CHAPTER

1

The Elements of a Complete Software Quality System

Contents

Starting a software quality program from scratch is time consuming and a task often doomed to failure before it is begun. Inadequate preparation, misused terms, lack of planning, and failure to recognize the roles of all individuals in the organization are only a few of the pitfalls waiting for the overanxious practitioner.

As stated in the Introduction, this is a what-to book. It is intended to serve as an introduction to the concepts involved in software quality systems and to suggest how the system's parts may be implemented. This chapter introduces some software quality terms, the basic elements of the software quality system, and a few important additional concerns. The balance of the book elaborates on each of the elements and concerns and discusses implementation of the overall software quality system.

1.1 Definitions

This text uses several terms that are granted many meanings throughout the computing and, in particular, the software industry. In this text, certain of these variably defined terms are used as defined in this section. Where available and appropriate, previously published definitions are used and their sources identified.

Copyright © 2003. Artech House. All rights reserved.

Horch, John. Practical Guide to Software Quality Management, Second Edition, Artech House, 2003. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/think/detail.action?docID=227656.
Created from think on 2025-07-07 18:31:17.

Activity: A task or body of effort directed at the accomplishment of an objective or the production of all or part of a product.

Anomaly: Any deviation from expected results or behavior.

Arithmetic defect: A software flaw in a mathematical computation.

Audit: "An activity to determine through investigation the adequacy of, and adherence to, established procedures, instructions, specifications, codes, and standards or other applicable contractual and licensing requirements, and the effectiveness of implementation" (ANSI N45.2.10-1973).

Client: That person or organization that causes the product to be developed or maintained. The client is often the customer.

Component: A general term for a portion of a product. A component could be a chapter of a document or a unit or module of software. A component may include the entire product.

Consumer: That person or organization that acquires a software product. The consumer may be either the customer or the user.

Control Defect: A software flaw in a decision process.

Customer: That person or organization that pays for the product.

Defect: A flaw in the product resulting from the commission of an error.

Element: *See* Unit.

Entity: Part of the overall company organization (e.g., software quality group, development group).

Error: A mistake made by a person resulting in a defect in the product.

Failure: The experienced manifestation of a defect being encountered in the product.

Fault: *See* Defect.

Guideline: A preferred practice or procedure that is encouraged, but not enforced, throughout the organization.

Input/Output defect: A software flaw in the process of passing information into or out of the software element.

Inspection: "A formal evaluation technique in which software requirements, design, or code is examined in detail by a person or group other than

Copyright © 2003. Artech House. All rights reserved.

the author to detect faults, violations of development standards, and other problems" (IEEE Standard 100–1996).

ISO 9000, et al.: International quality system standards published by the International Organization for Standardization (ISO). Intended to be used as the international definition of quality systems to be applied by producers or suppliers. Certification of an organization to ISO 9001 attests that the organization has a documented quality system and has evidence of its application.

Item: *See* Component.

Module: A group of units that together perform some convenient individual function or subfunction within the software system.

One-on-one review: The most informal examination, by a coworker of the producer, usually of a small portion of a product.

Peer review: A review of a product by peers of the producer. In some literature, the term *peer review* is used to mean any of the informal reviews.

Phase: Any of several convenient divisions of the software life cycle. These may typically include: concept development, requirements, design, coding, test, installation and acceptance, operation and maintenance, and retirement. Phases may or may not be sequential.

Process: The group of activities and procedures by which a producer develops or maintains a product.

Producer: The person or organization that, following a process, develops or maintains a product.

Product(s): The final, or intermediate, output(s) from any given phase of the software life cycle. These usually include specifications, code, test results, and so on.

Program: "A schedule or plan that specifies actions to be taken" (IEEE Standard 100–1992).

Quality: Compliance of a product with the expectations of the user, based on the product's requirements.

Quality assurance: The set of activities intended to detect, document, analyze, and correct process defects and to manage process changes.

Copyright © 2003. Artech House. All rights reserved.

Quality assurance practitioner: A person whose task is to perform one or more of the quality assurance functions or activities comprising the quality system.

Quality control: The set of activities intended to detect, document, analyze, and correct product defects and to manage product changes.

Quality control practitioner: A person whose task is to perform one or more of the quality control functions or activities comprising the quality system.

(Software) quality groups: The organizational entity responsible for monitoring and reporting the performance of the (software) product development functions and activities.

Quality management: The empowering, and encouraging, of the producer to identify and submit improvements to the product development process.

Quality practitioner: A person whose task is to perform one or more of the functions or activities comprising the quality system. The quality practitioner may or may not be assigned to a (software) quality group. This includes both quality assurance and quality control practitioners.

(Software) quality systems: The total set of quality control, quality assurance, and quality management activities dedicated to the provision of quality products.

Requirement: "A condition of capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents" (IEEE Standard 100–1996).

Review: A formal or informal meeting at which an output (product or component) of the software development life cycle is presented to the customer, user, or other interested parties for examination, evaluation, and approval.

SEI CMM: A five-level model of an organization's software process maturity, called the Capability Maturity Model (CMM), developed by the Software Engineering Institute (SEI).

Software: Computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.

Software development life cycle: The portion of the software life cycle devoted to the actual creation of the software system, generally

Copyright © 2003. Artech House. All rights reserved.

beginning with the requirements generation and ending with the installation of the software system into active production.

Software life cycle: The entire period during which a software system is active, beginning with its initial conceptual development and ending with its removal from active use and its archiving.

Software system: A total, integrated aggregation of software components that performs the set of specific functions as defined by its approved requirements.

Standard: A practice or procedure that is imposed and enforced throughout the organization.

Subsystem: A group of modules that together perform one of the major functions of the software system.

Supplier: The person or organization that provides the product.

Total quality: The culture that maximizes the likelihood that a product conforms to its requirements on an ongoing basis.

Total quality system: The set of activities required to provide decision-making, action-capable management with the information it needs to affect the product development process beneficially.

Unit: "A software component that is not subdivided into other components" (IEEE Standard 610.12–1990). This is also known as the "smallest replaceable component" and sometimes called an element.

Unit development folder: The "diary" of the development of a software component. It usually contains the portion of the approved requirements being addressed by the component, the design and test information that applies, and any additional information applicable to the understanding of the development approach used for the component.

User: That person who actually performs his or her job functions with the assistance of the product.

Vendor: A person or organization that sells part or all of a product, usually for inclusion in a larger product being developed by a producer.

Walk-through: A review method in which a producer leads one or more other members of the development team through a product, or portion thereof, that he or she has developed, while the other members ask questions and make comments about technique, style, possible errors, violations of development standards, and other problems.

Copyright © 2003. Artech House. All rights reserved.

1.2 The elements of a software quality system

There are two goals of the software quality system (SQS). The first goal is to build quality into the software from the beginning. This means assuring that the problem or need to be addressed is clearly and accurately stated, and that the requirements for the solution are properly defined, expressed, and understood. Nearly all the elements of the SQS are oriented toward requirements validity and satisfaction.

In order for quality to be built into the software system from its inception, the software requirements must be clearly understood and documented. Until the actual requirements, and the needs of the user that they fulfill, are known and understood, there is little likelihood that the user will be satisfied with the software system that is delivered. Whether we know them all before we start, or learn some of them as we go, all requirements must be known and satisfied before we are through. Further discussion of requirements is provided in Chapters 6 and 10.

The second goal of the SQS is to keep that quality in the software throughout the software life cycle (SLC). In this chapter, the 10 elements of the SQS are introduced and their contributions to the two goals indicated.

The 10 elements of the SQS are as follows:

1. Standards;
2. Reviewing;
3. Testing;
4. Defect analysis;
5. Configuration management (CM);
6. Security;
7. Education;
8. Vendor management;
9. Safety;
10. Risk management.

While each element can be shown to contribute to both goals, there are heavier relationships between some elements and one or the other of the two goals. These particular relationships will become obvious as each element is discussed in the chapters that follow.

Every SLC model has divisions, or periods of effort, into which the work of developing and using the software is divided. These divisions or periods are given various names depending on the particular life-cycle paradigm

Copyright © 2003. Artech House. All rights reserved.

being applied. For this discussion, the following periods of effort, together with their common names, are defined:

- Recognition of a need or problem (e.g., concept definition);
- Definition of the software solution to be applied (e.g., requirements definition);
- Development of the software that solves the problem or satisfies the need (e.g., design and coding);
- Proving that the solution is correct (e.g., testing);
- Implementing the solution (e.g., installation and acceptance);
- Using the solution (e.g., operation);
- Improving the solution (e.g., maintenance);

Regardless of their names, each division represents a period of effort directed at a particular part of the overall life cycle. They may be of various lengths and be applied in various sequences, but they will all exist in successful projects. Some newer life-cycle models or approaches for small to medium-sized projects (e.g., extreme programming, the rational unified process, and agile development) appear to circumvent some of these efforts. In reality, they merely reorder, reiterate, or resequence them.

There are also associations between certain elements and the various divisions or periods of the SLC. Again, most of the elements support most of the SLC, but certain elements are more closely associated with particular periods than with others.

Figure 1.1 displays the 10 elements as a cube supporting the goals of software quality and the periods of the SLC with which each element is most closely associated.

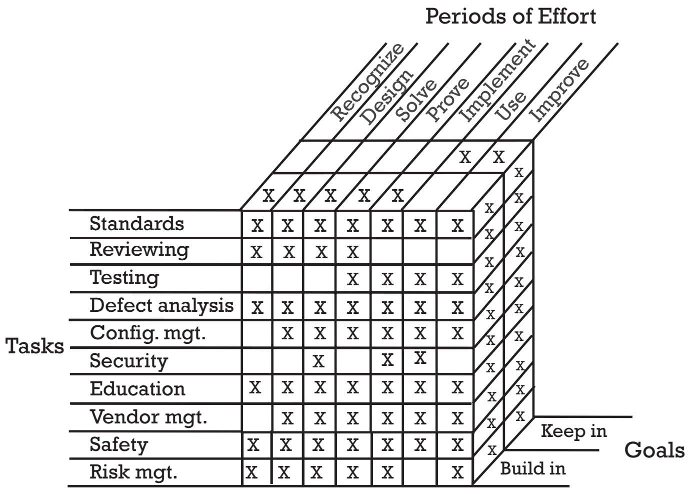
1.2.1 Standards

Software is becoming a science. The old days of free-form creativity in the development of software are gradually giving way to more controlled and scientific approaches. As some writers have said, "Software is moving from an arcane art to a visible science."

As implied by Figure 1.2, the standards manual can have inputs from many sources. Standards are intended to provide consistent, rigorous, uniform, and enforceable methods for software development and operation activities. The development of standards, whether by professional societies such as the Institute of Electrical and Electronics Engineers (IEEE), international groups such as International Organization for Standardization/

Copyright © 2003. Artech House. All rights reserved.

Horch, John. Practical Guide to Software Quality Management, Second Edition, Artech House, 2003. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/think/detail.action?docID=227656.
Created from think on 2025-07-07 18:31:17.



Tasks	Recognize	Design	Solve	Prove	Implement	Use	Improve
Standards	X	X	X	X	X	X	X
Reviewing	X	X	X	X			
Testing				X	X	X	X
Defect analysis	X	X	X	X	X	X	X
Config. mgt.		X	X	X	X	X	X
Security			X		X	X	
Education	X	X	X	X	X	X	X
Vendor mgt.		X	X	X	X	X	X
Safety	X	X	X	X	X	X	X
Risk mgt.	X	X	X	X	X		X

Figure 1.1 Quality tasks, life-cycle periods, and goals.

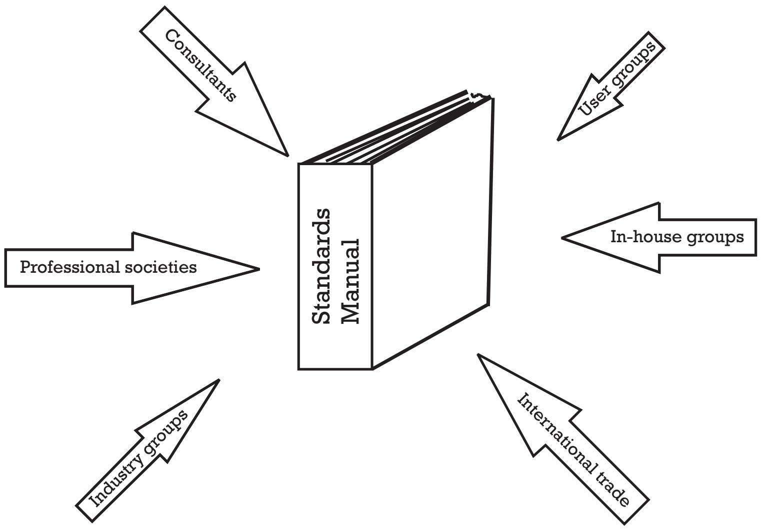


Figure 1.2 Standards sources.

International Electrotechnical Commission Joint Technical Committee One (ISO/IEC JTC1), industry groups, or software development organizations for themselves, is recognizing and furthering that movement.

Standards cover all aspects of the SLC, including the very definition of the SLC itself. More, probably, than any of the other elements, standards

Copyright © 2003. Artech House. All rights reserved.

can govern every phase of the life cycle. Standards can describe considerations to be covered during the concept exploration phase. They can also specify the format of the final report describing the retirement of a software system that is no longer in use.

Standards come into being for many reasons. They might document experience gained in the day-to-day running of a computer center, and the most efficient methods to be used. Laws and government regulations often impose standard procedures on business and industry. Industries can band together to standardize interfaces between their products such as in the communications areas. Contracts often specify standard methods of performance. And, in many cases, standards arise out of good common sense.

Whether a standard comes from within a company, is imposed by government, or is adopted from an industry source, it must have several characteristics. These include the following:

- *Necessity.* No standard will be observed for long if there is no real reason for its existence.
- *Feasibility.* Common sense tells us that if it is not possible to comply with the tenets of a standard, then it will be ignored.
- *Measurability.* It must be possible to demonstrate that the standard is being followed.

Each of these characteristics supports the total enforceability of the standard. An unenforceable standard is of no use to anyone.

Software standards should be imposed so that the producer of a software product or component can pay attention to the technical aspects of the task, rather than to the routine aspects that may be the same for every task. Standards, such as those for document formats, permit the producer to concentrate on technical issues and content rather than format or layout details.

Standards, while worthwhile, are less than fully effective if they are not supported by policies that clearly indicate their imposition. It should be the intent of responsible management to see that they are followed and enforced. Specific practices for standard implementation are often useful. In this way, adherence to the standard may be more uniform.

Lastly, not everything must be standardized. Guidelines that call out the preferred methods or approaches to many things are fully adequate. A set of standards that covers every minute aspect of an organization's activity can lose respect simply from its own magnitude. Competent and comprehensive guidelines give each person some degree of freedom in those areas where specific methods or approaches are not absolutely necessary. This leaves the

Copyright © 2003. Artech House. All rights reserved.

standards to govern those areas where a single, particular way of doing business is required.

1.2.2 Reviewing

Reviews permit ongoing visibility into the software development and installation activities.

Product reviews, also called technical reviews, are formal or informal examinations of products and components throughout the development phases of the life cycle. They are conducted throughout the software development life cycle (SDLC). Informal reviews generally occur during SDLC phases, while formal reviews usually mark the ends of the phases. Figure 1.3 illustrates this point.

Informal reviews include walk-throughs and inspections. Walk-throughs are informal, but scheduled, reviews, usually conducted in and by peer groups. The author of the subject component—a design specification, test procedure, coded unit, or the like—walks through his or her component, explaining it to a small group of peers. The role of the peers is to look for defects in or problems with the component. These are then corrected before the component becomes the basis for further development.

Inspections are a more structured type of walk-through. Though the basic goal of an inspection—removal of defects—is the same as that of the

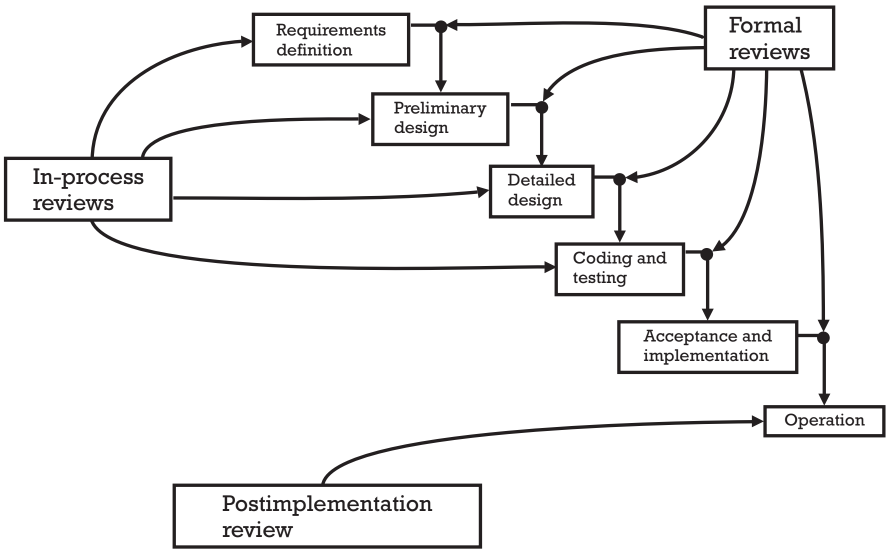


Figure 1.3 SDLC reviews.

Copyright © 2003. Artech House. All rights reserved.

walk-through, the format of the meeting and the roles of the participants are more strictly defined, and more formal records of the proceedings are prepared.

Process reviews may be held at any time. The purpose of a process review is to examine the success of the software process in effect. Data for the review is collected in the technical reviews and is usually based on defects identified by the technical reviews. Opportunities for improvements to the current process are sought. Management reviews are specialized process reviews, done on behalf of senior management, to examine project status and effective use of resources based on the current process.

Also included within the quality control activity of reviewing are audits. Audits are examinations of components for compliance with a content and format specification or for consistency with or comparison to a predecessor. An in-process audit of the unit development folder (UDF)—also called the software development file in some organizations—is usually informal. It compares the content and status of the UDF against standards governing the preparation and maintenance of the UDF. Its goal is to ascertain the UDFs are being used as required.

The physical audit (PA), often included as a part of the CM process, is an example of a formal audit. It compares the final form of the code against the final documentation for that code. The goal of the physical audit is to assure that the two products, documentation and code, are in agreement before being released to the user or customer. Another formal audit is the functional audit. The functional audit (FA), again often a CM responsibility, compares the test results with the currently approved requirements to assure that all requirements have been satisfied.

1.2.3 Testing

Tests provide increasing confidence and, ultimately, a demonstration that the software requirements are being satisfied. Test activities include planning, design, execution, and reporting. Figure 1.4 presents a simple conceptual view of the testing process. The basic test process is the same, whether it is applied to system testing or to the earliest module testing.

Test planning begins during the requirements phase and parallels the requirements development. As each requirement is generated, the corresponding method of test for that requirement should be a consideration. A requirement is faulty if it is not testable. By starting test planning with the requirements, nontestability is often avoided. In the same manner that requirements evolve and change throughout the software development, so,

Copyright © 2003. Artech House. All rights reserved.

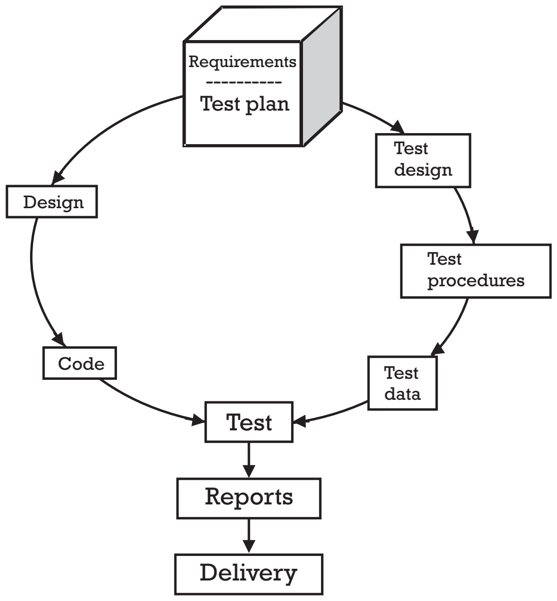


Figure 1.4 Simplified test process.

too, do the test plans evolve and change. This emphasizes the need for early, and continuing, CM of the requirements and test plans.

Test design begins as the software design begins. Here, as before, a parallel effort with the software development is appropriate. As the design of the software takes form, the test cases, scenarios, and data are developed that will exercise the designed software. Each test case also will include specific expected results so that a pass-fail criterion is established. As each requirement must be measurable and testable, so must each test be measurable. A test whose completion is not definitive tells little about the subject of the test. Expected results give the basis against which the success or failure of the test is measured.

Actual testing begins with the debugging and early unit and module tests conducted by the programmer. These tests are usually informally documented (perhaps by notations in the UDF) and are not closely monitored by the software quality practitioner since they are frequently experimental and meant to help the programmer in his or her day-to-day software generation. Formal test execution generally begins with integration tests in which modules are combined into subsystems for functional testing. In larger systems, it is frequently advisable to begin formal testing at the module level after the

Copyright © 2003. Artech House. All rights reserved.

programmer has completed his or her testing and is satisfied that the module is ready for formal testing.

An important aspect of complete testing is user acceptance testing. In these, usually last, tests, the concentration is on actual, delivered functionality and usability in the user's environment.

Test execution requires the use of detailed test procedures. These are step-by-step directions that tell the test conductor exactly what to do as the test is run. Every action, input, expected output, and response should be documented so that the test conductor is not put into the position of making test design decisions while the test is being run. Preparation of the test procedures is begun during the design phase and completed during the coding and debugging activities. By the time the coding phase is complete, all preparations for the formal testing activities should also be in place. Test cases, scenarios, data, and procedures, together with expected results and completion criteria, should be ready to be applied from module testing (if included on the particular project) through qualification and acceptance tests.

Test reports document the actual results of the testing effort as it progresses. For each test that is run, a report of the expected results, actual results, and the conclusions of the test conductor concerning success of the test should be prepared. Included in the report are the anomalies that were found and recommended action with respect to them. Errors, defects, faults, questionable or unexpected results, and any other nonpredicted outcomes are recorded and assigned for action. Once the anomaly has been addressed, the test, or an appropriate portion thereof, will be rerun to show that the defect has been corrected. As the tests progress, so do the levels of detail of the test reports, until the final acceptance test report is prepared that documents the fitness of the software system for use in its intended environment.

1.2.4 Defect analysis

Defect analysis is the combination of defect detection and correction, and defect trend analysis. Defect detection and correction, together with change control, presents a record of all discrepancies found in each software component. It also records the disposition of each discrepancy, perhaps in the form of a software problem report or software change request.

As shown in Figure 1.5, each needed modification to a software component, whether found through a walk-through, review, test, audit, operation, or other means is reported, corrected, and formally closed. A problem or requested change may be submitted by anyone with an interest in the

Copyright © 2003. Artech House. All rights reserved.

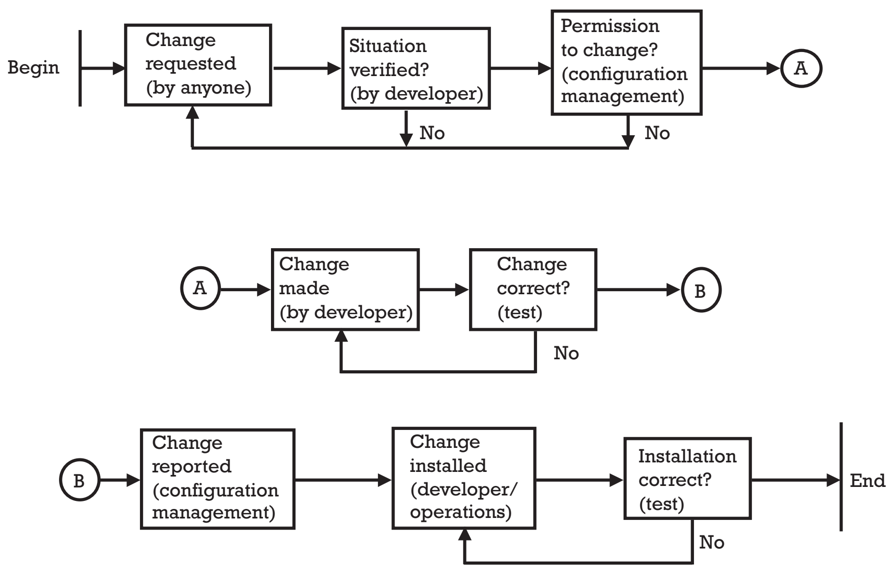

Begin
Change requested (by anyone)
Situation verified? (by developer)
Permission to change? (configuration management)
A
No
No
A
Change made (by developer)
Change correct? (test)
B
No
B
Change reported (configuration management)
Change installed (developer/ operations)
Installation correct? (test)
End
No

Figure 1.5 Typical change procedure.

software. The situation will be verified by the developers, and the CM activity will agree to the change. Verification of the situation is to assure that the problem or need for the change actually exists. CM may wish to withhold permission for the change or delay it until a later time; perhaps because of concerns such as interference with other software, schedule and budget considerations, the customer's desires, and so on. Once the change is completed and tested, it will be reported by CM to all concerned parties, installed into the operational software by the developers or operations staff, and tested for functionality and compatibility in the full environment.

This procedure is required for the ongoing project to make sure that all defects found are properly fixed and closed. It also serves future projects by providing a means for feeding defect information back into the development life cycle and modifying the software development process so that future occurrences of certain defects are reduced. Figure 1.6 places the change procedure into the larger picture of development process analysis and improvement.

A running record of defects, their solutions, and status is provided by the defect trend analysis effort. (The actual changes are made according to the configuration control process.) As previously mentioned, the record of defects and their solutions can serve to do the following:

Copyright © 2003. Artech House. All rights reserved.

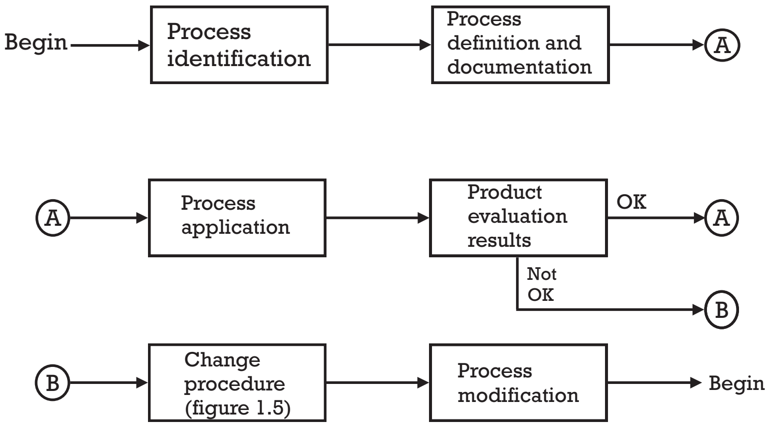

Begin
Process identification
Process definition and documentation
A
A
Process application
Product evaluation results
OK
A
Not OK
B
B
Change procedure (figure 1.5)
Process modification
Begin

Figure 1.6 Development process improvement.

- Prevent defects from remaining unsolved for inappropriate lengths of time;
- Prevent unwarranted changes;
- Point out inherently weak areas in the software;
- Provide analysis data for development process evaluation and correction;
- Provide warnings of potential defects through analysis of defect trends.

Formal recording and closure procedures applied to defects are insufficient if corresponding reports are not generated so that project management has visibility into the progress and status of the project. Regular reports of defect detection and correction activity keep management apprised of current defect areas and can warn of potential future trouble spots. Further, analysis of ongoing defect and change reports and activities provide valuable insight into the software development process and enhance the software quality practitioner's ability to suggest error-avoidance and software development process modification.

1.2.5 Configuration management

CM is a three-fold discipline. Its intent is to maintain control of the software, both during development and after it is put into use and changes begin.

Copyright © 2003. Artech House. All rights reserved.

As shown in Figure 1.7, CM is, in fact, three related activities: identification, control, and accounting. If the physical and functional audits are included as CM responsibilities, there are four activities. Each of the activities has a distinct role to play. As system size grows, so does the scope and importance of each of the activities. In very small, or one-time use, systems, CM may be minimal. As systems grow and become more complex, or as changes to the system become more important, each activity takes on a more definite role in the overall management of the software and its integrity. Further, some CM may be informal for the organization itself, to keep track of how the development is proceeding and to maintain control of changes, while others will be more formal and be reported to the customer or user.

Configuration identification is, as its name implies, the naming, and documentation, of each component (document, unit, module, subsystem, and system) so that at any given time, the particular component of interest can be uniquely identified. This is important when documenting, testing, changing, or delivering software to the customer; in other words, throughout the entire SLC. Unless it is known which specific version or component of the software is being affected, (i.e., coded, changed, tested) the software is out of control. Tests may be run on the wrong version of the code, changes may be made to an obsolete version of a document, or a system composed of the wrong versions of the various components may be delivered to the user or customer.

Configuration control is that activity that prevents unauthorized changes to any software product. Early in the SLC, documentation is the primary product. Configuration control takes on an increasingly formal role as the documents move from draft to final form. Once published, any changes to the documents are formally processed so that capricious, unnecessary, or unapproved changes are not made. As the life cycle moves into the coding, testing, and operation and maintenance phases, changes to either

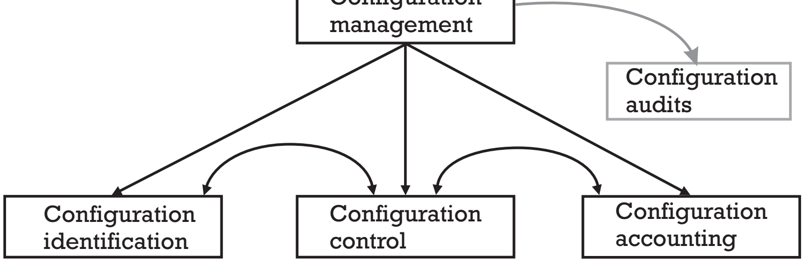


Figure 1.7 CM activities.

Copyright © 2003. Artech House. All rights reserved.

documents or code are closely controlled. Each change is verified for necessity and correctness before approval for insertion, so that control of the software can be maintained.

Configuration accounting keeps track of the status of each component. The latest version or update of each software component is recorded. Thus, when changes or other activities are necessary with respect to the component, the correct version of the component can be located and used. Each new edition of a document, each new assembly or compilation of the code, each new build of the software system is given a new specific identifier (through configuration identification) and recorded. All changes to that version or edition of a component are also referenced to it so that, if necessary, the history of activity with respect to any component can be recreated. This might be necessary in the loss of the current version or to return to a previous version for analysis or other purposes. Ultimately, the deliverable version, or build, of the software is created. Configuration accounting helps manage the builds and make the build process repeatable.

One last point should be made. That is, for very long term or long-lived systems, the development environment itself may need to be configuration managed. As operating systems, platforms, languages, and processes evolve over time, it may not be possible to recreate a given system without the original development environment and its components.

1.2.6 Security

Security activities are applied both to data and to the physical data center itself. These activities are intended to protect the usefulness of the software and its environment.

The highest quality software system is of no use if the data center in which it is to be used is damaged or destroyed. Such events as broken water pipes, fire, malicious damage by a disgruntled employee, and storm damage are among the most common causes of data center inoperability. Even more ominous is the rising incidence of terrorist attacks on certain industries and in various countries, including our own, around the world.

Another frequent damager of the quality of output of an otherwise high-quality software system is data that has been unknowingly modified. If the data on which the system is operating has been made inaccurate, whether intentionally or by accident, the results of the software will not be correct. To the user or customer, this appears to be inadequate software.

Additionally, though not really a software quality issue per se, is the question of theft of data. The security of stored or transmitted data is of paramount concern in most organizations. From the theft of millions of

Copyright © 2003. Artech House. All rights reserved.

dollars by interception of electronic funds transfers to an employee who just changes personnel or payroll records, data security is a major concern.

Finally, the recent onslaught of hackers and software attackers and the burgeoning occurrences of viruses also need to be considered. These threats to software quality must be recognized and countered.

The role of the software quality practitioner is, again, not to be the policeperson of the data or to provide the data or data center security. The software quality practitioner is responsible for alerting management to the absence, or apparent inadequacy, of security provisions in the software. In addition, the software quality practitioner must raise the issue of data center security and disaster recovery to management's attention.

1.2.7 Education

Education assures that the people involved with software development, and those people using the software once it is developed, are able to do their jobs correctly.

It is important to the quality of the software that the producers be educated in the use of the various development tools at his or her disposal. A programmer charged with writing object-oriented software in C++ cannot perform well if the only language he or she knows is Visual Basic. It is necessary that the programmer be taught to use C++ before beginning the programming assignment. Likewise, the use of operating systems, data modeling techniques, debugging tools, special workstations, and test tools must be taught before they can be applied beneficially.

The proper use of the software once it has been developed and put into operation is another area requiring education. It this case, the actual software user must be taught proper operating procedures, data entry, report generation, and whatever else is involved in the effective use of the software system's capabilities.

The data center personnel must be taught the proper operating procedures before the system is put into full operation. Loading and initializing a large system may not be a trivial task. Procedures for recovering from abnormal situations may be the responsibility of data center personnel. Each of the many facets of operating a software system must be clear so that the quality software system that has been developed may continue to provide quality results.

The software quality practitioner is not usually the trainer or educator. These functions are normally filled by some other group or means. The role of the software quality practitioner is, as always, to keep management attention focused on the needs surrounding the development and use of a quality

Copyright © 2003. Artech House. All rights reserved.

Horch, John. Practical Guide to Software Quality Management, Second Edition, Artech House, 2003. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/think/detail.action?docID=227656.
Created from think on 2025-07-07 18:31:17.

software system. In this case, the software quality practitioner is expected to monitor the requirements for, and the provision of, the education of the personnel involved in the SLC.

Lastly, the support personnel surrounding software development must know their jobs. The educators, CM and software quality practitioners, security and database administrators, and so on must be competent to maintain an environment in which quality software can be built, used, and maintained.

1.2.8 Vendor management

When software is purchased, the buyer must be aware of, and take action to gain confidence in, its quality. Not all purchased software can be treated in the same way, as will be demonstrated here. Each type of purchased software will have its own software quality system approach, and each must be handled in a manner appropriate to the degree of control the purchaser has over the development process used by producer. The following are three basic types of purchased software:

1. Off-the-shelf;
2. Tailored shell;
3. Contracted.

Off-the-shelf software is the package we buy at the store. Microsoft Office, Adobe Photoshop, virus checkers, and the like are examples. These packages come as they are with no warrantee that they will do what you need to have done. They are also almost totally outside the buyer's influence with respect to quality.

The second category may be called the tailored shell. In this case, a basic, existing framework is purchased and the vendor then adds specific capabilities as required by the contract. This is somewhat like buying a stripped version of a new car and then having the dealer add a stereo, sunroof, and other extras. The only real quality influence is over the custom-tailored portions.

The third category is contracted software. This is software that is contractually specified and provided by a third-party developer. In this case, the contract can also specify the software quality activities that the vendor must perform and which the buyer will audit. The software quality practitioner has the responsibility in each case to determine the optimum level of influence to be applied, and how that influence can be most effectively applied.

Copyright © 2003. Artech House. All rights reserved.

The purchaser's quality practitioners must work closely with the vendor's quality practitioners to assure that all required steps are being taken.

Attention to vendor quality practices becomes extremely important when the developer is off-shore or remote, such as in India, Nepal, or another country.

1.2.9 Safety

As computers and software grow in importance and impact more and more of our lives, the safety of the devices becomes a major concern. The literature records overdoses of medicines, lethal doses of radiation, space flights gone astray, and other catastrophic and near-catastrophic events. Every software project must consciously consider the safety implications of the software and the system of which it is a part. The project management plan should include a paragraph describing the safety issues to be considered. If appropriate, a software safety plan should be prepared.

1.2.10 Risk management

There are several types of risk associated with any software project. Risks range from the simple, such as the availability of trained personnel to undertake the project, to more threatening, such as improper implementation of complicated algorithms, to the deadly, such as failure to detect an alarm in a nuclear plant. Risk management includes identification of the risk; determining the probability, cost, or threat of the risk; and taking action to eliminate, reduce, or accept the risk. Risk and its treatment is a necessary topic in the project plan and may deserve its own risk management plan.

Copyright © 2003. Artech House. All rights reserved.

1.3 Additional issues

Several other issues can, and will, impact the scope and authority of the software quality system. These include maintaining the software once it is in operation, documenting the software's development and configuration, deciding where to place the software quality practitioners within the overall organization, and the concerns of implementing the quality system.

1.3.1 Maintenance

Software maintenance can best be viewed and treated as an extension or repetition of the development process.

Software maintenance includes two primary activities: correction of defects that were not found during development and testing, and enhancement of the software to meet new or changed requirements after installation. As suggested in Figure 1.8, each maintenance action or project is treated in much the same way as original development, and the parallels with the SDLC can be seen. The maintenance process begins with identifying the need for a change. The occurrence of a failure due to a previously unencountered defect will trigger a change. New requirements may come from user requests; the need for increased throughput in the data center; a change in processing technology, such as moving from mainframes to a client-server approach; or just the desire to reengineer old legacy code.

Whatever the reason for the change, effort is expended to determine exactly what will be needed (concept definition and requirements specification); how the change will be affected (design); the actual creation of the new or modified software (code and unit test); and the testing, approval, and installation of the change (integration, testing, and installation). Thus, in almost all cases (there are exceptions to most rules), maintenance can be seen primarily as a return to the regular SDLC activities, though usually on a smaller scale.

It is important to note the need for rigorous CM during the maintenance phase. Especially during periods of rapid change, such as might be found during the modification of software to address new government regulations, or the introduction of a new weapon system in a combat vehicle, there is

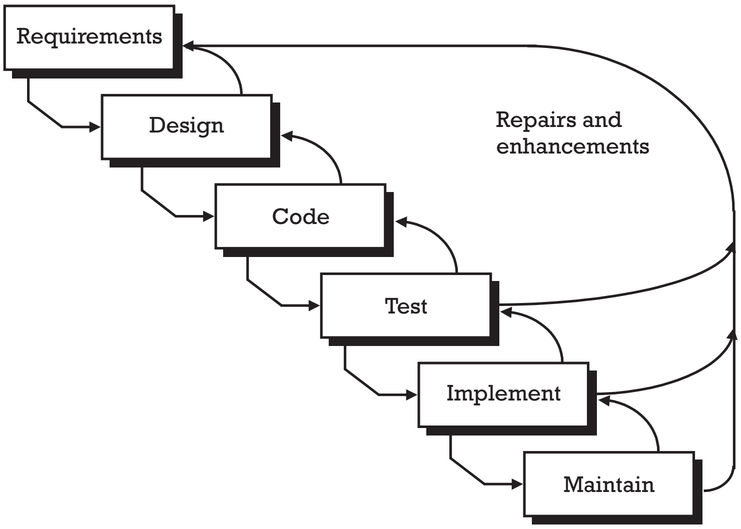


Figure 1.8 Recycling the life cycle.

Copyright © 2003. Artech House. All rights reserved.

significant danger of making changes to the wrong version of a module or subsystem. If multiple changes are being made simultaneously, as is often the case, one change may unknowingly affect another. In today's world of the Internet, World Wide Web, and e-commerce, changes are not only necessarily very rapid but also very visible. Errors may be seen or experienced by hundreds or even thousands of on-line users in the space of a short time.

CM must be part and parcel of changes in this environment, and the software quality practitioner must take an aggressive role in confirming that all CM procedures are being followed. This is in addition to the software quality practitioner's regular monitoring role in all software development activities, whether original development or maintenance.

1.3.2 Documentation

The purpose of documentation is to record what is required of the software, how it is to be accomplished, proof that it was provided, and how to use and maintain it. The role of the software quality practitioner is to monitor the documentation activities and keep management apprised of their status and quality.

It is like the old adage, if you don't know where you're going, any road will take you there (but it doesn't matter, because you won't realize that you've arrived, anyway). Without adequate documentation, the task at hand is never accurately specified. What is really wanted is not made clear. The starting and ending points are poorly specified, and no one is sure when the project is complete. Inadequate documentation is like not knowing where you are going. The system designers are not sure what the customer or user really wants, the programmer is not sure what the designer intends, and the tester is not sure what to look for. Finally, the customer or user is not sure that they got what they wanted in the first place.

The depth of the documentation depends on the scope of the specific project. Small projects can be successful with reduced documentation requirements. But, as the size of the project increases, the need for more complete documentation also increases. In the case of small or uncomplicated projects, the information contained in some documents can be provided in higher-level documents. As system size increases, additional documents may be needed to adequately cover such topics as interfaces and data design. More comprehensive test documentation will also be required such as specific test plans, cases, and reports.

Too much documentation can be as bad as too little. Overdocumentation can induce a user to say, "I'm not going to read all that!" The time spent documenting a project is wasted if the documentation does not add to the

Copyright © 2003. Artech House. All rights reserved.

required body of knowledge about the project. Overdocumentation can introduce inconsistencies, conflicting information, and other kinds of defects that detract from performance in the long run.

Documentation should be sufficient to accurately and completely tell what to do (concept and requirements), how to do it (plan and design), how to show that it was done (test), and how to use the system (user). The software quality practitioner monitors and reviews the documentation to see that it satisfies this need.

1.3.3 Organizational considerations

The placement of the software quality practitioner or group within the organization is a critical factor in its effectiveness.

While there are several acceptable structures, each dependent on the specific total business organization, there are certain conditions that must be observed to enable the SQS to be effective. Figure 1.9 depicts several possible organizational reporting arrangements. Each has its merits and faults, which will be explored in Chapter 11.

It is important to note that, in some companies, the SQS functions and activities may not be under the auspices of a formal software quality group at all. Remembering that the SQS functions are to be carried out by those parts of the organization best qualified to perform them, there are some companies that stop at that point and have various managers responsible for

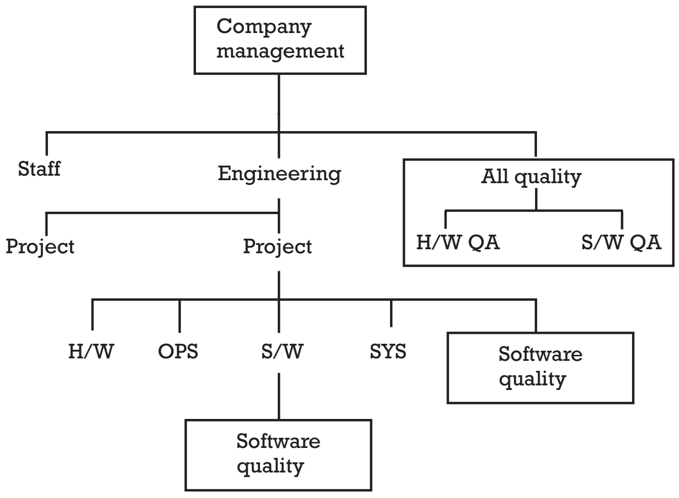


Figure 1.9 Traditional organizational styles.

Copyright © 2003. Artech House. All rights reserved.

Horch, John. Practical Guide to Software Quality Management, Second Edition, Artech House, 2003. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/think/detail.action?docID=227656.
Created from think on 2025-07-07 18:31:17.

individual SQS functions. This approach would seem to have some economies connected with it, since there is not the cost of a dedicated staff just for software quality. However, the coordination among the various responsible managers may, in fact, be time-consuming enough to actually cost more than a software quality group. In addition, when a manager has an assignment such as the development of a new software system, and some ancillary tasks such as documentation coordination, CM, training, security, or software quality, the development task usually gets the bulk of the manager's attention, and other tasks may be given less attention or effort.

Software quality is, as has been said frequently, everyone's individual responsibility. Each participant in the SLC is expected to do his or her job correctly. Unfortunately, this is often an unachieved goal. Software quality tasks, then, must be assigned to the group or individual who can, and will, be accountable for the assessment of, and reporting on, the quality of the software throughout its life cycle.

1.3.4 Implementation of the total software quality system

Implementation of a software quality system requires a delegation of authority (a charter to perform the activities), cooperation of the organization (which is usually gained through demonstration of usefulness over time), and order (a logical progression of steps leading to the actual application and performance of the SQS activities).

No activity should be started until a formal charter of responsibilities, accountabilities, and authority vested in the SQS is created and assigned to the software quality practitioner or group by management. This, however, only creates the SQS. It does not establish the set of functions and activities that must be performed, nor the order in which they will be inaugurated. The software quality practitioners themselves must plan, design, and implement the overall SQS.

The four major elements in a successful software quality system are the quality culture of "do it right the first time," a quality charter that specifies the responsibilities and authorities of each person with respect to quality, a software quality manual that details the various components of the organization's SQS, and the SQS standards and procedures themselves. Table 1.1 shows how the various affected parts of the organization must contribute to the elements for the institution of an effective and acceptable SQS.

One very important aspect of the whole process is the continued involvement of the development group from the very beginning. As each part of the SQS is conceived and planned, the quality charter established, the quality manual prepared, and the SQS implemented, the involvement of

Copyright © 2003. Artech House. All rights reserved.

Table 1.1 Key Software Quality Roles

Senior Management	SQS Program Element	Technical Personnel
Insist upon	Culture	Input to
Commit to	Charter	Input to
Input to	Manual	Input to
Fully support	Total SQS	Cooperate with

the producers will help to assure their acceptance and cooperation. Their participation, from the beginning, reduces the elements of surprise and, sometimes, distrust on the part of those whose work is the subject of the software quality activities.

Management, too, must be kept fully apprised of the activities and progress of the implementation of the SQS. They have provided the initial impetus for the SQS with their insistence on the concept of an organizational culture based on quality. Next, they have started the process through their demonstrated commitment to the quality charter. Continued support for the effort depends on their continued belief that an SQS will be beneficial in the long run. By maintaining close contact with management during the startup period, potential future pitfalls can be recognized and avoided.

Finally, a logical implementation plan must be worked out and accepted by all affected groups and by management. The needs of the various groups and their priorities must be reflected in the actual implementation schedule.

In the final analysis, the startup of the SQS closely resembles the creation of a software system. Each part of the SDLC is paralleled and each must be carefully addressed. Most of all, however, all affected organizations should be a party to the planning, design, and implementation of the SQS.

1.4 Summary

A total software quality system is more than reviews, testing, or standards. It is the comprehensive application of a 10-element discipline. The role of the software quality function is to review the state of the software development process and its products and report that state to decision-making, action-capable management. It is not the role of the software quality function to manage, direct, or control the software development process.

The ultimate objective of the software quality system is to provide, based on the results of the 10 elements, information that will permit decision-

Copyright © 2003. Artech House. All rights reserved.

making, action-capable management to beneficially affect the software development process.

While it is not an absolute necessity that the SQS functions be under the cognizance of a software quality organization, the accountability for the SQS functions becomes more visible and addressable if a software quality group does, in fact, exist. This group is not necessarily responsible for the actual performance of the SQS functions, but rather is responsible to alert management to the need for, and the efficacy of, those functions. The functions themselves are to be performed by the organizational entities most qualified to perform them (e.g., training by the training department, CM by the CM department, and so forth).

The software quality practitioner must be administratively and financially independent of the parts of the organization that it will monitor. This means at least on the same organizational level within a project or in a matrix management situation in which the SQS is administered by an organizational element completely outside the project organization.

In order that the software quality practitioners have the authority commensurate with their responsibilities and accountabilities, there should be a written charter from senior management that specifies the roles, objectives, and authority of the SQS and software quality practitioners. The preparation and approval of the charter will serve to get the commitment of senior management to whatever software quality system is finally implemented. Senior management commitment is a key requisite to the success, both near and long term, of the SQS. An SQS, and the software quality practitioners who execute it, are at high risk from political and financial variations within the organization without the formal commitment of senior management.

1.5 The next step

To delve deeper into the topic of software quality management, the following two texts might be of interest:

Shari Lawrence Pfleeger et al. *Solid Software,* Upper Saddle River, NJ: Prentice Hall, 2002.

Roger S. Pressman. *Software Engineering: A Practitioner's Approach,* New York: McGraw-Hill, 2001.

Copyright © 2003. Artech House. All rights reserved.

Horch, John. Practical Guide to Software Quality Management, Second Edition, Artech House, 2003. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/think/detail.action?docID=227656.
Created from think on 2025-07-07 18:31:17.

Additional reading

Boehm, B. W., *Software Engineering Economics,* Englewood Cliffs, NJ: Prentice Hall, 1981.

Cai, Kai-Yuan, *Handbook of Software Quality Assurance,* Englewood Cliffs, NJ: Prentice Hall, 1999.

Crosby, P. B., *Quality Is Free,* New York: McGraw-Hill, 1979.

Ginac, Frank P., *Customer-Oriented Software Quality Assurance,* Englewood Cliffs, NJ: Prentice Hall, 2000.

Humphrey, Watts S., *Managing the Software Development Process,* Reading, MA: Addison-Wesley, 1989.

Mai, Masaaki, *Kaisen—The Key to Japan's Competitive Success,* New York: Random House, 1996.

Schulmeyer, G. Gordon, and James I. McManus (eds.), *Handbook of Software Quality Assurance, Third Edition,* Englewood Cliffs, NJ: Prentice Hall, 1999.

Walton, Mary, *The Deming Management Method,* New York: Putnam, 1986.

Copyright © 2003. Artech House. All rights reserved.

Horch, John. Practical Guide to Software Quality Management, Second Edition, Artech House, 2003. ProQuest Ebook Central, http://ebookcentral.proquest.com/lib/think/detail.action?docID=227656.
Created from think on 2025-07-07 18:31:17.