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 14? Has software development been conducted according to preestablished 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.

16.2 Elements of Software Quality Assurance

WebRef

An in-depth discussion of SQA, including a wide array of definitions, can be obtained at **www.swqual.com/newsletter/vol2/no1/vol2no1.html**.

Software quality assurance encompasses a broad range of concerns and activities that focus on the management of software quality. These can be summarized in the following manner [Hor03]:

Standards. The IEEE, ISO, and other standards organizations have produced a broad array of software engineering standards and related documents. Standards may be adopted voluntarily by a software engineering organization or imposed by the customer or other stakeholders. The job of SQA is to ensure that standards that have been adopted are followed and that all work products conform to them.

Reviews and audits. Technical reviews are a quality control activity performed by software engineers for software engineers (Chapter 15). Their intent is to uncover errors. Audits are a type of review performed by SQA personnel with the intent of ensuring that quality guidelines are being followed for software engineering work. For example, an audit of the review process might be conducted to ensure that reviews are being performed in a manner that will lead to the highest likelihood of uncovering errors.

Testing. Software testing (Chapters 17 through 20) is a quality control function that has one primary goal—to find errors. The job of SQA is to ensure that testing is properly planned and efficiently conducted so that it has the highest likelihood of achieving its primary goal.

Error/defect collection and analysis. The only way to improve is to measure how you're doing. SQA collects and analyzes error and defect data to better understand how errors are introduced and what software engineering activities are best suited to eliminating them.

Change management. Change is one of the most disruptive aspects of any software project. If it is not properly managed, change can lead to confusion, and confusion almost always leads to poor quality. SQA ensures that adequate change management practices (Chapter 22) have been instituted.

Education. Every software organization wants to improve its software engineering practices. A key contributor to improvement is education of software engineers, their managers, and other stakeholders. The SQA organization takes the lead in software process improvement (Chapter 30) and is a key proponent and sponsor of educational programs.

Quote:

"Excellence is the unlimited ability to improve the quality of what you have to offer."

Rick Petin

Vendor management. Three categories of software are acquired from external software vendors—*shrink-wrapped packages* (e.g., Microsoft *Office*), a *tailored shell* [Hor03] that provides a basic skeletal structure that is custom tailored to the needs of a purchaser, and *contracted software* that is custom designed and constructed from specifications provided by the customer organization. The job of the SQA organization is to ensure that high-quality software results by suggesting specific quality practices that the vendor should follow (when possible), and incorporating quality mandates as part of any contract with an external vendor.

Security management. With the increase in cyber crime and new government regulations regarding privacy, every software organization should institute policies that protect data at all levels, establish firewall protection for WebApps, and ensure that software has not been tampered with internally. SQA ensures that appropriate process and technology are used to achieve software security.

Safety. Because software is almost always a pivotal component of human-rated systems (e.g., automotive or aircraft applications), the impact of hidden defects can be catastrophic. SQA may be responsible for assessing the impact of software failure and for initiating those steps required to reduce risk.

Risk management. Although the analysis and mitigation of risk (Chapter 28) is the concern of software engineers, the SQA organization ensures that risk management activities are properly conducted and that risk-related contingency plans have been established.

In addition to each of these concerns and activities, SQA works to ensure that software support activities (e.g., maintenance, help lines, documentation, and manuals) are conducted or produced with quality as a dominant concern.

INFO

Quality Management Resources

There are dozens of quality management resources available on the Web, including professional societies, standards organizations, and general information sources. The sites that follow provide a good starting point:

American Society for Quality (ASQ) Software Division
www.asq.org/software

Association for Computer Machinery **www.acm.org**

Data and Analysis Center for Software (DACS)
www.dacs.dtic.mil/

International Organization for Standardization (ISO)
www.iso.ch

ISO SPICE
www.isospice.com

Malcolm Baldridge National Quality Award
www.quality.nist.gov
Software Engineering Institute
www.sei.cmu.edu/
Software Testing and Quality Engineering
www.stickyminds.com
Six Sigma Resources
www.isixsigma.com/
www.asq.org/sixsigma/
TickIT International: Quality certification topics
www.tickit.org/international.htm
Total Quality Management (TQM)
General information:
www.gslis.utexas.edu/~rpollock/tqm.html
Articles: **www.work911.com/tqmarticles.htm**
Glossary:
www.quality.org/TQM-MSI/TQM-glossary.html

16.3 SQA Tasks, Goals, and Metrics

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 control activities) by applying solid technical methods and measures, conducting technical reviews, and performing well-planned software testing.

16.3.1 SQA Tasks

The charter of the SQA group is to assist the software team in achieving a high-quality end product. The Software Engineering Institute recommends a set of SQA actions that address quality assurance planning, oversight, record keeping, analysis, and reporting. These actions 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 as part of project planning and is reviewed by all stakeholders. Quality assurance actions 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 conducted, standards that are applicable to the project, procedures for error reporting and tracking, work products that are produced by the SQA group, and feedback that will be provided to the software 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 software engineering work products.

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

In addition to these actions, the SQA group coordinates the control and management of change (Chapter 22) and helps to collect and analyze software metrics.

16.3.2 Goals, Attributes, and Metrics

Quote:

"Quality is never an accident; it is always the result of high intention, sincere effort, intelligent direction and skillful execution; it represents the wise choice of many alternatives."

William A. Foster

The SQA actions described in the preceding section are performed to achieve a set of pragmatic goals:

Requirements quality. The correctness, completeness, and consistency of the requirements model will have a strong influence on the quality of all work products that follow. SQA must ensure that the software team has properly reviewed the requirements model to achieve a high level of quality.

Design quality. Every element of the design model should be assessed by the software team to ensure that it exhibits high quality and that the design itself conforms to requirements. SQA looks for attributes of the design that are indicators of quality.

Code quality. Source code and related work products (e.g., other descriptive information) must conform to local coding standards and exhibit characteristics that will facilitate maintainability. SQA should isolate those attributes that allow a reasonable analysis of the quality of code.

Quality control effectiveness. A software team should apply limited resources in a way that has the highest likelihood of achieving a high-quality result. SQA analyzes the allocation of resources for reviews and testing to assess whether they are being allocated in the most effective manner.

Figure 16.1 (adapted from [Hya96]) identifies the attributes that are indicators for the existence of quality for each of the goals discussed. Metrics that can be used to indicate the relative strength of an attribute are also shown.

FIGURE 16.3 **Software quality goals, attributes, and metrics**
Source: Adapted from [Hya96].

Goal	Attribute	Metric
Requirement quality	Ambiguity	Number of ambiguous modifiers (e.g., many, large, human-friendly)
	Completeness	Number of TBA, TBD
	Understandability	Number of sections/subsections
	Volatility	Number of changes per requirement
		Time (by activity) when change is requested
	Traceability	Number of requirements not traceable to design/code
	Model clarity	Number of UML models
		Number of descriptive pages per model
		Number of UML errors
Design quality	Architectural integrity	Existence of architectural model
	Component completeness	Number of components that trace to architectural model
		Complexity of procedural design
	Interface complexity	Average number of pick to get to a typical function or content
		Layout appropriateness
	Patterns	Number of patterns used
Code quality	Complexity	Cyclomatic complexity
	Maintainability	Design factors (Chapter 8)
	Understandability	Percent internal comments
		Variable naming conventions
	Reusability	Percent reused components
	Documentation	Readability index
QC effectiveness	Resource allocation	Staff hour percentage per activity
	Completion rate	Actual vs. budgeted completion time
	Review effectiveness	See review metrics (Chapter 14)
	Testing effectiveness	Number of errors found and criticality
		Effort required to correct an error
		Origin of error

16.4 Formal Approaches to SQA

In the preceding sections, I have argued that software quality is everyone's job and that it can be achieved through competent software engineering practice as well as through the application of technical reviews, a multi-tiered testing strategy, better control of software work products and the changes made to them, and the application of accepted software engineering standards. In addition, quality can be defined

WebRef
Useful information on SQA and formal quality methods can be found at **www.gslis.utexas.edu/~rpollock/tqm.html**.

in terms of a broad array of quality attributes and measured (indirectly) using a variety of indices and metrics.

Over the past three decades, a small, but vocal, segment of the software engineering community has argued that a more formal approach to software quality assurance is required. It can be argued that a computer program is a mathematical object. A rigorous syntax and semantics can be defined for every programming language, and a rigorous approach to the specification of software requirements (Chapter 21) is available. If the requirements model (specification) and the programming language can be represented in a rigorous manner, it should be possible to apply mathematic proof of correctness to demonstrate that a program conforms exactly to its specifications.

Attempts to prove programs correct are not new. Dijkstra [Dij76a] and Linger, Mills, and Witt [Lin79], among others, advocated proofs of program correctness and tied these to the use of structured programming concepts (Chapter 10).

16.5 Statistical Software Quality Assurance

Statistical quality assurance reflects a growing trend throughout industry to become more quantitative about quality. For software, statistical quality assurance implies the following steps:

? What steps are required to perform statistical SQA?

1. Information about software errors and defects is collected and categorized.
2. An attempt is made to trace each error and defect to its underlying cause (e.g., nonconformance to specifications, design error, violation of standards, poor communication with the customer).
3. Using the Pareto principle (80 percent of the defects can be traced to 20 percent of all possible causes), isolate the 20 percent (the *vital few*).
4. Once the vital few causes have been identified, move to correct the problems that have caused the errors and defects.

This relatively simple concept represents an important step toward the creation of an adaptive software process in which changes are made to improve those elements of the process that introduce error.

Quote:
"A statistical analysis, properly conducted, is a delicate dissection of uncertainties, a surgery of suppositions."
M. J. Moroney

16.5.1 A Generic Example

To illustrate the use of statistical methods for software engineering work, assume that a software engineering organization collects information on errors and defects for a period of one year. Some of the errors are uncovered as software is being developed. Others (defects) are encountered after the software has been released to its end users. Although hundreds of different problems are uncovered, all can be tracked to one (or more) of the following causes:

- Incomplete or erroneous specifications (IES)
- Misinterpretation of customer communication (MCC)

- Intentional deviation from specifications (IDS)
- Violation of programming standards (VPS)
- Error in data representation (EDR)
- Inconsistent component interface (ICI)
- Error in design logic (EDL)
- Incomplete or erroneous testing (IET)
- Inaccurate or incomplete documentation (IID)
- Error in programming language translation of design (PLT)
- Ambiguous or inconsistent human/computer interface (HCI)
- Miscellaneous (MIS)

Quote:

"20 percent of the code has 80 percent of the errors. Find them, fix them!"

Lowell Arthur

To apply statistical SQA, the table in Figure 16.2 is built. The table indicates that IES, MCC, and EDR are the vital few causes that account for 53 percent of all errors. It should be noted, however, that IES, EDR, PLT, and EDL would be selected as the vital few causes if only serious errors are considered. Once the vital few causes are determined, the software engineering organization can begin corrective action. For example, to correct MCC, you might implement requirements gathering techniques (Chapter 5) to improve the quality of customer communication and specifications. To improve EDR, you might acquire tools for data modeling and perform more stringent data design reviews.

It is important to note that corrective action focuses primarily on the vital few. As the vital few causes are corrected, new candidates pop to the top of the stack.

Statistical quality assurance techniques for software have been shown to provide substantial quality improvement [Art97]. In some cases, software organizations

FIGURE 16.2 Data collection for statistical SQA

	Total		Serious		Moderate		Minor	
Error	No.	%	No.	%	No.	%	No.	%
IES	205	22%	34	27%	68	18%	103	24%
MCC	156	17%	12	9%	68	18%	76	17%
IDS	48	5%	1	1%	24	6%	23	5%
VPS	25	3%	0	0%	15	4%	10	2%
EDR	130	14%	26	20%	68	18%	36	8%
ICI	58	6%	9	7%	18	5%	31	7%
EDL	45	5%	14	11%	12	3%	19	4%
IET	95	10%	12	9%	35	9%	48	11%
IID	36	4%	2	2%	20	5%	14	3%
PLT	60	6%	15	12%	19	5%	26	6%
HCI	28	3%	3	2%	17	4%	8	2%
MIS	56	6%	0	0%	15	4%	41	9%
Totals	942	100%	128	100%	379	100%	435	100%

have achieved a 50 percent reduction per year in defects after applying these techniques.

The application of the statistical SQA and the Pareto principle can be summarized in a single sentence: *Spend your time focusing on things that really matter, but first be sure that you understand what really matters!*

16.5.2 Six Sigma for Software Engineering

Six Sigma is the most widely used strategy for statistical quality assurance in industry today. Originally popularized by Motorola in the 1980s, the Six Sigma strategy "is a rigorous and disciplined methodology that uses data and statistical analysis to measure and improve a company's operational performance by identifying and eliminating defects' in manufacturing and service-related processes" [ISI08]. The term Six Sigma is derived from six standard deviations—3.4 instances (defects) per million occurrences—implying an extremely high quality standard. The Six Sigma methodology defines three core steps:

? What are the core steps of the Six Sigma methodology?

- *Define* customer requirements and deliverables and project goals via well-defined methods of customer communication.
- *Measure* the existing process and its output to determine current quality performance (collect defect metrics).
- *Analyze* defect metrics and determine the vital few causes.

If an existing software process is in place, but improvement is required, Six Sigma suggests two additional steps:

- *Improve* the process by eliminating the root causes of defects.
- *Control* the process to ensure that future work does not reintroduce the causes of defects.

These core and additional steps are sometimes referred to as the DMAIC (define, measure, analyze, improve, and control) method.

If an organization is developing a software process (rather than improving an existing process), the core steps are augmented as follows:

- *Design* the process to (1) avoid the root causes of defects and (2) to meet customer requirements.
- *Verify* that the process model will, in fact, avoid defects and meet customer requirements.

This variation is sometimes called the DMADV (define, measure, analyze, design, and verify) method.

A comprehensive discussion of Six Sigma is best left to resources dedicated to the subject. If you have further interest, see [ISI08], [Pyz03], and [Sne03].

16.6 Software Reliability

Quote:

"The unavoidable price of reliability is simplicity."

C. A. R. Hoare

There is no doubt that the reliability of a computer program is an important element of its overall quality. If a program repeatedly and frequently fails to perform, it matters little whether other software quality factors are acceptable.

Software reliability, unlike many other quality factors, can be measured directly and estimated using historical and developmental data. *Software reliability* is defined in statistical terms as "the probability of failure-free operation of a computer program in a specified environment for a specified time" [Mus87]. To illustrate, program *X* is estimated to have a reliability of 0.999 over eight elapsed processing hours. In other words, if program *X* were to be executed 1000 times and require a total of eight hours of elapsed processing time (execution time), it is likely to operate correctly (without failure) 999 times.

Whenever software reliability is discussed, a pivotal question arises: What is meant by the term *failure*? In the context of any discussion of software quality and reliability, failure is nonconformance to software requirements. Yet, even within this definition, there are gradations. Failures can be only annoying or catastrophic. One failure can be corrected within seconds, while another requires weeks or even months to correct. Complicating the issue even further, the correction of one failure may in fact result in the introduction of other errors that ultimately result in other failures.

16.6.1 Measures of Reliability and Availability

KEY POINT

Software reliability problems can almost always be traced to defects in design or implementation.

Early work in software reliability attempted to extrapolate the mathematics of hardware reliability theory to the prediction of software reliability. Most hardware-related reliability models are predicated on failure due to wear rather than failure due to design defects. In hardware, failures due to physical wear (e.g., the effects of temperature, corrosion, shock) are more likely than a design-related failure. Unfortunately, the opposite is true for software. In fact, all software failures can be traced to design or implementation problems; wear (see Chapter 1) does not enter into the picture.

There has been an ongoing debate over the relationship between key concepts in hardware reliability and their applicability to software. Although an irrefutable link has yet to be established, it is worthwhile to consider a few simple concepts that apply to both system elements.

KEY POINT

It is important to note that MTBF and related measures are based on CPU time, not wall clock time.

If we consider a computer-based system, a simple measure of reliability is *mean-time-between-failure* (MTBF):

$$\text{MTBF} = \text{MTTF} + \text{MTTR}$$

where the acronyms MTTF and MTTR are *mean-time-to-failure* and *mean-time-to-repair*,[2] respectively.

---

2 Although debugging (and related corrections) may be required as a consequence of failure, in many cases the software will work properly after a restart with no other change.

Many researchers argue that MTBF is a far more useful measure than other quality-related software metrics discussed in Chapter 23. Stated simply, an end user is concerned with failures, not with the total defect count. Because each defect contained within a program does not have the same failure rate, the total defect count provides little indication of the reliability of a system. For example, consider a program that has been in operation for 3000 processor hours without failure. Many defects in this program may remain undetected for tens of thousand of hours before they are discovered. The MTBF of such obscure errors might be 30,000 or even 60,000 processor hours. Other defects, as yet undiscovered, might have a failure rate of 4000 or 5000 hours. Even if every one of the first category of errors (those with long MTBF) is removed, the impact on software reliability is negligible.

Some aspects of availability (not discussed here) have nothing to do with failure. For example, scheduling downtime (for support functions) causes the software to be unavailable.

However, MTBF can be problematic for two reasons: (1) it projects a time span between failures, but does not provide us with a projected failure rate, and (2) MTBF can be misinterpreted to mean average life span even though this is *not* what it implies.

An alternative measure of reliability is *failures-in-time* (FIT)—a statistical measure of how many failures a component will have over one billion hours of operation. Therefore, 1 FIT is equivalent to one failure in every billion hours of operation.

In addition to a reliability measure, you should also develop a measure of availability. *Software availability* is the probability that a program is operating according to requirements at a given point in time and is defined as

$$\text{Availability} = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} \times 100\%$$

The MTBF reliability measure is equally sensitive to MTTF and MTTR. The availability measure is somewhat more sensitive to MTTR, an indirect measure of the maintainability of software.

16.6.2 Software Safety

ote:

"...safety of the ...shall be the ...low."

Cicero

Software safety is a software quality assurance activity that focuses on the identification and assessment of potential hazards that may affect software negatively and cause an entire system to fail. If hazards can be identified early in the software process, software design features can be specified that will either eliminate or control potential hazards.

A modeling and analysis process is conducted as part of software safety. Initially, hazards are identified and categorized by criticality and risk. For example, some of the hazards associated with a computer-based cruise control for an automobile might be: (1) causes uncontrolled acceleration that cannot be stopped, (2) does not respond to depression of brake pedal (by turning off), (3) does not engage when switch is activated, and (4) slowly loses or gains speed. Once these system-level hazards are identified, analysis techniques are used to assign severity and probability of occurrence.[3] To be effective, software must be analyzed in the context of the entire

3 This approach is similar to the risk analysis methods described in Chapter 28. The primary difference is the emphasis on technology issues rather than project-related topics.

Quote:

"I cannot imagine any condition which would cause this ship to founder. Modern shipbuilding has gone beyond that."

E. J. Smith, captain of the *Titanic*

system. For example, a subtle user input error (people are system components) may be magnified by a software fault to produce control data that improperly positions a mechanical device. If and only if a set of external environmental conditions is met, the improper position of the mechanical device will cause a disastrous failure. Analysis techniques [Eri05] such as fault tree analysis, real-time logic, and Petri net models can be used to predict the chain of events that can cause hazards and the probability that each of the events will occur to create the chain.

Once hazards are identified and analyzed, safety-related requirements can be specified for the software. That is, the specification can contain a list of undesirable events and the desired system responses to these events. The role of software in managing undesirable events is then indicated.

Although software reliability and software safety are closely related to one another, it is important to understand the subtle difference between them. Software reliability uses statistical analysis to determine the likelihood that a software failure will occur. However, the occurrence of a failure does not necessarily result in a hazard or mishap. Software safety examines the ways in which failures result in conditions that can lead to a mishap. That is, failures are not considered in a vacuum, but are evaluated in the context of an entire computer-based system and its environment.

WebRef

A worthwhile collection of papers on software safety can be found at **www.safeware-eng.com/**.

A comprehensive discussion of software safety is beyond the scope of this book. If you have further interest in software safety and related system issues, see [Smi05], [Dun02], and [Lev95].

16.7 The ISO 9000 Quality Standards[4]

A *quality assurance system* may be defined as the organizational structure, responsibilities, procedures, processes, and resources for implementing quality management [ANS87]. Quality assurance systems are created to help organizations ensure their products and services satisfy customer expectations by meeting their specifications. These systems cover a wide variety of activities encompassing a product's entire life cycle including planning, controlling, measuring, testing and reporting, and improving quality levels throughout the development and manufacturing process. ISO 9000 describes quality assurance elements in generic terms that can be applied to any business regardless of the products or services offered.

To become registered to one of the quality assurance system models contained in ISO 9000, a company's quality system and operations are scrutinized by third-party auditors for compliance to the standard and for effective operation. Upon successful registration, a company is issued a certificate from a registration body represented by the auditors. Semiannual surveillance audits ensure continued compliance to the standard.

---

4 This section, written by Michael Stovsky, has been adapted from "Fundamentals of ISO 9000," a workbook developed for *Essential Software Engineering*, a video curriculum developed by R. S. Pressman & Associates, Inc. Reprinted with permission.