FTRs tend to retard the "flow" of software engineering activities. Too few and the flow is "dirty." Too many and the flow slows to a trickle. Use metrics to determine which reviews work and which may not be effective. Take the ineffective ones out of the flow.

Software reviews are a "filter" for the software engineering process. That is, reviews are applied at various points during software development and serve to uncover errors and defects that can then be removed. Software reviews "purify" the software engineering activities that we have called *analysis, design,* and *coding.* Freedman and Weinberg [FRE90] discuss the need for reviews this way:

> Technical work needs reviewing for the same reason that pencils need erasers: *To err is human.* The second reason we need technical reviews is that although people are good at catching some of their own errors, large classes of errors escape the originator more easily than they escape anyone else. The review process is, therefore, the answer to the prayer of Robert Burns:
>
> *O wad some power the giftie give us*
> *to see ourselves as other see us*
>
> A review—any review—is a way of using the diversity of a group of people to:
>
> 1. Point out needed improvements in the product of a single person or team;
> 2. Confirm those parts of a product in which improvement is either not desired or not needed;
> 3. Achieve technical work of more uniform, or at least more predictable, quality than can be achieved without reviews, in order to make technical work more manageable.

Many different types of reviews can be conducted as part of software engineering. Each has its place. An informal meeting around the coffee machine is a form of review, if technical problems are discussed. A formal presentation of software design to an audience of customers, management, and technical staff is also a form of