Chapter X

Managing Quality

Well done is better than well said.

(Benjamin Franklin)

Quality is defined by PMI as conformance to requirements, specifications, standards and fitness for use (PMI, 2000). Part of this definition is mostly quantitative, namely the requirements, specifications, and standards (assuming that these three things have been carefully and correctly itemized), but "fitness for use" is mostly qualitative. Because of this qualitative portion of the definition, the quality of the product which is the subject of the project may be an area for potential conflict between the performing organization and benefiting organization. *In fact, as a project proceeds, quality is the most difficult area to keep on track, not because of its complexity, but because the project team may compromise it when a crunch arises* (Hallows, 1998).

Quality Management

Quality is more than just conformance to requirements. A good set of requirements is often difficult to devise for IT systems. If there are defects in the requirements (which there usually are), one could conceivably have a high-quality system that is useless. In addition, simply meeting the requirements will not guarantee that the customer or end users are satisfied with the product. Quality must also be distinguished from grade. A low-grade product may have just as much quality as a high-grade product. Different grades of a product are generally created for different classes of service, and they generally have different unit prices.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

Most people think of bugs when they think of quality in the context of IT systems. The term *bug* was originally coined by Dr. Grace Hopper, who developed the COBOL language. She had found that a computer crash was due to a moth that lodged inside of the hardware. For large IT systems of 1,000 function points or more, total defects will average five bugs per function point (Jones, 1994). Even with high-quality software development, for every 500 or so lines of procedural C-like code there is one bug (Linger, 1994). Your electric razor has dozens of bugs, your TV may have hundreds of bugs, and your car may have thousands of bugs. Most of these bugs are encountered only when a certain set of circumstances arises. Software bugs cost as much as $60 billion annually, as estimated by the National Institute of Standards and Technology. Inside sources reported that Microsoft Windows 2000 was released with 63,000 potential defects (Foley, 2000; ZDNet, 2000). Software bugs increase in number in our modern world as our dependency on IT deepens and our reliance on automation and embedded software grows.

In addition to bugs, however, the basic definition of *quality* needs to be further extended when we consider completion criteria and satisfaction criteria. A more complete definition of *quality* for IT projects should include:

- Conforms to requirements and specifications
- Meets "customer expectations"
- Is defect-free
- Is highly usability
- Is consistent with adopted standards
- Is reliability (does it do it right all the time)
- Is robust (can handle invalid/unusual data and usage)
- Is testable
- Is auditable
- Is maintainable and readable
- Is secure
- Is recoverable
- Is appropriately documented (external and internal)
- Is efficient (with respect to speed, storage, clicks, keystrokes, and other resources)
- Is platform independent (portability)
- Is flexible and adaptable

A less specific but official list of quality attributes is found in IEEE 83: portability, reliability, efficiency, accuracy, error, robustness, correctness. Another official list of quality attributes is found in ISO 9126: functionality, reliability, usability, efficiency, maintainability, and performance.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

Poor quality can result in a number of undesirable events and states, including increased costs, low morale in both the performing and the benefiting organization, lower stakeholder satisfaction, increased risk, lost business, and lower project business benefits, whereas meeting quality standards generally results in increased morale, lower risk, higher stakeholder satisfaction, and meeting project benefit expectations. *Quality in IT projects has a long-term effect.* Sometimes the effects of poor quality may not be noticed until well after an IT product is built and deployed. Often, an IT product will perform in a satisfactory manner but may not be economically maintainable or adaptable. For an IT project that is successful (and remember that most IT projects do not succeed), most of the long-term costs will be in the maintenance phase (about two-thirds of the life-cycle costs).

A PM needs to understand what quality looks and feels like for his or her project. *It can be disastrous if the PM of an IT project does not understand how quality manifests itself in the application area for his or her project!* Team members need the same understanding and appreciation, and if lacking appropriate training should be supplied. Quality management is the overall process required to ensure that the above IT quality metrics are achieved; this overall process includes the following three subprocesses: quality planning, quality assurance, and quality control. Figure 10.1 defines these three processes (PMI, 2000).

Quality Planning

The main focus on quality used to involved product inspection (checking items after development). Deming (1982) and others showed that quality could be improved more by preventative measures such as improving production processes rather than by postproduction inspection. With the advent of modern quality philosophies, companies have begun to spend more on planning and prevention. It was found that the cost of using prevention was less than the cost of quality nonconformance. A rule of thumb states that for every dollar spent on prevention, the cost of repair is reduced by $3 to $10 (Jones, 1994). Often, management is less concerned with prevention, because solving problems is a high-visibility effort with immediate rewards, whereas preventing problems has low visibility. Both conformance and nonconformance incur costs; the different cost issues of each are shown in Figure 10.2. Thus, the total costs involved with quality can be expressed as (Campenella, 1999; Crosby, 1979):

Figure 10.1. Quality processes

Quality Planning	Quality Assurance	Quality Control
Define quality for the project and how it will be measured	Take measurements; determine if measurements are appropriate	Compare measurements to plan and take corrective action
Planning project phase	Execution project phase	Controlling project phase

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

$$\text{Cost}_{\text{quality}} = \text{Cost}_{\text{conformance}} + \text{Cost}_{\text{non-conformance}}$$

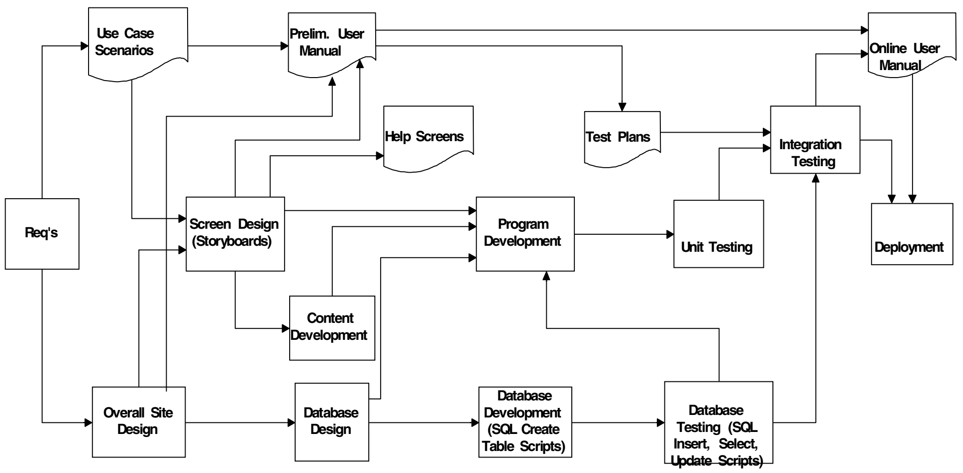
Complete quality planning comprises the design and plans for preventative measures (built-in quality) and product testing.

Nonconformance costs are classified as internal or external. Internal costs are incurred when the performing organization finds the defect, and external costs are incurred when the benefiting organization finds the defect. External costs are typically greater than internal. *The cost of correcting defects in IT systems is a function of when the defect is introduced and when the defect is discovered.* Defects introduced early and found late in the development process are the most expensive to correct. Consider the example methodology in Figure 10.3 for developing Web applications. Defects introduced in the requirements stage but not found until deployment may mean that all intervening steps have to be repeated in some degree. Defects introduced during program development (programming bugs) and found during integration testing will require the repetition of much fewer steps.

Figure 10.2. Cost of conformance and nonconformance

Cost of Conformance	**Cost of Non-Conformance**
Quality Planning and Training	Rework and Delay Costs
Effectiveness Studies	Wasted Effort
Validation and Verification	Warranty, Refund, Contract Penalty, and Legal Costs
Benefit Surveys	Loss of Customer Goodwill and Market Share

Figure 10.3. Web application development methodology



Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

Defects generally can be traced to five origins (Jones, 1994): requirements, design, coding, documentation, and bad fixes. It can cost 40 to 1,000 times more to fix a defect found late in the development cycle (Gause & Weinberg, 1989). Up to eighty percent of a typical software product's development time may be spent correcting errors not found early in the project (McConnell, 1997). In U.S. Department of Defense's (DoD) studies conduct by Hughes, it was found that it cost 100 times more to fix a problem that is not recognized until the application is released. A large number of errors are not found until the product is released and put in general use (about 22%) and most errors are introduced in the definition/requirements phase (about 55%). When defects are eventually fixed, there is a 40% change that other errors will be inadvertently introduced (for non-object-oriented software). Thus, the initial focus for IT project quality should be on prevention or built-in quality. Building in quality involves establishing defined rigorous procedures for how something is built or assembled. The discipline of software engineering is largely concerned with this topic, and this was covered in an earlier book chapter. For IT systems, this dimension of quality manifests itself mainly through: IT standards, requirements gathering techniques, design techniques, implementation techniques, and maintainability, adaptability, and reuse methods. Jones lists primary software defect prevention techniques as (Jones, 1994):

- Formal quality plans
- Use of joint application development (JAD)
- Use of prototyping
- Structured analysis and design techniques
- Reusable code and designs
- Software quality assurance teams
- Total quality management (TQM) methods
- Quality function deployment (QFD) methods
- "Clean room" development methods
- Quality measurement programs
- Reviews and inspections
- Use of defect estimation and measurement tools

The term *verification and validation* (V&V) has emerged in recent years and has been applied to software development projects; IEEE 1012-1998 is the new standard for V&V. Definitions from an IT perspective differ among authors concerning exactly which project activities are verification activities and which are validation activities; and some types of activities have ingredients of both. Formally, verification is proof of compliance with requirements, specifications, and standards. Verification is primarily concerned with built-in quality and is a *process-related notion* that asks the question, "Are we building the product in the correct manner?" Answering that question requires both the internal testing of the product and the inspection of processes. For IT projects, inspection involves primarily design walkthroughs, code walkthroughs, and method and tool

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

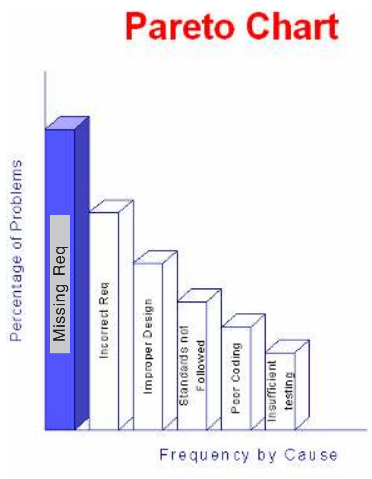
evaluations. These are often called *peer reviews* because peers are involved and typically provide more valuable feedback that management in these activities. These *defect prevention methods* can have a high defect removal rate of about 60% (Jones, 1994).

Formally, validation is proof that the customer and end users are satisfied with the system. Validation is a *product-related notion* that asks the question, "Have we built the correct product?" Answering that question requires inspection and acceptance (external) testing of the product, which for IT systems mainly comes at the end of the development cycle. Prior to the acceptance testing, which is typically performed by the benefiting organization (sometimes for contract compliance), there is normally several levels of internal testing (verification), including unit testing, integration testing, load testing (capacity), timing tests, destructive testing, and intrusion testing (security). Testing is discussed in more detail later in this chapter.

All validation need not be at the end of the development process. There are a number of procedures that can be carried out early in the overall process to help guarantee that the correct product is being built. These procedures involve getting the benefiting organization and other stakeholders more intimately involved in the review of preliminary product manifestations; this methodology is discussed in detail later in this chapter.

A part of the effort to build in quality for IT systems is to identify the major quality problem areas and then find the root causes of these problems. The Pareto chart is a classic quality engineering diagram that helps focus attention on the most critical problems of a project by presenting information in order of priority; this is illustrated in Figure 10.4. The philosophy is based on the old 80/20 rule (80% of the problems come from 20% of the activities). For a PM, instead of solving each problem as soon as it occurs, the Pareto diagram helps put the problems into perspective so a PM's time can be focused first on the key issues.

Figure 10.4. Pareto chart



Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

Quality Assurance

Quality assurance is done primarily during the execution of a project. Planned measurement systems are analyzed to see if all measurements are necessary or if more measurements need to be established. Overall project performance is regularly analyzed to make sure the project will satisfy the required quality standards. ***For IT projects, quality metrics should be defined in three areas: process, product, and perception; these three areas should encompass the critical success factors identified earlier in this book.*** For IT maturity models such as the SEI CMM model, maturity levels involve improvement of both the product and the process of building the product. Project control was covered in Chapter IX, and some of the control metrics identified and discussed in that chapter that are also quality related and are shown and classified in Figure 10.5. As well as product and process quality metrics, some metrics involve environmental factors into which the product is to be deployed.

Figure 10.5. IT quality metrics

Quality Metrics

Success Criteria			Metrics	Product	Process	Environment
Completion Factors						
	Project Management					
		Team Morale	Surveys & Interviews, Staff Attitude, Turnover		X	X
		Stakeholder Morale	Surveys & Interviews		X	X
	Verification					
		Defect Introduction	Defects per 1000 LOC	X	X	
		Defect Resolution	Defects Reported Defects Corrected		X	
	Technology					
		Process	Productivity (LOC per man hr)		X	
		Product	Product related metrics	X		X
Satisfaction Factors						
	Business Justification		ROI, IRR, Payback Period	X		X
	Validation					
		Stated Requirements	Requirements Document Approval	X		
		Unstated Requirements	Preliminary Product Manifestation Review	X		
		Acceptance Testing	Acceptance Exceptions (Changes & Defects)	X		
		External Quality Dimensions	Product Related Metrics	X		X
	Workflow & Content		Preliminary Product Manifestations Review	X		X
	Standards		Compliance Audits and Inspections	X		X
	Maintainability & Support		Comment Ratio, LOC per CO, Code Walkthrough	X		
	Adaptability		Reuse %, Avg Time per CO, Code Walkthrough	X		
	Trust/Security					
		Process Security	Security incidents, lost time		X	
		Product Security	Intrusion testing	X		

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

The product which is the subject of the IT project may also need to meet a certain *service level agreement* (SLA), which could even be part of the project contract. A SLA specifically defines the "quality" of service that is to be provided by the product within a certain deployment environment. Items typically found in SLAs are:

- Response time upper limits
- Percentage of log-ins or calls not answered first time
- Downtime maximum percentages (i.e., "99% uptime")
- Maximum downtime interval

The Software Engineering Institute's (SEI; www.sei.cmu.edu/cmm) CMM defines the necessary Level 2 practices for software quality assurance:

- Are quality assurance activities planned?
- Do these activities provide objective verification that software products and activities adhere to standards and requirements?
- Are the results of quality reviews and audits provided to affected parties?
- Are issues of non-compliance that are not resolved within the project addresses by upper management?
- Does the project follow a written policy for implementing quality assurance?
- Are adequate resources provided for performing quality assurance activities?
- Are measurements used to determine the cost and schedule status of quality assurance activities?
- Are these activities reviewed with upper management on a regular basis?

Quality Control

Quality control involves analyzing the measurements, comparing them to the plan, and taking corrective action if necessary, such as eliminating the cause of unsatisfactory performance. Analyzing the measurements typically involves quality control tools, such as statistical methods, control charts, trend analysis, and other flowcharting methods. Trend analysis may also involve using mathematical techniques to forecast future outcomes based upon historical values. Trend analysis is often used in IT projects to monitor technical performance metrics, such as how may defects have been introduced and detected and in which stages of the project. Finding many defects in later stages such as integration testing or user acceptance testing is an indication of earlier quality problems in analysis or design. This is illustrated in Figure 10.6.

Figure 10.7 shows another useful type of trend chart that of cumulative defect occurrence and correction. For this type of analysis, one hopes to see the gap between cumulative

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

Figure 10.6. Defects introduced and found

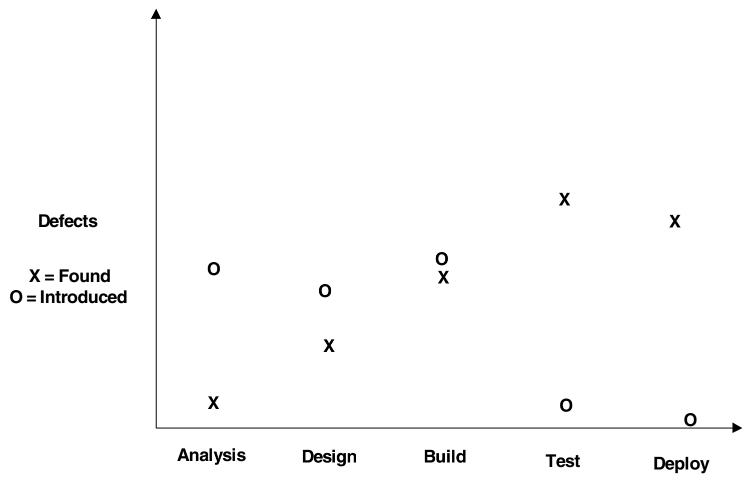
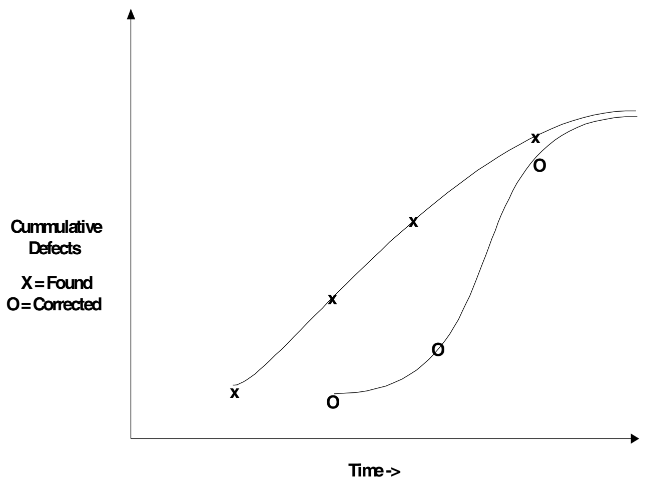


Figure 10.7. Cumulative defects



defects found and cumulative defects corrected closing as the project proceeds. If that gap (between the two curves) is becoming wider, then there is a problem in getting timely fixes to the defects.

A very useful technique for finding root causes of projects is the Ishikawa or Fishbone diagram (also called cause-and-effect diagrams). These diagrams show how various causes or potential causes (and their subcauses) relate to cause the overall problem. These diagrams also help stimulate project team thinking. Figure 10.8 shows this type of diagram for the problems involved with requirements analysis.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

Figure 10.8. Requirements cause and effect diagram

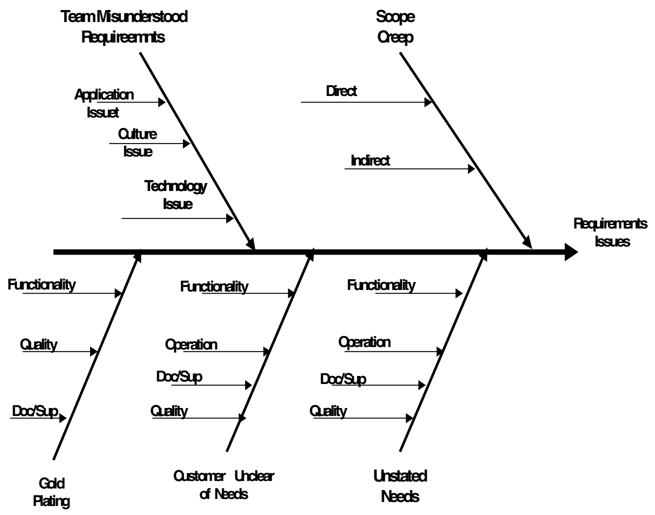


Figure 10.9 shows another diagram of this type for analyzing the problem of software defects.

Jones (1994) lists the eight *root* causes of quality problems as:

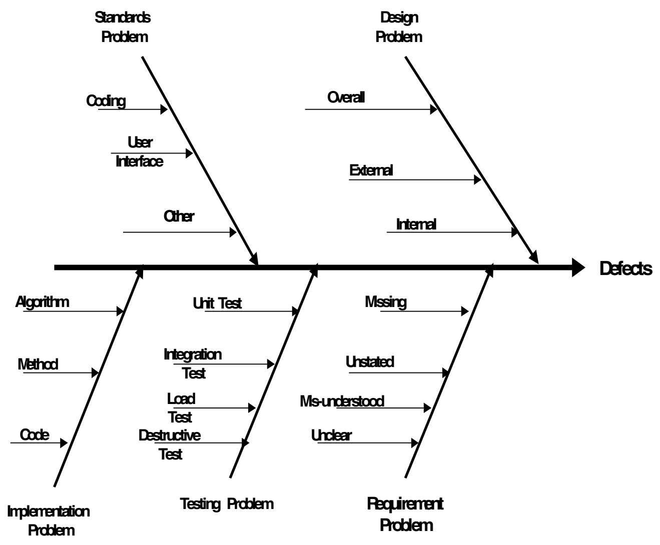
- Lack of an understanding of what quality means for software
- Inadequate defect prevention
- Inadequate use of reviews and inspections
- Insufficient or careless testing
- Lack of quality measurements
- Lack of understanding by project management that quality is critical to completion metrics and user satisfaction
- Excessive schedule pressure leading to reduced quality efforts
- Unstable and ambiguous user requirements

Often detailed and structured, *quality audits* are conducted upon project completion to evaluate both the software engineering and the project management processes in regard to quality aspects, and these become part of the project lessons-learned documentation.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

Figure 10.9. Defects cause and effect diagram



Software Testing

Techniques to build quality into software systems have been quite successful over the last several decades. The average software defect rate declined from about 8 per KLOC (thousand lines of code) in the 1970s to 5 per KLOC in 1990 ("The Quality Imperative," 1991). *However, when software is created using the most rigorous of methodologies and quality standards (producing 1 or 2 defects per KLOC), there will still be a significant number of defects in the product prior to testing. Software testing and defect correction typically takes between 30% to 60% of project development time and budget.* Complex application developments can spend more than half of their total program effort on testing. When a time crunch arises in a project, often the full testing of a product is compromised. A generally accepted rule is that testing costs will be 25% of the total development costs (Brown, 1998).

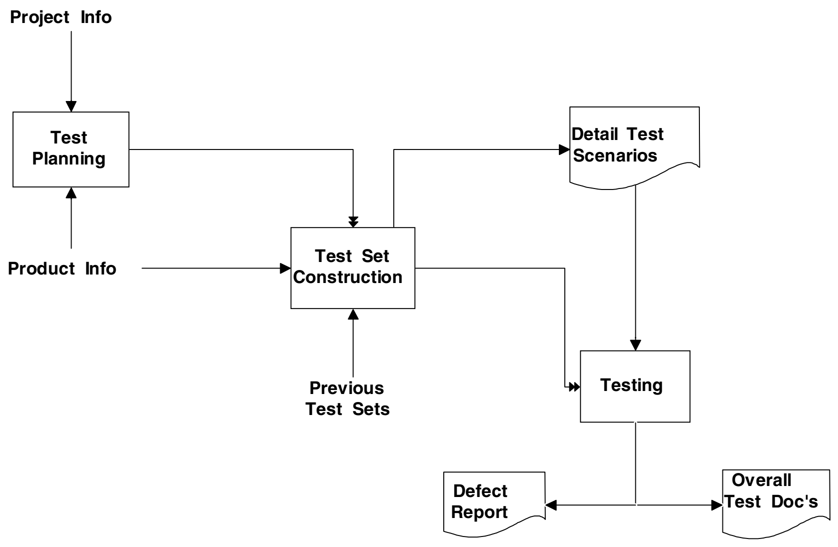
There may be times that testing is not necessary; the Software Program Managers Network has identified these circumstances (Brown, 1998):

- When the responsibility for failure can be shifted to someone else
- When the impact or significance of any problem is insignificant
- When the software does not have to work
- When no one is or will be using the system
- When the project has already failed

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

Figure 10.10. IEEE standard for software unit testing



Testing should be a multilevel process with different types of testing in different stages of the project. Single stage testing is about 30% effective, but multilevel testing can achieve a much higher efficiency (Jones, 1994). Testing should also be included for software components that are purchased or otherwise acquired.

Unit testing is typically done by the developers and is a process to make sure that each software module performs properly relatively independently from other modules. Input, output, and interface operations are typically simulated. Unit testing is typically mostly white-box ("structural") testing where knowledge of module internals is utilized to insure that each source code statement and path through the code is exercised, possibly as a function of different external and database conditions. Often, test harnesses are created for each module, which is a set of specific tests for that module. These harnesses are checked into the source code version control system, and they are usually automatically repeated for each change to that module. For cleanroom software engineering discussed earlier in this book, unit testing is not done by the coders but by separate testers; the coders review each others code and walk through the execution paths. The economic benefits of this clean room approach to unit testing are debatable. The unit testing of a module is typically included in the work breakdown structure (WBS) task to build that module.

Integration testing involves testing the entire system working together; this is often called an "end-to-end" or system test. This type of testing should not be done by the developers, but by an independent testing group. The project WBS should have separate WBS code(s) for integration testing. This testing process is typically composed of three phases: test planning, test set construction, and test execution. Figure 10.10 from the

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

IEEE Standard for Software Unit Testing (IEEE/ANSI STD 1008, 2002) illustrates these three testing phases.

The activities in each phase include the following (IEEE/ANSI STD 1008, 2002):

- Test planning
  - Plan the general approach
  - Plan resources and schedule
  - Determine test environment(s)
  - Determine features to be tested
  - Refine the general plan
- Test set construction
  - Design the set of tests and associated procedures
  - Implement that design (build test scripts)
- Testing
  - Execute the test sets
  - Check for correct behavior and results
  - Evaluate the test results, effort, and other relevant metrics
  - Document results

A test plan document is usually produced not only for the testing group, but also to be read by other interested stakeholders. It would include such items as:

- Identification of IT product and version
- Background of product/version including purpose
- Prior versions including revision history
- Purpose and objectives of this testing
- Reference to other product documents (i.e., requirements, design, etc.)
- Relevant standards
- Project team and stakeholder identification and contact info
- Test organization contact info
- Test resources (people, training, access to other people, hardware, space, etc.)
- Test environment(s), platforms, locations, and conditions
- Assumptions and dependencies
- Test scope and focus
- Relation to other validation activities
- Testing outline

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

Testing data and other setup

Test metrics

Testing tools

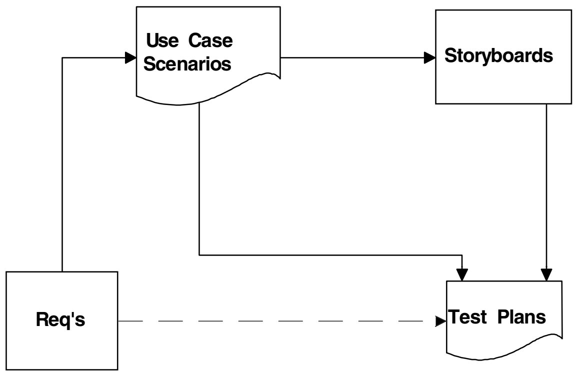
Test scripts

- Detail test procedures
- Test reporting, documentation, deliverables

A key part of this test process is the construction of the test sets, sometimes called test scripts. The test scripts should be based upon the product requirements either directly or via the use case scenarios and storyboards (screen design or "paper prototypes"); this is illustrated in Figure 10.11. *If the test scripts are taken from the use case scenarios, then the likelihood is much greater that the testing will involve features and code paths that are more important to the customer. Testing is expensive, so one does not want to waste time testing features that no one will use or cares little about. However, if the test scripts are not taken directly from the requirements, there still needs to be a check that each requirement is included somewhere in the test scripts (requirements traceability was discussed earlier in this book).*

These test scripts indicate specifically how the product is to be tested and what the expected results are at each point in the test. For example, in testing a form to add a new entity to a database, the script would indicate exactly what was to be typed into each field on the form, including typing of invalid entries (for which the product should give meaningful error messages). In conjunction with the generation of test scripts, there may be a need to generate test data. The ideal situation is to have the test data created as part of running earlier test scripts. However, to allow more testing to go on in parallel, the test data may need to be created before running the test scripts; also, for volume testing, huge amounts of data may need to be generated. The test environment also has to be defined and set up, and this environment may involve one or more platforms on both the server and client sides. For example, in building a modern Web application, one may need to test

Figure 10.11. Requirements-based testing



Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

using both Microsoft and Unix/Linux type servers, possibly from different vendors and versions; and one may need to test using multiple client configurations (browser types, browser versions, screen size/resolution, and PC/MAC).

The running of test scripts can often be automated by direct programming or by using a software-testing product; this is discussed later in this chapter. When running the test scripts, each failure, defect, standard deviation, concern, or suggestion for improvement must be noted in the test documentation. Failures may arise from a number of causes, and the test plan should indicate procedures for dealing with each of these causes:

- A fault in the test script
- A fault in the test data (including meta data [table/view definition, constraints, etc.])
- A fault in the test environment
- A fault in the implementation (i.e., coding bug)
- A fault in the algorithm
- A fault in the design
- A misunderstanding of the requirement

The ANSI/IEEE STD 829-1983 IEEE Standard for Software Test Documentation is a standard for the documentation produced in the testing phases. For each failure, detail documentation should be produced that would include items such as:

- Product name and version
- Location of defect within product
- Environment and platform info
- Related processing info (database conditions, etc.)
- Assigned/generated defect ID number
- Date/time
- Status (new, previous, etc.)
- Complete information about the failure (including screen shots, etc.)
- How to reproduce the problem (if reproducible as opposed to "random")
- Estimated severity of the problem
- Cause of defect (if known or suspected)
- Tester info (name, contact, organization, etc.)

When the defects were deemed fixed and/or rerested, that information would be appended to the aforementioned record (retest info, status, etc.). When developers indicate that a defect has been corrected, there is a need to retest. Defect corrections (and possibly other requested changes) are generally "batched" together to produce a new

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

version of the software product. Integration/system testing is then repeated for that new version. This retesting process is called "regression testing," and a key issue is how many of the original test scripts need to be repeated. One can simply repeat just the tests for new features and those that produced defects in the previous version, or alternatively, run all the tests again. Any specific fix cannot correct the defect (or partially correct it), but can also introduce other defects. The safest choice therefore would be to run all the tests again; this is often not a feasible choice in terms of time and cost, unless the testing has been considerably automated.

After integration testing, which focuses on features, there may be separate testing for loads, scalability, and timing. Timing may be a function of the load for "wall clock" measures of completion or response times, or it may be measured in the number of "clicks" or "keystrokes" needed to complete a task. These types of tests may be necessary to ensure compliance with SLAs during production operation. Destructive testing may also be carried out separately or in conjunction with the integration testing. A certain amount of destructive testing should be included in both unit testing and integration testing such as the effect of incorrect or missing user input. In destructive testing, one tries to destroy or compromise the integrity of the system via input errors, usage errors, executing features out of sequence, violating security measures, and intentionally tricking the system. Most integration testing utilizes black-box (behavioral) testing, in which only the external specifications of the product are used. However, destructive testing may involve types of "white-box" testing, in which knowledge of module internals is utilized. Separate auditability tests may also be run particularly for business applications in which both forward and reverse traceability is tested. Forward tracing follows the entry of a transaction through the system to where it is posted to account/entity totals. Reverse tracing takes account totals and works backward to display the transactions that make up that account total.

Acceptance testing, which involves the benefiting organization (i.e., the customer), is often carried out after integration testing. The project contract may require that acceptance testing be done; this testing is sometimes performed exclusively by the benefiting organization. Normally, acceptance testing is implemented on a much smaller scale than integration testing, and it represents a sampling of features and conditions. After (or instead of) acceptance testing, the product might be produced for alpha testing for use with a limited set of friendly users. If alpha testing is successful, beta testing is implemented, which involves a larger and not necessarily friendly group. Beta testing, involving many users (i.e., 1,000 or more), can achieve testing efficiencies of about 75% (Jones, 1994).

The Software Program Managers Network defines eight levels of software testing from white-box unit testing through final site black-box production testing (Brown, 1998). Each level has a focus, organizational responsibility, documentation basis, test type, and test activities. Their 10 commandments of testing are as follows:

1. Black-box test that validates requirements must trace to approved specifications and be executed by an independent organization
2. Test levels must be integrated into a consistent structure
3. Do not skip levels to save time or resources; test less at each level

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

4. All test products, configurations, tools, data, and so forth, need configuration (version) control
5. Always test to procedures
6. Change must be managed
7. Testing must be scheduled, monitored, and controlled
8. All test cases have to trace to something that is under configuration (version) control
9. You cannot run out of resources
10. Always specify and test to criteria

Extreme programming (XP) was discussed earlier in this book, and extreme testing is a key part of that XP methodology. Developers are expected to write unit test scripts first before the application is constructed. Test scripts are under source control (version control) along with the source code. The customers are expected to be a part of the project team and to help develop scenarios for the test scripts, at least acceptance scripts, if not all testing scripts. QA and test personnel are also expected to be a part of the project team.

Most testing of IT systems is still done manually. However, automated testing can often produce better results at a cheaper price. Newer, automated tools cover all phases of the testing process, including planning. The Quality Assurance Institute conducted benchmarks comparing manual and automated testing methods and found an overall reduction in person time of 75% (QA Quest, 1995). Reduction in the actual test runs was 95%, with little savings in the test planning phase. Automatic testing, along with saving time and cost (after an initial learning curve in time and dollars), has these further advantages (Brown, 1998):

- Test repeatability and consistency
- Expanded and practical test reuse
- Practical baseline suites

Automatic testing tools really come in handy when one has to repeat the same tests for a number of different client and/or server platform environments, as is the case with modern Web and e-commerce applications. Automated testing is also very useful when load testing is necessary with a high volume of users, transactions, and/or data. However, there are situations in which automatic testing is less useful, such as situations in which human judgment is necessary, and it is difficult to define criteria in a quantitative manner. Usability testing is one such area (usability of documentation as well as the product), and another area is "localization" particularly of global e-commerce applications.

There are a number of software testing products available to automate all or portions of the testing process: QAWizard (www.seapine.com), TestQuest (www.testquest.com), e-Test (www.empirix.com), HighTest (www.vtsoft. com), ApTest (www.aptest.com), QES (www.qestest.com), TestComplete (www.automatedqa.com), QACenter (www.compuware.com/), and TestWorks (www.soft.com/TestWorks/). Some testing

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

products are for specific environments (i.e., Web applications, windows applications, Java applications, etc.), some are for specific products (i.e., SAP, Peoplesoft, etc.), some are for specific problems (i.e., "memory leaks," bounds checkers, uninitialized variables, type mismatch, nonconformance to coding standards), and some are general purpose. For a list of current open source testing tools see OpenSourceTesting (www.opensourcetesting.org/). There are also a number of special purpose programs available for defect tracking as: Bug-Track (www.bug-track.com), ElementTool (www.elementtool.com), and TrackStudio (www.trackstudio.com).

Evaluating, selecting, procuring, and learning a test tool can be a significant expense and perhaps be a project in itself for an IT organization. Activities that are part of this effort include a needs analysis, a business justification for the procurement and implementation costs, a study of the available appropriate tools, implementation of the tool, and personnel training on the tool. These tools are difficult to evaluate and it is possible that it will take 6 months to discover that the tool will not fit the application (Hendrickson, 1999). "Try before you buy" is the best philosophy here. For example, with modern GUI applications, simple playback tools are inadequate, because if a single GUI widget is changed on a screen, then the whole script may have to be changed. For GUIs, more complex tools are needed that allow one to associate specific GUI widgets with specific series of valid/invalid script entries. Organizations should not underestimate the time and cost necessary to learn to use a specific tool, or the effort in setting up the tool for the application at hand (which typically involves test script programming). Dustin (1999) listed some lessons learned in test automation, including:

- Various tools used throughout the life cycle do not integrate easily
- Duplicate information had to be kept
- The testing tool drove the effort
- The staff was very busy learning to write the scripts
- Elaborate test scripts are developed, duplicating development effort
- Script creation is cumbersome
- The test tool(s) were introduced too late
- Training in the tools was too late
- Some testers resisted tools
- Expectations of early payback were not realistic
- Tool(s) had problems recognizing third party GUI controls
- Lack of test development guidelines
- Tool(s) was intrusive, and required "inserts" into code
- Reports reproduce by tool(s) was useless
- Tools were selected before overall system architecture was determined
- Upgrade to new tools had compatibility problems
- Tool(s) database was not scalable

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

There are also companies that offer testing and Q/A services, such as Systemware (www.sysemware.com). Several Web sites are available that provide links to numerous testing resources (tools, books, articles, etc.), such as the Software Q/A Test Resource Center (www.softwareqatest.com), the Software Testing and Quality Engineering Network (www.stqe.net), and Testing and Quality Engineering Magazine (www.stqemagazine.com).

Quality Stage Gates

User acceptance testing is a validation activity that comes late in the development process but not necessarily at the end. There are activities that can be carried out earlier in the overall process to help guarantee that the correct product is being built. These activities involve getting the benefiting organization and other stakeholders more intimately involved in the review of preliminary product manifestations. Sometimes these types of activities are called “static testing” because they do not actually involve running the system (final product). The cooperative review of these preliminary product manifestations are called quality stage gates. Management stage gates were discussed earlier in this book, and applying management stage gates to the classical waterfall methodology (and variations thereof) was also illustrated. Figure 2.6 in Chapter II illustrated my notion of dual gates.

Multiple quality stage gates may fall within one management stage gate or vice versa. At the management stage gate, the completion criteria are reviewed (either exhaustively or by exception), and a go/kill/hold decision is reached. Management stage gates can be set at fixed time intervals or upon completion of a major project phase. For methodologies in which phases overlap, or when using incremental or iterative approaches, the management stage gates can be set between increments or iterations or upon fixed time increments, such as 1 month. To minimize the time and cost associated with formal management reviews, it is recommended that the management stage gates occur at regular time intervals but that they take place only on an exception basis when key metrics (such as EVA indexes) indicate problems.

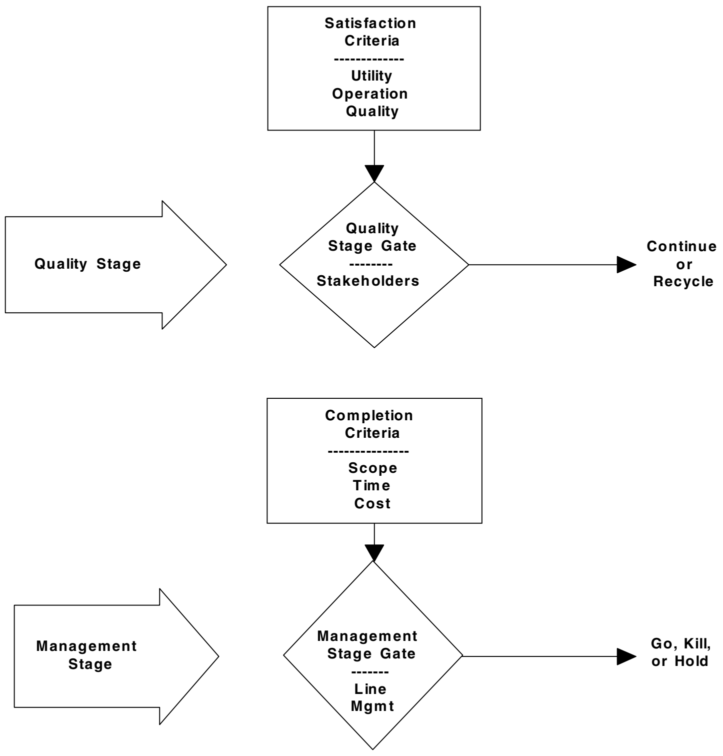
For each quality stage gate, the satisfaction factors typically are reviewed by focusing on a particular preliminary product manifestation by relevant stakeholders for that gate; the review does not make a go/kill/hold decision but rather a go-forward or recycle type decision. Figure 10.12 illustrates these distinctions.

Whereas the management stage gates review completion factors (i.e., actual costs to date, earned value, estimated cost at completion, completion status of activities, estimated time to complete, updated risk analysis, and need for more or less reserves), the quality stage gates focus on satisfaction factors as utility, operation, maintainability, auditability, and a revised business justification (i.e., revised cost-benefit analysis based on latest cost estimate and revised benefit numbers). Different stakeholders may be present at the two types of stage gate reviews, to minimize the cost and time of key individuals. Preliminary product manifestations that should be reviewed by concerned stakeholders would include requirements document, use case scenarios, preliminary

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

Figure 10.12. Dual stage gates



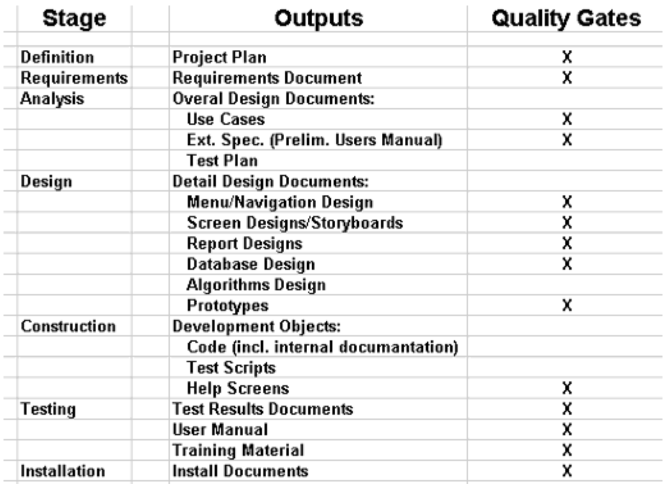
user manual, storyboards ("paper prototypes"), screen/report designs, and working prototypes. Wiegers (1999) reported a return on investment of up to 800% for requirements reviews. Figure 10.13 shows an example of how the quality gates might be set up for the classical waterfall methodology.

Figure 10.14 is an example of a stage gate review form. On this form, the metrics from the earned value analysis of the last management gate (typically at certain time period intervals) are recorded, which indicate how the project is going from a progress and cost perspective. The next portion of the form concerns the satisfaction factors, which are qualitatively evaluated by both the benefiting organization (customer) and the performing organization (project team). The basis of the satisfaction scoring is also specified, which would be the last quality gate in which a product artifact was available, such as a prototype. The risks that were identified and quantified at the start of the project are then reevaluated. The information on this form is then typically reviewed by line management of both organizations (either regularly or by exception) to determine if the project should continue.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

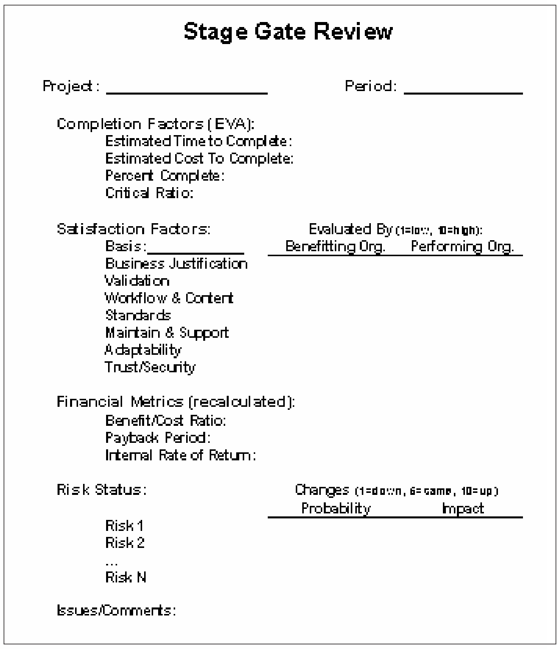
TEAM LinG

Figure 10.13. Event driven quality stage gates



Stage	Outputs	Quality Gates
Definition	Project Plan	X
Requirements	Requirements Document	X
Analysis	Overal Design Documents:	
	Use Cases	X
	Ext. Spec. (Prelim. Users Manual)	X
	Test Plan	
Design	Detail Design Documents:	
	Menu/Navigation Design	X
	Screen Designs/Storyboards	X
	Report Designs	X
	Database Design	X
	Algorithms Design	
	Prototypes	X
Construction	Development Objects:	
	Code (incl. internal documantation)	
	Test Scripts	
	Help Screens	X
Testing	Test Results Documents	X
	User Manual	X
	Training Material	X
Installation	Install Documents	X

Figure 10.14. Stage gate review form



Stage Gate Review

Project: ____________________ Period: ______________

Completion Factors (EVA):
- Estimated Time to Complete:
- Estimated Cost To Complete:
- Percent Complete:
- Critical Ratio:

Satisfaction Factors: Evaluated By (1=low, 10=high):

Basis: ___________	Benefitting Org.	Performing Org.
Business Justification		
Validation		
Workflow & Content		
Standards		
Maintain & Support		
Adaptability		
Trust/Security		

Financial Metrics (recalculated):
- Benefit/Cost Ratio:
- Payback Period:
- Internal Rate of Return:

Risk Status: Changes (1=down, 5=same, 10=up)

	Probability	Impact
Risk 1		
Risk 2		
...		
Risk N		

Issues/Comments:

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

Quality Programs

There are a number of modern quality programs sponsored by national and international organizations. These organizations foster quality by defining certain general or specific approaches to achieving quality and/or by setting standards for quality. Earlier in this book, the Software Engineering Institute's Capability Maturity Models (SEI CMM) and the Project Management Institute's Body of Knowledge (PMI PMBOK) were discussed. Quality assessment methods for IT (specifically software development) were also discussed, including the SPICE (ISO/IEC 15504), TickIT, CobiT, and ITIL approaches. These organizations foster quality by defining rather specifically the best practices for each of these overlapping disciplines. These best practices are the things organizations need to do in order to produce quality products via properly managed projects. Although the things that need to be done are itemized, exactly how to do them is left up to the discretion of the individual organization and/or PM. In this section we look at more general quality programs.

Total quality management (TQM) is a business philosophy that encourages organizations and their employees to focus on quality and finding ways to make continuous improvements to the organization's products/services and practices. TQM gained popularity in the United States in the 1990s after successful implementation by Ford, Boeing, and other major corporations. TQM is based on earlier work from around the world (particularly in Japan after World War II) by Shewart, Demming, and Ishikawa. The six principles of TQM are:

- Customer focus
- Focus on process as well as results
- Prevention versus inspection
- Mobilize the expertise of the workforce
- Fact-based decision making
- Feedback

The cornerstones, however, of TQM are continuous improvement, employee empowerment, and customer focus. Continuous improvement (or *kaizen*) is a continuous process of making improvements, typically by small steps. In Japanese, *kai* means to alter and *zen* means to make better. Employee empowerment means to allow employees to make suggestions on how to improve processes, because they are the people most intimate with the process, not their supervisors. TQM is based on the tenet that most employees want to be involved in improvement and to do their jobs well. TQM fosters teamwork among employees to identify quality issues and corrections. An application for the Malcolm Baldrige Award[1] may be made by companies that have fully implemented TQM and quantified the benefits thereof.

Quality function deployment (QFD) was founded in Japan in the late 1960s by Professors Shigeru Mizuno and Yoji Akao. Statistical quality control had become prevalent in the

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

Japanese manufacturing industry, and these quality activities were being integrated with the teachings of Ishikawa and Feigenbaum, which later became known as TQM. Mizuno and Akao sought to develop a quality assurance method that would design customer satisfaction into a product before that product was produced. The first major application was in Bridgestone Tire, in Japan, which used a Ishikawa ("fishbone diagram") to identify each customer requirement (effect) and to identify the design the process factors (causes) needed to control and measure it. With continued success, these fishbone diagrams of customer needs and expectations were refashioned into a spreadsheet or matrix format, with the rows being desired effects of customer satisfaction and the columns being the controlling and measurable causes. At about the same time, Ishihara introduced the value engineering principles that described business functions necessary to assure quality of the design process itself. QFD emerged as a combination of these two methods to ensure comprehensive quality design system for both product and business process (Akao, 1990). QFD reached the United States and Europe in the 1980s when the American Society for Quality Control published Akao's work and invited Akao to give QFD seminars in America. Today, QFD is used extensively in the United States, Japan, Sweden, Germany, Australia, Brazil, and Turkey. QFD is somewhat different from other quality principles in that it focuses on both customer-stated and unstated requirements. It seeks to maximize positive quality aspects, such as ease of use, fun to use, interesting, etc. QFD key principles are (Mizuno & Akao, 1994)

- Understanding customer requirements
- Quality systems thinking + psychology + knowledge/epistemology
- Maximizing positive quality to add value
- Comprehensive quality systems for customer satisfaction
- Strategies to "stay ahead of the game"

The International Organization for Standardization (ISO) also defines a set of quality management principles called ISO 9000, which encompasses much of TQM. Within the U.S. DoD, 2167A represents a similar set of principles and requirements. The latest version of ISO 9000 (ISO 9000:2000) is composed of these management principles:

- *Customer Focus:* Keep the customer's need at the forefront
- *Quality Focus through Leadership:* Leaders must always be concerned with quality and foster a similar attitude in their followers
- *People Focus:* Management must keep all stakeholders involved and foster open and honest communication
- *Process Focus:* Production and project activities should be viewed as a process so that process optimization techniques can be applied
- *Systems Approach:* Interrelated and dependent processes should be viewed as a system so that system optimization principles can be applied across all such processes

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

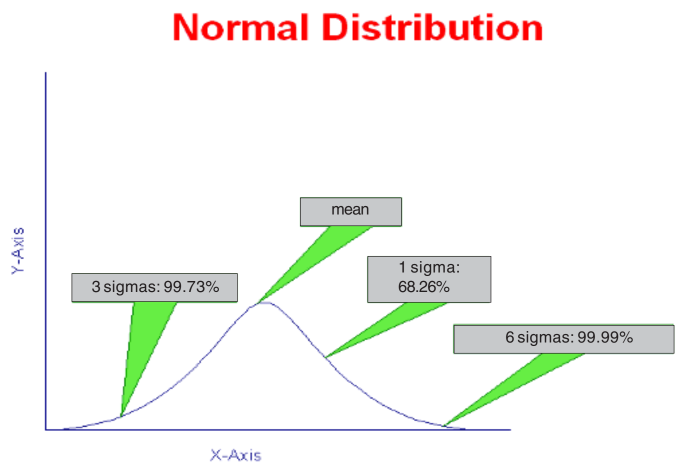
TEAM LinG

- *Continuous Improvement:* The organization should strive to continually improve its systems and processes via metrics and goals
- *Quantitative Decision Making:* Decisions should be based upon information and methods to formally analyze that information
- *Symbiotic Relations with Suppliers:* Relations with vendors should create value for both organizations for the long-term success and quality of both organizations

Organizations may seek ISO 9000 certification to prove that they have implemented these quality management principles and the business processes to support same. ISO 9000 certification can be sought under one of three programs: ISO 9001, ISO 9002, or ISO 9003. ISO 9001 is the certification that typically applies to IT organizations because it is for organizations that have design and development processes as well as production and support. If an IT organization does not do design and development (such as may be the case where all projects involve integration of off-the-shelf products), then ISO 9002 may apply. *The process standards for the ISO certifications basically make sure that organizations have and follow their own quality procedures; they do not define the quality procedures.* Companies desiring such certifications typically conduct a detailed self assessment first, then seek formal approval by a third party which is registered to conduct such audits. It has been suggested that jumping through the ISO hoops can be an expensive and depressing process for organizations as opposed to the application for a the TQM Baldrige award, which is more uplifting (Jones, 1994).

A bit more quantitative and specific quality program is *Six Sigma.* This program was started by Motorola in the 1980s and received considerable attention after very successful implementations by General Electric (GE) in the 1990s. Sigma (σ) is Greek and is the standard deviation in the normal distribution. This statistical distribution is the most common, and it is used in quality analysis, such as for measurement variations; it is illustrated in Figure 10.15. The *standard deviation* (sigma) is a measure of how far a point is from the mean (half of the distribution is on each side of the mean). Sigma ranges are

Figure 10.15. Normal distribution



Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

- 68.26 % of the distribution is within one sigma
- 95.46% of the distribution is within two sigma
- 99.73% of the distribution is within three sigma
- 99.99% of the distribution is within six sigma

For six sigma, about 99.99% of the products (or lines of code) are without defects. Inversely, for a metric like defects per million, the six-sigma level is only 3.4 defects per million.

Like other quality programs, Six Sigma focuses on processes; metrics are set up to measure defects per opportunity (DPO). A defect is essentially anything that results in stakeholder dissatisfaction. Defects are measured as early as possible during development rather that after the product is produced. For IT development, this involves setting up metrics at each stage in the development process: requirements gathering, analysis, design, implementation, testing, and deployment. Six Sigma differs from earlier quality programs (like TQM) in that it is not quality just for the sake of quality but specifically for the sake of the customer. The customer defines the defects. Once customers have defined what is most important to them, such as speed in completing the on-line ordering process, the organization determines which activities most affect its ability to speed up that process. Those selected activities become critical-to-quality (CTQ) activities (Shand, 2001). The Six Sigma program is composed of 12 steps (Harry & Schroeder, 2000):

Measure

1. Select critical-to-quality (CTQ) activities
2. Define performance standards
3. Validate measurement system

Analyze

4. Establish product capability
5. Define performance objectives
6. Identify variation sources (root causes)

Improve

7. Screen potential causes
8. Discover variable relationships
9. Establish operating tolerances

Control

10 Validate measurement system
11. Determine process capability
12. Implement process controls

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

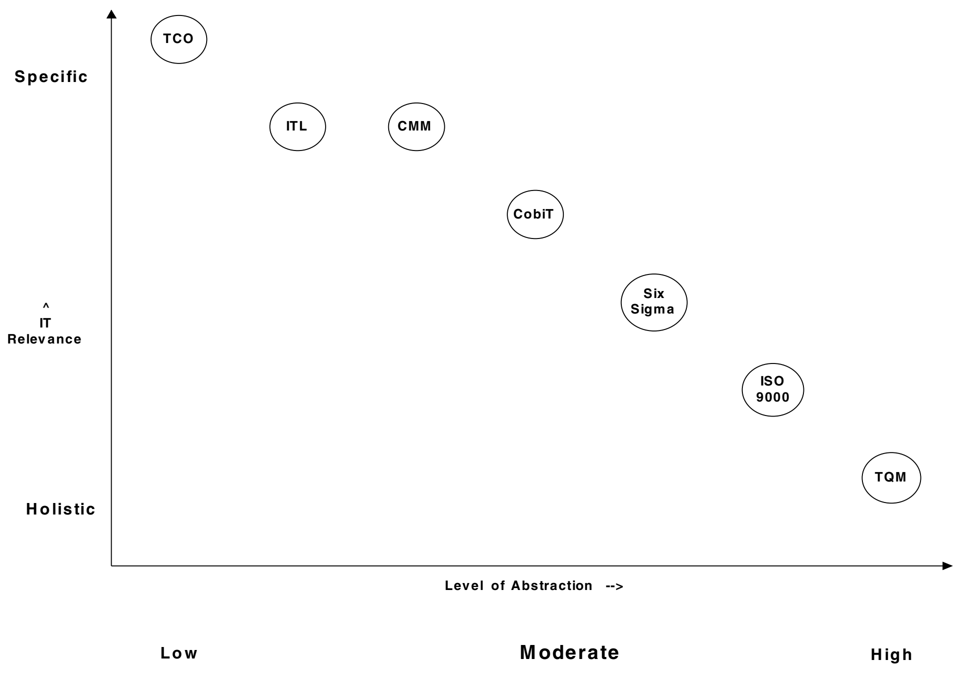
The basic Six Sigma process identifies CTQ activities, finds root causes for potential problems with those processes, discovers quantitative relationships between cause and effect, sets up metrics to measure causes, and controls the causes of the potential problems so that the effects remain within the six-sigma level. To implement Six Sigma requires a significant investment in a very specific training program.

Recently, *Computerworld* has defined many of these quality programs and summarized their degree of abstraction and specific IT relevance, as is illustrated if Figure 10.16 (Athens, 2004).

Software Development Standards

Quality programs and best practices focus mainly on processes, and most are fairly general rather than specific. Quality programs do not always handle the long-term effects of quality problems, particularly in IT projects. Issues like maintainability, adaptability, and reuse are not fully addressed in quality programs, particularly in customer-focused programs (because these aspects may never be seen by the customer). Standards focus mainly on product criteria, and standards can also be established to deal with the long-term effect of quality. *One of the best ways to ensure quality in IT processes, products,*

Figure 10.16. Quality programs



Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

and projects is by setting standards within the IT organization at the highest relevant level. Standards are formal documented specifications for how processes are to be specifically executed, how products are to be built, and/or how products are to perform.

There are many areas of technical standards. Some of the most common and important for IT development involve requirements, design, coding, database, testing, internal and external documentation, user interface, and other external interfaces. The IEEE has many important software engineering standards, and the ones most relevant in the quality area are:

- IEEE Standard for Software Quality Assurance Plans
- 829730-2002 -1998 IEEE Standard for Software Test Documentation
- 982.1-1988 IEEE Standard Dictionary of Measures to Produce Reliable Software
- 1008-1987 IEEE Standard for Software Unit Testing
- 1012-1998 IEEE Standard for Software Verification and Validation
- 1012a-1998 IEEE Standard for Software Verification and Validation—Supplement to 1012-1998
- 1061-1998 (R2004) IEEE Standard for Software Quality Metrics Methodology
- 1228-1994 (2002) IEEE Standard for Software Safety Plans
- 1465-1998 [Adoption of ISO/IEC 12119: 1994(E)], IEEE Standard Adoption of International Standard ISO/IEC 12119: 1994(E) Information Technology-Software packages: Quality requirements and testing

Details of these standards are available on the IEEE Web site (www.standards.ieee.org). Other relevant IT standards are also available from the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO); IEEE, ANSI, and ISO standards overlap in many areas and often correlate directly to each other. These are all fairly general standards which must be made specific for each organization, and perhaps for each platform and programming language. Consider, for example, an organization's coding standards, which may include such items as:

- Naming conventions for variables (static and instance), constants, classes, functions/methods, and so forth.
- Use and placement of constants (i.e., in one place such as a resource file, C++/PHP abstract class, or Java interface)
- Use of object orientation (inheritance, composition, polymorphism, interfaces)
- Use of comments and internal documentation
- Use of indentation and "white space"
- Use of parenthesis for expression evaluation order
- Conditional versus logical expressions
- Proximity of decisions and actions

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

- Iteration versus recursion
- Code density (i.e., one code line per statement) and readability
- Function/method sizes (i.e., only one logical function and under 100 lines of code)
- Types of classes (delegation, adapter, inner, anonymous, etc.)
- Access protection (public, private, package, protected, etc.)
- Minimization of coupling (friends, object arguments, etc.)
- Error handling (input checking, exceptions, etc.)
- Overloading (function, operator, etc.)

For communicating and enforcing standards, many companies probably still rely on some type of printed standards manual. For some companies, this is a large multivolume set of bound documents. Other organizations, who may or may not have such extensive standards manuals, do periodic "code walkthroughs," which have peers and/or managers examine samples of each programmer's code to make sure they are following the company standards. Some software engineering methodologies have a degree of code walkthrough built into the methodology, such as in pair programming, in XP.

The testing process is another way to make sure the programs have the desired "look and feel." Here, the results are easily viewed but not the underlying methods. Weinschenk (1997) stated that "in order for your guidelines to be successful, people must know they exist and be encouraged to use them." She outlines a list of several management actions which foster effective standards implementation.

The only way to really make sure that standards are being followed from both an external and internal perspective, however, is to require developers to utilize components in which the relevant standards have already been embedded (Brandon, 2000). The method of component utilization varies with the programming language and programming style from simple "include" techniques (such as a COBOL copy books, SHTML server-side includes, or PHP requires) to the object-oriented techniques of inheritance and composition that were discussed earlier in this book.

As an example, consider the user interface, which is often the key part of a modern IT application. There has been a lot of attention on user interface approaches and standards ever since Apple Computer was successful with the first commercial graphical user interface (GUI; Collins, 1995). Many books and some movies have been made concerning the technical, business, and social issues of the GUI saga. For a computer system to have lasting value, it must exist compatibly with users and other systems in an ever-changing IT world. Even if a system by itself is excellent, the fact they it may have a different look and feel to other systems in use will eventually spell its doom.

Companies building software systems, whether for their internal use or for resale, must build products that are compatible with other systems. As stated by Weinschenk and Yeo (1995), "interface designers, project managers, developers, and business units need a common set of look-and-feel guidelines to design and develop by. Coming up with these guidelines can be a long and difficult process. New GUI applications can't be held up for eight months while a task force argues about guidelines." There have been many books

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

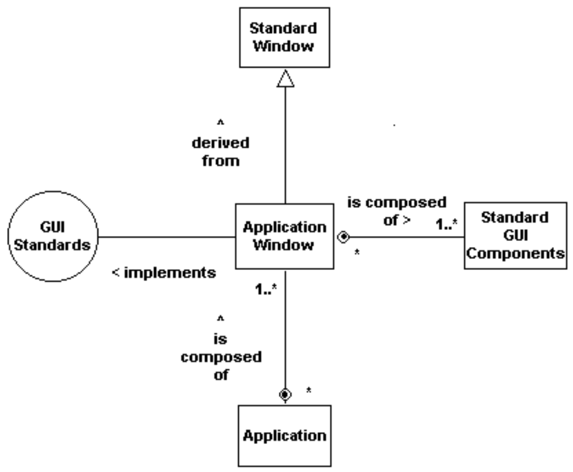
written on the subject of proper user interface design in general (Fowler, 1998; Galitz, 1997; Wood, 1998), and there are also books specifically devoted to certain platforms, such as Microsoft Windows (Cooper, 1995; Microsoft, 1995). Appendix A of Weinschenk's (1997) book also presents a list of user interface standards; *there are more than 300 items listed there*.

To enforce standards and to provide a flexible and adaptable programming base, an IT organization should adopt or develop a foundation for standards implementation. Figure 10.17 illustrates the concept of one such standards foundation with a unified modeling language (UML) diagram (Brandon, 2000).

In this example, an application is composed of a number of Application Windows. Each Application Window (i.e., PHP program, Java Server Page, Java Applet) is derived (inherited) from the Corporate Standard Window. The Application Window implements the GUI Standards Interface (or a window from which the Application was derived has implemented this interface). By interface, it is meant a Java interface or a C++ or PHP abstract class. The Application Window is composed of the Corporate Standard GUI Components. These standard components use the Corporate GUI Standards and these components may be derived from HTML/XML GUI objects, MFC, Java AWT, Java Swing, or third-party class libraries as long as these class library sources have been extended (customized) to use the data in the Corporate GUI Standards.

As well as implementing all the desired corporate standards, the standards foundation classes can also be constructed to encapsulate security mechanisms. If all application input is forced to take place through the standard foundation classes, then rogue programmers or contractors will be less likely be able to build-in "back doors," "Trojan horses," "time-bombs," and other illicit mechanisms into the system being developed. The standard foundation classes would be built and inspected by trusted sources, and those classes to be imported by other coders would be made final.

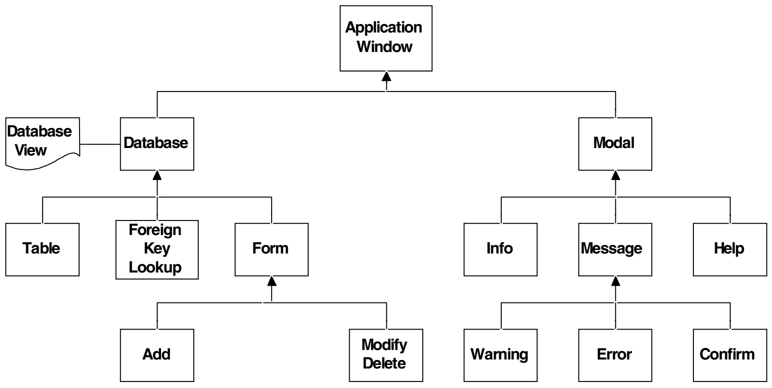
Figure 10.17. Standards foundation



Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

Figure 10.18. Standards implementation via window inheritance



Whereas Figure 10.17 shows a conceptual standards foundation, Figure 10.18 is an example of a typical implementation (via object-oriented inheritance) of a corporate standards foundation for windows in a business database type application. Each window would be further subclassed; for example, the general table would be subclassed for different entities such as employees, vendors, and so forth.

Chapter Summary

Earlier in this book, the many IT project management challenges were identified. These challenges can be met by a careful integration of modern project management and software engineering principles and practices. This was illustrated in Figure 1.4 in Chapter I; this chapter has discussed the quality aspects of project management. Critical IT project success factors were used as the basis for key quality metrics. A quality management plan for IT projects should include both verification and validation, and both of these processes should be on going throughout the project and controlled via a technique such as the quality stage gate process described herein. Other key quality topics were also discussed in this chapter, including the many types and methods of software testing, software development standards, and quality organizations and programs.

References

Akao, Y. (1990). *Quality function deployment: Integrating customer requirements into product design* (G. Mazur, Trans.). New York: Productivity Press.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

Athens, G. (2004, March). Model mania. *Computerworld*.

Brandon, D. (2000). An object oriented approach to user interface standards. *Challenges of information technology management in the 21st century* (pp. 753-756). Hershey, PA: Idea Group Publishing.

Brown, N. (1998). *Little book of testing: Vols. I & II.* Chesapeake, VA: Software Program Managers Network.

Collins, D. (1995). *Designing object oriented user interfaces*. Boston: Benjamin/ Cummings.

Cooper, A. (1995). *About face.* Boston: IDG Books.

Crosby, P. (1979). *Quality is free.* Princeton, NJ: McGraw-Hill.

Deming, W. (1982). *Out of the crisis.* MIT Press.

Dustin, E. (1999, September/October). Lessons in test automation. *Testing & Quality, 1*(5).

Foley, M. (2000, February 11). Can Microsoft squash 63000 bugs in Windows 2000. *Smart Reseller.*

Fowler, S. (1998). *GUI design handbook*. McGraw-Hill.

Galitz, W. (1997). *The essential guide to user interface design*. New York: Wiley.

Gause, D. & Weinberg, G. (1989). *Exploring requirements: Quality before design.* New York: Dorset House.

Hallows, J. (1998). *Information systems project management.* New York: American Management Association.

Harry, M. & Schroeder, R. (2000). *Six Sigma: The breakthrough management strategy revolutionizing the world's top corporations*. New York: Doubleday.

Hendrickson, E. (1999, May-June). Making the right choice. *Software Testing and Quality Engineering.*

IEEE/ANSI STD 1008. (2002). *IEEE standard for software unit testing.* Piscataway, NJ: IEEE Computer Society.

Jones, C. (1994). *Assessment and control of software risks*. Englewood Cliffs, NJ: Yourdon Press Computing Series.

Linger, R. (1994). Cleanroom process model. *IEEE Software, 11*(2)

McConnell, S. (1997). *Software project survival guide*. Seattle, WA: Microsoft Press.

Microsoft. (1995). *The Windows interface guidelines for software design.* Microsoft Press.

Mizuno, S. & Akao, Y. (1994). *QFD: The customer-driven approach to quality planning and deployment* (G. Mazur, Trans.). Newton Square, PA: Quality Resources.

PMI. (2000). *The project management body of knowledge (PMBOK).* ISBN 1-880410-22-2.

QA Quest. (1995, November). *The newsletter of the Quality Assurance Institute*. Orlando, FL.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG

The Quality Imperative. (1991, Bonus Issue, Fall). *Business Week.*

Shand, D. (2001, March 5). Six Sigma.*Computerworld,* p. 38.

Weinschenk, S. & Yeo, S. (1995). *Guidelines for enterprise wide GUI design.* New York: Wiley.

Weinschenk, S., Pamela J., & Sarah Y. (1997). *GUI design essentials.* Wiley.

Wiegers, K. (1999). *Software requirements.* Seattle, WA: Microsoft Press.

Wood, L. (1998). *User interface design.* Boca Raton, FL: CRC Press.

ZDNet. (2000, February). Retrieved from www.zdnet.com

Endnote

[1] The Malcolm Baldrige National Quality Award (presented by NIST – National Institute of Standards and Technology) was created by Public Law 100-107, signed into law on August 20, 1987. Principal support for the program comes from the Foundation for the Malcolm Baldrige National Quality Award, established in 1988. The award is named for Malcolm Baldrige, who served as Secretary of Commerce from 1981 until his tragic death in a rodeo accident in 1987.

Copyright © 2006, Idea Group Inc. Copying or distributing in print or electronic forms without written permission of Idea Group Inc. is prohibited.

TEAM LinG