

KEY
CONCEPTS

elements of software
quality assurance . 450
formal approaches. 456
goals 454
ISO 9001:2008
standard. 462
quality management
resources 452
Six Sigma 458
software
reliability 459
software safety .. 460
SQA plan 463
SQA tasks 453
statistical software
quality assurance . 456

The software engineering approach described in this book works toward a single goal: to produce on-time, high-quality software. Yet many readers will be challenged by the question: “What is software quality?”

Philip Crosby [Cro79], in his landmark book on quality, provides a wry answer to this question:

The problem of quality management is not what people don't know about it. The problem is what they think they do know . . .

In this regard, quality has much in common with sex. Everybody is for it. (Under certain conditions, of course.) Everyone feels they understand it. (Even though they wouldn't want to explain it.) Everyone thinks execution is only a matter of following natural inclinations. (After all, we do get along somehow.) And, of course, most people feel that problems in these areas are caused by other people. (If only they would take the time to do things right.)

QUICK
LOOK

What is it? It's not enough to talk the talk by saying that software quality is important. You have to (1) explicitly define what is meant when you say “software quality,” (2) create a set of activities that will help ensure that every software engineering work product exhibits high quality, (3) perform quality control and assurance activities on every software project, (4) use metrics to develop strategies for improving your software process and, as a consequence, the quality of the end product.

Who does it? Everyone involved in the software engineering process is responsible for quality.

Why is it important? You can do it right, or you can do it over again. If a software team stresses quality in all software engineering activities, it reduces the amount of rework that it must do. That results in lower costs, and more importantly, improved time to market.

What are the steps? Before software quality assurance (SQA) activities can be initiated,

it is important to define *software quality* at a number of different levels of abstraction. Once you understand what quality is, a software team must identify a set of SQA activities that will filter errors out of work products before they are passed on.

What is the work product? A Software Quality Assurance Plan is created to define a software team's SQA strategy. During modeling and coding, the primary SQA work product is the output of technical reviews (Chapter 20). During testing (Chapters 22 through 26), test plans and procedures are produced. Other work products associated with process improvement may also be generated.

How do I ensure that I've done it right? Find errors before they become defects! That is, work to improve your defect removal efficiency (Chapter 30), thereby reducing the amount of rework that your software team has to perform.

Indeed, quality is a challenging concept—one that we addressed in some detail in Chapter 19.¹

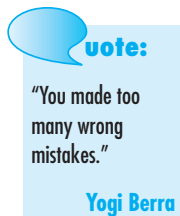
Some software developers continue to believe that software quality is something you begin to worry about after code has been generated. Nothing could be further from the truth! *Software quality assurance* (often called *quality management*) is an umbrella activity (Chapter 3) that is applied throughout the software process.

Software quality assurance (SQA) encompasses: (1) an SQA process, (2) specific quality assurance and quality control tasks (including technical reviews and a multi-tiered testing strategy), (3) effective software engineering practice (methods and tools), (4) control of all software work products and the changes made to them (Chapter 29), (5) a procedure to ensure compliance with software development standards (when applicable), and (6) measurement and reporting mechanisms.

In this chapter, we focus on the management issues and the process-specific activities that enable a software organization to ensure that it does “the right things at the right time in the right way.”

21.1 BACKGROUND ISSUES

Quality control and assurance are essential activities for any business that produces products to be used by others. Prior to the twentieth century, quality control was the sole responsibility of the craftsperson who built a product. As time passed and mass production techniques became commonplace, quality control became an activity performed by people other than the ones who built the 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 [Dem86] as key elements of quality management.

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 [IEEE93a]. Extending the definition presented earlier, software quality assurance is a “planned and systematic pattern of

¹ If you have not read Chapter 19, you should do so now.

actions” [Sch98c] 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? Have software engineering practices 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.

21.2 ELEMENTS OF SOFTWARE QUALITY ASSURANCE

WebRef

An in-depth discussion of SQA, including a wide array of definitions, can be obtained at http://www.swqual.com/images/FoodforThought_Jan2011.pdf.

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 20). 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 22 through 26) 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 29) 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 37) and is a key proponent and sponsor of educational programs.

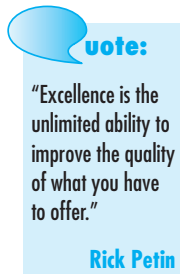
Vendor management. Three categories of software are acquired from external software vendors—*shrink-wrapped packages* (e.g., Microsoft Office), a *tailored shell* [Hor03] that provides a basic skeletal structure that is custom tailored to the needs of a purchaser, and *contracted software* that is custom 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 (Chapter 27).

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 35) 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.





Quality Management Resources

Dozens of quality management resources are 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

Cyber Security and Information Systems Information
Analysis Center (CSIAAC)
<https://sw.thecsiac.com/>

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

ISO SPICE **<http://www.spiceusergroup.org/>**

Malcolm Baldrige National Quality Award
<http://www.nist.gov/baldrige/>

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)
<http://www.isixsigma.com/methodology/total-quality-management-tqm/>
<http://asq.org/learn-about-quality/total-quality-management/overview/overview.html>

21.3 SQA PROCESSES AND PRODUCT CHARACTERISTICS

As we begin a discussion of software quality assurance, it's important to note that SQA procedures and approaches that work in one software environment may not work as well in another. Even within a company that adopts a consistent approach² to software engineering, different software products may exhibit different levels of quality [Par11].

The solution to this dilemma is to understand the specific quality requirements for a software product and then select the process and specific SQA actions and tasks that will be used to meet those requirements. The Software Engineering Institute's CMMI and ISO 9000 standards are the most commonly used process frameworks. Each proposes "a syntax and semantics" [Par11] that will lead to the implementation of software engineering practices that improve product quality. Rather than instantiating either framework in its entirety, a software organization can "harmonize" the two models by selecting elements of both frameworks and matching them to the quality requirements of an individual product.

21.4 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


² For example, CMMI-defined process and practices (Chapter 37).

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.

21.4.1 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 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 as part of project planning and is reviewed by all stakeholders. 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 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.

note:

"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**

In addition to these activities, the SQA group coordinates the control and management of change (Chapter 29) and helps to collect and analyze software metrics.

SafeHome



Software Quality Assurance

The scene: Doug Miller's office as the *SafeHome* software project begins.

The players: Doug Miller (manager of the *SafeHome* software engineering team) and other members of the product software engineering team.

The conversation:

Doug: How are things going with the informal reviews?

Jamie: We're conducting informal reviews of the critical project elements in pairs as we code but before testing. It's going faster than I thought.

Doug: That's good, but I want to have Bridget Thornton's SQA group conduct audits of our work products to ensure that we're following our processes and meeting our quality goals.

Venod: Aren't they already doing the bulk of the testing?

Doug: Yes, they are. But QA is more than testing. We need to be sure that our documents are evolving along with our code and that we're making sure we don't introduce errors as we integrate new components.

Jamie: I really don't want to be evaluated based on their findings.

Doug: No worries. The audits are focused on conformance of our work products to the requirements and process our activities. We'll only be using audit results to try to improve our processes as well as our software products.

Vinod: I have to believe it's going to take more of our time.

Doug: In the long run it will save us time when we find defects earlier. It also costs less to fix defects if they're caught early.

Jamie: That sounds like a good thing then.

Doug: It's also important to identify the activities where defects were introduced and add review tasks to catch them in the future.

Vinod: That'll help us determine if we're sampling carefully enough with our review activities.

Doug: I think SQA activities will make us a better team in the long run.

21.4.2 Goals, Attributes, and Metrics

The SQA activities 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

FIGURE 21.1 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

result. SQA analyzes the allocation of resources for reviews and testing to assess whether they are being allocated in the most effective manner.

Figure 21.1 (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.

21.5 FORMAL APPROACHES TO SQA

In the preceding sections, we 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 and process frameworks. In addition, quality can be defined 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 28) 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 14).

21.6 STATISTICAL SOFTWARE QUALITY ASSURANCE

note:

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

Lowell Arthur



What steps are required to perform statistical SQA?

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

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.

21.6.1 A Generic Example

Note:

"A statistical analysis, properly conducted, is a delicate dissection of uncertainties, a surgery of suppositions."

M. J. Moroney

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. Other 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).

To apply statistical SQA, the table in Figure 21.2 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

FIGURE 21.2

Data collection
for statistical
SQA

Error	Total		Serious		Moderate		Minor	
	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%

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 8) 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 (e.g., [Rya11], [Art97]). In some cases, software organizations 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!*

21.6.2 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:

- *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.



What are the core steps of the Six Sigma methodology?

- 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].

21.7 SOFTWARE RELIABILITY

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.

note:

“The unavoidable price of reliability is simplicity.”

C. A. R. Hoare

KEY POINT

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

21.7.1 Measures of Reliability and Availability

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.



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*,³ respectively.

Many researchers argue that MTBF is a far more useful measure than other quality-related software metrics discussed in Chapter 30. 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 thousands 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.

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 1 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. For a comprehensive discussion of software reliability measures, see [Laz11].

21.7.2 Software Safety

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.



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.

note:

"The safety of the people shall be the highest law."

Cicero

³ 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.

note:

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

E. I. Smith,
captain of the
Titanic

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.⁴ To be effective, software must be analyzed in the context of the entire 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.

WebRef

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

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.

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 [Fir12], [Har12], [Smi05], and [Lev95].

21.8 THE ISO 9000 QUALITY STANDARDS⁵

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

⁴ This approach is similar to the risk analysis methods described in Chapter 35. The primary difference is the emphasis on technology issues rather than project-related topics.

⁵ 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.

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.

The requirements delineated by ISO 9001:2008 address topics such as management responsibility, quality system, contract review, design control, document and data control, product identification and traceability, process control, inspection and testing, corrective and preventive action, control of quality records, internal quality audits, training, servicing, and statistical techniques. In order for a software organization to become registered to ISO 9001:2008, it must establish policies and procedures to address each of the requirements just noted (and others) and then be able to demonstrate that these policies and procedures are being followed. If you desire further information on ISO 9001:2008, see [Coc11], [Hoy09], or [Cia09].

WebRef

Extensive links to ISO 9000/9001 resources can be found at www.tantara.ab.ca/info.htm.



The ISO 9001:2008 Standard

The following outline defines the basic elements of the ISO 9001:2000 standard.

Comprehensive information on the standard can be obtained from the International Organization for Standardization (www.iso.ch) and other Internet sources (e.g., www.praxiom.com).

Establish the elements of a quality management system.

- Develop, implement, and improve the system.
- Define a policy that emphasizes the importance of the system.

Document the quality system.

- Describe the process.
- Produce an operational manual.
- Develop methods for controlling (updating) documents.
- Establish methods for record keeping.

Support quality control and assurance.

- Promote the importance of quality among all stakeholders.
- Focus on customer satisfaction.

Define a quality plan that addresses objectives, responsibilities, and authority.

Define communication mechanisms among stakeholders.

Establish review mechanisms for the quality management system.

- Identify review methods and feedback mechanisms.
- Define follow-up procedures.

Identify quality resources including personnel, training, and infrastructure elements.

Establish control mechanisms.

- For planning.
- For customer requirements.
- For technical activities (e.g., analysis, design, testing).
- For project monitoring and management.

Define methods for remediation.

- Assess quality data and metrics.
- Define approach for continuous process and quality improvement.

INFO

21.9 THE SQA PLAN

The *SQA Plan* provides a road map for instituting software quality assurance. Developed by the SQA group (or by the software team if an SQA group does not exist), the plan serves as a template for SQA activities that are instituted for each software project.

A standard for SQA plans has been published by the IEEE [IEEE93]. The standard recommends a structure that identifies: (1) the purpose and scope of the plan, (2) a description of all software engineering work products (e.g., models, documents, source code) that fall within the purview of SQA, (3) all applicable standards and practices that are applied during the software process, (4) SQA actions and tasks (including reviews and audits) and their placement throughout the software process, (5) the tools and methods that support SQA actions and tasks, (6) software configuration management (Chapter 29) procedures, (7) methods for assembling, safeguarding, and maintaining all SQA-related records, and (8) organizational roles and responsibilities relative to product quality.

SOFTWARE TOOLS



Software Quality Management

Objective: The objective of SQA tools is to assist a project team in assessing and improving the quality of software work product.

Mechanics: Tools mechanics vary. In general, the intent is to assess the quality of a specific work product. Note: A wide array of software testing tools (see Chapters 22 through 26) are often included within the SQA tools category.

Representative Tools:⁶

QA Complete, developed by SmartBear (<http://smartbear.com/products/qa-tools/test-management>), QA management ensures complete

test coverage through every stage of the software development process.

QPR Suite, developed by QPR Software (<http://www.qpr.com>), provides support for Six Sigma and other quality management approaches.

Quality Tools and Templates, developed by iSixSigma (<http://www.isixsigma.com/tools-templates/>), describe a wide array of useful tools and methods for quality management.

NASA Quality Resources, developed by the Goddard Space Flight Center (<http://www.hq.nasa.gov/office/codeq/software/ComplexElectronics/checklists.htm>) provides useful forms, templates, checklists, and tools for SQA.

21.10 SUMMARY

Software quality assurance is a software engineering umbrella activity that is applied at each step in the software process. SQA encompasses procedures for the effective application of methods and tools, oversight of quality control activities such as technical reviews and software testing, procedures for change

⁶ Tools noted here do not represent an endorsement, but rather a sampling of tools in this category. In most cases, tool names are trademarked by their respective developers.

management, procedures for assuring compliance to standards, and measurement and reporting mechanisms.

To properly conduct software quality assurance, data about the software engineering process should be collected, evaluated, and disseminated. Statistical SQA helps to improve the quality of the product and the software process itself. Software reliability models extend measurements, enabling collected defect data to be extrapolated into projected failure rates and reliability predictions.

In summary, you should note the words of Dunn and Ullman [Dun82]: “Software quality assurance is the mapping of the managerial precepts and design disciplines of quality assurance onto the applicable managerial and technological space of software engineering.” The ability to ensure quality is the measure of a mature engineering discipline. When the mapping is successfully accomplished, mature software engineering is the result.

PROBLEMS AND POINTS TO PONDER

- 21.1. Some people say that “variation control is the heart of quality control.” Since every program that is created is different from every other program, what are the variations that we look for and how do we control them?
- 21.2. Is it possible to assess the quality of software if the customer keeps changing what it is supposed to do?
- 21.3. Quality and reliability are related concepts but are fundamentally different in a number of ways. Discuss the differences.
- 21.4. Can a program be correct and still not be reliable? Explain.
- 21.5. Can a program be correct and still not exhibit good quality? Explain.
- 21.6. Why is there often tension between a software engineering group and an independent software quality assurance group? Is this healthy?
- 21.7. You have been given the responsibility for improving the quality of software across your organization. What is the first thing that you should do? What’s next?
- 21.8. Besides counting errors and defects, are there other countable characteristics of software that imply quality? What are they and can they be measured directly?
- 21.9. The MTBF concept for software is open to criticism. Explain why.
- 21.10. Consider two safety-critical systems that are controlled by computer. List at least three hazards for each that can be directly linked to software failures.
- 21.11. Acquire a copy of ISO 9001:2000 and ISO 9000-3. Prepare a presentation that discusses three ISO 9001 requirements and how they apply in a software context.

FURTHER READINGS AND INFORMATION SOURCES

Books by Chemuturi (*Mastering Software Quality Assurance*, J. Ross Publishing, 2010), Hoyle (*Quality Management Essentials*, Butterworth-Heinemann, 2007), Tian (*Software Quality Engineering*, Wiley-IEEE Computer Society Press, 2005), El Emam (*The ROI from Software Quality*, Auerbach, 2005), Horch (*Practical Guide to Software Quality Management*, Artech

House, 2003), and Nance and Arthur (*Managing Software Quality*, Springer, 2002) are excellent management-level presentations on the benefits of formal quality assurance programs for computer software. Books by Deming [Dem86], Defoe and Juran (*Juran's Quality Handbook*, 6th ed., McGraw-Hill, 2010), Juran (*Juran on Quality by Design*, Free Press, 1992), and Crosby ([Cro79] and *Quality Is Still Free*, McGraw-Hill, 1995) do not focus on software, but are must reading for senior managers with software development responsibility. Gluckman and Roome (*Everyday Heroes of the Quality Movement*, Dorset House, 1993) humanizes quality issues by telling the story of the players in the quality process. Kan (*Metrics and Models in Software Quality Engineering*, Addison-Wesley, 1995) presents a quantitative view of software quality.

Work by Evans (*Quality & Performance Excellence*, South-Western College Publishing, 2007) and (*Total Quality: Management, Organization and Strategy*, 4th ed., South-Western College Publishing, 2004), Bru (*Six Sigma for Managers*, McGraw-Hill, 2005), and Dobb (*ISO 9001:2000 Quality Registration Step-by-Step*, 3rd ed., Butterworth-Heinemann, 2004) are representative of the many books written on TQM, Six Sigma, and ISO 9001:2000, respectively.

O'Connor and Kleyner (*Practical Reliability Engineering*, Wiley, 2012), Naik and Tripathy (*Software Testing and Quality Assurance: Theory and Practice*, Wiley-Spektrum, 2008), Pham (*System Software Reliability*, Springer, 2006), Musa (*Software Reliability Engineering: More Reliable Software, Faster Development and Testing*, 2nd ed., McGraw-Hill, 2004) and Peled (*Software Reliability Methods*, Springer, 2001) have written practical guides that describe methods for measuring and analyzing software reliability.

Vincoli (*Basic Guide to System Safety*, Wiley, 2006), Dhillon (*Computer System Reliability, Safety and Usability*, CRC Press, 2013) and (*Engineering Safety*, World Scientific Publishing Co., 2003), Hermann (*Software Safety and Reliability*, Wiley-IEEE Computer Society Press, 2010), Verma, Ajit, and Karanki (*Reliability and Safety Engineering*, Springer, 2010), Storey (*Safety-Critical Computer Systems*, Addison-Wesley, 1996), and Leveson [Lev95] are the most comprehensive discussions of software and system safety published to date. In addition, van der Meulen (*Definitions for Hardware and Software Safety Engineers*, Springer-Verlag, 2000) offers a complete compendium of important concepts and terms for reliability and safety; Gardiner (*Testing Safety-Related Software*, Springer-Verlag, 1999) provides specialized guidance for testing safety critical systems; Friedman and Voas (*Software Assessment: Reliability Safety and Testability*, Wiley, 1995) provide useful models for assessing reliability and safety. Ericson (*Hazard Analysis Primer*, CreateSpace Independent Publishing Platform, 2012) and (*Hazard Analysis Techniques for System Safety*, Wiley, 2005) addresses the increasingly important domain of hazard analysis.

A wide variety of information sources on software quality assurance and related topics is available on the Internet. An up-to-date list of World Wide Web references can be found under "software engineering resources" at the SEPA website www.mhhe.com/pressman.