UNIT –I

FUNDAMENTALS OF SOFTWARE QUALITY ASSURANCE

Software Quality Assurance

"What is not tracked is not done"

In software, so many things need to be done management cannot track all of them. So, some organization needs to do the tracking. That is the role of software quality assurance (SQA

SQA is designed to insure that officially established pro- cesses are being implemented and followed. Specifically, SQO insures that: •An appropriate development methodology is in place; •The projects use standards and procedures in their work; •Independent reviews and audits are conducted; •Documentation is produced to support maintenance and enhancement; •Documentation is produced during development and not after development; •Mechanisms are in place and used to control changes; •Testing emphasizes all the high-risk product areas; •Each software task is satisfactorily completed before the next one begins; •Deviations from standards and procedures are exposed ASAP;

•The project is audible by external professionals; •The quality control work is itself performed against estab- lished standards; •The SQA plan and the software development plan are compatible.

The **goals of SQA** are:

•To improve software quality by approximately monitoring both the software and the development process that pro- duces it; •To insure full compliance with the established standards and procedures for the software and the software pro- cess; •To insure that inadequacies in the product, the process, or the standards are brought to managements' attention so these inadequacies can be fixed.

THE ROLE OF SQA The people responsible for the software projects are the only ones who can be responsible for quality. The role of SQA is to monitor the way these groups perform their responsibilities. In doing this, there are several pitfalls: •It is a mistake to assume that SQA staff can do anything about quality; •The existence of SQA does not insure that standards and procedures will be followed; •Unless management demonstrates its support for SQA by following their recommendations, SQA will be ineffec- tive; •Unless line management requires that SQA try to resolve their issues with project management before escalation, SQA and development will not work together effectively.

All SQA can do is alert management to deviations from established standards and practices. Management must then insist that the quality problems be fixed before the software is shipped; otherwise, SQA becomes an expensive bureau- cratic exercise.

SQA RESPONSIBILITIES SQA can be effective when it reports through an independent management chain, when it is properly staffed, and when it sees its role as supporting the development and maintenance personnel in improving product quality. Then, SQA should be given the following responsibilities: •Review all development and quality plans for complete- ness; •Participate as inspection moderators in design and code inspections; •Review all test plans for adherence to standards; •Review a significant sample of all test results to deter- mine adherence to plans; •Periodically audit SCM performance to determine adher- ence to standards;

•Participate in all project quarterly and phase reviews and register non-concurrence if appropriate standards and procedures have not be reasonably met.

SQA FUNCTIONS Before establishing an SQA function, the basic organizational framework should include the following: •Quality assurance practices - Adequate development tools, techniques, methods, and standards are defined and available for Quality Assurance review; •Software project planning evaluation - If not defined at the outset, they will not be implemented; •Requirements evaluation - Initial requirements must be reviewed for conformance to quality standards; •Evaluation of the design process- •Evaluation of coding practices -

•Evaluation of the software integration and test process - •In-process evaluation of the management and project control process -

SQA REPORTING SQA reporting should not be under the software development manager. SQA should report to a high enough management level to have some chance of influencing priorities and obtaining enough resources to fix key problems. However, lower-level reporting normally results in better working rela- tionships with developers, while the ability to influence priori- ties is reduced.

Some general guidelines are: •SQA should not report to the project manager; •SQA should report somewhere within the local company or plant organization; •There should be no more than one management position between SQA and the senior location manager; •SQA should always have a “dotted-line” relationship to a senior corporate executive; •SQA should report to someone having a vested interest in software quality, like the staff head responsible for field services.

SQA CONSIDERATIONS •SQA organizations are rarely staffed with sufficiently experienced or knowledgable people because such peo- ple usually prefer development/design work, and man- agement often wants them in the latter, too; •The SQA management team often is not capable of negotiating with development. This depends on the cali- ber of the SQA team; •Senior management often backs development over SQA on a very large percentage of issues. Development then ignores he SQA issues, and SQA degenerates into a series of low-level, useless debates.

•Many SQA organizations operate without suitably docu- mented and approved development standards and pro- cedures; without such standards, they do not have a sound basis for judging developmental work, and every issue becomes a matter of opinion. Development also wins such generalized debates when schedules are tight. •Software development groups rarely produce verifiable quality plans. SQA is then trapped into arguments over specific defects rather than overall quality indicators. SQA may win the battle but lose the war.

SQA PEOPLE Getting good people into SQA can be a problem. Possible solutions include putting new hires there (but must also have experienced people there, too), rotating personnel through SQA (which may result in only poor developers being assigned there), and requiring that all new development man- agers be promoted from SQA after spending at least 6 months there (which can be very effective).

INDEPENDENT VERIFICATION AND VALIDATION In DoD contracts, independent verification and validation (IV&V) is often specified. The IV&V organization provides an independent assessment of the quality of the software. How- ever, do not confuse SQA and IV&V. SQA works for the developer; IV&V works for the customer.

Software Configuration Management

The most frustrating software problems are often caused by poor software configuration management (SCM). For exam- ple, a bug fixed at one time reappears; a developed and tested feature is missing; or a fully tested program suddenly doesn't work.

SCM helps to reduce these problems by coordinating the work products of many different people who work on a com- mon project. With such control, can get problems such as; •Simultaneous update - When two or more programmers work separately on the same program, the last one to make changes can easily destroy the other work. •Shared code - Often, when a bug is fixed in code shared by several programmers, some of them are not notified

•Common code - In large systems, when common pro- gram functions are modified, all the users need to know. •Versions - Most large programs are developed in evolu- tionary releases. With one release in customer use, one in test, and a third in development, bug fixes must be propagated between them.

These problems stem from a lack of control. The key is to have a control system that answers the following questions: •What is my current software configuration? •What is its status? •How do I control changes to my configuration? •How do I inform everyone else of my changes? •What changes have been made to my software? •Do anyone else's changes affect my software?

SOFTWARE PRODUCT NOMENCLATURE •System - The package of all the software that meet's the user's requirements. •Subsystem - Comprise large systems, such as communi- cations, display, and processing; •Product - Components of subsystems, such as control program, compliers, and utilities of an operating system. •Component - Components of a product, such as supervi- sor and scheduler of a control program. •Module - Lowest level of components is modules. Typi- cally implement individual functions that are relatively small and self-contained, such as queue management and interrupt dispatcher.

During implementation, two things are happening. First, the modules are being developed, enhanced, tested, and repaired from detailed design and implementation through system test. Second, the modules are being assembled into components, products, subsystems, and systems. During this building process, the modules are consistently being changed to add functions or repair problems. This process is supported by a hierarchy of tests: •Unit test - A separate test for each individual module; •Integration test - As the modules are integrated into com- ponents, products, subsystems, and systems, their inter- faces and interdependencies are tested to insure they are properly designed and implemented. •Function test - When integration results in a functionally operable build, it is tested in a component test, product test, subsystem test, and finally in a system test;

•Regression test - At each integration test (here called a spin), a new product is produced. This is the first tested to insure that it hasn't regressed, or lost functions present in a previous build.

Once an initial product is stabilized, a first baseline is estab- lished. Each baseline is a permanent database, together with all the changes that produced it. Only tested code and approved changes are in the baseline, which is fully pro- tected.

The key SCM tasks are: Configuration control Change management Revisions Deltas Conditional code.

CONFIGURATION CONTROL The task of configuration control revolves around one official copy of the code. The simplest way to protect every system revision is to keep a separate official copy of each version.

However, when two or more groups work on separate copies of the same or similar versions of common code, they often make different changes to correct the same problem. A good rule of thumb is that no two separate copies of a program can be kept identical. If separate copies exist, they must be assumed to differ; even if they were the same, they will soon diverge.

Only keep one official copy of any code that is used by sev- eral groups. Working copies may occasionally be used, but a common library must be the official source for all this com- mon code, and only officially approved changes can be per- mitted into this library.

REVISIONS Keep track of every change to every module and test case. There is one latest official version and every prior version is identified and retained; these obsolete copies can be used to trace problems.

A numbering system must separately identify each test, mod- ule, component, product, and system.

VERSIONS Often, several different functions can be implemented by the same module with only modest coding differences. For exam- ple, different memory management code may be needed to handle expansion beyond the standard 512K. Then, a differ- ent use a standard management module below 512K, and one using a mode switch beyond 512K.

Since these are different programs, they have different desig- nations, such as MEM and MEML. Each would have its own sequence of revisions and revision numbering schemes.

DELTAS Versions solves the problem of different functional needs for the same module but introduces multiple copies of the same code. The reason is that most of the code in the two modules would be identical. One, however, may have an additional routine to handle memory limits testing and mode switching (in the case of the 512K memory limits problem). Since they are stored as separate modules, however, there is no way to make sure all changes made to one are incorporated into the other.

One way to handle this is with deltas. This involves storing the base module (MEM) with those changes required to make it into MEML When maintenance is required on MEM, these changes can be made directly, so long as they do not interfere with the delta code. Changes to MEML are made to the delta, with MEM left alone.

There are disadvantages to this system. It is possible to have versions of both versions, so tracking get extremely compli- cated. If an element is lost or corrupted in a chain of deltas, it may be difficult to resurrect the entire chain. If deltas live a long time, they could grow into large blocks of code.

An answer is to use deltas only for temporary variations; then incorporate them into the baseline. However, the deltas have to be incorporated separately.

CONDITIONAL CODE Another way of handling slight variations between modules is to use some form of conditional program construction. For example, a billing program might use different functions depending on the need for a state sales tax. The source pro- gram would contain various tax versions, but none would be included in the final system unless called for at system instal- lation.

The use of conditional code simplifies code control because there is only one official copy of each module. The number of version combinations is also minimized.

There are disadvantages, however. The most important is that end users must specify all parameters and then perform a special and perhaps complex installation process. Also, thew system generation process becomes more complex with system growth.

BASELINES The baseline is the foundation for SCM. It provides the official standard on which subsequent work is based and to which only authorized changes are made. After an initial baseline is established and frozen, every subsequent change is recorded as a delta until the next baseline is set.

While a baseline should be established early in a project, establishing one too early will impose unnecessary proce- dures and slow the programmers' work. As long as program- mers can work on individual modules with little interaction, a code baseline is not needed. As soon as integration begins, formal control is needed.

BASELINE SCOPE. Items to be included in the implementation phase are: •The current level of each module, including source and object code •he current level of each test case, including source and object code •The current level of each assemble, compiler, editor, or other tool used •The current level of any special test or operational data •The current level of all macros, libraries, and files •The current level of any installation or operating proce- dures •The current level of operating systems and hardware, if pertinent

Retain every change, no matter how minor it seems.

BASELINE CONTROL Controlled flexibility is accomplished by providing the pro- grammers with private working copies of any part of the base- line. They can try new changes, conduct tests, etc. without disturbing anyone else. When ready, new changes can be incorporated into the baseline, after assuring the changes are compatible and no new code causes regressions.

Every proposed change must be tested against a trial version of the new baseline to make sure it does not invalidate any other changes

CONFIGURATION MANAGEMENT RECORDS Every change proposal is documented and authorized before being made. The documentation includes the reason for the change, the potential cost in time, the person responsible for the change, and the products affected.

Detailed records are especially crucial when hardware and software changes are made simultaneously. Just because the programs run doesn't mean they will continue to run if hardware changes.

Should always have problem reports, which document every problem and the precise conditions that caused it.

The expect list details every function and planned feature for every component in each new baseline.

CONFIGURATION MANAGEMENT RESPONSIBILITIES The configuration manager is the central control point for sys- tem changes and has the following responsibilities: •Develop, document, and distribute the SCM procedures; •Establish the system baseline, including backup provi- sions; •Insure that no unauthorized changes are made to the baseline; •Insure that all baseline changes are recorded in sufficient detail so the can be reproduced or backed out; •Insure that all baseline changes are regression tested; •Provide the focal point for exception resolution.

MODULE OWNERSHIP To insure integrity of modules, each module should have an owner. Of course, each person usually owns more than one module at a time.

The module owner's responsibilities are: •Know and understand the module design; •Provide advice to anyone who works on or interfaces with the module; •Serve as a technical control point for all module modifica- tions, including both enhancement and repair; •Insure module integrity by reviewing all changes and conducting periodic regression tests

Module ownership insures design continuity by providing a single focus for all module changes. Its disadvantages are that it depends on the skill and availability of individuals and it only provides design control at the detailed level. These dis- advantages can be countered by using a back-up "buddy" system between module owners and by maintaining an over- all design responsibility to monitor and control the software structure, interfaces, macros, and conventions.

THE CHANGE CONTROL BOARD On moderate through large projects, a change control board (CCB) (sometimes called the Configuration Control Board) is needed to insure every change is properly considered and coordinated.

The CCB should include some members from development, documentation, test, assurance, maintenance, and release. The CCB reviews each request for change and approves it, disapproves it, or requests more information.

Depending on project size, several CCBs may be needed, each with expertise authority over a specific area. Some examples are: overall design and module interfaces, the con- trol program, the applications component, user interfaces, and development tools. With multiple CCBs, a system-level CCS is needed to resolve disputes between these lower-level boards.

A CCB typically needs the following information on each pro- posed change: •Size - How many new/changed LOC? •Alternatives - How else can it be done? •Complexity - Is the change within a single component or does it involve others? •Schedule - When? •Impact - What are future consequences? •Cost - What are potential costs and savings? •Relationship with other changes - Will another change supersede or invalidate this one, or does it depend on other changes? •Test - Are there special test requirements? •Resources - Are the needed people available to do the work? •System impact - What are the memory, performance

•Benefits - What are the expected benefits? •Politics - Are there special considerations such as who is requesting the change or whom it will affect? •Change maturity - How long has the change been under consideration?

If the change is to fix a customer-related problem, other infor- mation may be required: •A definition of the problem the change is intended to fix; •The conditions under which the problem was observed; •A copy of the trouble report; •A technical description of the problem; •The names of other programs affected

CCB PROBLEMS Do not waive a review just because change activity has become a bottleneck. It is precisely at the time of heaviest change and testing activity that loss of control is most likely and CCB review is most needed.

Select CCB members with care. They have the power to block any part of the project so these assignments should not be treated lightly. The project’s software development man- ager should personally chair the highest-level CCB.

THE NEED FOR AUTOMATED TOOLS Without automated tools, changes can easily be lost or done without proper procedures.

UNIT -II

Managing Software Quality

1.MANAGING SOFTWARE ORGANISATION

Commitment is the essential foundation for large scale project where many professionals coordination are involved.commitment is an agreement by one person to do something for another with a planned completion date and some consideration or payment.

Commitment Discipline

Commitment discipline is the foundation of software project management which focuses on

Ensuring that the organisation meets its commitment which is satisfied by committed people.

Commitment + Discipline = Success

Making a Commitment :

The overall mission for a commitment culture is "Keep your Promises".In Short ,in a commitment culture each individual actively seeks new commitments ,agree on these,and do his utmost to fulfil them.We have chosen to use the concept of commitment as the underlying principle for learning as it gives the student a realistic ,professional and challenging approach to their future profession.

- Voluntariness
- Agreement
- Infrastructure
- Openness
- Fulfillment
- Renegotiation

The Commitment Hierarchy:

As long as the professionals felt that the commitment are satisfied,the commitments are not done even the commitments are met by commited individuals.

If the commitment is not met,then the management team takes care in making commitment and then insists on extraordinary efforts to meet the committed individuals.

The Software commitment process:

The effective software commitment process must reach the top of the organisation.The senior executive's personal involvement is what motivates the entire commitment process.

Requirement:

The requirements of the software commitment process are

- All commitment are made by the senior exeutive
- Commitment are made only after the formal review and concurrent process.
- Enforcement mechanism is needed to conduct the review.

Establishing a commitment process:

A commitment process is established by the senior executive who is willing to insist that the required planning be done before any commitment is made.

Training course is required for the people to know how to make the schedules and estimates

As per the specific estimationg,review and approval procedures,Once the senior executive is decided to implement, then those items are readily accomplished.

MANAGING SOFTWARE QUALITY

Software product quality is the key measure ofthe software process to evaluate as softwareorganization.It provides a clear record of development progress,a basis for setting objectives and a framework for current action.

Basic Quality Principles:

1.Unless aggressive quality goals are established ,nothing will change.

2.If these goals are not numerical ,the quality program will remain just talk.

3.without quality plans,only you are committed to quality.

4.Quality plans are just paper unless plans are tracked and reviewed.

Measurement Criteria

Quality measures fall in to following classes

1. Development
   - Defects
   - Change activity
2. Product
   - Error Seeding
   - Software structure
   - Controlled tests
3. Acceptance
   - Problem
   - Install efoort

4.Usage
- Problems
- Operating efforts
- Surveys
- Availability

5.Repair
- Defects
- Repair effort

Establishing Software Quality program

The steps that are needed to establish a software quality program are listed as follows

1.Senior management establishes aggressive and explicit numerical quality goals.Without numerical measures,the quality effort will be just another motivational program with little lasting impsct.

2. The quality measures used are objective ,requiring a minimum of human judgement.

3.These measures are precisely defined and documented so computer programs can be written to gather and process them.

4.A quality plan is produced at the beginning of each project.This plan commits to specific numerical targets and it is update at every significant project change and milestone.

5.These plans are reviewed for compliance with management quality goals.

6.Quality performance is tracked and published.

7.Since no single measures can adequately represent a complex product,the quality measures are treated as indicators of overall performance.

Estimating Software Quality

The following factors are considered to estimate the software quality

1.Customer installation rate for the product

2.Product release history

3.Distribution plan

Removal Efficiency

Removal efficiency is used to calculate the overall efficiency of the development process which indicates the cumulative percent of the previously injected errors that have been removed by the end of each project phase.Since defect removal costs can be expected to roughly double with each project phase,attention should be focused on early removal.

Quality goals

Senior management in every organisation must establish its own quality goals and clearly state the goals to the people.A reasonable starting point would be:

1.Every new product or product release must have better quality than its predecesoor.

2.The corporate quality organisation,with the assistance of the software groups, will establish the quality measures to be used.

3.Each product manager is responsible for producing a documented quality plan to meet these goals.

4.Product quality performance will be judged by:

The degree to which quality plan shows improvement

The effectiveness of the action plans to address areas of deficient performance

Quality plans

The quality plan documents

1.The quality actions management intends to implement

2.The deviations of the quality measures.

3.The identifications of the planned process changes.

4.The anticipated quality improvements.

* Refactoring tools
* QVT or Model transformation Tools
* Configuration management tools including revision control

Preventing, Discovering and Removing Defects

To reduce the number of defects delivered with a software project an organization can engage in a variety of activities. While defect prevention is much more effective and efficient in reducing the number of defects, most organization conduct defect discovery and removal. Discovering and removing defects is an expensive and inefficient process. It is much more efficient for an organization to conduct activities that prevent defects.

Defect Removal Efficiency

If an organization has no defect prevention methods in place then they are totally reliant on defect removal efficiency.

1. Requirements Reviews up to 15% removal of potential defects
2. Design Reviews up to 30% removal of potential defects
3. Code Reviews up to 20% removal of potential defects
4. Formal Testing up to 25% removal of potential defects

In other words, if your organization is great at defect removal the maximum percentage of defects your organization can expect to remove is 90%. If a software project is 100 function points, the total number of maximum (or potential) defects could be 120. If you were perfect at defect removal your project would still have up to 12 defects after all your defect discovery and removal efforts. The far majority of organization would receive a B (medium) or even a D (poor) at defect removal efficiency.

Activity	Perfect	Medium		Poor
Requirements Reviews		15%	5%	0%
Design Reviews	30%	15%	0%	
Code Reviews	20%	10%	0%	
Formal Testing	25%	15%	15%	
Total Percentage Removed	90%	45%	15%	

Defect Discovery and Removal

Size in Function Points

		Totals Defects Remaining			
Points	Max Defects	Perfect	Medium		Poor
100	120	12	66	102	
200	240	24	132	204	
500	600	60	330	510	
1,000	1,200	120	660	1,020	
2,500	3,000	300	1,650	2,550	
5,000	6,000	600	3,300	5,100	
10,000	12,000	1,200	6,600	10,200	

20,000	24,000	2,000	13,200	20,400

An organization with a project of 2,500 function points and was about medium at defect discovery and removal would have 1,650 defects remaining after all defect removal and discovery activities. The calculation is 2,500 x 1.2 = 3,000 potential defects. The organization would be able to remove about 45% of the defects or 1,350 defects. The total potential defects (3,000) less the removed defects (1,350) equals the remaining defects of 1,650.

Defect Prevention

If an organization concentrates on defect prevention (instead of defect detection) then the number of defects inserted or created is much less. The amount of time and effort required to discover and remove this defects is much less also.

1. Roles and Responsibilities Clearly Defined up to 15% reduction in number of defects created
2. Formalized Procedures up to 25% reduction in number of defects created
3. Repeatable Processes up to 35% reduction in number of defects created
4. Controls and Measures in place up to 30% reduction in number of defects created

Imagine an organization with items 1 and 2 in place. A project with 100 function points would have a potential of 120 defects, but since they have preventative measures in place, they can reduce the number of potential defects by 48 (40% = 25% + 15%). That makes the potential number of defects 72 compared to 120 with no preventative efforts. Assuming that an organization was medium at defect discovery and removal they could remove 45% of the remaining defects or have 40 remaining when the project rolled to production.

Defect Removal	Max Defects	Prevention	Medium		
100	120	72	40		
200	240	144	79		
500	600	360	198		
1,000	1,200	720	396		
2,500	3,000	1,800	990		
5,000	6,000	3,600	1,980		
10,000	12,000		7,200	3,960	
20,000	24,000		14,400		7,920

The above table represents the number of defects that an organization that does items 1 and 2 above and is medium at discovery and removal.

The problem for estimating defects is multidimensional. First the total number of defects must be estimated. Second the impact of defect prevention needs to be understood and the estimated number of defects adjusted. Third an assessment needs to be done to understand how many defects can be discovered and removed by an organization.

Clearly, the fewer number of defects that an organization must discover and remove the better. The way this is accomplished is by better process, a more stable organization and repeatable processes. The focus of software organizations needs to be on defect prevention instead of defect detection.

Software Quaity Assurance Management

- The aim of Software Quality Management (SQM) is to manage the quality of software and of its development process.
- A quality product is one which meets its requirements and satisfies the user
- A quality culture is an organizational environment where quality is viewed as everyone's responsibility.

Software Quality Plan (SQP) layer

A project level quality plan written by each project for declaring project commitment to follow an applicable set of standards, regulations, procedures and tools during the development lifecycle. In addition, SQP should contain quality goals to be achieved, expected risks and risk management. SQP sources are derived from

- SQA components that are adopted as is or customized to the project's needs
- New procedures, standards and tools complementing missing or not-applicable SQA components that have been written in particular for the project, or imported from outside the organization.

Any deviation of an SQP from SQA should be justified by the project manager and be confirmed by the company management.

Software Quality Control (SQC) layer

Ensures in-process that both SQA and SQP are being followed by the development teams.

SQC activities include

- Mentoring how to produce artifacts, such as well-defined engineering documents using standard templates
- Mentoring how to conduct standard processes, such as quality reviews
- Perform in-process quality reviews to verify, evaluate and confirm artifacts
- Verify and evaluate to improve the use of methods, procedures and adopted software tools

SQM Roles

- to ensure that the required level of quality is achieved in a software product
- to encourage a company-wide "Quality Culture" where quality is viewed as everyone's responsibility
- to reduce the learning curve and help with continuity in case team members change positions within the organization
- to enable in-process fault avoidance and fault prevention through proper development

Many people use the terms SQM and SQA (Software quality assurance) interchangeably.

Software quality management

Software quality management can be realized in various ways depending on organization and type of realized project,[2] but it should support whole software development lifecycle, meaning:

- Collecting requirements and defining scope of IT project, focused on verification if defined requirements will be testable. One of the products can be test strategy.
- Designing the solution, focused on planning test process e.g. what type of tests will be performed, how they will be performed in context of test environments and test data. One of the products can be test plan including test schedule.
- Solution implementation supported by creating test cases and scenarios, executing them and registering defects including coordination of fixing them. Products can be test cases and scenarios, reports from test iteration realization.
- Change management, supported by verification how planned changes can influence the quality of created solution and eventual change of test plan. One of the products can be changes in test plan, test cases and scenarios.
- Closing project, supported by realization number of tests focused on complex verification of overall quality of created solution. It can include System Integration Tests, User Acceptance Tests and Operational Acceptance Tests. One of the products can be recommendation about production start of the system.[3]

UNIT-III

SOFTWARE QUALITY ASSURANCE METRICS

Software Quality:

Kitchen ham (1989 b) refers to software quality "fitness for needs" and claims quality involves matching expectations.

Two features of a piece of quality software:

-> Conformance to its specification
-> Fitness for its intended purpose.

The Department of Defense (DOD, 1985) in the USA defines software quality as "the degree to which the attributes of the software enable it to perform its intended end use".

Software product quality

- Product quality
  - conformance to requirements or program specification; related to Reliability
- Scalability
  - Correctness
- Completeness
- absence of bugs
- Fault-tolerance
  - Extensibility
  - Maintainability
- Documentation

Software was particularly problematical for the following reasons:

-> Software has no physical existence
-> The lack of knowledge of client needs at the start
-> The change of client needs over time
-> The rapid rate of change on both hardware and software
-> The high expectations of customers, particularly with respect to adaptability.

Within the software quality area, the need to provide a solution that matches user needs is often considered as "design quality", whilst ensuring a match to the specification is considered as "manufacturing quality".

Software Quality Factors

A software quality factor is a non-functional requirement for a software program which is not called up by the customer's contract, but nevertheless is a desirable requirement which enhances the quality of the software program.

Understandability

Completeness
Maintainability
Conciseness
Portability
Consistency
Testability
Usability
Reliability
Structured ness
Efficiency
Security

Views of Quality:

Quality is a multidimensional construct. It may therefore be considered using a polyhedron metaphor. Within this metaphor, a three-dimensional solid represents quality. Each face represents a different aspect of quality such as correctness, reliability, and efficiency.

It has been classified according to a number of 'views' or perspective. These views are often diverse and may conflict with each other. Each view comes from a particular context.
The views are generally presented in adversarial pairs such as versus designers.

The software project has the following roles

Project manager
Business analyst
Implementation programmer
Quality auditor
End user
Line manager
Project sponsor

Views of Quality

User	Designer
What I Want	Good Specification
Fast response	technically correct
Control information	Fits within systems structure
Easy to use help menus	Easy to maintain
Available as required	Difficult for user to manage
Exception data	Fast development
Reacts to business change	Low maintenance
Input data once	well documents

In an attempt to classify different and conflicting views of quality, Garvin (1984) has suggested five different views of quality

1. The transcendent view
   - Innate excellence
   - Classical definition
2. The product-based view
   - Higher the quality higher the cost
   - Greater functionality
   - Greater care in development
3. The user-based view
   - Fitness for purpose
   - Very hard to quantify
4. The manufacturing view
   - Measures quality in terms of conformance
   - Zero defects
5. The value-based view
   - Provides the data with what the customer requires at a price.

Quality is determined by people because

- It is people and human organizations who have problems to be solved by computer software
- It is people who define the problems and specify the solutions
- It is still currently people who implement designs and product code.
- It is people who test code

HIERARCHICAL MODEL OF QUALITY:

To compare quality in different situations, both qualitatively and quantitatively, it is necessary to establish a model of quality.

Many model suggested for quality.
Most are hierarchical in nature.
A quantitative assessment is generally made, along with a more quantified assessment.

Two principal models of this type, one by Boehm (1978) and
one by McCall in 1977. A hierarchical model of software quality is based upon a set of quality criteria, each of which has a set of measures or metrics associated with it.

The issues relating to the criteria of quality are:

-> What criteria of quality should be employed?
-> How do they inter-relate?
-> How may the associated metrics be combined into a meaningful overall measure of Quality?

THE HIERARCHICAL MODELS OF BOEHM AND MCCALL

THE GE MODEL (MCCALL, 1977 AND 1980) / (McCall Model)

-> This model was first proposed by McCall in 1977.

-> It was later revised as the MQ model, and it is aimed by system developers to be used during the development process.

-> In early attempt to bridge the gap between users and developers, the criteria were chosen in an attempt to reflect user' s views as well as developer' s priorities.

-> The criteria appear to be technically oriented, but they are described by a series of questions which define them in terms to non specialist managers.

The three areas addressed by McCall's model (1977):

Product operation: requires that it can be learnt easily, operated efficiently And it results are those required by the users.

Product revision: it is concerned with error correction and
Adaptation Of the system and it is most costly part of software development.

Product transition: it is an important application and it is
distributed processing and the rapid rate of change in hardware is Likely to increase.

McCall' s criteria of quality defined

Efficiency is concerned with the use of resources e.g. processor time, storage. It falls into two categories: execution efficiency and storage efficiency.

Usability is the ease of use of the software.

Integrity is the protection of the program from unauthorized access.

Correctness is the extent to which a program fulfils its specification.

Reliability is its ability not to fail.

Maintainability is the effort required to locate and fix a fault in the program within its operating environment.

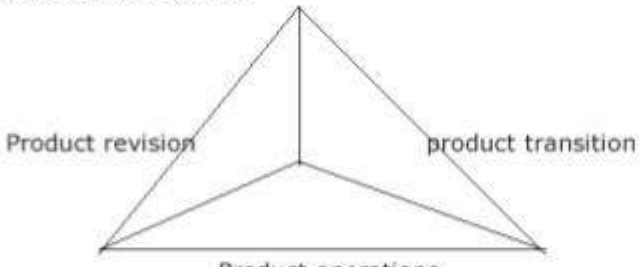
Flexibility is the ease of making changes required by changes in the operating environment.

Testability is the ease of testing the programs, to ensure that it is error-free and meet its specification.

Portability is the effort required to transfer a program from one environment to another.

Reusability is the ease of refusing software in a different context.

Interoperability is the effort required to couple the system to another system.



The GE model after McCall (1977)

The Boehm model (1978)

It is to provide a set of well-defined, well-differentiated characteristics of software quality.

It is hierarchical in nature but the hierarchy is extended, so that quality criteria are subdivided.

According to the uses made of the system and they are classed into ' general' or ' as is' and the utilities are a subtype of the general utilities, to the product operation.

There are two levels of actual quality criteria, the intermediate level being further split into primitive characteristics which are amenable to measurement.

This model is based upon a much larger set of criteria than McCall' s model, but retains the same emphasis on technical criteria.

The two models share a number of common characteristics are.

- The quality criteria are supposedly based upon the user's view.
- The models focus on the parts that designers can more readily analyze.
- Hierarchical models cannot be tested or validated. It cannot be shown that the metrics accurately reflect the criteria.
- The measurement of overall quality is achieved by a weighted summation of the characteristics.

Boehm talks of modifiability where McCall distinguishes expandability from adaptability and documentation, understandability and clarity.

HOW THE QUALITY CRITERIA INTERRELATE

The individual measure of software quality provided do not provide an over all measure of software quality.

The individual measures must be combined.

The individual measures of quality may conflict with each other.

Some of these relationships are described below;

Integrity vs. efficiency (inverse) the control of access to data or software requires additional code and processing leading to a longer runtime and additional storage requirement.

Usability vs. efficiency (inverse) Improvements in the human / computer interface may significantly increase the amount of code and power required.

Maintainability and testability vs. efficiency (inverse) Optimized and compact code is

not easy to maintain.

Portability vs. efficiency (inverse) the use of optimized software or system utilities will lead to decrease in probability.

Flexibility and reusability vs. integrity (inverse) the general flexible data structures required for flexible and reusable software increase the security and protection problem.

Interoperability vs. integrity (inverse) Coupled system allow more avenues of access to more and different users.

Reusability vs. reliability (inverse) reusable software is required to be general: maintaining accuracy and error tolerance across all cases is difficult.

Maintainability vs. flexibility (direct) maintainable code arises from code that is well structured.

Maintainability vs. reusability (direct) well structured easily maintainable code is easier to reuse in other programs either as a library of routines or as code placed directly within another program.

Portability vs. reusability (direct) portable code is likely to be free of environment-specific features.

Correctness vs. efficiency (neutral) the correctness of code, i.e. its conformance to specification does not influence its efficiency.

MEASURING SOFTWARE QUALITY

MEASURING QUALITY

Quality measurement, where it is considered at all, is usually
expressed in terms of metrics.

Software metric is a measurable property which is an indicator of one or more of the quality criteria that we are seeking to measure. As such, there are a number of conditions that a quality metric must meet. It must:

-> Be clearly linked to the quality criterion that it seeks to measure
-> Be sensitive to the different degrees of the criterion
-> Provide objective determination of the criterion that can be mapped onto a suitable scale.
-> Metrics are not same as direct measures.

Measurement techniques applied to software are more akin to the social sciences, where properties are similarly complex and ambiguous.

A typically measurable property on which a metric may be based is structured ness.

Structured ness as it simplest may be calculated in terms of the average length of code modules within the programs.

$$\text{Structured ness} \propto \text{modularity lines of code} \propto \frac{\text{lines of code}}{\text{Number of modules}}$$

SOFTWARE METRICS

Metrics are classified into two types according to whether they are predictive or descriptive.

A predictive metric is used to make predictions about the software later in the lifecycle. Structured ness is used to predict the maintainability of the software product in use.

A descriptive metric describes the state of the software at the time of measurement.

Different authors have taken different approaches to metrics.

Structured ness is measured by questions such as:

-> Have the rules for transfer of control between modules been followed?(y/n)
-> Are modules limited in size?(y/n)
-> Do all modules have only one exit point ?(y/n)
-> Do all modules have only one entry point?(y/n)
-> A well-structured program will produce positive answers to such questions.

McCall's approach is more quantities, using scores derived from equations such as

$$\text{McCall s structured ness metric} = \frac{n01}{ntot}$$

Where: n01 = no of modules containing one or zero exit points only ntot = total number of modules

Generally, in this approach, scores are normalized to a range between 0 and 1, to allow for easier combination and comparison.

This appears attractive, to give unjustified credibility to the results obtained.

To validate this relationship and determine whether it is a linear relationship or more complex in nature.

It is also possible to validate whether the dependence of maintainability structured ness in identical to that of adaptability or reusability.

What makes a good metric?

Seven criteria for a good metric, after Watts (1987)

Objectivity the results should be free from subjective influences. It must not matter who the measurer is.

Reliability the results should be precise and repeatable.

Validity the metric must measure the correct characteristic.

Standardization the metric must be unambiguous and allow for comparison.

Comparability the metric must be comparable with other measures of the same Criterion.

Economy the simpler and therefore, the cheaper the measure is to use, the Better.

Usefulness the measure must address a need, not simply measure a property for its own sake.

A further important feature is consistency.

Automation is also desirable.

Metrics cited in the literature:

Metrics available for each criterion (after Watts, 1987) \

Quality criteria	number of metrics cited
Maintainability	18
Reliability	12
Usability	4
Correctness	3
Integrity	1
Expandability	1
Portability	1
Efficiency	0
Adaptability	0
Interoperability	0
Reusability	0

The metrics cited depends to a very large extent upon just seven distinct measurable properties: readability, error prediction, error detection, complexity, and mean time to failure (MTTF), modularity, testability.

1. Readability as a measure of usability may be applied to documentation in order to assess how such documentation may assist in the usability of a piece of software.

2. Error prediction as a measure of correctness this measure is depends upon the stable software development environment.

3. Error detection as measure of correctness

4. Mean time to failure (MTTF) as a measure of reliability

5. Complexity as a measure of reliability the assumption underpinning these measures is that as complexity increases, so reliability decrease.

6. Complexity as a measure of maintainability is also indicative of maintainability.

7. Readability of code as a measure of maintainability has also been suggested as a measure of maintainability.

8. Modularity as a measure of maintainability increased modularity is generally assumed to increase maintainability. Four measures have been suggested. Yau and Collofello (1979) measured " stability" as the number of modules affected by program modification. Kentger (1981) defined a four-level hierarchy of module types:

Control modules.
Problem-oriented modules.
Management modules for abstract data.
Realization modules for abstract data.

9. Testability as a measure of maintainability the ease and effectiveness of testing will have an impact upon the maintainability of a product.

An overall measure of quality

Much of the work in this area has been concerned with simple reduction of a set of scores to a single 'figure-of-merit'.

Five such methods are detailed by Watts (1987) as part of the MQ approach.

1. Simple scoring: In this method, each criterion is allocated a score. The overall quality is given by the mean of the individual scores.
2. Weighted scoring: This scheme allows the user to weight each criterion according to how important they consider them to be. Each criterion is evaluated to produce a score between 0 and 1. Each score is weighted before summation and the resulting figure reflects the relative importance if the different factors.
3. Phased weighting factor method: This is an extension of weighted scoring. A weighting is assigned to a group characteristics before each individual weighting is considered.

Total Quality Management

What is Total Quality Management?

TQM is a management philosophy, a paradigm, a continuous improvement approach to doing business through a new management model. The TQM philosophy evolved from the continuous improvement philosophy with a focus on *quality* as the main dimension of business. Under TQM, emphasizing the quality of the product or service predominates. TQM expands beyond statistical process control to embrace a wider scope of management activities of how we manage people and organizations by focusing on the entire process, not just simple measurements.

TQM is a comprehensive management system which:

- Focuses on meeting owners'/customers' needs by providing quality services at a cost that provides value to the owners/customers
- Is driven by the quest for continuous improvement in all operations
- Recognizes that everyone in the organization has owners/customers who are either internal or external
- Views an organization as an internal system with a common aim rather than as individual departments acting to maximize their own performances
- Focuses on the *way* tasks are accomplished rather than simply *what* tasks are accomplished
- Emphasizes teamwork and a high level of participation by all employees

TQM beliefs

Presented here are universal total quality management beliefs.

- Owner/customer satisfaction is the measure of quality
- Everyone has owners/customers; everyone is an owner/customer
- Quality improvement must be continuous
- Analyzing the processes used to create products and services is key to quality improvement
- Measurement, a skilled use of analytical tools, and employee involvement are critical sources of quality improvement ideas and innovations
- Sustained total quality management is not possible without active, visible, consistent, and enabling leadership by managers at all levels
- If we do not continuously improve the quality of products and services that we provide our owners/customers, someone else will

Characteristics of Successful TQM Companies

The construction industry has arrived late to TQM, probably due to the tendency to easily brush aside anything in management that is new, or to dismiss TQM as a fad.

Continuous improvement is not a fad but a necessary part of management's obligation to properly run its company. Gone are the boom days when quality did not matter due to the volume of work available and the ease of obtaining work. The attitude of construction managers and contractors was simply to *add it to the bill, because the owner will pay for it*. In other words, in those boom days *Cost plus Profit equaled Price*. Now, however, the new attitude is *Price minus Cost equals Profit*. Owners are now demanding higher quality work, and at a lower cost. In attempting to keep pace with the new attitude, a quality management system that helps keep costs down is well worth implementing.

The characteristics that are common to companies that successfully implement TQM in their daily operations are listed here.

- Strive for owner/customer satisfaction and employee satisfaction
- Strive for accident-free jobsites
- Recognize that the owner/customer provides the revenue while the employees are responsible for the profit
- Recognize the need for measurement and fact-based decision making
- Arrange for employees to become involved in helping the company improve
- Train extensively
- Work hard at improving communication inside and outside the company
- Use teams of employees to improve processes
- Place a strong emphasis on the right kind of leadership, and provide supervisors with a significant amount of leadership training
- Involve subcontractors and suppliers, requiring them to adopt TQM
- Strive for **continuous** improvement

Quality principles that successful TQM companies recognize

The quality principles that successful TQM companies recognize and attempt to continually incorporate into their actions are the following:

- People will produce quality goods and services when the meaning of quality is expressed daily in their relations with their work, colleagues, and organization.
- Inspection of the *process* is as important as inspection of the *product*. Quality improvement can be achieved *by the workers closest to the process.*
- Each system with a certain degree of complexity has a *probability of variation*, which can be understood by scientific methods.
- Workers work *in* the system to improve the system; *managers work on the system to improve the system.*
- Total quality management is a strategic choice made by top management, and must be *consistently translated* into guidelines provided to the whole organization.
- Envision what you desire to be as an organization, but *start working from where you actually are.*
- Studies have indicated that people like working on a quality-managed jobsite especially due to the cleaner site and safer place to work.

- Accept the responsibility for quality. Establish datums for measurement.
- Use the principle of *get it right, the first time, every time*.
- Understand that quality is a journey, not a destination. It consists of steps that form a process that is continuous.

Quality improvement team tasks are . . .

Quality improvement teams are active in the following task areas:

- Identify the customers of the process
- Determine customer expectations
- Flowchart the process
- Identify all of the inputs and interfaces
- Identify the output(s)
- Systematically review the procedures currently being used in the process
- Collect and analyze available quantitative data
- Determine the need for additional data
- Identify the problem(s)
- Determine the root cause of the problem
- Determine potential solutions
- Select a trial solution
- Present recommendations to the steering committee
- Implement the solution on a pilot-project basis
- Analyze the data to discern if there has been improvement
- The quality improvement team (QIT) is responsible for planning and managing the TQM implementation process for the organization. The QIT is responsible for **making TQM happen**.
- The QIT must take the lead in managing the cultural changes that TQM will require.
- At least one member of the Senior Management Team (SMT) should be a member of the QIT.
- The QIT reports to the Chief Executive Officer (CEO) and the SMT.
- The QIT and SMT should hold a joint meeting to review the TQM effort at least quarterly.

Quality improvement team structure is . . .

A quality improvement team (QIT) meets on a regular basis. During the first months of the TQM effort, the QIT will probably wish to meet once per week for 3 to 5 hours. In addition, QIT members will spend an additional 6 to 10 hours per week on training, education, and QIT assignments. After the TQM implementation plan is complete and underway, the QIT should meet once or twice per month.

The QIT determines and defines the duties of each QIT member. Typical positions for members are TQM Director or Coordinator; Communications, Education, and Training Coordinator; Quality Measurement; Customer Satisfaction; and Employee

Involvement and Satisfaction. The types of quality improvement teams are permanent, temporary, preparation, planning, and implementation. Facilitators hold an extremely important role in the success of implementation.

In the order presented here, the QIT plans the ongoing TQM education for: (1) QIT and SMT members, (2) facilitators and supervisors, and (3) all employees.

The QIT reviews, approves and helps implement quality improvement plans, establishes the TQM organizational structure and team structure for the company, procures (from management) and manages the resources required for TQM implementation, solicits and evaluates Quality Improvement Opportunities (QIOs), selects and commissions teams to work on QIOs, and proffers rewards and recognition.

The permanent teams of the quality improvement process are based on the following areas of concern:

- Key Objective Teams
  - Customer Satisfaction
  - Employee Morale/Satisfaction
  - Stakeholder Relationships
- Functional Area QI Teams
- Employee Involvement Teams
- Natural Work Teams

The temporary teams of the quality improvement process are:

- Task Force QITs
- Project Quality Teams

The facilitators of the quality improvement process should retain the following characteristics:

Qualities of a good facilitator:

Is respected by people at all levels of the organization

Is organized

Is a good listener and communicator

Understands TQM principles and philosophies

Is objective and open-minded

Is a team player, one who likes to accomplish things through others

Roles and duties

Organizes team meetings

Keeps meetings on track

Is record keeper

Procures needed resources and outside support

Communicates progress to QIT

SOFTWARE QUALTIY METRICS

A definition of software quality metrics is:-

A measure of some property of a piece of software or its specifications.

Basically, as applied to the software product, a software metric measures (or quantifies) a characteristic of the software.
Some common software metrics (discussed later) are:-

- Source lines of code.
- Cyclomatic complexity, is used to measure code complexity.
- Function point analysis (FPA), is used to measure the size (functions) of software.
- Bugs per lines of code.
- Code coverage, measures the code lines that are executed for a given set of software tests.
- Cohesion, measures how well the source code in a given module work together to provide a single function.
- Coupling, measures how well two software components are *data* related, i.e. how independent they are.

The above list is only a small set of software metrics, the important points to note are:-

- They are all measurable, that is they can be quantified.
- They are all related to one or more software quality characteristics.

The last point, related to software characteristics, is important for software process improvement. Metrics, for both process and software, tell us to what extent a desired characteristic is present in our processes or our software systems. **Maintainability** is a desired characteristic of a software component and is referenced in all the main software quality models (including the ISO 9126). One good measure of maintainability would be *time required to fix a fault*. This gives us a handle on maintainability but another measure that would relate more to the *cause* of poor maintainability would be *code complexity*. A method for measuring code complexity was developed by **Thomas McCabe** and with this method a quantitative assessment of any piece of code can be made. Code complexity can be specified and can be known by measurement, whereas *time to repair* can only be measured after the software is in support. Both *time to repair* and *code complexity* are software metrics and can both be applied to software process improvement.

From our previous definition of SQA and SQC, we now see the importance of measurement

(metrics) for the SDLC and SPI. It is metrics that indicate the value of the standards, processes, and procedures that SQA assures are being implemented correctly within a software project. SQA also collectsrelevant software metrics to provide input into a SPI (such as a CMMi continuous improvement initiative). This exercise of constantly measuring the outcome, then looking for a causal relationship to standards, procedures and processes makes SQA and SPI pragmatic disciplines.

The whole process of setting up a SDLC, then selecting the correct metrics and then establishing causal relationships to parts of the SDLC is **more of an art than a science**. It is for this reason that there a few, if any, off the shelf answers to SQA, SQC and SPI. It is a question of where are the risks and challenges of a given environment. For example if you have one version of a system and this runs on a central server, then configuration issues are unlikely and you are less likely to be concerned with **Portability** issues than someone who targets multiple platform, multiple versions is concerned with portability (and configuration management).

That said the following section tries to pull the ideas of quality metrics, quality characteristics, SPI, SQC and SQA together with some examples by way of clarifying the definition of these terms.

The Software Assurance Technology Center (SATC), NASA, Software Quality Model includes Metrics

The table below cross references Goals, Attributes (software characteristics) and Metrics. This table is taken from the Software Assurance Technology Center (SATC) at NASA. Although the software quality model has different quality characteristics than those previously discussed on this website, namely ISO 9126, the relationship with Goals lends itself to giving examples of how this could be used in CMMi. If you look at the other quality models they have a focus on what comes under the *Product (Code) Quality* goal of the SATC model.

The SATC model of documenting metrics is useful for implementing the CMMi SPI for these reasons:-

- The SATC model includes goals for processes, (i.e. Requirements, Implementation and Testing).
- The SATC model can be used to reference all of the CMMi software engineering process areas, for example Requirements management (which includes traceability).
- If desired the SATC model can be expanded to accommadate grater risk mitigation in the specified goal areas, or other goal areas can be created.
- Demonstrating the relationship of metrics to quality characteristics and SPI (CMMI) is well served by the SATC quality model.

The SATC Software Quality Model (which includes Goals and Metrics as well as the software attributes)

GOALS	ATTRIBUTES	METRICS
	Ambiguity	Number of Weak Phrases. Number of Optional Phrases.
	Completeness	Number of To Be Determined (TBDs) and To be Added (TBAs).
Requirements Quality	Understandability	Document Structure. Readability Index.
	Volatility	Count of Changes / Count of Requirements. Life cycle stage when the change is made.
	Traceability	Number of software requirements not traced to system requirements. Number of software requirements not traced to code and tests.
	Structure/Architecture	Logic complexity. GOTO usage. Size.
	Maintainability	Correlation of complexity/size.
Product (Code) Quality	Reusability	Correlation of complexity/size.
	Internal Documentation	Comment Percentage.
	External Documentation	Readability Index.
Implementation Effectivity	Resource Usage	Staff hours spent on life cycle activities.
	Completion Rates	Task completions. Planned task completions.

Testing Effectivity	Correctness	Errors and criticality. Time of finding of errors. Time of error fixes. Code Location of fault.

SATC's relationship with CMMi

The SATC Goals can be mapped to the following CMMi development processes:-

GOALS	CMMi process.
Requirements Quality	Requirements Development, Requirements Management.
Product (Code) Quality	Technical Solution
Implementation Effectivity	Project Management.
Testing Effectivity	Verification and Validation.

UNIT IV

SOFTWARE QUALITY PROGRAM

Software Quality program concepts

SQP stands for software quality program.Software quality program is a framework for building quality in to the software and for the actions necessary to verify that the required functionality and performance have been achieved.

The software quality program is more than “traditional” quality assurance.It goes beyond what is normally performed by “traditional” quality assurance functions and defines the enterprise-wide actions necessary for achieving quality in software development such as,

- Establishing the quality requirements for the software product
- Defining,implementing and evaluating processes and methodologies for the development,operationand maintenance of the software.
- Defining and using productivity,process quality and product quality measures
- Defining documentation requirements for the software
- Performing evaluations of the software development processes and products.
- Planning,implementing and managing a software quality program.

Objective

The objective of the software quality program is to assure the quality of

- Deliverable software and its documentation.
- The processes used to produce deliverable software and
- Non deliverable software

Responsibility for the Software quality program

Contractor personnel responsible for ensuring compliance with the software quality program requirements shall have the resources,responsibility,authority and organizational freedom to permit objective evaluations and to initiate and verify corrective actions.

The persons conducting the evaluation of a product or activity shall not be the persons who developed the product,performed the activity,or responsible for the product or activity.This does not preclude members of the development team from participating in these evaluations.The contractor shall assign responsibility for the fulfilment of and for ensuring compliance with the software quality program requirements .

Documentation for the Software Quality Program

The software quality program,including procedures,processesand products shall be documented in contractor format and shallprovide implementing instructions for each of the requirements in standard.The software quality program is subject to review by the contracting agency and may be disapproved by the contracting agencywhenever the program does not meet the requirements of the contract.

Software Quality program planning

The contractor shall conduct a complete review of the contract to identify and make timely provision for acquiring or developing the resources and skills required for implementing the software quality program.The contactor shall prepare the plan for applying the documented software quality program plan.the contractor shall place the SQPP under configuration control prior to implementation.

Software quality program procedures ,tools and records

Procedures used by SQP are as follows

- Product reviews
- In-process reviews
- Management review
- Process audit
- Code walkthrough
- Static code analysis
- FEMA

Software quality program implementation

The contractor shall implement the software quality program in accordance with the SQPP and shall adhere to the program for the duration of the contract.The software quality program shall be fully integrated with the activities required by the contract.

ESTABLISHMENT OF SOFTWARE QUALITY PROGRAM

Tasks:

Swift and accurate collection of data.

Develop a plan for quality

Scope:

The requirements to establish and implement the software quality program must be defined and made public knowledge are listed below

- Planning for and conducting assessments of the quality of the software
- Planning for and conducting assessments of the quality of the documentation.
- Planning for and conducting assessments of the quality of workmanship of all the contractors for the software

- Planning for and conducting assessments of the quality of all work whish will need to be performed for the on-going maintenance of the software.
- All deliverable and non deliverable items which are to be developed aspart of the project need to be clearly identified and labelled.
- System boundaries need ot be clearly defined.

Minimal Quality Assurance Effort
The following things must be done when a minimum investment quality plan is produced.

- Something is better than nothing
- Concentrate efforts for greatest effect
- Most major system failures have been caused by interface problems.
- Early error control
- Search for critical / risky fuctions
- Encourage developer's cooperation.

Quality Plan

- A Quality plan helps you schedule all of the tasks needed to make surethat your project meets the need of your customer.

It comprises 2 parts

Quality Assurance Plan

Quality control plan

Purpose:

Define the techniques,procedures and methodologies that will be used to assure timely delivery of the software and that the development system meets the specified requirements within project resources

Barriers:

The barriers prevent top management from understanding and implementing constant improvements in quality and productivity in all areas of their organisation are as follows

- Top management must understand the direct relationship of improved quality to productivity and from there to lowering of costs and expenses.
- Top management must understand that its controls the system and subsystemsthat determine the performance of the people in the organisation.
- Implementation of a quality program is dependent upon the management of the organisation.

Technical definition

Consist of the following 3 parts

1.Requirements

2.Confidence

3.Constant improvement

Software Quality Assurance Planning

Scope and intent of Software Quality Assurance (SQA) activities
The SQA team's objective is to ensure that the product does not deviate far from the original design specifications. If it is discovered that deviation has occurred, the SQA team will notify the development team to prevent future deviations and to correct the previous deviations. Also, the SQA team will perform a walkthrough to analyze the product's quality at any particular stage of development. Error detection and possible enhancements are also expressed to the development team.
SQA organizational role
The SQA organizational role is to review the product(s) at specific times during product implementation. Upon reviewing, the SQA team's duties will be to evaluate the software at its current development stage and recognize any defects in the subsequent stage (design or implementation). The SQA team will directly interact with the software engineering team in group discussions, discussing any errors or possible enhancements that have been identified. In addition, the SQA team will ensure that the software engineering team has not deviated in any way from the initial design specifications.
SQA Tasks
Task Overview
Description of SQA Task 1
The Engine Software Engineer will check with the Requirements Specification on a weekly basis to make sure that what he is coding conforms to the original design. This process will ensure that the product meets the client's expectations and standards and that the engine, up to its current point, is working properly.
.

Work products and documentation for Task 1
As a result of Task 1, any major deviations that occur will be expressed to the other group members and documented on a separate defect log. Documentation will ensure that each group member is aware of the change(s) made to the engine so that each part of the project can be adjusted accordingly.
.
Description of SQA Task 2
The User-Interface Software Engineer will check with the Requirements Specification on a weekly basis to make sure that what he is coding conforms to the original design. This process will ensure that the product meets the client's expectations and standards and that the user-interface, up to its current point, is working properly.

Work products and documentation for Task 2
As a result of Task 2, any major deviations that occur will be expressed to the other group members and documented on a separate defect log. Documentation will ensure that each group member is aware of the change(s) made to the interface, so that each part of the project can be adjusted accordingly.

Description of SQA Task 3
Each member of the group will routinely perform a hands-on evaluation of the user-interface. Noted evaluation criteria will be: ease of use, principle of least astonishment, unobtrusiveness, and overall attractiveness. This is done to ensure that the user-interface is evaluated honestly, and remains easily understandable and attractive.

Work products and documentation for Task 3
As a result of Task 3, all suggestions or concerns are expressed to the User-Interface Engineer. These are recorded in the defect log. Based on these concerns, the User-Interface Engineer takes note and makes the appropriate adjustments to the user-interface to make sure that the final product is satisfactory.

Description of SQA Task 4

Each member of the group will routinely perform a hands-on evaluation of the DirectX engine. Noted evaluation points will be: any logic errors and/or software glitches that occur, and any desired enhancements. This is done to ensure that the DirectX engine is evaluated honestly, and that it is defect free, sufficiently powerful, and efficient.

Work products and documentation for Task 4
As a result of Task 4, all suggestions or concerns are expressed to the Engine Software Engineer. These are recorded in the defect log. Based on these concerns the Engine Software Engineer takes note and makes the appropriate adjustments to the DirectX engine to make sure that the final product is satisfactory.

Description of SQA Task 5
An SQA leader will be appointed to (1) control the frequent SQA reviews; (2) keep track of all SQA meetings; and (3) manage the flow of information to the correct software engineer. In addition, the SQA leader will review each product defect or enhancement that has been reported, then assign a priority rank to each. The higher the priority rank, the more important it is to fix the defect or enhancement. Priority ranking will be determined by a group discussion, involving the software engineers, and headed by the SQA leader.

Work products and documentation for Task 5
As a result of Task 5, all suggestions or concerns expressed during each evaluation will be recorded. In addition, each recorded item will be assigned a priority ranking and the date the item was reviewed. All requests for defect fixes and enhancement implementations will be recorded.

Standards, Practices and Conventions (SPC)
Every software engineer's work will be evaluated weekly to ensure that the project is continuing smoothly and on schedule. Upon review, the current prototype will also be checked to determine if the software engineer is deviating

from the original specification.
Unscheduled reviews of every software engineer's work will be conducted to ensure that each subsystem is given ample attention during implementation. By making unscheduled evaluations, it can be determined whether the software engineer is allocating enough time for each subsystem or if the work is being rushed without attention to detail.

All software engineers are responsible for submitting any major design changes or implementation variances to the SQA leader. This procedure will account for all impacts on the rest of the software resulting from the change(s). Every major change will be recorded by the SQA leader, including which software engineer requested the change and the date requested.
All software engineers are expected to thoroughly test each completed subsystem of the software following guidelines outlined in the Test Specification. This is to ensure that each subsystem is working properly and efficiently. Any major defect found will be reported to the SQA leader.

SQA Resources

An SQA leader will be assigned to control the flow of information from the SQA team to the software engineers. The SQA leader will oversee the software quality control to ensure that the software engineers are conforming to the standards of the Requirements Specification document. Furthermore, the SQA leader will have the duty of assigning a relative rank of priority to every defect reported - functional or cosmetic. The priority will be used to determine which defects or enhancements are deemed the most important for the software engineers to correct first. Every defect or enhancement request is submitted to the SQA leader for review. The SQA leader will be in control of all SQA activities including SQA meetings and reviews.

Because software engineers are most familiar with implementation of their particular part of the software, each software engineer will perform software quality analyses. Before and after each subsystem is complete, the software engineer will review the Requirements Specification document to ensure that the subsystem is implemented within the bounds of the original design.
Each team member will actively and frequently test the current prototype of the software for possible defects or necessary enhancements. This ensures that each subsystem of the software is functioning properly and follows the Requirements Specification document. Completely debugging one's own source code can be difficult; therefore, each team member will be responsible for checking every major iteration of the software prototype to ensure that many or most defects are intercepted in early programming stages.
No special software or hardware will be needed to conduct the software quality assurance walkthroughs. It may, however, be helpful for each group member to have access to a central defect-report database, which will be reviewed and updated frequently. Although this is not necessary, it will be helpful in keeping software defects and enhancements organized and easily accessible to every group member for review.

.

Roles and Responsibilities

The SQA leader will oversee any formal technical reviews. Any defects

or enhancements will be discussed and recorded by the SQA leader. Each defect or enhancement will be given a priority rank, which will be recorded. Once the review is complete, the SQA leader will make a summary of each defect or enhancement and distribute them to the appropriate software engineer.
Each software engineer will be responsible for reviewing his own software module during module creation and upon module completion. Once each major software module is complete, it is the software engineer's duty to inform the SQA leader that the module is ready for review.

Review work products

The SQA leader will keep a defect log. The defect log contains all defects and enhancements, as well as a priority rank for each. The following will also be noted in the defect log: (1) whether or not the defect or enhancement has been handled, (2) which software engineer oversaw the correction, and (3) what date the correction was completed.

Formal Technical Reviews

Description of System Specification review

Description of focus of the System Specification review

The System Specification Review will provide a forum to analyze the proposed design of the software. To determine any obvious design flaws, the focus of this review will be to analyze the major software functions outlined in the System Specification. Once a design defect has been recognized, the SQA team will discuss with the software engineers any ideas or suggestions on how to compensate for the design flaw.

Timing of the review

The System Specification Review will be held upon completion of the System Specification. This should occur within the first few weeks of the software's development. This is necessary so that the underlying design of the software is sound and will not create any serious problems for the software engineers in the future.

Work products produced

The SQA leader will create a summary report of the System Specification Review. The summary report will include any changes to the software's major subsystem design. Once subsystem design defects have been identified, the SQA team will discuss possible solutions with the software engineers. Each possible solution will be noted and reviewed to determine if the solution will have an impact on the rest of the design. Once all obvious design defects have been handled, the System Specification will be amended to account for the design changes.

Review checklist

- Is the proposed design the best possible solution?
- Is there a better way to break up the software into subsystems? If yes, how?
- Is there any obvious design flaws that have not been accounted for? If yes, what?
- Are there any necessary enhancements for the software?
- Is the proposed System Specification within the time frame?

- Is each subsystem possible to implement in languages of choice?

SQA Tools, Techniques, Methods

All SQA activities will follow the same guidelines and methods. Every SQA meeting will include every group member. Every group member is expected to participate in the discussion. Any group member not attending the review will be notified by the SQA leader of what took place at the review. The SQA leader will oversee the discussion and will take notes of any defects or enhancements that need to be analyzed.

The SQA team will analyze the defects or enhancements and determine their complexity, impact on the system, and priority. Once prioritized, the SQA leader will assign each item to the software engineers along with their priority.

After a defect has been eliminated or an enhancement added, the software engineer will inform the SQA leader at the next SQA review. The SQA leader will take note of the correction.

No special tools will be necessary for SQA although access to a central database that all group members can access would be helpful to cut down time and duplication of error.

Factors affecting Intensity of SQA Activities

- SQA Activities are linked to the completion of a project phase
  - Requirements, design, etc.
- The SQA activities need to be integrated into the development plan that implements one or more software development models, such as the waterfall, prototyping, spiral, …
- They need to be activities just like other more traditional activities – be entered in plan, scheduled, etc.
- SQA planners need to determine
  - A **list** of SQA activities needed for the project
  - And then **for each activity**, they need to decide on
    - Timing
    - **Type** of QA activity to be applied (there are several)
    - **Who** performs the activity and resources required.
      - Important to note that many participate in SQA activities
        - Development team
        - Department staff members
        - Independent bodies

    - **Resources required** for the removal of defects and introduction of changes.

- Sad testimony that few want to allocate the necessary **time** for SQA activities.
  - This means **time** for SQA activities and then **time** for subsequent removal of defects.
  - Often, there is no time for follow-on work!!
- Activities are not simply cranked in and absorbed!
- So, **time** for SQA activities and defect correction actions needs to be examined.
- Project Factors
  - Magnitude of the project – how big is it?
  - Technical complexity and difficulty **Discuss**
  - Extent of reusable software components – a real factor
  - Severity of failure outcomes if project fails – essential!
- Team Factors
  - Professional qualifications of the team members
  - Team acquaintance w/ project and experience in the area
  - Availability of staff members who can professionally support the team, and
  - Percentage of new staff members in the team.

UNIT – V

SOFTWARE QUALITY ASSURANCE STANDARDIZATION

Types of standardization process:

- Emergence in *de facto* use: tradition (old standards on countries) and/or domination (Microsoft ex.).
- Fixed by a *standard body*:
  - in an impositive process: usually in *mandatory norms* (like dictatorial Laws).
  - in a consensus process: usually for *voluntary standards*

standardization is defined as: The development and implementation of concepts, doctrines, procedures and designs to achieve and maintain the required levels of compatibility, interchangeability or commonality in the operational, procedural, material, technical and administrative fields to attain interoperability

2.ISO 9000 Series:

ISO 9000 is a family of standards for quality management systems. ISO 9000 is maintained by ISO, the International Organization for Standardization and is administered by accreditation and certification bodies.

ISO-9001 covers the entire process from product design through after sales service.

ISO-9002 covers only the manufacture (or a specific service such as a QC/QA laboratory) of the product.

ISO-9003 covers the "final inspection" of the product only.

ISO-9004 is an "Internal Use" standard - it cannot be registered and is not subject to the third party audits.

❖ **The IS09000 series: a generic quality management standard**

- The 1S09000 series of standards are the international standards defined t quality management systems.
- The series dates from 1979, when BS 5750 r introduced in the UK. In 1987, the corresponding ISO, BS and EN standard were harmonized to produce three identical series of standards. In this text, shall use the ISO numbers for consistency.
- Minor modifications were introduced in 1994. The corresponded European and British standards are given in Table 7.3, which also lists the futon of each standard.
- Applied within software development. 1S09002 is intended for many manufacturing situations where the product is produced to a predefined specification and 1S09 for easy applications where the quality can be determined by a simple final inspection and testing procedure.
- ISO9000 provides guidance on which standard to adopt and 1S09004 assistance on how to establish a QMS which meets the

The contents of the standard

- In this section, we shall deal with the requirements of the 1S09001 standard.
- The 1S09002 and 1S09003 standards may be thought of as subsets of the 1S0900 standard, and in any case most software applications will require the full range of 1S09001 activity.
- The standard is based around a model specification for a quality management System.
- This underlying model is based around two fundamental principles:
    - Right first time.
    - Fitness for purpose
- The standard is intended to be realistic and implement able and, therefore, sets no prescriptive quality performance targets, referring instead to standards agreed as part of the contract with the customer and acceptable to them.
- The standard focuses upon ensuring that procedures are carried out in a systematic. Way and that the results are documented, again in a systematic manner.
- The main requirements are dealt with in Clause 4 of the standard under 20 subclasses the headings of each sub clause in Clause 4 are summarized in Table.
- Those clauses also found in 1S09002 and 1S09003 are marked with a tick in the right-hand columns. It should be noted that in 1S09003, some of the clauses are simplified Standards.

Table 7.4 Comparison of the requirements of the three principal standards

Clause	ISO9001	ISO9002	ISO9003
4.1	Management responsibility	✓	✓
4.2	Quality system	✓	✓
4.3	Contract review	✓	
4.4	Design control		
4.5	Document control	✓	✓
4.6	Purchasing	✓	
4.7	Purchaser supplied product	✓	
4.8	Product identification and traceability	✓	✓
4.9	Process control	✓	
4.10	Inspection and testing	✓	✓
4.11	Inspection, measuring and testing equipment	✓	✓
4.12	Inspection and test status	✓	✓
4.13	Control of non-conforming product	✓	✓
4.14	Corrective action	✓	
4.15	Handling, storage, packaging and delivery	✓	✓
4.16	Quality records	✓	✓
4.17	Internal quality audits	✓	
4.18	Training	✓	✓
4.19	Servicing		
4.20	Statistical techniques	✓	✓

The function of each section is detailed below.

Clause 4.1: Management responsibility

- The model recognizes the importance of management responsibility for quality throughout the organizatio3 whilst it is impossible for senior management to oversee everything personally, the standard explicitly provides for a management representative who is directly responsible for quality and is accountable to senior management.
- Clause also sets out the batik principles for establishing the quality sys tem within the organization and sets out many of its function which are then described in greater detail in later sections.

Clause 4.2: Quality system

- The model requires the organization to set up a quality system. The system ' should be documented and a quality plan and manual prepared.

- The scope of the plan is determined by the activities undertake and consequently the standard (1S09001/213) employed.
- Focus of the plan should be to ensure that activities are carried out in a systematic way and documented

Clause 4.3: Contract review

Contract review specifies that each customer order should be regarded as a con tract. Order entry procedures should be developed and documented aim of these procedures is to:

- Ensure that customer requirements are clearly defined in writing.
- Highlight differences between the order and the original quotation, so that they may be agreed.
- Ensure that the requirements can be met.
- The aim of this clause is to ensure that both the supplier and customer under stand the specified requirements of each order and to document this agreed specification to prevent misunderstandings and conflict at a later data

Clause 4.4: Design control

Design control procedures are required to control and verify design activity to take the results from market research through to practical designs. Key activities covered are: Planning for research and development.

- Assignment of activities to qualified staff.
- Identify interfaces between relevant groups.
- Preparation of a design brief.
- Production of technical data.
- Verification that the outputs for the design phase meets the input requirements.
- Identification and documentation of all changes and modifications.

The aim of this section is to ensure that the design phase is carried out effectively and to ensure that the output from the design phase accurately reflects the input requirements The importance of this process in a software context cannot be underestimated.

Clause 4.5: Document control

Three levels of documentation are recognized by the standard:

Level 1: planning and policy documents.

Level 2: procedures.

Level 3: detailed instructions

- The top level documents the quality plan and sets out policy on key quality issues.
- Level adds more detail to the documentation. Where possible, existing documentation should be incorporated.
- The aim should be to provide systematic documentation, rather than simply to provide more documents.
- It is important that each level of documentation is consistent with the one above it, providing greater detail as each level is descended.
- It is a common complaint that the standard requires a prohibitive amount of documentation to be produced.
- Supporters of the standard argue that systematizing of documentation can actually lead to a reduction in volume due to the removal of obsolete and surplus documents.
- It is more likely that some reduction will be achieved, which will offset greater volumes in other areas.
- Good existing documentation should be incorporated into any new stem and this is facilitated by the standard not specifying a particular form but merely specifying that documents be fit for their intended purpose.

Clause 4.6: Purchasing

- The purchasing system is designed to ensure that all purchased products and ser vices conform to the requirements and standards of the organization.
- The emphasis is placed on verifying the supplier' s own quality main procedures.

Clause 4.10: Inspection and testing

Inspection and testing are required to ensure conformance in three stages:

- Incoming materials or services.
- In process.
- Finished product and/or service.
  - ❖ All incoming supplies must be checked in some way. The method will vary according to the status of the supplier' s quality management procedures, from full examination to checking evidence supplied with the goods.
  - ❖ Monitoring ' in processes is required to ensure that all is going according to plan.
  - ❖ At the end of the process, any final inspection tests documented in the quality plan must be carried out. Evidence of conformity to quality standards, together with details of any supporting ' in-process' monitoring may be included. In an effective system, however, the final inspection and test should not have to be as extensive as it otherwise would be.
  - ❖ In addition, it should not reveal many problems, since they should have been eliminated by this stage.

Clause 4.11: Inspection, measuring and testing equipment and maintained.

- ❖ Procedures to ensure that calibration and maintenance activities are properly implemented should be documented, identifying the measurements required and the precision associated with each.
- ❖ Records must be kept of all activity.
- ❖ Checking and calibration activities should become part of regular maintenance.
- ❖ Management should ensure that checks are carried out at the prescribed intervals and efficient records kept.

Clause 4.15: Handling, storage, packaging and delivery

- Handling and associated activities must be designed to protect the quality built into the product.
- Subcontractors employed for transportation should be subject to the same documented procedures as internal employees.
- The scope of this clause is determined by the contract with the customer.
- The clause covers al activities which are the contractual obligation of the supplier.

Clause 4.16: Quality records

- Do not have to conform to a prescribed format, but must be fit for their intended purpose.
- As many will exist before the accredited system is implemented, the aim is to systematize and assimilate existing practice wherever possible, to reduce wasted effort in reproducing previous work in this area.

Clause 4.17: Internal quality audits

- The quality system should be' policed from within the organization and not dependent upon external inspection.
- Procedures should be established to set up regular internal audits as part of normal management procedure.
- The role of internal audits should be to identify problems early in order to minimize their impact and cost.

Clause 4.18: Training

Training activities should be implemented and documented. In particular, writ ten procedures are required:

- To establish training needs.
- To carry out training activity.
- To record the training requirements and completed activities for each member of staff.

Clause 4.12: Inspection and testing status

All material and services may be classified in one of three categories:

- Awaiting inspection or test.
- Passed inspection.
- Failed inspection.

This status should be clearly identifiable at any stage. It is important that material awaiting inspection is not mistakenly allowed to miss inspection at any stage, as non-conformance may go undetected.

Clause 4.13: Control of non-conforming product

- Standard defines non-conforming product as all products or services falling but side tolerance limits agreed in advance with the custom Once again it is not prescriptive about performance levels non-conforming products or ser vices should be clearly identified, documented and, if possible, physically separated from the conforming product.
- Procedures should be established to handle non-conforming products by reworking, disposal, re-grading or other acceptable documented courses of action.
- There are circumstances where the standard permits the sale of non-con forming product provided that the customer is clearly aware of the circum stances and is generally offered a concession.
- Representatives of accreditation bodies suggest that this an area where organizations often become lax after a while, relaxing procedures and allowing non-conforming product through.

Clause 4.14: Corrective action

- Corrective action is the key to continual improvement. Such action should be implemented via a systematic programmed which provides guidance and defines the duties of all parties.
- Records should be kept of any action taken so that future audits can investigate its effectiveness.

- Here a supplier has also obtained external accreditation for their quality management systems, checks may be considerably simplified. As with all procedures, they should be documented

Clause 4.7: Purchaser-supplied product

- CMI Services and products supplied by the customer must be checked for suit ability, in the same way as supplies purchased from any other supplier.
- In order to ensure this, procedures should be put in place and documented, so that these services and products may be traced through all processes and storage.

Clause 4.8: Product identification and traceability

- Ensure effective process control and to correct any non-conformance, it is necessary to establish procedures to identify and trace materials from input to output.
- This also enables quality problems to be traced to root causes may be that t1 problem can be traced back to supplied materials, in which case the problem may lie outside the quality system altogether

Clause 4.9: Process control

- Process control requires a detailed knowledge of the process itself. This must be documented, often in graphical form, as a process flow chart or similar.
- Procedures for setting up or calibration must also be recorded.
- Documented instructions should be available to staff to ensure that they have the capability to carry out the task as specified.
- It is staggering how often organizations do not understand their own pro ceases properly.
- The discipline of documenting the actual process precisely and unambiguously for accreditation purposes can be very educational.

Clause 4.20: Statistical techniques

- Statistical techniques are required to be used where appropriate.
- The standard does not specify particular techniques or methods but does specify that once again they should be appropriate for the intended purpose.
- Their use may be necessary in order to satisfy other requirements, notably process control,

ISO-9001:

Some of the requirements in ISO 9001 (which is one of the standards in the ISO 9000 family) would include

- **a set of procedures that cover all key processes in the business;**
- **monitoring processes to ensure they are effective;**
- **keeping adequate records;**
- **checking output for defects, with appropriate corrective action where necessary;**
- **regularly reviewing individual processes and the quality system itself for effectiveness; and**
- **facilitating continual improvement**

A company or organization that has been independently audited and certified to be in conformance with ISO 9001 may publicly state that it is "ISO 9001 certified" or "ISO 9001 registered." Certification to an ISO 9000 standard does not guarantee the compliance (and therefore the quality) of end products and services; rather, it certifies that consistent business processes are being applied.
Although the standards originated in manufacturing, they are now employed across a wide range of other types of organizations. A "product", in ISO vocabulary, can mean a physical object, or services, or software. In fact, according to ISO in 2004, *"service sectors now account by far for the highest number of ISO 9001:2000 certificates - about 31% of the total"*

ISO 9001:2000 specifies requirements for a quality management system where an organization

1. needs to demonstrate its ability to consistently provide product that meets customer and applicable regulatory requirements, and
2. aims to enhance customer satisfaction through the effective application of the system, including processes for continual improvement of the system and the assurance of conformity to customer and applicable regulatory requirements.

All requirements of this International Standard are generic and are intended to be applicable to all organizations, regardless of type, size and product provided.

Where any requirement(s) of this International Standard cannot be applied due to the nature of an organization and its product, this can be considered for exclusion.

Where exclusions are made, claims of conformity to this International Standard are not acceptable unless these exclusions are limited to requirements within clause 7, and such exclusions do not affect the organization's ability, or responsibility, to provide product that meets customer and applicable regulatory requirements.

ISO-9002:

Developing products to a standard of consistent high quality, Scorpion Research is a recognised leader in its field and proud to announce its achievement of the coveted ISO 9002 quality certification.

The ISO 9002 standard is an emerging global standard for product and process quality, adopted by 91 countries which comprise the International Organisation for Standardisation (ISO).

ISO 9002 is best known to European countries who use the certification as a means of identifying companies dedicated to providing a customer with the best product and service possible.

Scorpion Research has always emphasised its quality through a continuous improvement program complementing its other quality initiatives. Achieving ISO 9002 certification further demonstrates commitment to a consistent level of quality recognised under International standards.

According to ISO assessors, the certification process is rigorous and only a minority of the applicants who strive to become certified are successful on their first audit. Scorpion Research is amongst this minority of organisations that passed the required certification audit on the first assessment demonstrating its consistent emphasis towards a high quality standard.

Certification is an ongoing process and Scorpion Research will undergo regular internal and external audits by third party ISO assessors in order to retain its certification. The ISO certification complements the quality program by establishing an externally verified unbiased process of self-monitoring and problem-solving.

Scorpion Research joins firms such as British Telecom, Sony, IBM, Compaq, Digital Equipment, Xerox, Toshiba and Hewlett-Packard in ISO certification.

ISO-9003:

This is the least complex and easiest to install of three standards in the ISO 9000 series. This standard is for organizations that do not participate in design and development, purchasing or have

production controls. It is designed for organizations that only require final inspection and testing of their products and services to ensure that they have met the specified requirements. This system is generally only relevant to simple products and services. It is also an option for organizations that cannot justify the expense of one of the other systems but still desire a quality management system for their organization.

ISO-9004:

- Your quality system must balance two needs:
  - Your customers' need to have quality products at a reasonable price.
  - Your company's need to make quality products at a reasonable cost.

3.ISO9000-3 for software development:

ISO 9000-3	*4.4 Software development and design*
4.4.1 General	**Develop and document procedures to control the product design and development process. These procedures must ensure that all requirements are being met.**

Software development	**Control your software development project and make sure that it is executed in a disciplined manner.**
	• **Use one or more life cycle models to help organize your software development project.**
	• **Develop and document your software development procedures. These procedures should ensure that:**
	  • **Software products meet all requirements.**
	  • **Software development follows your:**
	  • **Quality plan.**
	  • **Development plan.**

Software design	**Control your software design process and make sure that it is performed in a systematic way.**
	• **Use a suitable software design method.**
	• **Study previous software design projects to avoid repeating old mistakes.**
	Design software that is:
	• **Easy to test, install, use, and maintain.**
	• **Safe and reliable when failure could cause:**
	  • **Human injury.**
	  • **Property damage.**
	  • **Environmental harm.**
	Develop and document rules to control:
	• **Coding activities.**
	• **Naming conventions.**
	• **Commentary practices.**

	• Programming languages. Apply configuration management techniques to document and control the use and review of all: • Analysis tools. • Design techniques. • Compilers and assemblers. Train personnel in the use of such tools and techniques.
4.4.2 Design and development planning	Create design and development planning procedures. Your product planning procedures should ensure that: • Plans are prepared for each design activity or phase. • Responsibility for implementing each plan, activity, or phase is properly defined. • Qualified personnel are assigned to the product design and development process. • Adequate resources are allocated to the product design and development process. • Plans are updated, and circulated to the appropriate participants, as designs change.

Software design and development planning	Prepare a software development plan. Your plan should be documented and approved before it is implemented. Your plan should control:
	• Technical activities.
	  • Requirements analyses.
	  • Design processes.
	  • Coding activities.
	  • Integration methods.
	  • Testing techniques.
	  • Installation work.
	  • Acceptance testing.
	• Management activities.
	  • Project supervision.
	  • Progress reviews.
	  • Reporting requirements.
	Your software development plan should:
	• Define your project.
	• Identify related plans and projects.
	• List your project objectives.
	• Define project inputs and outputs.
	  • Define inputs for each project activity.
	  • Define outputs for each project activity.
	• Explain how your project will be organized.
	  • Explain how your teams will be structured.
	  • Explain who will be responsible for what.
	  • Explain how subcontractors will be used.
	  • Explain how project participants will interact.

	• Explain how all resources will be managed. • Discuss project risks and potential problems. • Identify important project assumptions. • Present your project schedule. • Define project phases and dependencies. • Specify project time lines and milestones. • Introduce your project budget. • Describe the work that will be done. • Describe each task. • Describe the inputs for each task. • Describe the outputs for each task. • Identify all relevant control strategies. • Identify all relevant standards and conventions. • Identify all relevant rules and regulations. • Identify all relevant practices and procedures. • Identify configuration management practices. • Identify backup and recovery procedures. • Identify archiving procedures. • Identify all relevant methods and approaches. • Identify methods used to control nonconforming products. • Identify methods used to control software development software • Identify methods used to control virus protection activities. • Identify all relevant tools and techniques. • Identify methods used to qualify all tools and techniques. • Identify methods used to control all tools and techniques.
4.4.3 Organizational and technical interfaces	Identify the groups who should be routinely involved in the product design and development process, and ensure that their design input is properly documented, circulated, and reviewed. Make sure that your software development plan or your subcontractors' plans: • Define how the responsibility for software development will be distributed between all participants. • Define how technical information will be shared and transmitted between all participants. Explain how all project participants will provide input. • Explain how subcontractors will provide input during design, installation, maintenance, and training. • Explain how regulatory authorities will provide input during design, installation, maintenance, and training.

	• Explain how help desk staff will provide input during design, installation, maintenance, and training.
	• Explain how other related projects will provide input during design, installation, maintenance, and training.
	• Explain how end users will provide input during design, installation, maintenance, and training.
	Make sure that your customer has accepted the responsibility to:
	• Cooperate and support your project.
	• Provide the information you need when you need it.
	• Resolve outstanding issues in a timely manner.
	Make sure that your customer representative has been given the responsibility and authority to:
	• Clarify requirements and expectations.
	• Answer questions and solve problems.
	• Make and implement agreements.
	• Approve plans and proposals.
	• Establish acceptance criteria.
	• Provide appropriate customer-supplied products.
	• Define and distribute authorities and responsibilities.
	Schedule joint progress reviews. You and your customer together should review:
	• Activities.
	  • Your developmental activities.
	  • Your customer's activities.
	  • Your users' activities.
	  • Training activities.
	  • Conversion activities.
	• Results.
	  • Results of verification activities.
	  • Results of acceptance tests.
	  • Results of conformance evaluations.

4.4.4 Design input	Develop procedures to ensure that all design-input requirements are identified, documented, and reviewed; and that all design flaws, ambiguities, contradictions, and deficiencies are resolved. Design input requirements can be classified as follows: • Customer expectations. • Contractual conditions. • Statutory imperatives. • Regulatory requirements. • Environmental constraints. • Safety considerations. • Performance standards. • Functional specifications. • Descriptive prescriptions. • Aesthetic preferences.

Software design input	**Design input requirements should be specified by the customer. However, sometimes the customer will expect you to develop the design input specification. In this case, you should:**
	• **Prepare procedures that you can use to develop the design input specification. These procedures should be documented and should explain:**
	  • **How interviews, surveys, studies, prototypes, and demonstrations will be used to develop your design-input specification.**
	  • **How you and your customer will formally agree:**
	  • **To accept the official specification.**
	  • **To accept changes to the official specification.**
	  • **How changes to specifications will be controlled.**
	  • **How prototypes and product demonstrations will be evaluated.**
	  • **How system oriented input requirements will be met through the use of hardware, software, and interface technologies.**
	  • **How reviews, evaluations, and other discussions between you and your customer will be recorded.**
	• **Work closely with your customer in order to avoid misunderstandings and to ensure that the specification meets the customers needs.**
	• **Express your specification using terms that will make it easy to validate during product acceptance.**
	• **Ask your customer to formally approve the resulting design input specification.**
	Your design input specification may address the following kinds of characteristics or requirements:
	• **Functional requirements.**
	• **Reliability requirements.**
	• **Usability requirements.**
	• **Efficiency requirements.**
	• **Maintainability requirements.**
	• **Portability requirements.**
	• **Interface requirements.**
	  • **Hardware interface requirements.**
	  • **Software interface requirements.**
	Your design input specification may also need to address the following kinds of requirements:
	• **Operational requirements.**
	• **Safety requirements.**
	• **Security requirements.**
	• **Statutory requirements.**

4.4.5	**Develop procedures to control design outputs.**

Design output	• **Design outputs are usually documents. They include drawings, parts lists, process specifications, servicing procedures, and storage instructions. These types of documents are used for purchasing, production, installation, inspection, testing, and servicing.** • **Design outputs must be expressed in terms that allow them to be compared with design input requirements.** • **Design output documents must identify those aspects of the product that are crucial to its safe and effective operation. These aspects can include operating, storage, handling, maintenance, and disposal requirements.** • **Design output documents must be reviewed and approved before they are distributed.** • **Design outputs must be accepted only if they meet official acceptance criteria.**
Software design output	• **Prepare design output documents using standardized methods and make sure that your documents are correct and complete.** • **Software design outputs can include design specifications, source code, user guides, etc.**
4.4.6 Design review	**Develop procedures that specify how design reviews should be planned and performed. Design review procedures should:** • **Be formally documented.** • **Ensure that reviews are recorded.** • **Ensure that representatives from all relevant areas are involved in the process of review.**
Software design review	**Plan and perform design reviews for software development projects. Your reviews should ensure that all:** • **Design activities and results are reviewed.** • **Product nonconformities are identified and addressed.** • **Process deficiencies are identified and addressed.** • **Review conclusions and observations are recorded.**

Software design review procedures	**Develop and document design review procedures. Your procedures should make sure that you:**
	• **Clarify exactly what is being reviewed.**
	• **Distinguish between different types of reviews.**
	• **Organize and schedule design review meetings.**
	• **Indicate when design reviews should be performed.**
	• **Maintain a record of all design review meetings.**
	• **Invite all appropriate groups to participate.**
	  • **Invite customers to participate (when required).**
	  • **Confirm that customers agree with your results.**

	• Allow design activities to continue only if all:
	  • Deficiencies and nonconformities have been addressed.
	  • Risks and consequences have been assessed.
	Your design review procedures may also:
	• Define the methods that should be used to ensure that all rules and conventions are being followed.
	• Define what needs to be done to prepare for a design review. This may include:
	  • Defining roles.
	  • Listing objectives.
	  • Preparing an agenda.
	  • Collecting documents.
	• Define what should be done during the review. This may include:
	  • Defining the guidelines that should be followed.
	  • Defining the techniques that should be used.
	• Define the criteria that constitute a successful review.
	• Define the follow-up methods that should be used to ensure that all outstanding issues will be addressed.

4.4.7 Design verification	**Develop procedures that specify how design outputs, at every stage of the product design and development process, should be verified.** **These procedures should:** • **Verify that outputs satisfy design-input requirements.** • **Ensure that objective evidence is used to verify outputs.** • **Ensure that all design verifications are recorded.** • **Ensure that all design documents are verified.** **These design verification procedures may also:** • **Use alternative calculations to verify design outputs.** • **Use tests and demonstrations to verify outputs.** • **Compare design outputs with proven designs.**

Software design verification	**Verify design outputs by:**
	• **Performing design reviews.**
	• **Performing demonstrations.**
	  • **Evaluating prototypes.**
	  • **Carrying out simulations.**
	• **Performing tests.**
	Maintain a record of design verifications.
	• **Record verification results.**
	• **Record remedial actions.**
	• **Record action completions.**
	Accept design outputs for subsequent use only if they have been properly verified and only if all remedial actions have been taken.

4.4.8 Design validation	**Develop procedures that validate the assumption that your newly designed products will meet customer needs. Develop design validation procedures that:** • **Confirm that your new product performs properly under all real-world operating conditions.** • **Confirm that your new product will meet every legitimate customer need and expectation.** • **Ensure that validations are carried out early in the design process whenever this will help guarantee that customer needs will be met.**

Software design validation	• **Prove that your product is ready for its intended use before you ask your customer to accept it.**
	• **Accept validated products for subsequent use only if they have been properly verified and only if all remedial actions have been taken.**
	• **Maintain a record of design validations.**
	  • **Record validation results.**
	  • **Record remedial actions.**
	  • **Record action completions.**

4.4.9 Design changes	**Develop procedures to ensure that all product design modifications are documented, reviewed, and formally authorized before the resulting documents are circulated and the changes are implemented.**

Software design changes	• **Develop procedures to control design changes that may occur during the product life cycle. Your procedure should ensure that you:**
	  • **Document the design change.**
	  • **Evaluate the design change.**
	  • **Justify the design change.**
	  • **Verify the design change.**
	  • **Approve the design change.**
	  • **Implement the design change.**
	  • **Monitor the design change.**
	Your configuration management process may be used to control design changes.

4.CMM and CMMI

❖ CAPABLITY MATURITY MODEL

- ❖ To determine an organization's current start of process maturity the SET uses an assessment that results in a 5 point grading scheme.
- ❖ The grading scheme determines compliance with a capability maturity model [CMM] that defines key activities required at different levels of process maturity.
- ❖ The SET approach provides a measure of the global effectiveness of a company's S/W engineering practice & establishments *5 process maturity levels* that are defined in the following manner:

LEVEL

- ➢ *INITIAL*
- ➢ *REPETABLE*
- ➢ *DEFINED*
- ➢ MANAGED
- ➢ *OPTIMIZED*

The set has associated key process areas (KPA's) with each of the maturity levels. Each KPA is described by identifying the following *characteristics*:

- *GOALS*
- *COMMITMENTS*
- *ABILITIES*
- *ACTIVITIES*
- *METHODS FOR MONITORING INFORMATION*
- *METHODS FOR VERIFYING IMPLIMENTATION*

18 KPA's are defined across the maturity model & mapped into different levels of process maturity the following KPA's should be achieved at each process maturity level

PROCESS MATURITY LEVEL 2:

- *S/W CONFIGURATION MANAGEMENT*
- *S/W QUALITY ASSURANCE*
- *S/W SUBCONTRACT MANAGEMENT*
- *S/W PROJECT TRACKING & MANAGEMENT*
- *S/W PROJECT PLANNING*
- *REQUIREMENTS, MANAGEMENT*

PROCESS MATURITY LEVEL 3:

- *PEER REVIEWS*
- *INTERGROUP CORDINATION*
- *S/W PROJECT ENGINEERING*
- *INTEGRATED S/W MANAGEMENT*
- *ORGANISATION PROCESS DEFINITION*
- *ORGANISATION PROCESS FOCUS*

PROCESS MATURITY LEVEL 4:

- *S/W QUALITY MANAGEMENT*
- *QUANTITATIVE PEOCESS AMNAGEMENT*

PROCESS MATURITY LEVEL 5:

- *PROCESS CHANGE MANAGEMENT*

- *TECHNOLOGY CHANGE MANAGEMENT*
- *DEFECT PRESERVATION*

- Each of the KPA is defined by a set of key practice that contribute to satisfying its goals.
- The key practices are policies, procedures & activities that must occur before a key process area has been fully instituted.
- The SET defines key indicators as "those key practices or components of key practices that offer the greatest in sight into whether the goals of a process area have been achieved.

The **Capability Maturity Model** (**CMM**), also sometimes referred to as the **Software CMM** (**SW-CMM**).The CMM is a *process* capability model based on software development organisation processes/practices.

Though the CMM was retired in 1997 and has not been updated since, having been superseded by **CMMI** (**Capability Maturity Model Integration**), it has been used as a generally applicable model to assist in understanding the process capability maturity of organisations in diverse areas. For example, software engineering, system engineering, project management, risk management, system acquisition, information technology (IT) or personnel management, against a scale of five maturity levels, namely: Initial, Repeatable, Defined, Managed and Optimized.

The **Capability Maturity Model** (**CMM**) is a process capability maturity model which aids in the definiton and understanding of an organisation's processes. The CMM was originally used to enable the assessment of software development processes.

Capability Maturity Model

• CMM model strives to achieve predictability and consistency as a precursor to continuous improvements by following a set of process in a well defined framework.
• Level 1 is Initial level
• Level 2 is repeatable which helps in achieving repeatability of performance and quality should the organizations undertake a similar project again.
• Level 3 is defined level
• Level four is measured level.
• Level 5 is optimistic level , here people always work towards a target.

The Role of CMM

The role of CMM is increasing. This may be attributed to a number of factors:

- The maturity of the model itself
- The increasing general awareness of the need for externally recognized quality standards

- The adoption of model by key software purchases such as national departments of defense.

Advantages of CMM

- It allows for improvement and evolution
- Also used in conjunction with other quality standards
- It highlights the defects as they occur.
- The CMM prioritizes tasks for improvement
- It provides a matrix for strengths and weakness

Schematic view of CMM

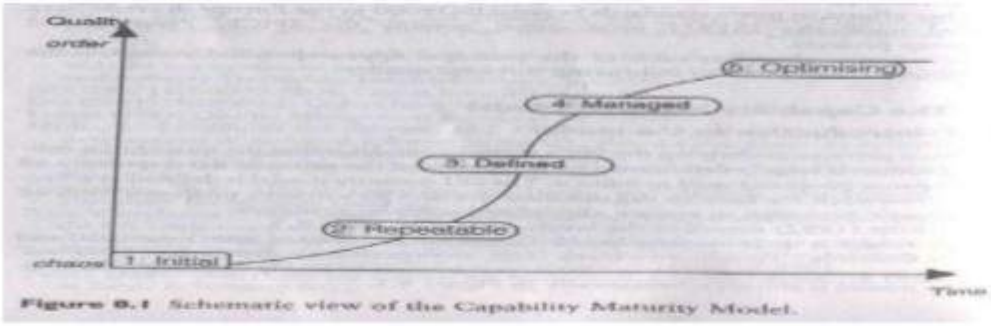


Figure 8.1 Schematic view of the Capability Maturity Model.

The maturity model seeks to measure how well these processes are carried out.There are five stages to six measurement categories in subjectively rating an organization's quality operation.

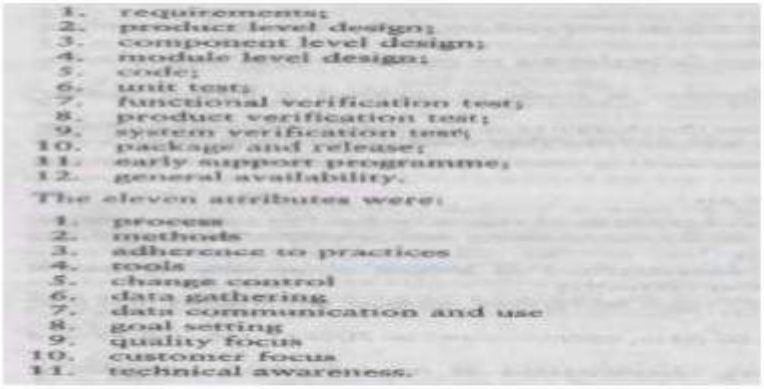
The five stages are:

- **Uncertainty**, where management is confused and uncommitted regarding quality management tool
- **Awakening**, where management is beginning to recognize that quality management can help
- **Enlightenment**, where the decision is made to conduct a formal quality improvement programme
- **Wisdom**, where the company has the chance to make changes permanent (things are basically quiet and people wonder why they used to have problems);
- **Certainty**, where quality management is considered an absolutely vital part of company management.

The six measurement categories are:

- management understanding and attitude, characterized as 'no comprehension of quality as a management tool' at uncertainty and 'an essential part of the company system' at certainty;
- quality organization status, characterized as hidden at uncertainty and a thought leader/main concern at certainty;
- problem handling, which are fought when they occur at uncertainty and prevented at certainty;
- cost of quality as percentage of sales, characterized as 20% at uncertainty and 2.5% at certainty;
- quality improvement actions, characterized as no organized activities at uncertainty and a normal and continued activity at certainty;
- Summation of company quality posture, summarized as 'we do not know why we have problems with quality' at uncertainty and 'we know why we do not • have problems with quality' at certainty.

As a precursor to the maturity model itself, Radice et.al. (1985) working with Humphrey, identified 12 process stages. The process stages were stages in the lifecycle:



1. requirements;
2. product level design;
3. component level design;
4. module level design;
5. code;
6. unit test;
7. functional verification test;
8. product verification test;
9. system verification test;
10. package and release;
11. early support programme;
12. general availability.

The eleven attributes were:

1. process
2. methods
3. adherence to practices
4. tools
5. change control
6. data gathering
7. data communication and use
8. goal setting
9. quality focus
10. customer focus
11. technical awareness.

Evolution of the CMM

Year	Version published
1987	Software process maturity framework
1987	Preliminary maturity questionnaire
1987	Characterizing the software process
1989	Managing the software process
1990	Draft version of CMM v0.2
1991	CMM v0.6 discussion
1991	CMM v1.0
1993	CMM v1.1

The CMMI is organized around a set of Process Areas (PAs). The PAs are divided into groups associated with what are called maturity levels. In the CMMI, most basic management practices are considered part of maturity level 2, while most software engineering practices are associated with level 3. Level 4 is about process and product quality management, while level 5 includes processes for process optimization and technology change management.

An organization that has practices that meet all the goals of maturity level 2 is characterized as a level 2 organization. An organization that cannot demonstrate that its practices meet all the applicable level 2 goals is considered level 1 by default. An organization that meets the goals of all the PAs at level 2 and level 3 is a level 3 maturity organization and so on.

The CMMI also offers an alternate approach call the "continuous representation", where individual PA's are rated on a maturity scale of 1 - 5. This allows the organization the flexibility to tailoring its maturity goals based on business needs for particular PA's.

As organizations move up the maturity ladder, they are more likely to produce higher quality products with more predictable costs and cycle times. The higher levels are more likely to correlate with higher productivity and shorter cycle times as well.

The idea of model based software improvement is very simple. First pick the applicable Process Areas. Next perform an assessment of organizational practices relative to the model. The organization practices do not have to conform exactly to the representative practices included in the model. They just have to meet the stated goals of each PA. Based on this comparison, assign a maturity level to the organization and produce a list of strengths and weakness relative to the model.

The output of the assessment is used to prioritize areas for improvement. The idea is to improve deficient practices at the lower maturity levels first and systematically move up in maturity level over time using additional assessments to measure progress.

Focusing on the lowest level practices puts a firm foundation in place before tacking the higher level processes and it give the organization time to internalize the changes. This approach nicely avoids the twin problems of putting practices in place before required supporting practices are available and trying to do too much too soon.

So model based improvement has a lot of very attractive features. One of the most attractive ones is the level goal itself. The numerical level gives organizations a metric that that can be easily understood, can used to measure progress, and can used to benchmark against other organizations.

SEI has defined a formal appraisal methodology and provided a lead assessor training and certification program. This means that assessment results, particular those obtained using a third party assessor, will be reasonably consistent and can be used to benchmark against other organizations. In fact SEI maintains a publicly accessible database of assessment results giving the number of organizations at each level by industry and the average time for an organization to improve from one level to the next.

6.Six Sigma Concept:

Six Sigma at many organizations simply means a measure of quality that strives for near perfection. Six Sigma is a disciplined, data-driven approach and methodology for eliminating defects (driving towards six standard deviations between the mean and the nearest specification limit) in any process -- from manufacturing to transactional and from product to service.

The term Six Sigma represents a stringent level of quality. It is a specific defect rate: 3.4 defective parts per million (ppm). It was made known in the industry by Motorola, Inc.,

Six Sigma has become an industry standard as an ultimate quality goal.
Sigma (s) is the Greek symbol for standard deviation.
As the following figure1 indicates, the areas under the curve of normal distribution defined by standard deviations are constants in terms of percentages, regardless of the distribution parameters.

The area under the curve as defined by plus and minus one standard deviation (sigma) from the mean is
68.26%.

The area defined by plus/minus two standard deviations is 95.44%, and so forth. The area defined by plus/minus six sigma is 99.9999998%. The area outside the six sigma area is thus 100% -99.9999998% = 0.0000002%.

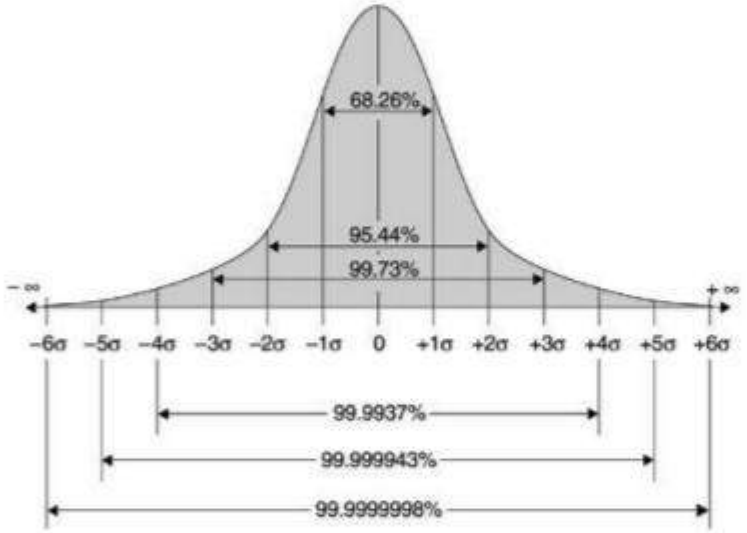


Fig. 1

The area within the six sigma limit as the percentage of defect-free parts and the area outside the limit as the percentage of defective parts, it is found that six sigma is equal to 2 defectives per billion parts or 0.002 defective parts per million.

The interpretation of defect rate as it relates to the normal distribution will be clearer if we include the specification.

Given the specification limits (which were derived from customers' requirements), our purpose is to produce parts or products within the limits. Parts or products outside the specification limits do not conform to requirements. If we can reduce the variations in the production process so that the six sigma (standard deviations) variation of the production process is within the specification limits, then we will have six sigma quality level.

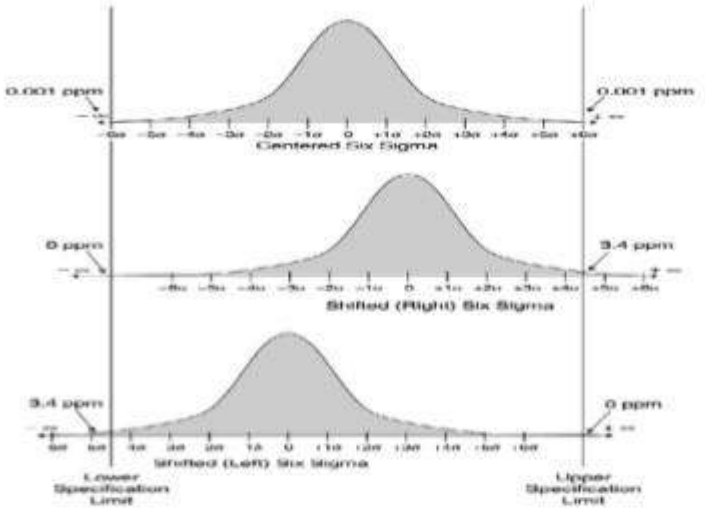


Fig. 2 Specification Limits, Centered six sigma**, and Shifted** six sigma

The six sigma value of 0.002 ppm is from the statistical normal distribution. It assumes that each execution of the production process will produce the exact distribution of parts or products centered with regard to the specification limits. In reality, however, process shifts and drifts always result from variations in process execution.

The maximum process shifts as indicated by research is 1.5 sigma. If we account for this 1.5-sigma shift in the production process, we will get the value of 3.4 ppm. Such shifting is illustrated in the two lower panels of fig.2.

Given fixed specification limits, the distribution of the production process may shift to the left or to the right. When the shift is 1.5 sigma, the area outside the specification limit on one end is 3.4 ppm, and on the other it is nearly zero.

The slight difference between the centered six sigma and the shifted six sigma may imply something significant. The former is practically equivalent to zero defects, which may invite the debate whether it is feasible to achieve such a goal.

In order to reach six sigma, we have to improve the process. Specifically, we must reduce process variations so that the six sigma variation is still within the specification limits.

The concept and approach of six sigma has been expanded and applied to the improvement of management systems and total quality management.

The statistical representation of Six Sigma describes quantitatively how a process is performing. To achieve Six Sigma, a process must not produce more than 3.4 defects per million opportunities. A Six Sigma defect is defined as anything outside of customer specifications. A Six Sigma opportunity is then the total quantity of chances for a defect. Process sigma can easily be calculated using a Six-Sigma calculator.

The fundamental objective of the Six Sigma methodology is the implementation of a measurement-based strategy that focuses on process improvement and variation reduction through the application of Six Sigma improvement projects. This is accomplished through the use of two Six Sigma sub-methodologies: DMAIC and DMADV. The Six Sigma DMAIC process (define, measure, analyze, improve, control) is an improvement system for existing processes falling below specification and looking for incremental improvement. The Six Sigma DMADV process (define, measure, analyze, design, verify) is an improvement system used to develop new processes or products at Six Sigma quality levels. It can also be employed if a current process requires more than just incremental improvement. Both Six Sigma processes are executed by Six Sigma Green Belts and Six Sigma Black Belts, and are overseen by Six Sigma Master Black Belts.

Six Sigma is a set of practices originally developed by Motorola to systematically improve processes by eliminating defects.A defect is defined as nonconformity of a product or service to its specifications.

While the particulars of the methodology were originally formulated by Bill Smith at Motorola in 1986, Six Sigma was heavily inspired by six preceding decades of quality improvement methodologies such as quality control, TQM, and Zero Defects. Like its predecessors, Six Sigma asserts the following:

- Continuous efforts to reduce variation in process outputs is key to business success
- Manufacturing and business processes can be measured, analyzed, improved and controlled
- Succeeding at achieving sustained quality improvement requires commitment from the entire organization, particularly from top-level management

Sigma (the lower-case Greek letter σ) is used to represent standard deviation (a measure of variation) of a population (lower-case 's', is an estimate, based on a sample). The term "six sigma process" comes from the notion that if one has six standard deviations between the mean of a process and the nearest specification limit, there will be practically no items that fail to meet the specifications. This is the basis of the Process Capability Study, often used by quality professionals. The term "Six Sigma" has its roots in this tool, rather than in simple process standard deviation, which is also measured in sigmas. Criticism of the tool itself, and the way that the term was derived from the tool, often sparks criticism of Six Sixma.