SOFTWARE ENGINEERING

# MODULE– 4

**CODING AND TESTING**

**CODING:** The objective of the coding phase is to transform the design of a system into code in a high-level language, and then to unit test this code.

**Coding Standards and Guidelines**

Normally, good software development organisations require their programmers to adhere to some well-defined and standard style of coding which is called their **coding standard**.

It is mandatory for the programmers to follow the coding standards.

Compliance of their code to coding standards is verified during code inspection. Any code that does not conform to the coding standards is **rejected** during code review and the code is reworkedby the concerned programmer.

Good software development organisations usually develop **their own coding standards** and guidelines depending on what suits their organization best and based on the specific types of software they develop.

## Coding Standards

1. **Rules for limiting the use of global:** These rules list what types of data can be declared global and what cannot, with a view to limit the data that needs to be defined with global scope.
2. **Standard headers for different modules**: The header of different modules should have standard format and information for ease of understanding and maintenance.

The following is an example of header format that is being used in some companies:

* 1. Name of the module.
  2. Date on which the module was created.
  3. Author’s name.
  4. Modification history.
  5. Synopsis of the module.
  6. Different functions supported in the module, along with their input/output parameters.
  7. Global variables accessed/modified by the module

1. **Naming conventions for global variables**, **local variables, and constant identifiers**: A popular naming convention is that variables are named using mixed case lettering. Global variable names would always start with a capital letter (e.g., GlobalData) and local variable names start with small letters (e.g., localData). Constant names should be formed using capital letters only (e.g., CONSTDATA).
2. **Conventions regarding error return values and exception handling mechanisms** : The way error conditions are reported by different functions in a program should be standard within an organization. For example, all functions while encountering an error condition should either returna 0 or 1 consistently, independent of which programmer has written the code. This facilitates reuse and debugging.

## Coding Guidelines

1. **Do not use a coding style that is too clever or too difficult to understand**: Code should be easy to understand. Many inexperienced engineers actually take pride in writing cryptic and incomprehensible code. Clever coding can obscure meaning of the code and reduce code understandability; thereby making maintenance and debugging difficult and expensive.
2. **Avoid obscure side effects**: The side effects of a function call include modifications to the parameters passed by reference, modification of global variables, and I/O operations. An obscure side effect is one that is not obvious from a casual examination of the code.
3. **Do not use an identifier for multiple purposes**: Programmers often use the same identifier to denote several temporary entities.
4. Each variable should be given a descriptive name indicating its purpose.
5. Use of variables for multiple purposes usually makes future enhancements more difficult.
6. **Code should be well-documented**: As a rule of thumb, there should be at least one comment line on the average for every three source lines of code.
7. **Length of any function should not exceed 10 source lines**: A lengthy function is usually very difficult to understand as it probably has a large number of variables and carries out many different types of computations. For the same reason, lengthy functions are likely to have disproportionately larger number of bugs.
8. **Does not use GO TO statements**: Use of GO TO statements makes a program unstructured. This makes the program very difficult to understand, debug, and maintain

**CODE REVIEW**

Review is a very effective technique to remove defects from **source code**. In fact, review has been acknowledged to be more cost-effective in removing defects as compared to testing.

Testing is an effective defect removal mechanism. However, testing is applicable to only

## executable code.

The reason behind why code review is a much more **cost-effective** strategy to eliminate errors from code compared to testing is that reviews directly **detect errors**.

Normally, the following two types of reviews are carried out on the code of a module:

* + Code walkthrough.
  + Code inspection.

## Code walkthrough:

1. The main objective of code walkthrough is to discover the algorithmic and logical errors in thecode.
2. Code walkthrough is an informal code analysis technique.
3. In this technique, a module is taken up for review after the module has been coded, successfully compiled, and all syntax errors have been eliminated.
4. A few members of the development team are given the code a couple of days before thewalkthrough meeting.
5. Each member selects some test cases and simulates execution of the code by hand (i.e., tracesthe execution through different statements and functions of the code).
6. The members note down their findings of their walkthrough and discuss those in awalkthrough meeting where the coder of the module is present.

## Code Inspection:

1. The principal aim of code inspection is to check for the presence of some common types of errors that usually creep into code due to programmer mistakes and oversights and to checkwhether coding standards have been adhered to.
2. The programmer usually receives feedback on programming style, choice of algorithm, andprogramming techniques.

Following is a list of some classical programming errors which can be checked during code inspection:

* + Use of uninitialized variables.
  + Jumps into loops.
  + Non-terminating loops.
  + Incompatible assignments.
  + Array indices out of bounds.
  + Improper storage allocation and deallocation.
  + Use of incorrect logical operators or incorrect precedence among operators.
  + Dangling reference caused when the referenced memory has not been allocated.

**SOFTWARE DOCUMENTATION**

When software is developed, in addition to the executable files and the source code, several kinds of documents such as users’ manual, software requirements specification (SRS) document, design document, test document, installation manual, etc., are developed as part of the software engineering process. All these documents are considered a vital part of any good software development practice. Good documents are helpful in the following ways:

Good documents help enhance understandability of code

Documents help the users to understand and effectively use the system. Good documents help to effectively tackle the manpower turnover1 problem.

Production of good documents helps the manager to effectively track the progress of theproject.

Different types of software documents can broadly be classified into the following:

## Internal documentation:

1. These are provided in the source code itself.
2. Internal documentation can be provided in the code in several forms.
3. The important types of internal documentation are the following:
   1. Comments embedded in the source code.
   2. Use of meaningful variable names.
   3. Module and function headers.
   4. Code indentation.
   5. Code structuring (i.e., code decomposed into modules and functions).
   6. Use of enumerated types.
   7. Use of constant identifiers.
   8. Use of user-defined data types

**External documentation**: These are the supporting documents such as SRS document, installationdocument, user manual, design document, and test document

## Gunning’s fog index:

Gunning’s fog index Gunning’s fog index (developed by Robert Gunning in 1952) is a metric that has been designed to measure the readability of a document. The computed metric value (fog index) of a document indicates the number of years of formal education that a person should have, in order to be able to comfortably understand that document.

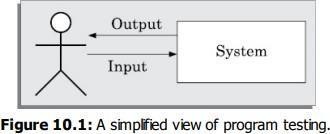


Example 10.1 Consider the following sentence: “The Gunning’s fog index is based on the premise that use of short sentences and simple words makes a document easy to understand.” Calculate its Fog index. The fog index of the above example sentence.

**Tool link:** <http://gunning-fog-index.com/>

# TESTING

**Definition:** Testing a program involves executing the program with a set of test inputs and observing ifthe program behaves as expected. If the program fails to behave as expected, then the input data and the conditions under which it fails are noted for later debugging and error correction.



# BLACK-BOX TESTING

In black-box testing, test cases are designed from an examination of the input/output values only and no knowledge of design or code is required. The following are the two main approaches available to design black box test cases:

* Equivalence class partitioning
* Boundary value analysis

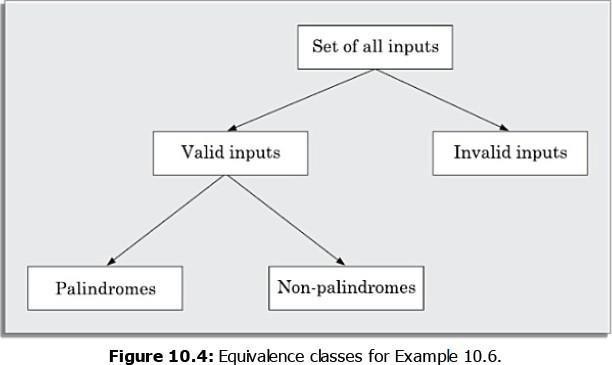
## Equivalence Class Partitioning

In the equivalence class partitioning approach, the domain of input values to the program under test ispartitioned into a set of equivalence classes. The partitioning is done such that for every input data belonging to the same equivalence class, the program behaves similarly.

**Example 1 :** For a software that computes the square root of an input integer that can assume values in the range of 0 and 5000. Determine the equivalence classes and the black box test suite.

**Answer:** There are three equivalence classes—The set of negative integers, the set of integers in the range of 0 and 5000, and the set of integers larger than 5000. Therefore, the test cases must include representatives for each of the three equivalence classes. A possible test suite can be: **{–5,500,6000**}.

**Example 2:** Design equivalence class partitioning test suite for a function that reads a character



string of size less than five characters and displays whether it is a palindrome.

**Answer:** The equivalence classes are the leaf level classes shown in Figure 10.4. The equivalence classes are palindromes, non-palindromes, and invalid inputs. Now, selecting one representative value from each equivalence class, we have the required test suite:

## {abc,aba,abcdef}.

******Boundary Value Analysis**

Boundary value analysis-based test suite design involves designing test cases using the values at theboundaries of different equivalence classes.

Example, programmers may improperly use < instead of <=, or conversely <= for <, etc.

**Example 10.9** For a function that computes the square root of the integer values in the range of 0 and5000, determine the boundary value test suite.

**Answer:** There are three equivalence classes—The set of negative integers, the set of integers in therange of 0 and 5000, and the set of integers larger than 5000. The boundary value-based test suite is:

## {0,-1,5000,5001}.

**Important steps in the black-box test suite design approach:**

1. Examine the input and output values of the program.
2. Identify the equivalence classes.
3. Design equivalence class test cases by picking one representative value from each equivalenceclass.
4. Design the boundary value test cases as follows. Examine if any equivalence class is a range ofvalues. Include the values at the boundaries of such equivalence classes in the test suite.

**WHITE-BOX TESTING**

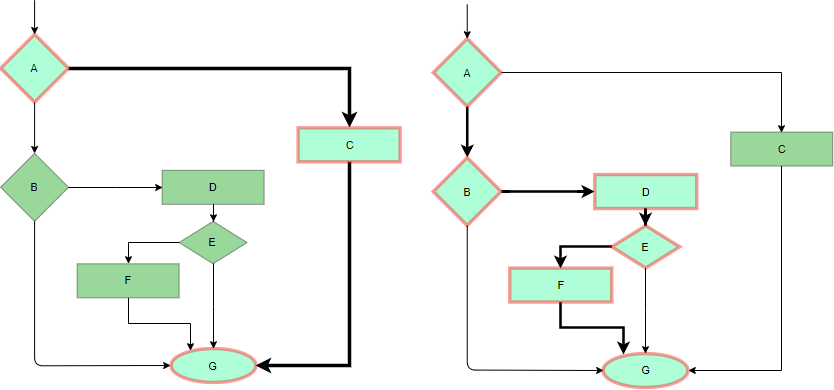
* White box testing is a [software testing technique](https://www.geeksforgeeks.org/software-testing-techniques/) that involves testing the internal structure and workings of a [software application .](https://www.geeksforgeeks.org/what-is-application-software/)
* The tester has access to the source code and uses this knowledge to design test cases that can verify the correctness of the software at the code level.
* White box testing is also known as [structural testing](https://www.geeksforgeeks.org/structural-software-testing/) or [code-based testing,](https://www.geeksforgeeks.org/what-is-code-driven-testing-in-software-testing/) and it is used to test the software’s internal logic, flow, and structure.
* The tester creates test cases to examine the code paths and logic flows to ensure they meet the specified requirements.

## White Box Testing Techniques

One of the main benefits of white box testing is that it allows for testing every part of an application. To achieve complete code coverage, white box testing uses the following techniques:

## Statement Coverage

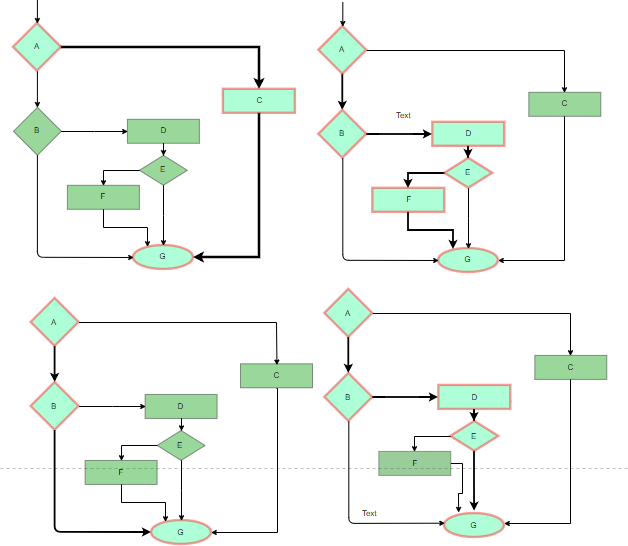
In this technique, the aim is to traverse all statements at least once. Hence, each line of code is tested. In the case of a flowchart, every node must be traversed at least once. Since all lines of code are covered, it helps in pointing out faulty code.



*Statement Coverage Example*

## Branch Coverage

In this technique, test cases are designed so that each branch from all decision points is traversed at least once. In a flowchart, all edges must be traversed at least once.



*4 test cases are required such that all branches of all decisions are covered, i.e, all edges of the flowchart are covered*

## Condition Coverage

In this technique, all individual conditions must be covered as shown in the following example:

* + READ X, Y
  + IF(X == 0 || Y == 0)
  + PRINT ‘0’
  + #TC1 – X = 0, Y = 55
  + #TC2 – X = 5, Y = 0

## Multiple Condition Coverage

In this technique, all the possible combinations of the possible outcomes of conditions are tested at least once. Let’s consider the following example:

* + READ X, Y
  + IF(X == 0 || Y == 0)
  + PRINT ‘0’
  + #TC1: X = 0, Y = 0
  + #TC2: X = 0, Y = 5
  + #TC3: X = 55, Y = 0
  + #TC4: X = 55, Y = 5

## Loop Testing

Loops are widely used and these are fundamental to many algorithms hence, their testing is very important. Errors often occur at the beginnings and ends of loops.

* + **Simple loops:** For simple loops of size n, test cases are designed that:

1. Skip the loop entirely
2. Only one pass through the loop
3. 2 passes
4. m passes, where m < n
5. n-1 ans n+1 passes
   * **Nested loops:** For nested loops, all the loops are set to their minimum count, and we start from the innermost loop. Simple loop tests are conducted for the innermost loop and this is worked outwards till all the loops have been tested.
   * **Concatenated loops:** Independent loops, one after another. Simple loop tests are applied for each. If they’re not independent, treat them like nesting.

# DEBUGGING

## What is Debugging?

Debugging is the process of finding and resolving defects or problems within a computer program that prevent the correct operation of computer software or a system.

## Need for debugging

Once errors are known during a program code, it’s necessary to initially establish the precise program statements liable for the errors and so to repair them.

## Challenges in Debugging

There are a lot of problems at the same time as acting the debugging. These are the following:

1. Debugging is finished through the individual that evolved the software program and it’s miles difficult for that person to acknowledge that an error was made.
2. Debugging is typically performed under a tremendous amount of pressure to fix the supported error as quick as possible.
3. It can be difficult to accurately reproduce input conditions.
4. Compared to the alternative software program improvement activities, relatively little research, literature, and formal preparation exist in the procedure of debugging.

## Debugging Approaches

The following are a number of approaches popularly adopted by programmers for debugging.

## Brute Force Method

This is the foremost common technique of debugging however is that the least economical method. during this approach, the program is loaded with print statements to print the intermediate values with the hope that a number of the written values can facilitate to spot the statement in error. This approach becomes a lot of systematic with the utilization of a symbolic program (also known as a source code debugger), as a result of values of various variables will be simply checked and breakpoints and watch-points can be easily set to check the values of variables effortlessly.

## Backtracking

This is additionally a reasonably common approach. during this approach, starting from the statement at which an error symptom has been discovered, the source code is derived backward till the error is discovered. sadly, because the variety of supply lines to be derived back will increase, the

quantity of potential backward methods will increase and should become unimaginably large so limiting the utilisation of this approach.

## Cause Elimination Method

In this approach, a listing of causes that may presumably have contributed to the error symptom is developed and tests are conducted to eliminate every error. A connected technique of identification of the error from the error symptom is that the package fault tree analysis.

## Program Slicing

This technique is analogous to backtracking. Here the search house is reduced by process slices. A slice of a program for a specific variable at a particular statement is that the set of supply lines preceding this statement which will influence the worth of that variable.

## Examples of error during debugging

Some common example of error during debugging are:

* + Syntax error
  + Logical error
  + Runtime error
  + Stack overflow
  + Index Out of Bound Errors
  + Infinite loops
  + Concurrency Issues
  + I/O errors
  + Environment Dependencies
  + Integration Errors
  + Reference error
  + Type error

**PROGRAM ANALYSIS TOOLS**

Program analysis tools help developers analyze, debug, optimize, and verify software code. These tools can be categorized based on their purpose and method of analysis:

## Static Analysis Tools

These tools analyze code **without executing it**, identifying potential issues such as syntax errors, security vulnerabilities, and coding standard violations.

## Examples:

* + **SonarQube** – Code quality and security analysis.
  + **ESLint** – JavaScript linting.
  + **Pylint** – Python linting.
  + **FindBugs/SpotBugs** – Java bug detection.
  + **Cppcheck** – C/C++ static analysis.

## Dynamic Analysis Tools

These tools analyze code **during execution**, detecting runtime errors, performance issues, and memory leaks.

## Examples:

**Valgrind** – Memory leak detection for C/C++.

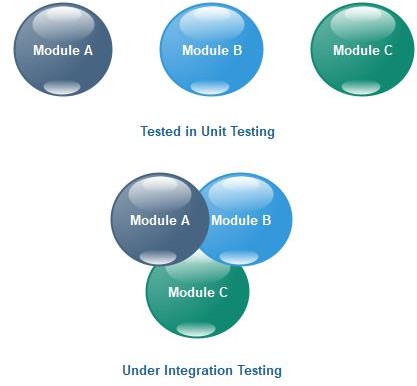
**AddressSanitizer** – Detects memory corruption.

**GDB (GNU Debugger)** – Debugging tool for various languages.

**JProfiler** – Java performance profiling.

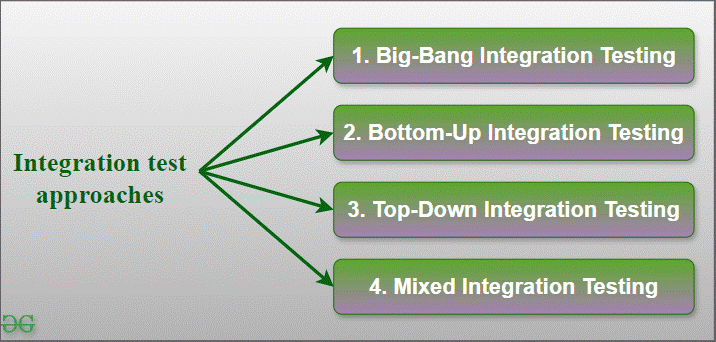
# INTEGRATION TESTING

Integration testing is the second level of the software testing process comes after unit testing. In this testing, units or individual components of the software are tested in a group. The focus of the integration testing level is to expose defects at the time of interaction between integrated components or units.



# INTEGRATION TEST APPROACHES

There are four types of integration testing approaches. Those approaches are the following:



***Integration test approaches***

## Big-Bang Integration Testing

* + It is the simplest integration testing approach, where all the modules are combined and the functionality is verified after the completion of individual module testing.
  + In simple words, all the modules of the system are simply put together and tested.

## Bottom-Up Integration Testing

* + In bottom-up testing, each module at lower levels are tested with higher modules until all modules are tested.
  + The primary purpose of this integration testing is that each subsystem tests the interfaces among various modules making up the subsystem.
  + This integration testing uses test drivers to drive and pass appropriate data to the lower-level modules.

## Top-Down Integration Testing

* + Top-down integration testing technique is used in order to simulate the behaviour of the lower-level modules that are not yet integrated.
  + In this integration testing, testing takes place from top to bottom.
  + First, high-level modules are tested and then low-level modules and finally integrating the low-level modules to a high level to ensure the system is working as intended.

## Mixed Integration Testing

* + A mixed integration testing is also called sandwiched integration testing. A mixed integration testing follows a combination of top down and bottom-up testing approaches.
  + In top-down approach, testing can start only after the top-level module have been coded and unit tested.
  + In bottom-up approach, testing can start only after the bottom level modules are ready. This sandwich or mixed approach overcomes this shortcoming of the top-down and bottom-up approaches.
  + It is also called the hybrid integration testing. also, stubs and drivers are used in mixed integration testing.

# TESTING OBJECT-ORIENTED PROGRAMS

## What is Object-Oriented Testing?

Object-Oriented Testing (OOT) is a [software testing](https://precisetestingsolution.com/tag/best-software-testing-company) technique that focuses on testing the individual objects within an object-oriented system. This approach is based on the principles of object-oriented programming (OOP), which emphasizes encapsulation, inheritance, and polymorphism. By testing objects in isolation and in combination, OOT helps ensure the quality and reliability of object- oriented software systems.

## Developing Test Cases in Object-Oriented Testing

To effectively develop test cases for object-oriented testing, it’s crucial to consider the following aspects:

**Constructor Testing:** Verify that objects are initialized correctly. **Method Testing:** Test each method’s functionality, input validation, error handling, and output correctness.

**Attribute Testing:** Ensure that attributes are assigned and accessed correctly.

**Valid State Testing:** Verify that objects can be created in valid states. **Invalid State Testing:** Test how objects handle invalid states and recover gracefully. **State Transition Testing:** Ensure that objects can transition between different states correctly.

**Inheritance Hierarchy Testing:** Verify that inheritance relationships are defined correctly. **Method Overriding Testing:** Test that overridden methods behave as expected. **Polymorphism Testing:** Ensure that polymorphic behavior is implemented correctly.

**Interface Compliance Testing:** Verify that objects adhere to the specified interfaces.

**Interface Method Testing:** Test the functionality of interface methods.

**Object Interaction Testing:** Test how objects interact with each other, including message passing

**Dependency Testing:** Verify that objects depend on the correct components and services.

# SMOKE TESTING

Smoke testing is a software testing method that verifies the basic functionality of a new build. It's also known as build verification testing or confidence testing.

## Purpose

* Identify potential major defects early in the development cycle
* Determine if the build is stable enough to proceed with more in-depth testing
* Prevent the entire team from wasting time or resources

## Goal of Smoke Testing

The aim of Smoke Testing is:

1. **Stop Wasting Resources:** Refrain from wasting resources on extensive testing if the core functions aren’t working properly.
2. **Time Management**: Save time by recognizing show-stopping concerns early on, so that development teams may rapidly handle important issues.
3. **Making Objective Decisions:** Establish a transparent and impartial framework for determining whether a software build is ready for more, in-depth testing or if it has to be fixed right away.
4. **Continuous Integration:** Make sure that before every new build is integrated into the bigger codebase, it satisfies basic quality criteria in order to support the continuous integration approach.
5. **Communication**: Give quick feedback on the stability of the build to the development and testing teams to help them communicate effectively.

## Tools for Smoke Testing

✅ **Selenium** – Web app automation testing

✅ **JUnit/TestNG** – Java-based testing frameworks

✅ **Cypress** – JavaScript-based UI testing

✅ **Postman** – API testing

✅ **Appium** – Mobile application testing

# SOME GENERAL ISSUES ASSOCIATED WITH TESTING

Key points about testing issues:

## Documentation problems:

Missing or outdated documentation makes it hard for testers to understand requirements and design effective tests.

## Communication gaps:

Misunderstandings between developers and testers regarding features or expectations can lead to incomplete or incorrect testing.

## Unstable test environment:

Fluctuations in network, hardware, or other system components can cause unreliable test results.

## Time pressure:

Tight deadlines often force teams to rush through testing, potentially skipping critical areas

## Resource limitations:

Lack of skilled testers or inadequate testing tools can hinder comprehensive testing

## Test case management issues:

Duplication of test cases or poor organization can lead to inefficient testing processes

## Defect management challenges:

Not having a robust system to track and prioritize identified defects can result in issues slipping into production

## Automation complexities:

Choosing the right automation tools and maintaining test scripts can be difficult, especially for complex applications

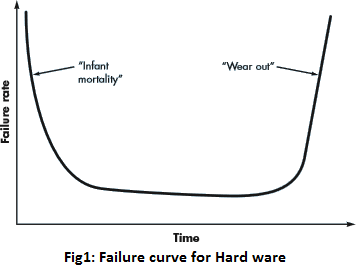
**PART-2**

**SOFTWARE RELIABILITY AND QUALITY MANAGEMENT**

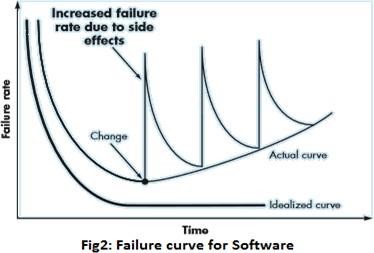
**SOFTWARE RELIABILITY**

Reliability in software is software that has no failure and works in a special time period with a special environment. Hardware reliability is the probability of the absence of any hardware-related system malfunction for a given mission on the other hand software reliability is the probability that the software will provide a failure-free operation in a fixed environment for a fixed interval of time. The article focuses on discussing the difference between Hardware Reliability and Software Reliability.

**Software is deterministic because is doesn’t wear out**: Software is deterministic but hard ware is probabilistic because it does not wear out. Hard ware has high failure rates. Hardware fails because of failure modes like poor quality of fabrication, design error, overload of the component, wear due to old age etc



Software also fails because of some errors in the Software, human operator errors. The important point here is hard ware wears out because of old age but this is not the case in software. Hard ware’s efficiency may decrease day by day but not of software’s.



**STATISTICAL TESTING**

Statistical Testing is a testing method whose objective is to work out the undependable software package products instead of discovering errors. check cases are designed for applied mathematics testing with a wholly different objective than those of typical testing.

Characteristics of Software Statistical Testing

## The characteristics of the software statistical testing are listed below −

* Statistical testing is a critical test method used to analyze the quality of software using the information obtained from the various testing methodologies.
* The statistical testing uses the statistical procedures in the test data to conclude on the characteristics, quality, and robustness of the software.
* The statistical testing detects bugs, and weaknesses in the software by evaluating the information gathered at the time of testing.
* The statistical testing measures the efficiency of the testing process, and helps to make informed decisions on whether the software is ready to be deployed to the production environment.
* The statistical testing guides how to share the test results with the project stakeholders, by highlighting the defects identified, and confirming whether the software meets the user requirements.

## Advantages of Software Statistical Testing

The advantages of the software statistical testing are listed below −

* The statistical testing verifies the different parameters of the software that are supposedly used which ultimately helps to improve the quality of the software.
* The estimation of the reliability of the software using the statistical testing is far more accurate than the other procedures namely ROCOF, POFOD etc.

# 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.

# 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.

## Software Engineering Institute Capability Maturity Model (SEICMM)

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

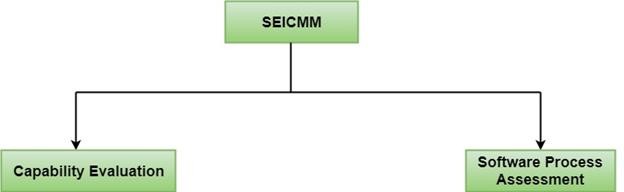
The model defines a five-level evolutionary stage of increasingly organized and consistently more mature processes.

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

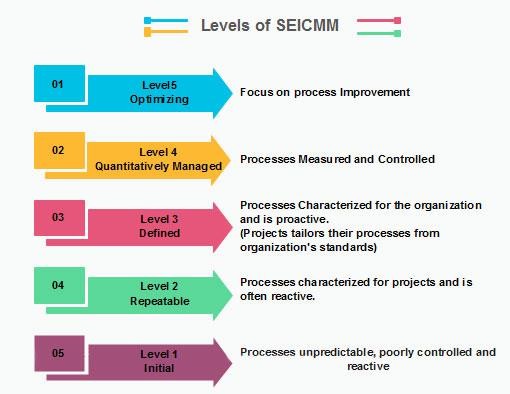
**There are two methods of SEICMM**:



**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.

**FEW OTHER IMPORTANT QUALITY STANDARDS**

## Code reviews:

Regularly reviewing code written by peers to identify potential issues, improve code quality, and maintain coding standards.

# CI/CD:

Automatically building, testing, and deploying code changes frequently to catch issues early and ensure smooth integration.

## Version control:

Tracking changes made to code using tools like Git to manage different versions and collaborate effectively.

## Performance testing:

Evaluating how a system performs under different load conditions to identify bottlenecks and optimize performance.

## Security testing:

Identifying vulnerabilities in the system to prevent security breaches.

## Usability testing:

Observing users interacting with the software to identify areas for improvement in user interface design and functionality.

## Code complexity analysis:

Measuring the complexity of code to identify areas that might be difficult to maintain or understand.

## Defect tracking:

Systematically managing and prioritizing identified software defects throughout the development process.

## Configuration management:

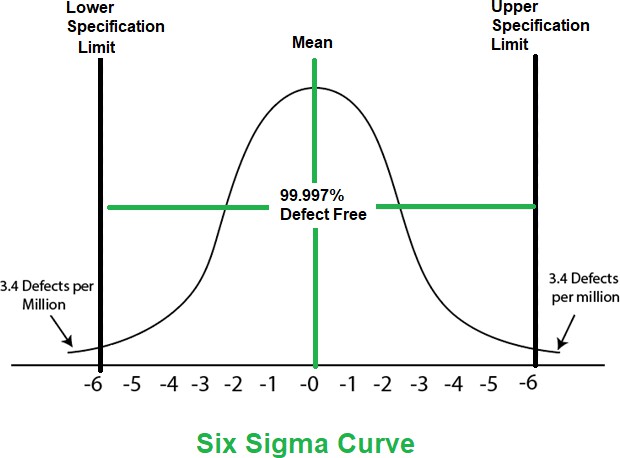
Controlling and managing all components of a software system to ensure consistency and stability.

## Industry standards (e.g., ISO 9001):

Following established guidelines for quality management systems to ensure consistent quality across projects.

**SIX SIGMA**

* Six Sigma is a methodology used by most organizations for process improvement, and It is a statistical concept that aims to define the variation found in any process.
* Six Sigma is a process of producing high and improved quality output. This can be done in two phases – identification and elimination.
* The cause of defects is identified and appropriate elimination is done, which reduces variation in whole processes.
* Six Sigma processes have a failure rate of only 3.4 per million opportunities i.e. 99.99966 percent of Six Sigma products are free from defect, while Five Sigma processes have a failure rate of only 233 errors per million opportunities.



## Characteristics of Six Sigma

The Characteristics of Six Sigma are as follows:

1. Statistical Quality Control: Six Sigma is derived from the Greek Letter ? which denote Standard Deviation in statistics. Standard Deviation is used for measuring the quality of output.
2. Methodical Approach: The Six Sigma is a systematic approach of application in DMAIC and DMADV which can be used to improve the quality of production. DMAIC means for Design-Measure- Analyze-Improve-Control. While DMADV stands for Design-Measure- Analyze-Design-Verify.
3. Fact and Data-Based Approach: The statistical and methodical method shows the scientific basis of the technique.
4. Project and Objective-Based Focus: The Six Sigma process is implemented to focus on the requirements and conditions.
5. Customer Focus: The customer focus is fundamental to the Six Sigma approach. The quality improvement and control standards are based on specific customer requirements.
6. Teamwork Approach to Quality Management: The Six Sigma process requires organizations to get organized for improving quality.