

Black Box Testing

In this testing, the test engineer will analyze the software against requirements, identify the defects or bug, and sends it back to the development team.

Then, the developers will fix those defects, do one round of White box testing, and send it to the testing team.

Here, fixing the bugs means the defect is resolved, and the particular feature is working according to the given requirement.

The main objective of implementing the black box testing is to specify the business needs or the customer's requirements.

In other words, we can say that black box testing is a process of checking the functionality of an application as per the customer requirement. The source code is not visible in this testing; that's why it is known as **black-box testing**.

- It examines the requirement specifications of the system
- Selects a set of proper inputs and determines the possible outcomes
- Prepares test cases for the inputs and executes the test cases. Compares the results with expected outcomes
- If they are any undesired results or errors then correct the errors and repeat the testing

(Note: Identification of equivalence class – Partition any input domain into a minimum of two sets: **valid values** and **invalid values**. For example, if the valid range is 0 to 100 then select one valid input like 49 and one invalid like 104.)

Types of Black Box Testing

Black box testing further categorizes into two parts, which are as discussed below:

- **Functional Testing**
- **Non-function Testing**

Functional Testing

The test engineer will check all the components systematically against requirement specifications is known as **functional testing**. Functional testing is also known as **Component testing**.

In functional testing, all the components are tested by giving the value, defining the output, and validating the actual output with the expected value.

Functional testing is a part of black-box testing as its emphasizes on application requirement rather than actual code. The test engineer has to test only the program instead of the system.

Types of Functional Testing

Just like another type of testing is divided into several parts, functional testing is also classified into various categories.

The diverse **types of Functional Testing** contain the following:

- **Unit Testing**
- **Integration Testing**
- **System Testing**

Unit Testing

Unit testing is the first level of functional testing in order to test any software. In this, the test engineer will test the module of an application independently or test all the module functionality is called **unit testing**.

The primary objective of executing the unit testing is to confirm the unit components with their performance. Here, a unit is defined as a single testable function of a software or an application. And it is verified throughout the specified application development phase.

Integration Testing

Once we are successfully implementing the unit testing, we will go [integration testing](#). It is the second level of functional testing, where we test the data flow between dependent modules or interface between two features is called **integration testing**.

The purpose of executing the integration testing is to test the statement's accuracy between each module.

Types of Integration Testing

Integration testing is also further divided into the following parts:

- **Incremental Testing**
- **Non-Incremental Testing**

Incremental Integration Testing

Whenever there is a clear relationship between modules, we go for incremental integration testing. Suppose, we take two modules and analysis the data flow between them if they are working fine or not.

If these modules are working fine, then we can add one more module and test again. And we can continue with the same process to get better results.

In other words, we can say that incrementally adding up the modules and test the data flow between the modules is known as **Incremental integration testing**.

Types of Incremental Integration Testing

Incremental integration testing can further classify into two parts, which are as follows:

1. **Top-down Incremental Integration Testing**

2. **Bottom-up Incremental Integration Testing**
3. **1. Top-down Incremental Integration Testing**
4. In this approach, we will add the modules step by step or incrementally and test the data flow between them. We have to ensure that the modules we are adding are the **child of the earlier ones**.
5. **2. Bottom-up Incremental Integration Testing**
6. In the bottom-up approach, we will add the modules incrementally and check the data flow between modules. And also, ensure that the module we are adding is the **parent of the earlier ones**.

Non-Incremental Integration Testing/ Big Bang Method

Whenever the data flow is complex and very difficult to classify a parent and a child, we will go for the non-incremental integration approach. The non-incremental method is also known as **the Big Bang method**.

3. System Testing

Whenever we are done with the unit and integration testing, we can proceed with the system testing.

In system testing, the test environment is parallel to the production environment. It is also known as **end-to-end** testing.

In this type of testing, we will undergo each attribute of the software and test if the end feature works according to the business requirement. And analysis the software product as a complete system.

[Alpha Testing](#) is a type of software testing performed to identify bugs before releasing the product to real users or to the public. Alpha Testing is one of the user acceptance tests.

[Beta Testing](#) is performed by real users of the software application in a real environment. Beta testing is one type of User Acceptance Testing.

Difference between Alpha and Beta Testing:

The difference between Alpha and Beta Testing is as follows:

Alpha Testing

- Alpha testing involves both the white box and black box testing.
- Alpha testing is performed by testers who are usually internal employees of the organization.
- Alpha testing is performed at the developer's site.

Beta Testing

- Beta testing commonly uses black-box testing.
- Beta testing is performed by clients who are not part of the organization.
- Beta testing is performed at the end-user of the product.

Alpha Testing

- Reliability and security testing are not checked in alpha testing.
- Alpha testing ensures the quality of the product before forwarding to beta testing.
- Alpha testing requires a testing environment or a lab.
- Alpha testing may require a long execution cycle.
- Developers can immediately address the critical issues or fixes in alpha testing.
- Multiple test cycles are organized in alpha testing.

Beta Testing

- Reliability, security and robustness are checked during beta testing.
- Beta testing also concentrates on the quality of the product but collects users input on the product and ensures that the product is ready for real time users.
- Beta testing doesn't require a testing environment or lab.
- Beta testing requires only a few weeks of execution.
- Most of the issues or feedback collected from the beta testing will be implemented in future versions of the product.
- Only one or two test cycles are there in beta testing.

Non-function Testing

The next part of black-box testing is **non-functional testing**. It provides detailed information on software product performance and used technologies.

Non-functional testing will help us minimize the risk of production and related costs of the software.

Non-functional testing is a combination of **performance, load, stress, usability and, compatibility testing**.

Types of Non-functional Testing

Non-functional testing categorized into different parts of testing, which we are going to discuss further:

- **Performance Testing**
- **Usability Testing**
- **Compatibility Testing**

Advantages of Black Box Testing:

- The tester does not need to have more functional knowledge or programming skills to implement the Black Box Testing.
- It is efficient for implementing the tests in the larger system.
- Tests are executed from the user's or client's point of view.
- Test cases are easily reproducible.
- It is used in finding the ambiguity and contradictions in the functional specifications.

Disadvantages of Black Box Testing:

- There is a possibility of repeating the same tests while implementing the testing process.
- Without clear functional specifications, test cases are difficult to implement.
- It is difficult to execute the test cases because of complex inputs at different stages of testing.
- Sometimes, the reason for the test failure cannot be detected.
- Some programs in the application are not tested.
- It does not reveal the errors in the control structure.
- Working with a large sample space of inputs can be exhaustive and consumes a lot of time.

White Box Testing

In white-box testing, the developer will inspect every line of code before handing it over to the testing team or the concerned test engineers.

the code is noticeable for developers throughout testing; that's why this process is known as **WBT (White Box Testing)**.

In other words, we can say that the **developer** will execute the complete white-box testing for the particular software and send the specific application to the testing team.

The purpose of implementing the white box testing is to emphasize the flow of inputs and outputs over the software and enhance the security of an application.

White box testing is also known as **open box testing, glass box testing, structural testing, clear box testing, and transparent box testing.**

White box testing techniques analyze the internal structures the used data structures, internal design, code structure and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing or clear box testing or structural testing.

The test cases are designed in such a way that every statement in a program is executed at least once

If some statement is not executed then there is a bug in that statement

Advantages:

- White box testing is very thorough as the entire code and structures are tested.
- It results in the optimization of code removing error and helps in removing extra lines of code.
- It can start at an earlier stage as it doesn't require any interface as in case of black box testing.
- Easy to automate.

Disadvantages:

- Main disadvantage is that it is very expensive.

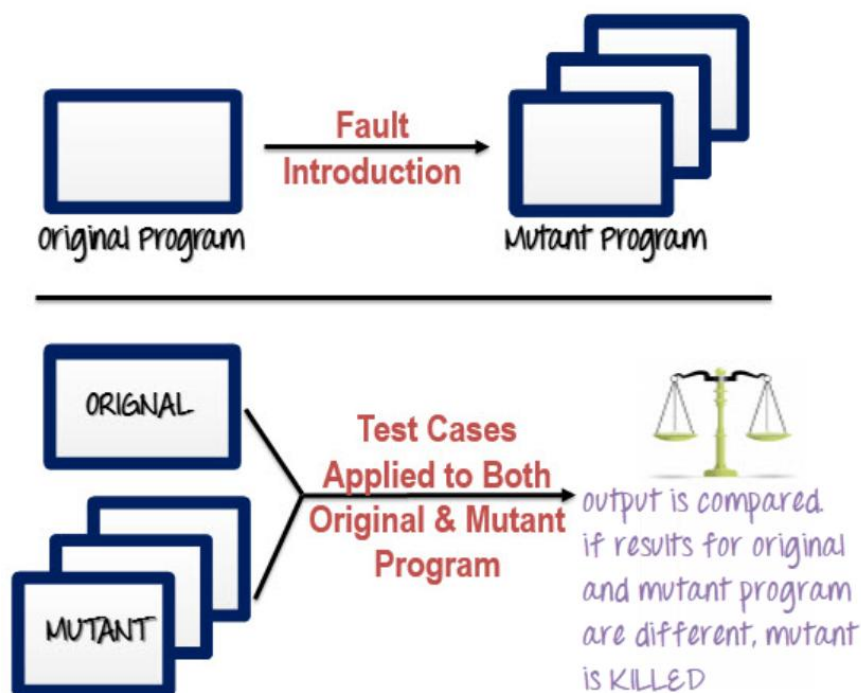
- Redesign of code and rewriting code needs test cases to be written again.
- Testers are required to have in-depth knowledge of the code and programming language as opposed to black box testing.
- Missing functionalities cannot be detected as the code that exists is tested.
- Very complex and at times not realistic.

What is Mutation Testing?

Mutation Testing is a type of software testing in which certain statements of the source code are changed/mutated to check if the test cases are able to find errors in source code. The goal of Mutation Testing is ensuring the quality of test cases in terms of robustness that it should fail the mutated source code.

The changes made in the mutant program should be kept extremely small that it does not affect the overall objective of the program. The different faults to be introduced may be the change of the data type, altering the constant value, changing the order of the operations in an expression, etc. Mutation Testing is also called Fault-based testing strategy as it involves creating a fault in the program and it is a type of White Box Testing which is mainly used for Unit Testing.

How to execute Mutation Testing?



Following are the steps to execute mutation testing (mutation analysis):

Step 1: Faults are introduced into the source code of the program by creating many versions called mutants. Each mutant should contain a single fault, and the goal is to cause the mutant version to fail which demonstrates the effectiveness of the test cases.

Step 2: Test cases are applied to the original program and also to the mutant program. A [Test Case](#) should be adequate, and it is tweaked to detect faults in a program.

Step 3: Compare the results of an original and mutant program.

Step 4: If the original program and mutant programs generate the different output, then that the mutant is killed by the test case. Hence the test case is good enough to detect the change between the original and the mutant program.

Step 5: If the original program and mutant program generate the same output, Mutant is kept alive. In such cases, more effective test cases need to be created that kill all mutants.

How to Create Mutant Programs?



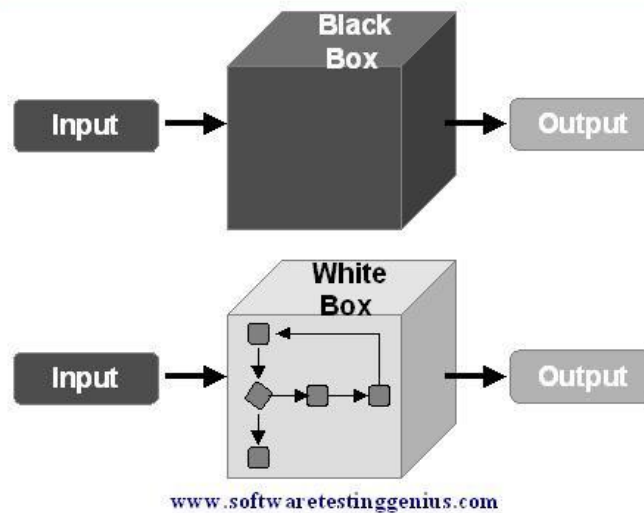
A mutation is nothing but a single syntactic change that is made to the program statement. Each mutant program should differ from the original program by one mutation.

Original Program	Mutant Program
If ($x > y$)	If ($x < y$)
Print "Hello"	Print "Hello"
Else	Else
Print "Hi"	Print "Hi"

S. No.	Black Box Testing	
1.	It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.	It is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software.
2.	Implementation of code is not needed for black box testing.	Code implementation is necessary for white box testing.
3.	It is mostly done by software testers.	It is mostly done by software developers.
4.	No knowledge of implementation is needed.	Knowledge of implementation is required.
5.	It can be referred to as outer or external software testing.	It is the inner or the internal software testing.
6.	It is a functional test of the software.	It is a structural test of the software.
7.	This testing can be initiated based on the requirement specifications document.	This type of testing of software is started after a detail design document.
8.	No knowledge of programming is required.	It is mandatory to have knowledge of programming.

S. No.	Black Box Testing	
9.	It is the behavior testing of the software.	It is the logic testing of the software.
10.	It is applicable to the higher levels of testing of software.	It is generally applicable to the lower levels of software testing.
11.	It is also called closed testing.	It is also called as clear box testing.
12.	It is least time consuming.	It is most time consuming.
13.	It is not suitable or preferred for algorithm testing.	It is suitable for algorithm testing.
14.	Can be done by trial and error ways and methods.	Data domains along with inner or internal boundaries can be better tested.
15.	Example: Search something on google by using keywords	Example: By input to check and verify loops
16.	Black-box test design techniques- <ul style="list-style-type: none"> • Decision table testing • All-pairs testing • Equivalence partitioning • Error guessing 	White-box test design techniques- <ul style="list-style-type: none"> • Control flow testing • Data flow testing • Branch testing
17.	Types of Black Box Testing: <ul style="list-style-type: none"> • Functional Testing • Non-functional testing • Regression Testing 	Types of White Box Testing: <ul style="list-style-type: none"> • Path Testing • Loop Testing • Condition testing
18.	It is less exhaustive as compared to white box testing.	It is comparatively more exhaustive than black box testing.

Comparison among Black-Box & White-Box Tests



DEBUGGING

debugging is the process of fixing a bug in the software. In other words, it refers to identifying, analyzing, and removing errors. This activity begins after the software fails to execute properly and concludes by solving the problem and successfully testing the software. It is considered to be an extremely complex and tedious task because errors need to be resolved at all stages of debugging.

Debugging Approaches/Strategies:

- **Brute Force:** Study the system for a larger duration in order to understand the system. It helps the debugger to construct different representations of systems to be debugging depending on the need. A study of the system is also done actively to find recent changes made to the software.
- **Backtracking:** Backward analysis of the problem which involves tracing the program backward from the location of the failure message in order to identify the region of faulty code. A detailed study of the region is conducted to find the cause of defects.
- **Forward analysis** of the program involves tracing the program forwards using breakpoints or print statements at different points in the program and studying the results. The region where the wrong outputs are obtained is the region that needs to be focused on to find the defect.
- **Using the past experience** of the software debug the software with similar problems in nature. The success of this approach depends on the expertise of the debugger.
- **Cause elimination:** it introduces the concept of binary partitioning. Data related to the error occurrence are organized to isolate potential causes.

SOFTWARE MAINTENANCE

Software Maintenance is the process of modifying a software product after it has been delivered to the customer. The main purpose of software maintenance is to modify and update software applications after delivery to correct faults and to improve performance.

Need for Maintenance –

Software Maintenance must be performed in order to:

- Correct faults.
- Improve the design.
- Implement enhancements.
- Interface with other systems.
- Accommodate programs so that different hardware, software, system features, and telecommunications facilities can be used.
- Migrate legacy software.
- Retire software.

Types of Software Maintenance

Maintenance can be divided into the following:

1. Corrective maintenance:

Corrective maintenance of a software product may be essential either to rectify some bugs observed while the system is in use, or to enhance the performance of the system.

2. Adaptive maintenance:

This includes modifications and updations when the customers need the product to run on new platforms, on new operating systems, or when they need the product to interface with new hardware and software.

3. Perfective maintenance:

A software product needs maintenance to support the new features that the users want or to change different types of functionalities of the system according to the customer demands.

4. Preventive maintenance:

This type of maintenance includes modifications and updations to prevent future problems of the software. It goals to attend problems, which are not significant at this moment but may cause serious issues in future.

Reverse Engineering –

Reverse Engineering is processes of extracting knowledge or design information from anything man-made and reproducing it based on extracted information. It is also called back Engineering.

Software Reverse Engineering –

Software Reverse Engineering is the process of recovering the design and the requirements specification of a product from an analysis of it's code. Reverse Engineering is becoming important, since several existing software products, lack proper documentation, are highly unstructured, or their structure has degraded through a series of maintenance efforts.

Why Reverse Engineering?

Providing proper system documentation.

Recovery of lost information.

Assisting with maintenance.

Facility of software reuse.

Discovering unexpected flaws or faults.

Used of Software Reverse Engineering –

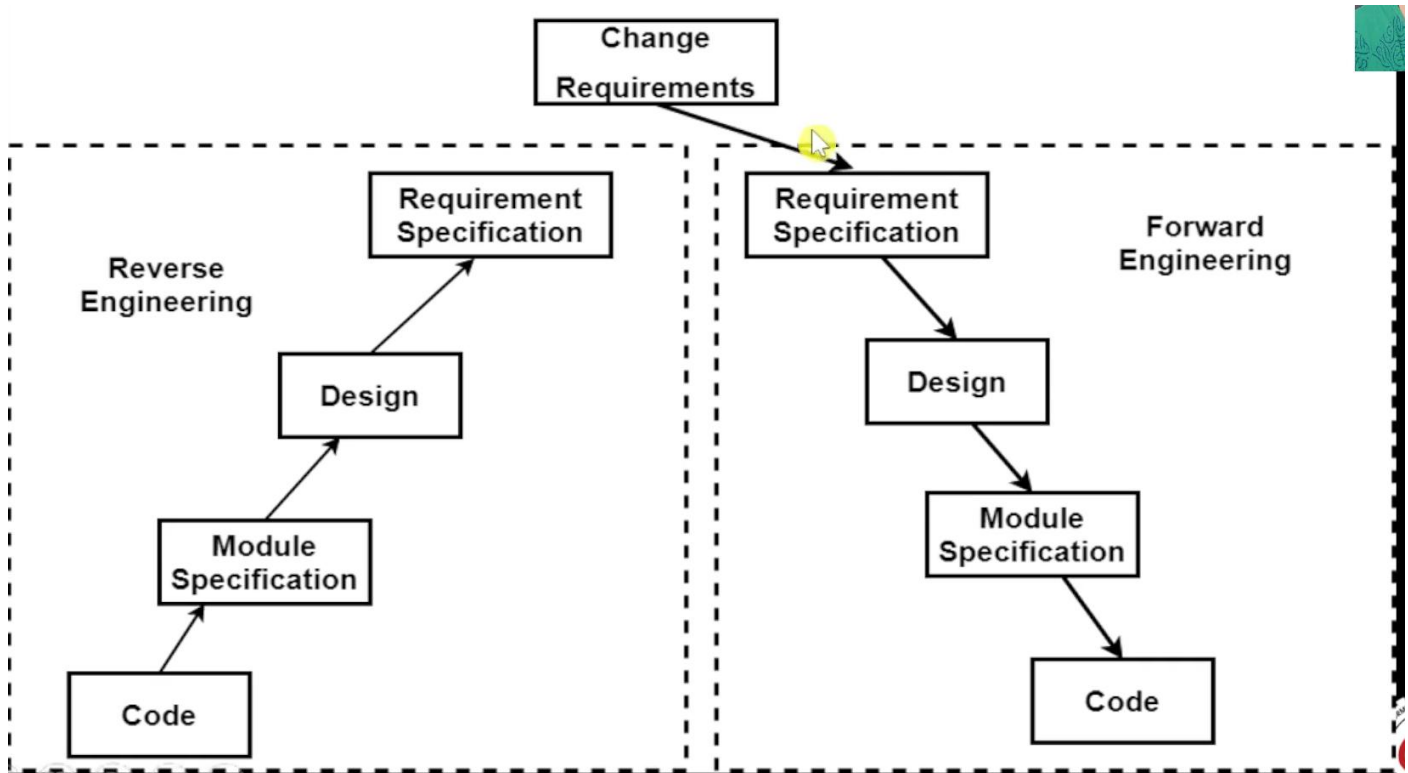
Software Reverse Engineering is used in software design, reverse engineering enables the developer or programmer to add new features to the existing software with or without knowing the source code.

Reverse engineering is also useful in software testing, it helps the testers to study the virus and other malware code .


Re- Engineering

Software Re-engineering is a process of software development which is done to improve the maintainability of a software system. Re-engineering is the examination and alteration of a system to reconstitute it in a new form. This process encompasses a combination of sub-processes like reverse engineering, forward engineering, reconstructing etc.

Re-engineering is the reorganizing and modifying existing software systems to make them more maintainable.



Reliability Issues

- **The quality of the software is determined by its reliability.**
- **The reliability is a failure-free software operation in a specific environment.**
- **The software reliability is difficult as it is complex. For example, the aircraft of this generation will have billions of lines of code.**
- **Software failure is due to errors, incorrect usage of software, incompetent developers, or inadequate testing.** 
- **The reliability may be hardware or software.**
- **Hardware reliability refers to the failure of electronic or mechanical parts due to continuous usage.**

Software Reliability

- **Reliability determines whether the software product is correctly working over a given time period.**
- **Reliability denotes the trustworthiness or dependability of the software.**
- **Reliability is difficult to measure, as it changes when the errors are removed, it relies on the observation and location of the error.**

Hardware and Software Reliability

- **Hardware reliability refers to the component's wear and tear over time, and there is hardware failure.**
- **The software fault will remain till the error is tracked or the design or program is changed.**
- **After the hardware repair, the reliability is maintained, whereas the software reliability may increase or decrease.**
- **Hardware reliability study is related to stability.**
- **Software reliability aims at reliability growth.**

Reliability Metrics

- The software reliability metrics are expressed in a quantity form by using the reliability metrics.
 - There are different software metrics used in different phases of SDLC. Rate of occurrence of failure
 - (ROCOF): this metric is used to measure the occurrence of software failure.
-
- The behavior of the software is observed over a period and the total number of failures is measured.
-
- 1) Mean Time to Failure (MTTF): This metric is used to measure the average time between two successive failures. For this metric, only run time is used and not the booting time, system shutdown, etc.
 - 2) Mean Time to Repair (MTTR): This metric is the measurement of time to fix the error.

3) Duration for tracking an error and fixing is considered in this metric.

4) Mean Time Between Failure (MTBR): This metric combines MTTF and MTTR by considering the real-time and not the execution time.

$$\text{MTBF} = \text{MTTF} + \text{MTTR}$$

5) Probability of Failure on Demand (POFOD): This metric is used to measure the likelihood of the system failure when a request is made, and it does not include time measurement.

For example, a POFOD of 0.002 would mean that 1 out of every 2000 service requests would fail.

6) Availability: This metric is used to determine the failure and repair time of the system.

Availability indicates how likely the system will be available to clients during a certain time.

Software Quality

- Software quality is not determined like a traditional product such as the working of a fan or any machine.
- Software is not considered a quality product if it only satisfies the functionality.
- If the software is working as per requirement specification but it takes a long time and memory, then it is not quality software.
- The software which is defect-free, delivered within the allocated budget and time, meets the client's requirements, and can be easily maintained is known as quality software.

Factors for Software Quality

- **Portability:** When a software product can be made to work on different operating system environments, different systems, etc.,
- **Usability:** When both expert seniors and new persons can use the software without any difficulty.
- **Reusability:** Different components can be reused several times to develop new software.

- **Correctness:** The software is correctly implemented according to the requirement specifications.
- **Maintainability:** The software product can be easily corrected, and new functions can be added without much overhead.

Software Quality Management

- A quality management system or quality system is a fundamental methodology used by organizations to give assurance that the products have desirable quality.
- A quality management system must contain the following.
 - **Managerial Structure and Individual Responsibilities**
 - **Quality System Activities**

Managerial Structure and Individual Responsibilities:

- Quality is a responsibility of the whole organization with a quality department in every organization.
- Quality management is used to establish a framework for the organization's standards.

- **The standard should be applied to software-related documentation such as system requirements, design, and code.**
- **At the project level, every process is checked whether produced products are of the expected quality standard.**

Quality System Activities

- **The quality system activities are related to auditing the projects, reviewing the quality, developing guidelines, methods, etc. for organizations, generating reports for the top management, etc.**
- **The quality goals are defined in the quality plan to define what processes are to be used.**
- **The most used terms are quality assurance or quality control in the manufacturing industry.**

ISO 9000 Certification

- ISO (International Standards Organization) is an international standard development organization.**
- It is composed of representatives from the national standards organizations of member countries.**
- ISO published its 9000 series of standards in 1987.**
- It provides guidelines for maintaining quality ISO certification.**
- ISO serves as a reference for contracts between independent parties.**
- The operations and responsibilities, and report aspects are addressed by the ISO for producing high-quality development.**
- The ISO 9000 standard gives guidelines for producing the product and it is not concerned with the product itself.**

Types of ISO 9000

- 1. ISO 9001**
- 2. ISO 9002**
- 3. ISO 9003**

ISO 9001

- This standard applies to most software organizations.**
- The design, development, production, and servicing of the products use this standard.**

ISO 9002

ISO 9002

- This standard applies to those organizations which are not involved in the design of the product but are concerned with the production.**
- The industries such as car or steel manufacturing companies buy their design from external parties and only focus on manufacturing. Therefore, this standard does not apply to software products.**

ISO 9003

- **ISO 9003:** This standard is applied only to organizations that are involved only in the installation and testing of the products.

Need for ISO Certification

- With the use of ISO certification, the customer gets confidence in the product.
- ISO 9000 uses a well-structured and documented process in place, and this assures that the developed software is of high quality.
- The development process is much more efficient and cost-effective with the ISO 9000.
- If there are any weak points, then they are pointed out by the ISO 9000 and the remedies are given.

- Thus, the use of ISO 9000 sets the basic framework for an efficient process and results in total quality management (TQM).
- However, the ISO 9000 sets the steps for the software production process but does not guarantee the high quality of the process.
- It is not a full-proof agency, and it may downplay the domain experts.
- It requires a heavy emphasis on documentation and takes a lot of time and effort.

Software Engineering Institute Capability Maturity Model (SEI CMM)

- The quality of the software can be improved using the SEI CMM.
- With the help of the SEI CMM model, there will be business benefits.
- SEI CMM can be used for capability evaluation and software process assessment.

- With the help of capability evaluation, we can understand a way to assess the software process capability of an organization.
- To improve the process capability, the software process assessment is used by an organization.
- There are five maturity levels of software development industries using SEE CMM.
- Initial
- Repeatable
- Defined
- Managed
- Optimized

3

CMM Level	Focus	Key Process Areas
Initial	Competent people	
Repeatable	Project Management	Software project planning and configuration management
Defined	Defining of processes	Process definition, training programs, and peer reviews

CMM Level	Focus	Key Process Areas
Managed	Product and process quality	Software metrics and quality management
Optimizing	Improvement of continuous process	Defect prevention, process change management, technology change management.

ISO 9000 certification vs. SEI/CMM

- **ISO 9000 includes a set of international standards on quality management, and it is awarded by the international standards body.**
- **ISO 9000 helps companies to get efficient documentation for quality whereas the SEI CMM was developed specifically for the software industry and therefore addresses many issues which are specific to the software industry alone.**
- **The ISO 9000 is focused on the customer and supplier relationship, whereas SEI/CMM is only on the improvement of intermediate processes to achieve a high-quality product.**
- **ISO 9000 is accepted by most countries, but the SEI CMM is used in USA and less in other countries.**

- **SEI CMM model provides a list of KPAs in an organization for gradual quality improvement from one level to the next.**

- 1) Explain different versions of ISO 9000**
- 2) What is the need for ISO 9000? Explain in brief.**
- 3) Define software and hardware reliability**
- 4) What are the common approaches in debugging?**
- 5) Explain the CMM model**