# UNIT - 5

# Software Reliability

Software reliability is also defined as the probability that a software system fulfills its assigned task in a given environment for a predefined number of input cases, assuming that the hardware and the input are free of error.

Software Reliability is an essential connection of software quality, composed with functionality, usability, performance, serviceability, capability, installability, maintainability, and documentation. Software Reliability is hard to achieve because the complexity of software turns out to be high. While any system with a high degree of complexity, containing software, will be hard to reach a certain level of reliability, system developers tend to push complexity into the software layer, with the speedy growth of system size and ease of doing so by upgrading the software.

**For example**, large next-generation aircraft will have over 1 million source lines of software on-board; next-generation air traffic control systems will contain between one and two million lines; the upcoming International Space Station will have over two million lines on-board and over 10 million lines of ground support software; several significant life-critical defense systems will have over 5 million source lines of software. While the complexity of software is inversely associated with software reliability, it is directly related to other vital factors in software quality, especially functionality, capability, etc.

# Software Failure Mechanisms

The software failure can be classified as:

**Transient failure:** These failures only occur with specific inputs.

**Permanent failure:** This failure appears on all inputs.

**Recoverable failure:** System can recover without operator help.

**Unrecoverable failure:** System can recover with operator help only.

**Non-corruption failure:** Failure does not corrupt system state or data.

**Corrupting failure:** It damages the system state or data.

Software failures may be due to bugs, ambiguities, oversights or misinterpretation of the specification that the software is supposed to satisfy, carelessness or incompetence in writing code, inadequate testing, incorrect or unexpected usage of the software or other unforeseen problems.

## Hardware vs. Software Reliability

| **Hardware Reliability** | **Software Reliability** |
| --- | --- |
| Hardware faults are mostly physical faults. | Software faults are design faults, which are tough to visualize, classify, detect, and correct. |
| Hardware components generally fail due to wear and tear. | Software component fails due to bugs. |
| In hardware, design faults may also exist, but physical faults generally dominate. | In software, we can simply find a strict corresponding counterpart for "**manufacturing**" as the hardware manufacturing process, if the simple action of uploading software modules into place does not count. Therefore, the quality of the software will not change once it is uploaded into the storage and start running |
| Hardware exhibits the failure features shown in the following figure:  Software Failure Mechanisms  It is called the **bathtub curve**. Period A, B, and C stand for burn-in phase, useful life phase, and end-of-life phase respectively. | Software reliability does not show the same features similar as hardware. A possible curve is shown in the following figure:  Software Failure Mechanisms  If we projected software reliability on the same axes. |

## What is Statistical Testing (ST)?

Statistical Testing makes use of statistical methods to determine the reliability of the program. Statistical testing focuses on how faulty programs can affect its operating conditions.

## How to perform ST?

Software is tested with the test data that statistically models the working environment.

Failures are collated and analyzed.

From the computed data, an estimate of program's failure rate is calculated.

A Statistical method for testing the possible paths is computed by building an algebraic function.

Statistical testing is a bootless activity as the intent is NOT to find defects.

# Software Quality

Software quality product is defined in term of its fitness of purpose. That is, a quality product does precisely what the users want it to do. For software products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in the SRS document. Although "fitness of purpose" is a satisfactory interpretation of quality for many devices such as a car, a table fan, a grinding machine, etc.for software products, "fitness of purpose" is not a wholly satisfactory definition of quality.

**Example:** Consider a functionally correct software product. That is, it performs all tasks as specified in the SRS document. But, has an almost unusable user interface. Even though it may be functionally right, we cannot consider it to be a quality product.

**The modern view of a quality associated with a software product several quality methods such as the following:**

**Portability:** A software device is said to be portable, if it can be freely made to work in various operating system environments, in multiple machines, with other software products, etc.

**Usability:** A software product has better usability if various categories of users can easily invoke the functions of the product.

**Reusability:** A software product has excellent reusability if different modules of the product can quickly be reused to develop new products.

**Correctness:** A software product is correct if various requirements as specified in the SRS document have been correctly implemented.

**Maintainability:** A software product is maintainable if bugs can be easily corrected as and when they show up, new tasks can be easily added to the product, and the functionalities of the product can be easily modified, etc.

## Software Quality Management System

A quality management system is the principal methods used by organizations to provide that the products they develop have the desired quality.

**A quality system subsists of the following:**

**Managerial Structure and Individual Responsibilities:** A quality system is the responsibility of the organization as a whole. However, every organization has a sever quality department to perform various quality system activities. The quality system of an arrangement should have the support of the top management. Without help for the quality system at a high level in a company, some members of staff will take the quality system seriously.

**Quality System Activities:** The quality system activities encompass the following:

Auditing of projects

Review of the quality system

Development of standards, methods, and guidelines, etc.

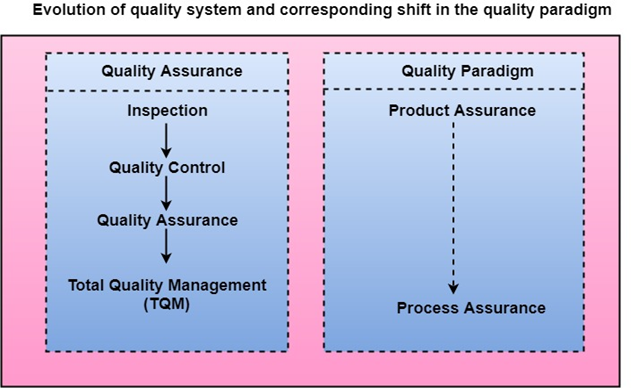
Production of documents for the top management summarizing the effectiveness of the quality system in the organization.

## Evolution of Quality Management System

Quality systems have increasingly evolved over the last five decades. Before World War II, the usual function to produce quality products was to inspect the finished products to remove defective devices. Since that time, quality systems of organizations have undergone through four steps of evolution, as shown in the fig. The first product inspection task gave method to quality control (QC).

Total quality management (TQM) advocates that the procedure followed by an organization must be continuously improved through process measurements. TQM goes stages further than quality assurance and aims at frequently process improvement. TQM goes beyond documenting steps to optimizing them through a redesign. A term linked to TQM is Business Process Reengineering (BPR).

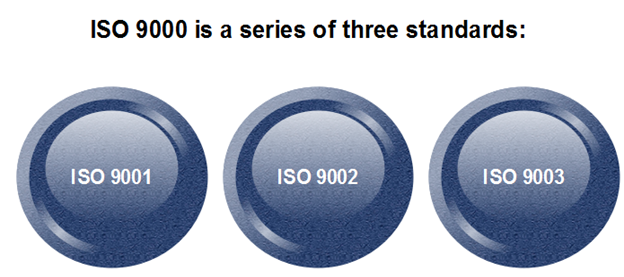
BPR aims at reengineering the method business is carried out in an organization. From the above conversation, it can be stated that over the years, the quality paradigm has changed from product assurance to process assurance, as shown in fig.



# ISO 9000 Certification

ISO (International Standards Organization) is a group or consortium of 63 countries established to plan and fosters standardization. ISO declared its 9000 series of standards in 1987. It serves as a reference for the contract between independent parties. The ISO 9000 standard determines the guidelines for maintaining a quality system. The ISO standard mainly addresses operational methods and organizational methods such as responsibilities, reporting, etc. ISO 9000 defines a set of guidelines for the production process and is not directly concerned about the product itself.

## Types of ISO 9000 Quality Standards



The ISO 9000 series of standards is based on the assumption that if a proper stage is followed for production, then good quality products are bound to follow automatically. The types of industries to which the various ISO standards apply are as follows.

1. **ISO 9001:** This standard applies to the organizations engaged in design, development, production, and servicing of goods. This is the standard that applies to most software development organizations.
2. **ISO 9002:** This standard applies to those organizations which do not design products but are only involved in the production. Examples of these category industries contain steel and car manufacturing industries that buy the product and plants designs from external sources and are engaged in only manufacturing those products. Therefore, ISO 9002 does not apply to software development organizations.
3. **ISO 9003:** This standard applies to organizations that are involved only in the installation and testing of the products. For example, Gas companies.

## How to get ISO 9000 Certification?

An organization determines to obtain ISO 9000 certification applies to ISO registrar office for registration. The process consists of the following stages:



1. **Application:** Once an organization decided to go for ISO certification, it applies to the registrar for registration.
2. **Pre-Assessment:** During this stage, the registrar makes a rough assessment of the organization.
3. **Document review and Adequacy of Audit:** During this stage, the registrar reviews the document submitted by the organization and suggest an improvement.
4. **Compliance Audit:** During this stage, the registrar checks whether the organization has compiled the suggestion made by it during the review or not.
5. **Registration:** The Registrar awards the ISO certification after the successful completion of all the phases.
6. **Continued Inspection:** The registrar continued to monitor the organization time by time.

# Capability Maturity Model (CMM)

The Capability Maturity Model (CMM) is a procedure used to develop and refine an organization's software development process.

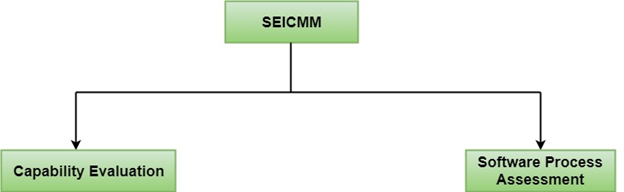
The model defines a five-level evolutionary stage of increasingly organized and consistently more mature processes.

CMM was developed and is promoted by the Software Engineering Institute (SEI), a research and development center promote by the U.S. Department of Defense (DOD).

Capability Maturity Model is used as a benchmark to measure the maturity of an organization's software process.

## Methods of SEICMM

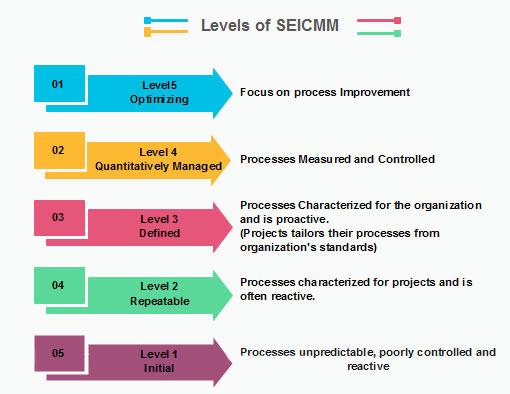
There are two methods of SEICMM:



**Capability Evaluation:** Capability evaluation provides a way to assess the software process capability of an organization. The results of capability evaluation indicate the likely contractor performance if the contractor is awarded a work. Therefore, the results of the software process capability assessment can be used to select a contractor.

**Software Process Assessment:** Software process assessment is used by an organization to improve its process capability. Thus, this type of evaluation is for purely internal use.

SEI CMM categorized software development industries into the following five maturity levels. The various levels of SEI CMM have been designed so that it is easy for an organization to build its quality system starting from scratch slowly.



### Level 1: Initial

Ad hoc activities characterize a software development organization at this level. Very few or no processes are described and followed. Since software production processes are not limited, different engineers follow their process and as a result, development efforts become chaotic. Therefore, it is also called a chaotic level.

### Level 2: Repeatable

At this level, the fundamental project management practices like tracking cost and schedule are established. Size and cost estimation methods, like function point analysis, COCOMO, etc. are used.

### Level 3: Defined

At this level, the methods for both management and development activities are defined and documented. There is a common organization-wide understanding of operations, roles, and responsibilities. The ways through defined, the process and product qualities are not measured. ISO 9000 goals at achieving this level.

### Level 4: Managed

At this level, the focus is on software metrics. Two kinds of metrics are composed.

**Product metrics** measure the features of the product being developed, such as its size, reliability, time complexity, understandability, etc.

**Process metrics** follow the effectiveness of the process being used, such as average defect correction time, productivity, the average number of defects found per hour inspection, the average number of failures detected during testing per LOC, etc. The software process and product quality are measured, and quantitative quality requirements for the product are met. Various tools like Pareto charts, fishbone diagrams, etc. are used to measure the product and process quality. The process metrics are used to analyze if a project performed satisfactorily. Thus, the outcome of process measurements is used to calculate project performance rather than improve the process.

### Level 5: Optimizing

At this phase, process and product metrics are collected. Process and product measurement data are evaluated for continuous process improvement.

SOFTWARE MAINTENANCE

Software maintenance is a part of the Software Development Life Cycle. Its primary goal is to modify and update software application after delivery to correct errors and to improve performance. Software is a model of the real world. When the real world changes, the software require alteration wherever possible.

Software Maintenance is an inclusive activity that includes error corrections, enhancement of capabilities, deletion of obsolete capabilities, and optimization.

## Need for Maintenance

Software Maintenance is needed for:-

* Correct errors
* Change in user requirement with time
* Changing hardware/software requirements
* To improve system efficiency
* To optimize the code to run faster
* To modify the components
* To reduce any unwanted side effects.

Thus the maintenance is required to ensure that the system continues to satisfy user requirements.

## Types of Software Maintenance

### 1. Corrective Maintenance

Corrective maintenance aims to correct any remaining errors regardless of where they may cause specifications, design, coding, testing, and documentation, etc.

### 2. Adaptive Maintenance

It contains modifying the software to match changes in the ever-changing environment.

### 3. Preventive Maintenance

It is the process by which we prevent our system from being obsolete. It involves the concept of reengineering & reverse engineering in which an old system with old technology is re-engineered using new technology. This maintenance prevents the system from dying out.

### 4. Perfective Maintenance

## Software Maintenance

It defines improving processing efficiency or performance or restricting the software to enhance changeability. This may contain enhancement of existing system functionality, improvement in computational efficiency, etc.

# **Software Maintenance Models**

Software maintenance may require different approaches based on your business goals, the industry you function in, the expertise of your tech team, and the predictive trends of the market.

Therefore, along with understanding the different types of software maintenance, you also have to explore various models of software. Based on the kind of problem you are trying to solve, your team can choose the right model from the following options:

## **1. Quick-Fix Model**

A quick-fix model in software maintenance is a method for addressing bugs or issues in the software by prioritizing a fast resolution over a more comprehensive solution.

This approach typically involves making a small, localized change to the software to address the immediate problem rather than fully understanding and addressing the underlying cause. However, organizations adopt this approach of maintenance only in the case of emergency situations that call for quick resolutions.

Under the quick-fix model, tech teams carry out the following software maintenance activities:

* Annotate software changes by including change IDs and code comments
* Enter them into a maintenance history detailing why they made the change and the techniques used by them
* Note each location and merge them via the change ID if there are multiple points in the code change

## **2. Iterative Enhancement Model**

The iterative model is used for small-scale application modernization and scheduled maintenance. Generally, the business justification for changes is ignored in this approach as it only involves the software development team, not the business stakeholders. So, the software team will not know if more significant changes are required in the future, which is quite risky.

The iterative enhancement model treats the application target as a known quantity. It incorporates changes in the software based on the analysis of the existing system. The iterative model best suits changes made to confined application targets, with little cross-impact on other apps or organizations.

## **3. Reuse-Oriented Model**

The reuse-oriented model identifies components of the existing system that are suitable to use again in multiple places. In recent years, this model also includes creating components that can be reused in multiple applications of a system..

There are three ways to incorporate the reuse-oriented model — object and function, application system, and component.

* **Object and function reuse:** This model reuses the software elements that implement a single well-defined object.
* **Application system reuse:** Under this model, developers can integrate new components in an application without making changes to the system or re-configuring it for a specific user to reuse.
* **Component reuse:** Component reuse refers to using a pre-existing component rather than creating a new one in software development. This can include using pre-built code libraries, frameworks, or entire software applications.

## **4. Boehm’s Model**

Introduced in 1978, Boehm’s model focuses on measuring characteristics to get non-tech stakeholders involved with the life cycle of software. The model represents a hierarchical structure of high-level, intermediate, and primitive characteristics of software that define its overall quality.

The high-level characteristics of quality software are:

* **Maintainability:** It should be easy to understand, evaluate, and modify the processes in a system.
* **Portability:** Software systems should help in ascertaining the most effective way to make environmental changes
* **As-is utility:** It should be easy and effective to use an as-is utility in the system.

The intermediate level of characteristics represented by the model displays different factors that validate the expected quality of a software system. These characteristics are:

* **Reliability:** Software performance is as expected, with zero defects.
* **Portability:** The software can run in various environments and on different platforms.
* **Efficiency:** The system makes optimum utilization of code, applications, and hardware resources.
* **Testability:** The software can be tested easily and the users can trust the results.
* **Understandability:** The end-user should be able to understand the functionality of the software easily and thus, use it effectively.
* **Usability:** Efforts needed to learn, use, and comprehend different software functions should be minimum.

The primitive characteristics of quality software include basic features like device independence, accessibility, accuracy, etc.

## **5. Taute Maintenance Model**

Developed by B.J. Taute in 1983, the Taute maintenance model facilitates development teams to update and perform necessary modifications after executing the software.

The Taute model for software maintenance can be carried out in the following phases:

* **Change request phase:** In this phase, the client sends the request to make changes to the software in a prescribed format.
* **Estimate phase:** Then, developers conduct an impact analysis on the existing system to estimate the time and effort required to make the requested changes.
* **Schedule phase:** Here, the team aggregates the change requests for the upcoming scheduled release and creates the planning documents accordingly.
* **Programming phase:** In the programming phase, requested changes are implemented in the source code, and all the relevant documents, like design documents and manuals, are updated accordingly.
* **Test phase:** During this phase, the software modifications are carefully analyzed. The code is tested using existing and new test cases, along with the implementation of regression testing.
* **Documentation phase:** Before the release, system and user documentation are prepared and updated based on regression testing results. Thus, developers can maintain the coherence of documents and code.
* **Release phase:** The customer receives the new software product and updated documentation. Then the system’s end users perform acceptance testing.

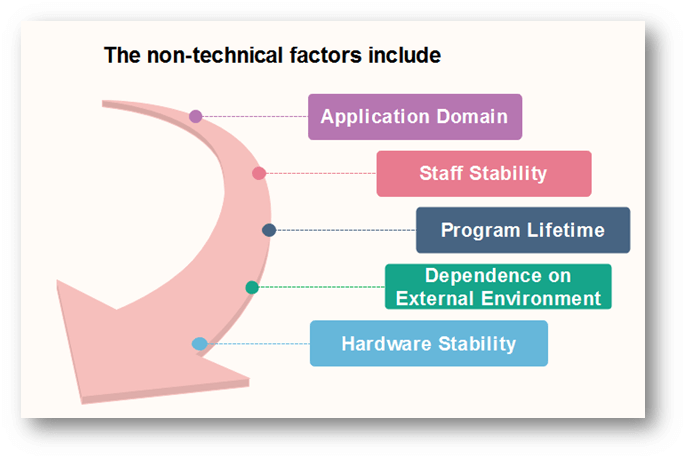
# Software Maintenance Cost Factors

There are two types of cost factors involved in software maintenance.

These are

* Non-Technical Factors
* Technical Factors

## Non-Technical Factors



**1. Application Domain**

* If the application of the program is defined and well understood, the system requirements may be definitive and maintenance due to changing needs minimized.
* If the form is entirely new, it is likely that the initial conditions will be modified frequently, as user gain experience with the system.

**2. Staff Stability**

* It is simple for the original writer of a program to understand and change an application rather than some other person who must understand the program by the study of the reports and code listing.
* If the implementation of a system also maintains that systems, maintenance costs will reduce.
* In practice, the feature of the programming profession is such that persons change jobs regularly. It is unusual for one user to develop and maintain an application throughout its useful life.

**3. Program Lifetime**

* Programs become obsolete when the program becomes obsolete, or their original hardware is replaced, and conversion costs exceed rewriting costs.

**4. Dependence on External Environment**

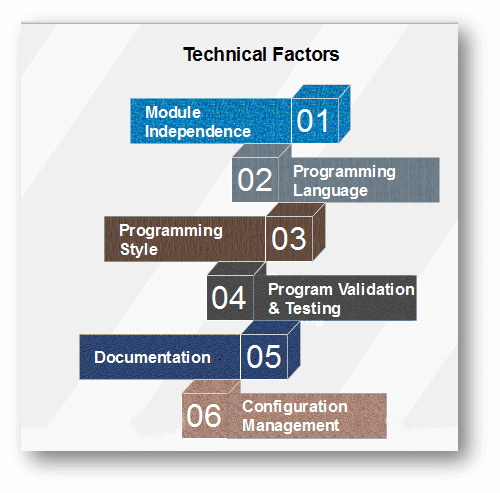
* If an application is dependent on its external environment, it must be modified as the climate changes.
* For example:
* Changes in a taxation system might need payroll, accounting, and stock control programs to be modified.
* Taxation changes are nearly frequent, and maintenance costs for these programs are associated with the frequency of these changes.
* A program used in mathematical applications does not typically depend on humans changing the assumptions on which the program is based.

**5. Hardware Stability**

* If an application is designed to operate on a specific hardware configuration and that configuration does not changes during the program's lifetime, no maintenance costs due to hardware changes will be incurred.
* Hardware developments are so increased that this situation is rare.
* The application must be changed to use new hardware that replaces obsolete equipment.

## Technical Factors

Technical Factors include the following:



**Module Independence**

It should be possible to change one program unit of a system without affecting any other unit.

**Programming Language**

Programs written in a high-level programming language are generally easier to understand than programs written in a low-level language.

**Configuration Management Techniques**

* One of the essential costs of maintenance is keeping track of all system documents and ensuring that these are kept consistent.
* Effective configuration management can help control these costs.

# Software Configuration Management

When we develop software, the product (software) undergoes many changes in their maintenance phase; we need to handle these changes effectively.

Several individuals (programs) works together to achieve these common goals. This individual produces several work product (SC Items) e.g., Intermediate version of modules or test data used during debugging, parts of the final product.

The elements that comprise all information produced as a part of the software process are collectively called a software configuration.

As software development progresses, the number of Software Configuration elements (SCI's) grow rapidly.

**These are handled and controlled by SCM. This is where we require software configuration management.**

A configuration of the product refers not only to the product's constituent but also to a particular version of the component.

Therefore, SCM is the discipline which

* Identify change
* Monitor and control change
* Ensure the proper implementation of change made to the item.
* Auditing and reporting on the change made.

Configuration Management (CM) is a technic of identifying, organizing, and controlling modification to software being built by a programming team.

**The objective is to maximize productivity by minimizing mistakes (errors).**

CM is used to essential due to the inventory management, library management, and updation management of the items essential for the project.

## Why do we need Configuration Management?

Multiple people are working on software which is consistently updating. It may be a method where multiple version, branches, authors are involved in a software project, and the team is geographically distributed and works concurrently. It changes in user requirements, and policy, budget, schedules need to be accommodated.

## Importance of SCM

It is practical in controlling and managing the access to various SCIs e.g., by preventing the two members of a team for checking out the same component for modification at the same time.

**It provides the tool to ensure that changes are being properly implemented.**

It has the capability of describing and storing the various constituent of software.

SCM is used in keeping a system in a consistent state by automatically producing derived version upon modification of the same component.