Software Reliability

Software Reliability

Basic Concepts

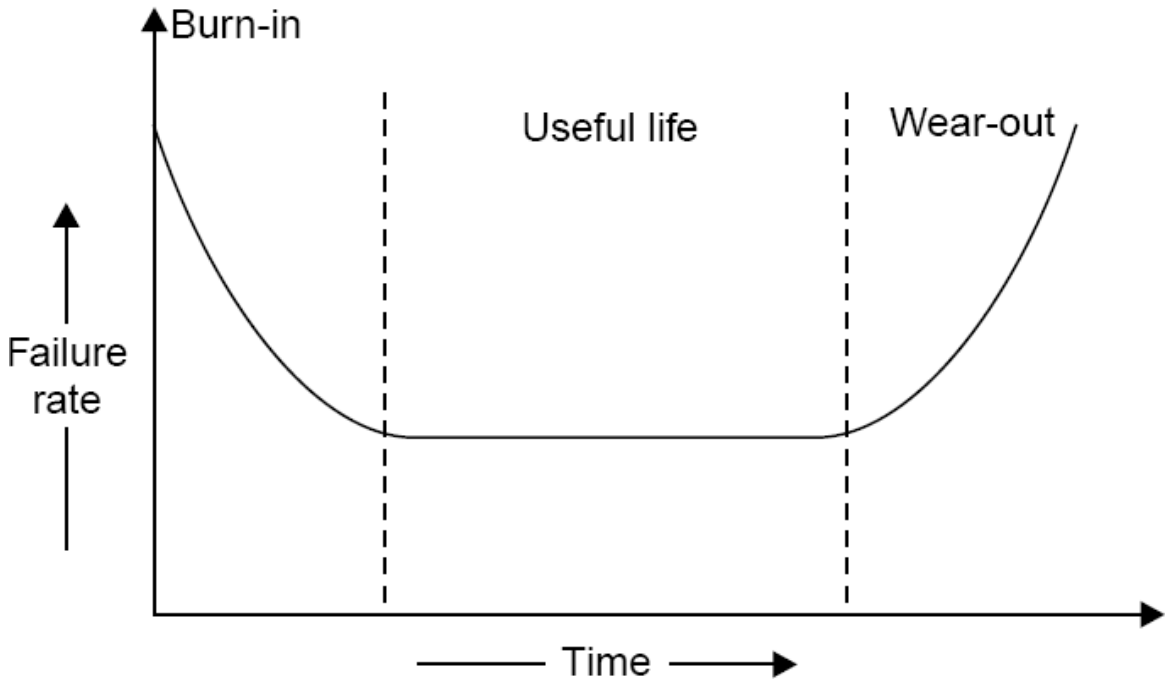
There are three phases in the life of any hardware component i.e., burn-in, useful life & wear-out.

In **burn-in phase,** failure rate is quite high initially, and it starts decreasing gradually as the time progresses.

During **useful life period**, failure rate is approximately constant.

Failure rate increase in **wear-out phase** due to wearing out/aging of components. The best period is useful life period. The shape of this curve is like a “bath tub” and that is why it is known as bath tub curve. The “bath tub curve” is given in Fig.next

Software Reliability



Bath tub curve of hardware reliability.

Software Reliability

We do not have wear out phase in software. The expected curve for software is given in fig. 7.2.

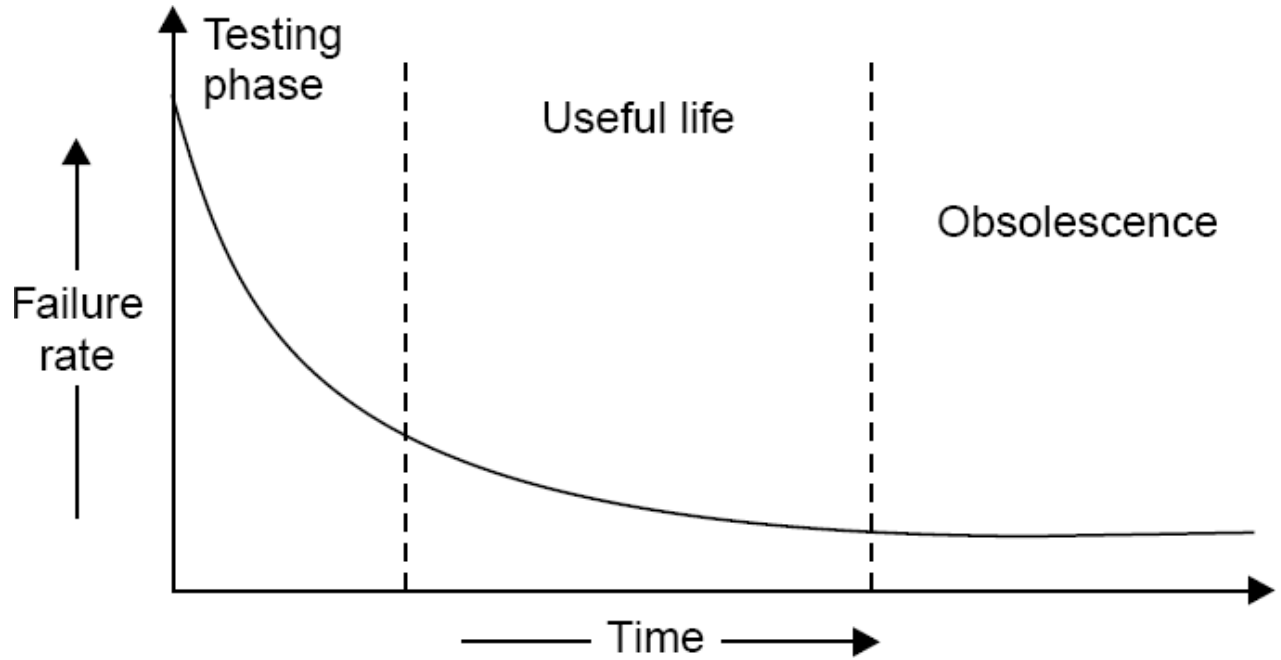


Fig. 7.2: Software reliability curve (failure rate versus time)

Software Reliability

Software may be retired only if it becomes obsolete. Some of contributing factors are :

- ✓ change in environment
- ✓ change in infrastructure/technology
- ✓ major change in requirements
- ✓ increase in complexity
- ✓ extremely difficult to maintain
- ✓ deterioration in structure of the code
- ✓ slow execution speed
- ✓ poor graphical user interfaces

Software Reliability

What is Software Reliability?

"Software reliability means operational reliability. Who cares how many bugs are in the program?

As per IEEE standard: "Software reliability is defined as the ability of a system or component to perform its required functions under stated conditions for a specified period of time".

Can be measured directly and estimated using historical and developmental data (unlike many other software quality factors)
Software reliability problems can usually be traced back to errors in design or implementation

Software Reliability

- Failures and Faults

A fault is the defect in the program that, when executed under particular conditions, causes a failure.

The execution time for a program is the time that is actually spent by a processor in executing the instructions of that program. The second kind of time is calendar time. It is the familiar time that we normally experience.

Software Reliability

There are four general ways of characterising failure occurrences in time:

1. time of failure,
2. time interval between failures,
3. cumulative failure experienced up to a given time,
4. failures experienced in a time interval.

Software Reliability

Uses of Reliability Studies

There are at least four other ways in which software reliability measures can be of great value to the software engineer, manager or user.

1. you can use software reliability measures to evaluate software engineering technology quantitatively.

2. Software reliabilty measures offer you the posibility of evaluating development status during the test phases of a project
3. one can use software reliability measures to monitor the operational performance of software and to control new features added and design changes made to the software.

4. a quantitative understanding of software quality and the various factors influencing it and affected by it enriches into the software product and the software development process.

Software Reliability Metric

- Reliability metrics are units of measure for system reliability
- System reliability is measured by counting the number of operational failures and relating these to demands made on the system at the time of failure
- A long-term measurement program is required to assess the reliability of critical systems

Reliability Metrics - 1

- Probability of Failure on Demand (POFOD)

  POFOD = 0.001

  For one in every 1000 requests the service fails per time unit

- Rate of Fault Occurrence (ROCOF)

  ROCOF = 0.02

  Two failures for each 100 operational time units of operation

Reliability Metrics - 2

- Mean Time to Failure (MTTF)
  - average time between observed failures (aka MTBF)

- Availability = MTBF / (MTBF+MTTR)
  - MTBF = Mean Time Between Failure
  - MTTR = Mean Time to Repair

- Reliability = MTBF / (1+MTBF)

Software Quality Assurance

What is Quality?

- Conformance to software requirements is the foundation from which software quality is measured.
- Specified standards are used to define the development criteria that are used to guide the manner in which software is engineered.
- Software must conform to implicit requirements (ease of use, maintainability, reliability, etc.) as well as its explicit requirements.

What is Quality? -1

Quality – developed product meets it's specification

Problems:

• Development organization has requirements exceeding customer's specifications (added cost of product development)

• Certain quality characteristics can not be specified in unambiguous terms (i.e. maintainability)

• Even if the product conforms to it's specifications, users may not consider it to be a quality product (because users may not be involved in the development of the requirements)

Quality Concepts - 1

- Variation control is the heart of quality control
- Software engineers strive to control the
  - process applied
  - resources expended
  - end product quality attributes
- Quality of design
  - refers to characteristics designers specify for the end product to be constructed

Quality Concepts - 2

- Quality of conformance
  - degree to which design specifications are followed in manufacturing the product
- Quality control
  - series of inspections, reviews, and tests used to ensure conformance of a work product to its specifications
- Quality assurance
  - auditing and reporting procedures used to provide management with data needed to make proactive decisions

Quality Costs

- Prevention costs
  - quality planning, formal technical reviews, test equipment, training
- Appraisal costs
  - in-process and inter-process inspection, equipment calibration and maintenance, testing
- Failure costs
  - rework, repair, failure mode analysis
- External failure costs
  - complaint resolution, product return and replacement, help line support, warranty work

What are SQA, SQP, SQC, and SQM?

SQA includes all 4 elements…

- **Software Quality Assurance** – establishment of network of organizational procedures and standards leading to high-quality software
2. **Software Quality Planning** – selection of appropriate procedures and standards from this framework and adaptation of these to specific software project
3. **Software Quality Control** – definition and enactment of processes that ensure that project quality procedures and standards are being followed by the software development team
4. **Software Quality Metrics** – collecting and analyzing quality data to predict and control quality of the software product being developed

Why are Standards Important?

- Standards provide encapsulation of best, or at least most appropriate, practice
- Standards provide a framework around which the quality assurance process may be implemented
- Standards assist in continuity of work when it's carried out by different people throughout the software product lifecycle

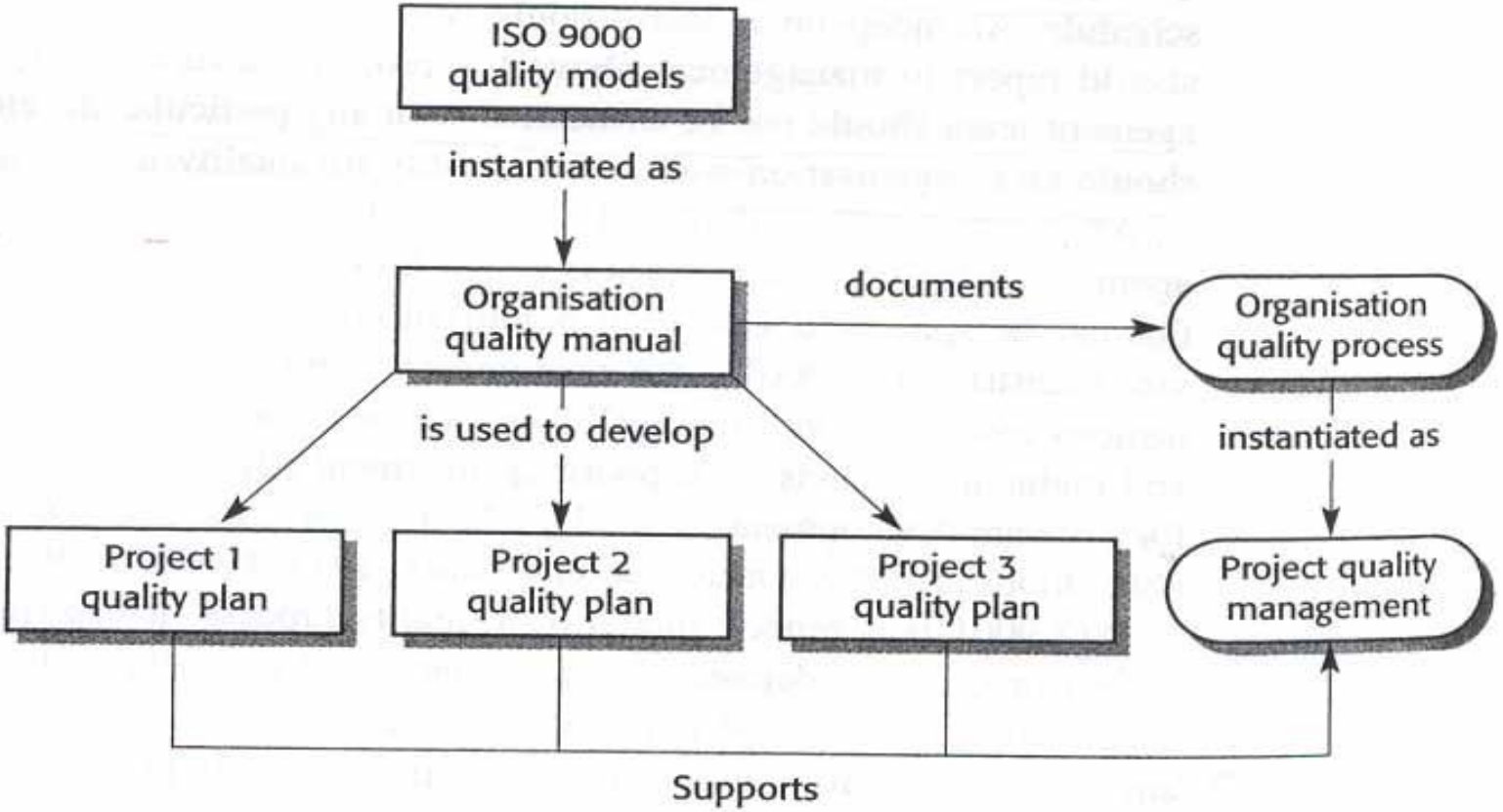
Standards should not be avoided. If they are too extensive for the task at hand, then they should be tailored.

Why are Standards Important?

- Standards provide encapsulation of best, or at least most appropriate, practice
- Standards provide a framework around which the quality assurance process may be implemented
- Standards assist in continuity of work when it's carried out by different people throughout the software product lifecycle

Standards should not be avoided. If they are too extensive for the task at hand, then they should be tailored.

SDS a Simplistic approach

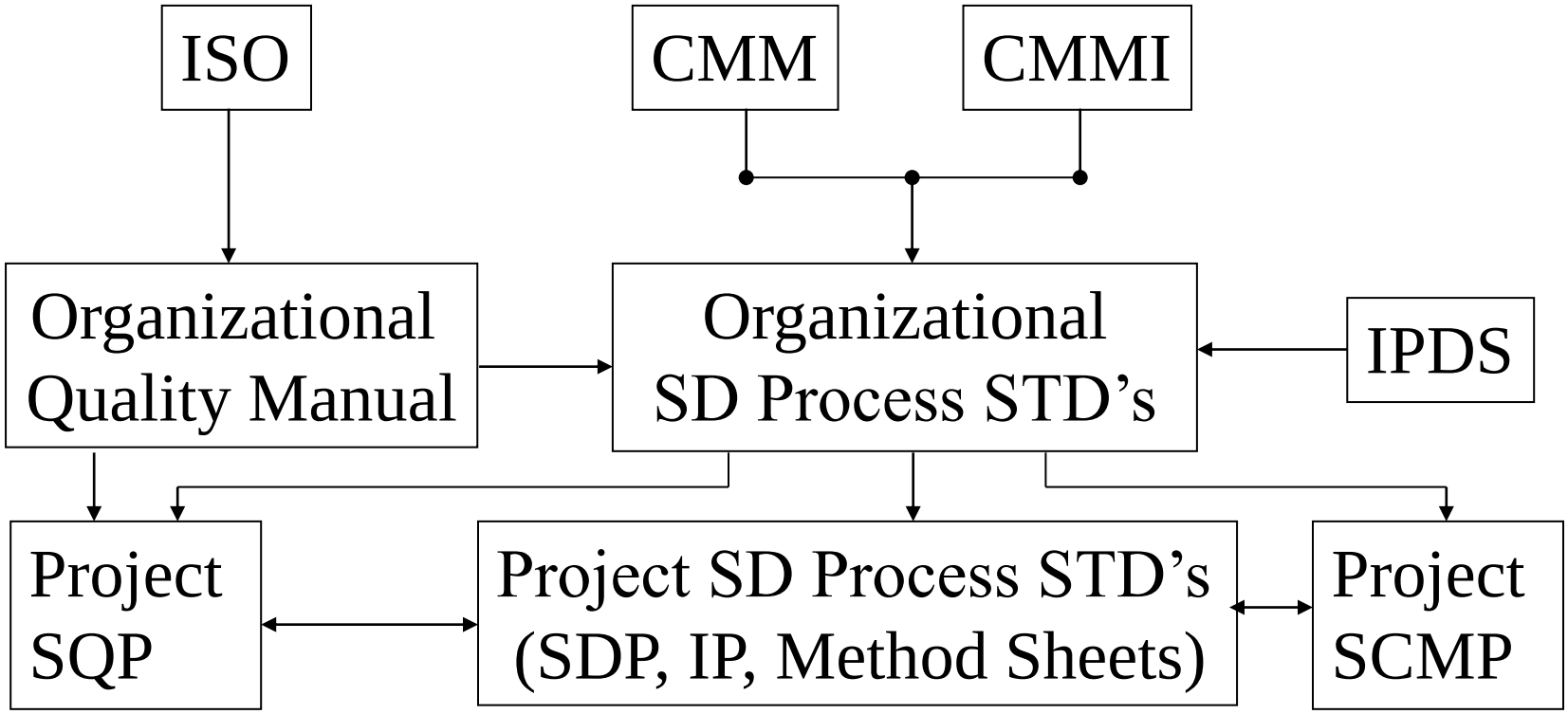


In most mature organizations:

- ISO is not the only source of SDS
- Process and Product standards are derived independently
- Product standards are not created by SQA

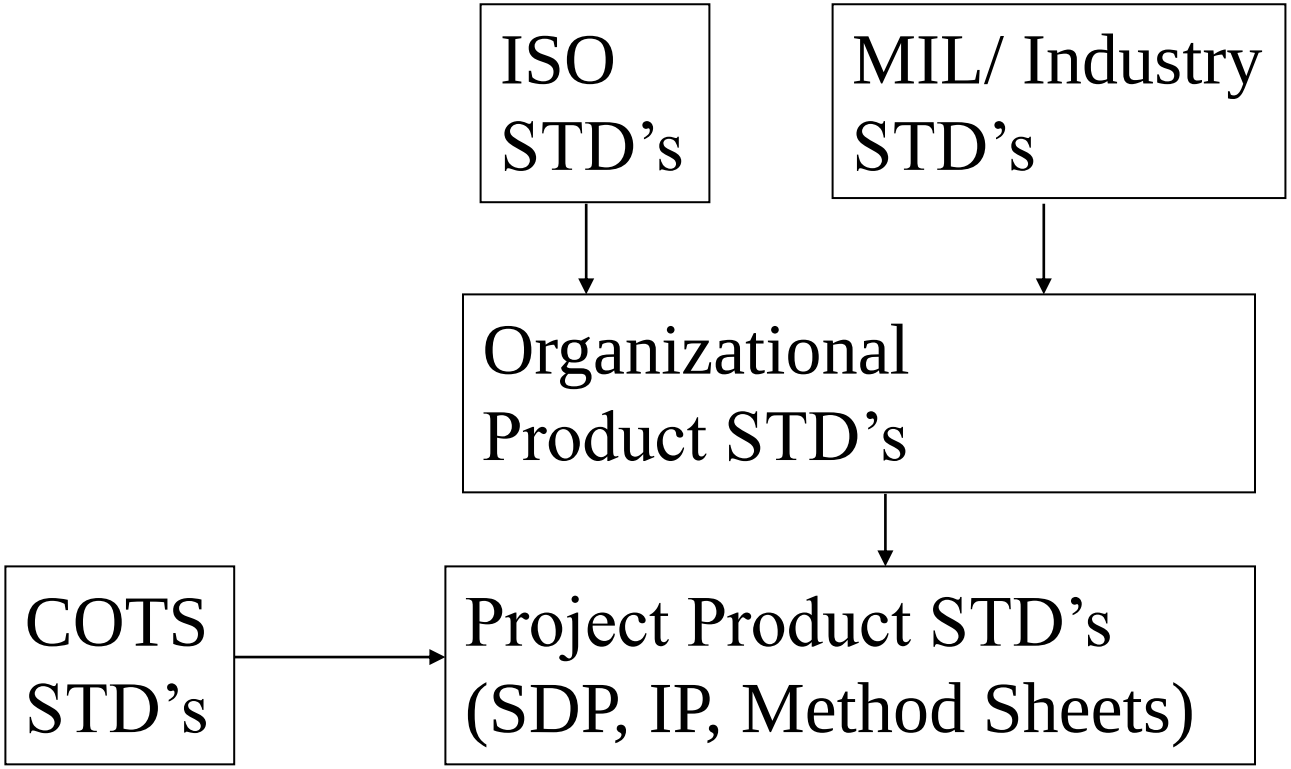
Process Standards

Process Standards – standards that define the process which should be followed during software development



Product Standards

Product Standards – standards that apply to software product being developed



Quality Models

McCall 's Quality model

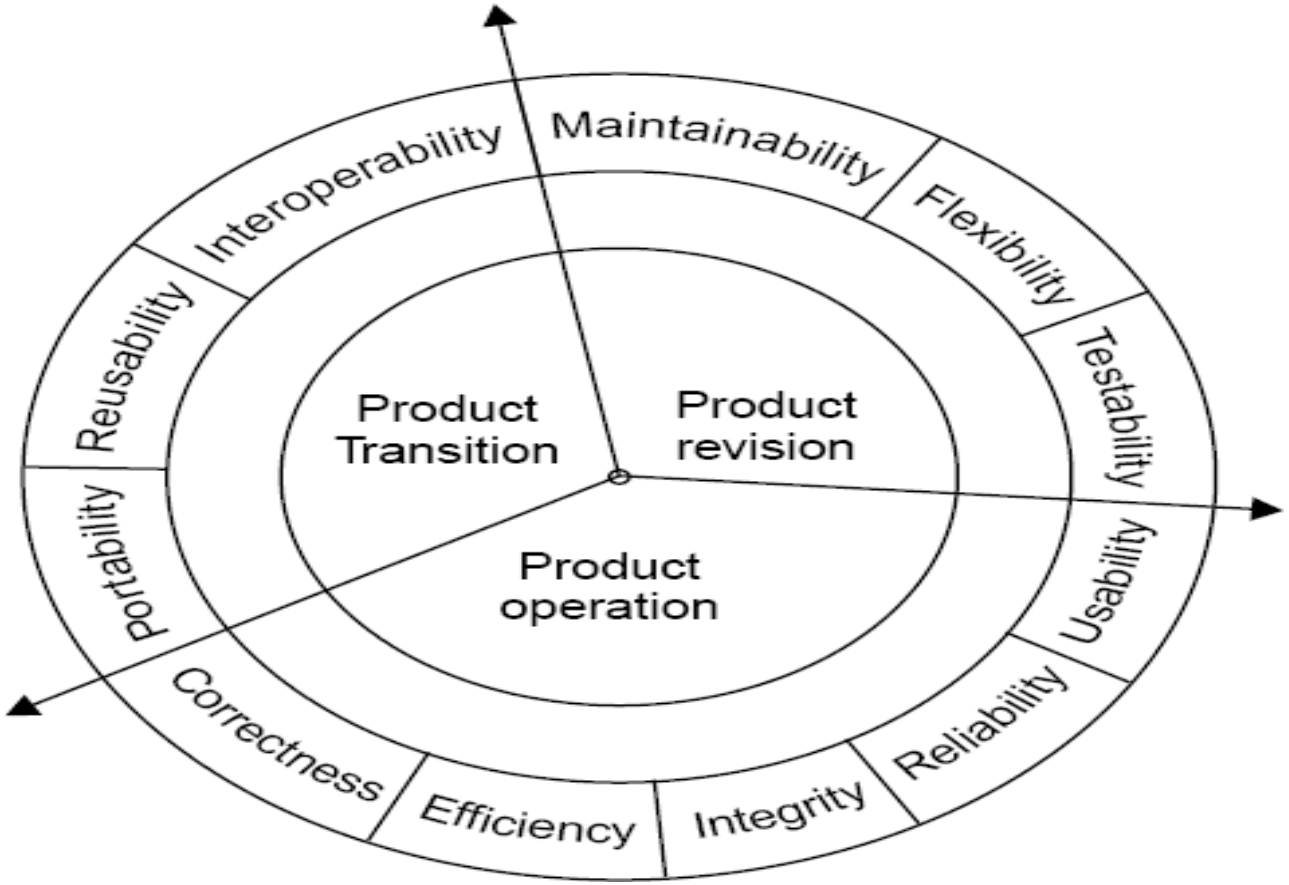


Fig 7.9: Software quality factors

McCall 's Quality model - 1

i. Product Operation

Factors which are related to the operation of a product are combined. The factors are:

- Correctness
- Efficiency
- Integrity
- Reliability
- Usability

These five factors are related to operational performance, convenience, ease of usage and its correctness. These factors play a very significant role in building customer's satisfaction.

McCall 's Quality model - 2

ii. Product Revision

The factors which are required for testing & maintenance are combined and are given below:

- Maintainability
- Flexibility
- Testability

These factors pertain to the testing & maintainability of software. They give us idea about ease of maintenance, flexibility and testing effort. Hence, they are combined under the umbrella of product revision.

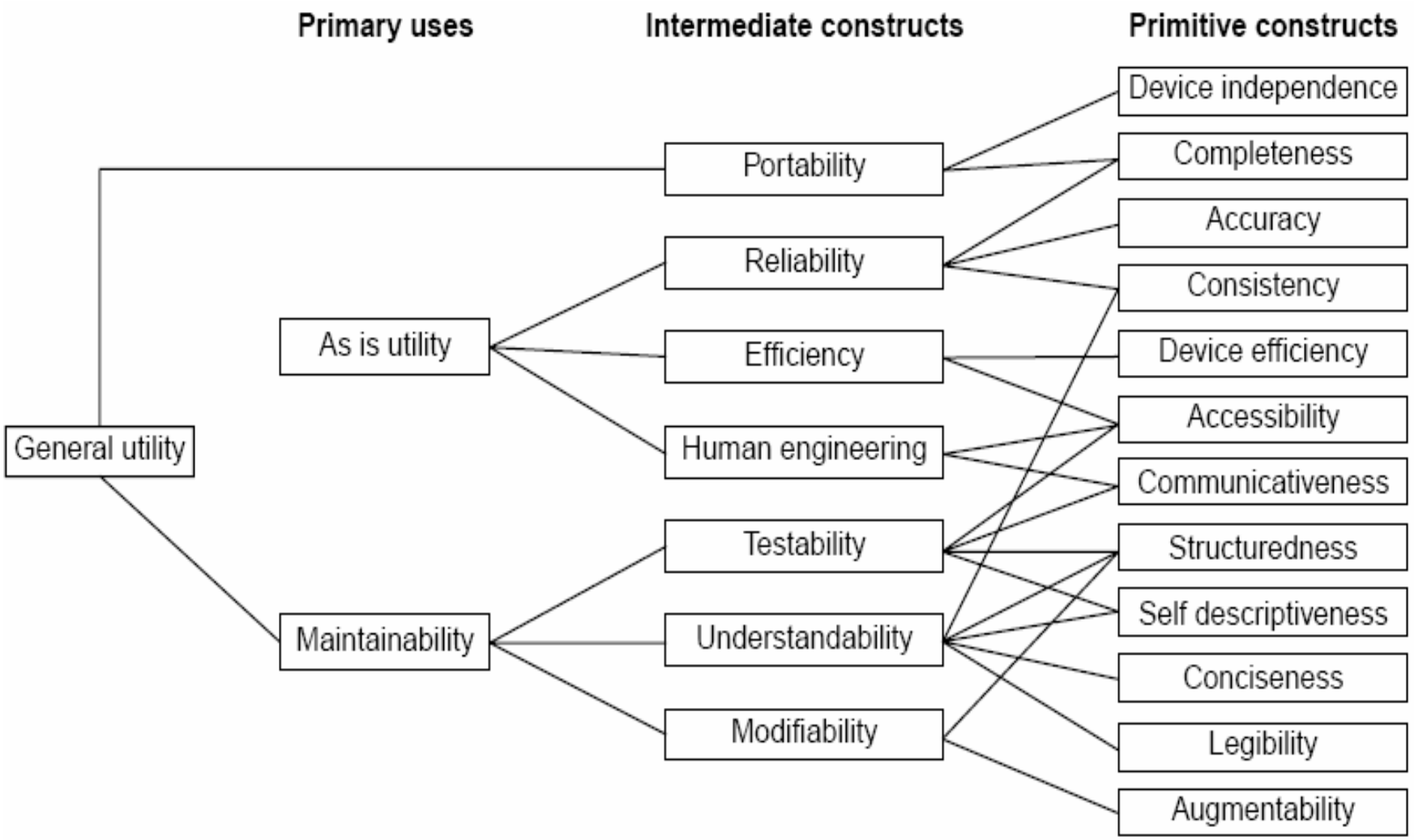
McCall 's Quality model - 3

iii. Product Transition

We may have to transfer a product from one platform to an other platform or from one technology to another technology. The factors related to such a transfer are combined and given below:

- Portability
- Reusability
- Interoperability

Boehm software Quality Model



FURPS Quality Factors

- Functionality
- Usability
- Reliability
- Performance
- Supportability

ISO 9126 Quality Factors

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

Capability Maturity Model

It is a strategy for improving the software process, irrespective of the actual life cycle model used.

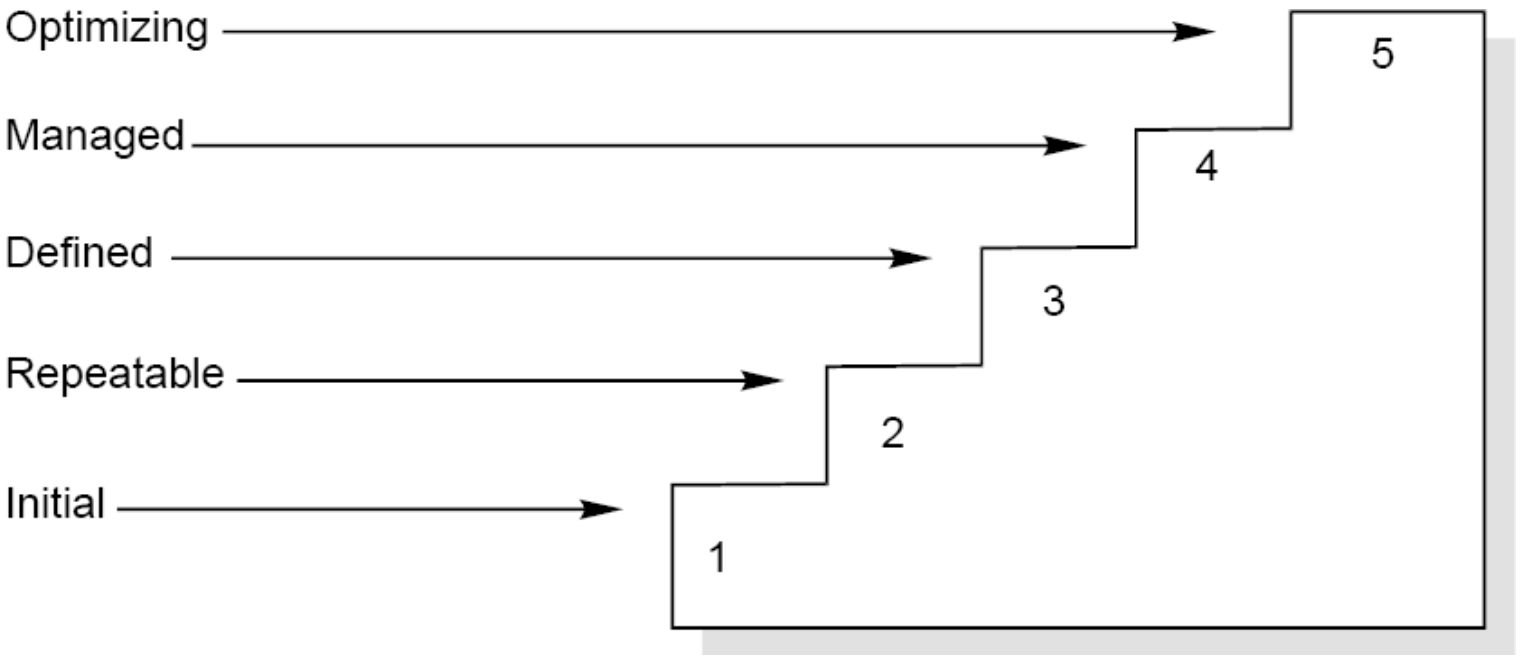


Fig.7.23: Maturity levels of CMM

Capability Maturity Model - 1

Maturity Levels:

- ✓ Initial (Maturity Level 1)
- ✓ Repeatable (Maturity Level 2)
- ✓ Defined (Maturity Level 3)
- ✓ Managed (Maturity Level 4)
- ✓ Optimizing (Maturity Level 5)

Capability Maturity Model - 2

Maturity Level	Characterization
Initial	Adhoc Process
Repeatable	Basic Project Management
Defined	Process Definition
Managed	Process Measurement
Optimizing	Process Control

Fig.7.24: The five levels of CMM

Capability Maturity Model - 3

- Key Process Areas

The key process areas at level 2 focus on the software project's concerns related to establishing basic project management controls, as summarized below:

Requirements Management (RM)	Establish a common relationship between the customer requirements and the developers in order to understand the requirements of the project.
Software Project Planning (PP)	Establish reasonable plans for performing the software engineering and for managing the software project.
Software Project Tracking and Oversight (PT)	Establish adequate visibility into actual progress so that management can take effective actions when the software project's performance deviates significantly from the software plans.
Software Subcontract Management (SM)	Select qualified software subcontractors and manage them effectively.
Software Quality Assurance (QA)	Provide management with appropriate visibility into the process being used by the software project and of the products being built.
Software Configuration Management (CM)	Establish and maintain the integrity of the products of the software project throughout the project's software life cycle.

Capability Maturity Model - 4

The key process areas at level 3 address both project and organizational issues, as summarized below:

Organization Process Focus (PF)	Establish the organizational responsibility for software process activities that improve the organization's overall software process capability.
Organization Process Definition (PD)	Develop and maintain a usable set of software process assets that improve process performance across the projects and provide a basis for cumulative, long-term benefits to the organization.
Training Program (TP)	Develop the skills and knowledge of individuals so that they can perform their roles effectively and efficiently.
Integrated Software Management (IM)	Integrate the software engineering and management activities into a coherent, defined software process that is tailored from the organization's standard software process and related process assets.

(Contd.)...

Capability Maturity Model - 5

Software Product Engineering (PE)	Consistently perform a well-defined engineering process that integrates all the software engineering activities to produce correct, consistent software products effectively and efficiently.
Inter group Coordination (IC)	Establish a means for the software engineering group to participate actively with the other engineering groups so the project is better able to satisfy the customer's needs effectively and efficiently.
Peer Reviews (PR)	Remove defects from the software work products early and efficiently. An important corollary effect is to develop a better understanding of the software work products and of the defects that can be prevented.

Capability Maturity Model - 6

The key process areas at level 4 focus on establishing a quantitative understanding of both the software process and the software work products being built, as summarized below:

Quantitative Process Management (QP)	Control the process performance of the software project quantitatively.
Software Quality Management (QM)	Develop a quantitative understanding of the quality of the project's software products and achieve specific quality goals.

Capability Maturity Model - 7

The key process areas at level 5 cover the issues that both the organization and the projects must address to implement continuous and measurable software process improvement, as summarized below:

Defect Prevention (DP)	Identify the causes of defects and prevent them from recurring.
Technology Change Management (TM)	Identify beneficial new technologies (i.e., tools, methods, and processes) and transfer them into the organization in an orderly manner.
Process Change Management (PC)	Continually improve the software processes used in the organization with the intent of improving software quality, increasing productivity, and decreasing the cycle time for product development.

ISO 9000

The SEI capability maturity model initiative is an attempt to improve software quality by improving the process by which software is developed.

ISO-9000 series of standards is a set of document dealing with quality systems that can be used for quality assurance purposes. ISO-9000 series is not just software standard. It is a series of five related standards that are applicable to a wide variety of industrial activities, including design/ development, production, installation, and servicing. Within the ISO 9000 Series, standard ISO 9001 for quality system is the standard that is most applicable to software development.

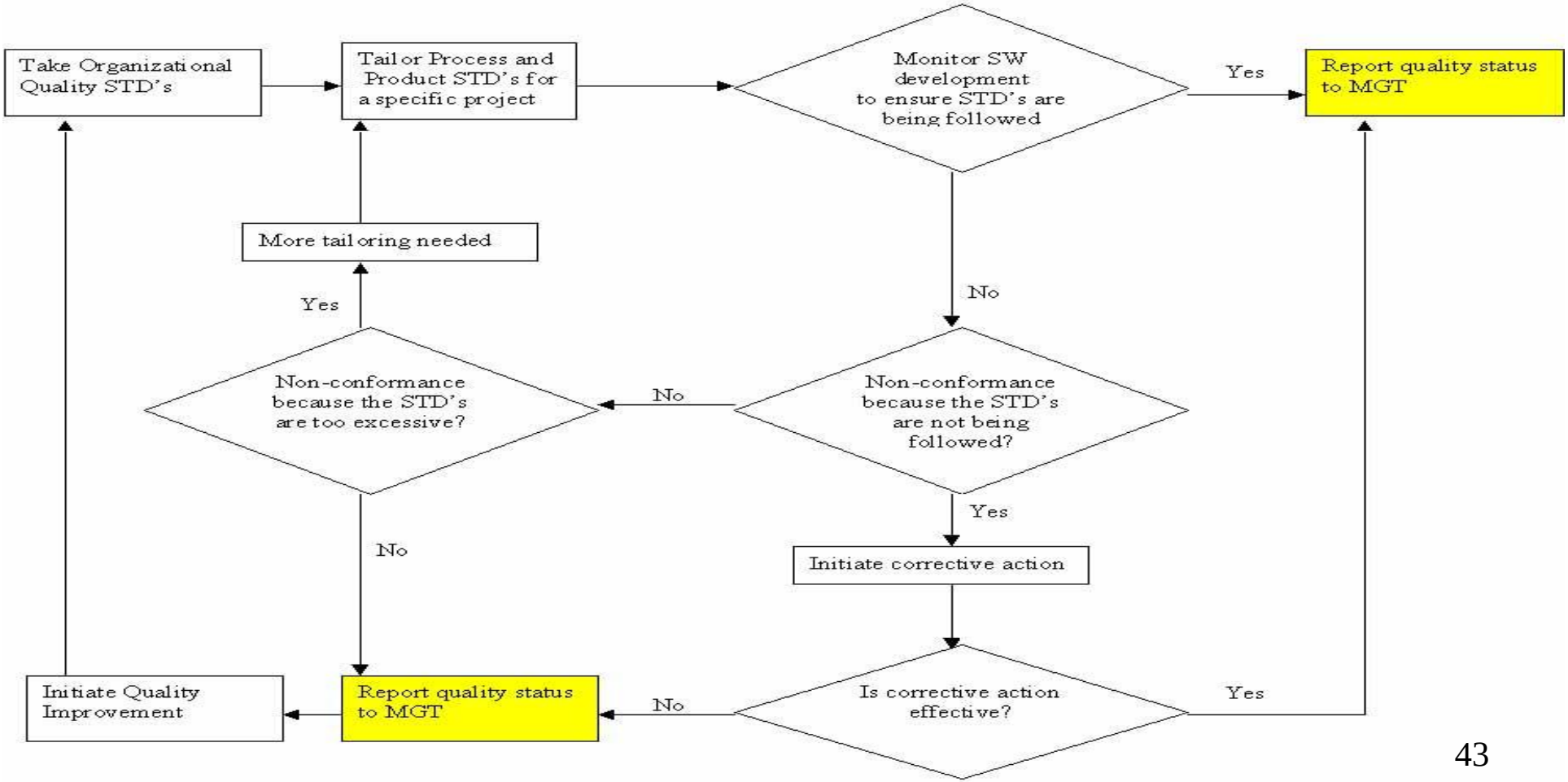
ISO - 9001 Elements

Quality System Requirements
Management Responsibility
Quality system
Contract review
Design Control
Document control
Purchasing
Purchaser supplied product
Product identification and traceability
Process control
Inspection and testing
Inspection, measuring and test equipment
Inspection and test status
Control of non-conforming product
Corrective action
Handling, storage, preservation, packaging and shipping
Quality records
Internal quality audits
Training
Servicing
Statistical techniques

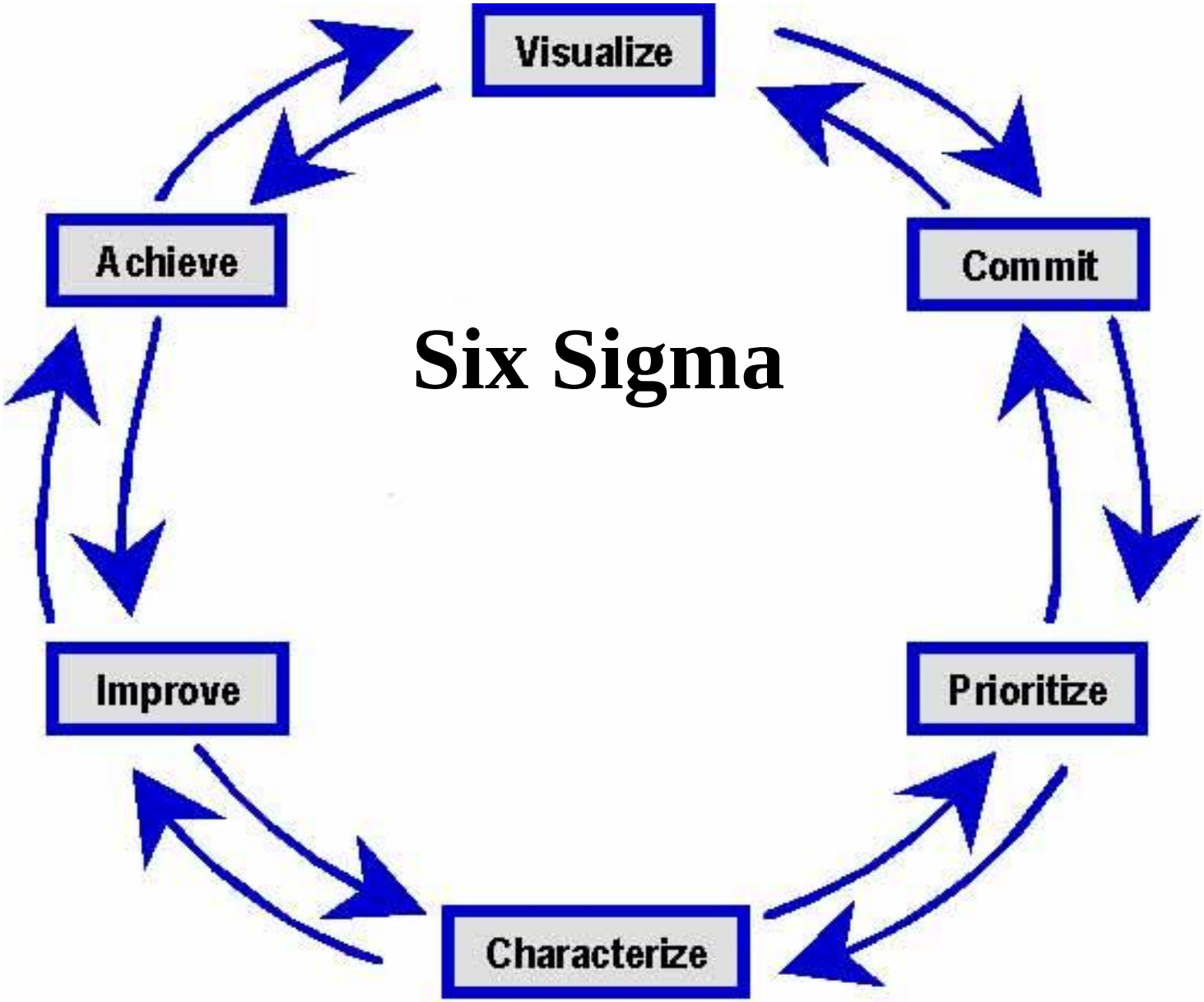
Software Quality Responsibilities
Management Responsibility
Quality system
Contract review
Design Control
Document control
Purchasing
-Product identification and traceability
Process control
Inspection and testing
-Inspection and test status
-Corrective action
-Quality records
Internal quality audits
Training
-
Statistical techniques

Process and Product Quality Creative Approach

- **Quality Improvement** – identifying good quality products, examining the processes used to develop these products, and then generalizing these processes so that they can be applied everywhere



Quality Improvement – The Wheel of 6 Sigma



Quality Improvement – Six Sigma Process

• **Visualize** – Understand how it works now and imagine how it will work in the future

• **Commit** – Obtain commitment to change from the stakeholders

• **Prioritize** – Define priorities for incremental improvements

• **Characterize** – Define existing process and define the time progression for incremental improvements

• **Improve** – Design and implement identified improvements

• **Achieve** – Realize the results of the change

Continuity and Independence of SQA

• Software Quality Assurance team must be independent in order to take an objective view of the process and report problems to senior management directly

• If prescribed process is inappropriate for the type of software product which is being developed, then it should be tailored

• The standards must be upheld no matter how small the task. Prototyping doesn't mean no standards. It means tailored standards.

• Quality is **FREE**, if it's **Everyone's Responsibility**!

Software Quality Control

Methods of Software Quality Control

SQC involves overseeing the software development process to ensure that the procedures and STD's are being followed

The following activities constitute SQC:

• **Quality Reviews -** in-process reviews of processes and products

Reviews are the **most widely used method** of **validating** the quality of processes and products. Reviews make **quality everyone's responsibility**. **Quality** must be **built-in.** SQE is responsible for writing Quality Engineering Records (QERs) documenting their participation in these reviews.

• **Tests -** end-result verifications of products. These verifications are conducted after the software has been developed. Test procedures are followed during conduct of these activities. SQE is responsible for keeping the logs and some times for writing the test report.

• **Quality Audits -** in-process verifications of processes. These audits are conducted periodically (twice a month) to assess compliance to the process STD's.

Quality Reviews

• **Peer reviews -** reviews of processes and products by groups of people. These reviews require pre-review preparation by all participants. If a participant is not prepared, then the review is not effective. This type of review requires participation of the SQE, moderator, recorder, author(s), and one or more critical reviewers. All issues found during these reviews are documented on AR forms.

• **Walkthroughs -** reviews of products by groups of people mostly without preparation. For example a requirements traceability review is a walkthrough. It involves tracing a requirement from customer requirements to the test procedures. All issues found during these reviews are documented on CAR forms.

• **Desk inspections -** reviews of products by individuals. These reviews involve people reviewing products by themselves (not in a group) and then submitting their comments to the author(s). The issues found during these reviews are treated in informal manner.

Tests

• **Engineering Dry-run -** test conducted by engineering without SQE. These tests include Unit Tests and engineering dry-runs of the formal tests. These engineering dry-runs are used to verify correctness and completeness of the test procedures. Also, these is the final engineering verification of the end-product before sell-off to SQE. All issues found during these tests are documented on STR forms.

• **SQE Dry-run -** test conducted by SQE. These tests include PQT, FAT and SAT dry-runs. These tests are used to verify the end-product before the formal test with the customer. An SQE is sometimes responsible for writing the test report. However, if a separate test group is available, then SQE is relived of this obligation. All issues found during these tests are documented on STR forms.

• **TFR -** test conducted as "RFR - run-for-record" with the SQE and the customer. These tests include FAT and SAT. These tests are conducted to sell the end-product off to the customer. SQE is present at all such tests. All issues found during these tests are documented on STR forms.

Quality Audits

• **SQE Audits -** audits conducted by SQE to verify that the process STD's are being followed. Examples of these audits are IPDS compliance, Configuration Control, and Software Engineering Management. All findings for these audits are documented on QER forms. The results of the audits are distributed to the next level of management (above project level). If the issue(s) are not fixed then the findings are elevated to upper management.

• **Independent Audits -** audits conducted by ISO generalists or other independent entities to verify that the process STD's are being followed. These audits are usually conducted on a division/facility level. The results of these audits are distributed to upper management.

Software Configuration Management

SCM – activities assuring that software products are properly identified and their transition is tracked. In many mature organizations SCM is not part of SQA responsibilities.

- **Baseline Identification** – identification of initial state of the product
- **Change Identification** – identification of changes made to the baseline
- **Change Control** – documentation of changes via revision history, change summary, or using automated development tools (ClearCase or Apex)
- **Status Accounting** – reporting changes to others and monitoring completeness of the project archives
- **Preservation** – keeper of the software products