MCA 401C: SOFTWARE TESTING

UNIT I

The role of process in software quality: Testing as a process - Overview of the Testing Maturity Model (TMM) - Basic definitions - Software testing principles - Origins of defects - Defect classes, Defect repository - Test design - Defect example: the coin problem

UNIT II

Test case design strategies : Black box approach - Random testing - Equivalence class partitioning - Boundary value analysis - Cause and Effect graphing - State transition testing - Error guessing - White box approach - Test adequacy criteria - Coverage and control flow graphs - Covering code logic - Data flow and white box test design - Loop testing - Mutation testing - Evaluating test adequacy criteria.

UNIT III

Levels of testing : Unit test: functions, procedures, classes and methods as units - Unit test planning - Designing test units - The class as a testable unit - The test harness - Integration test: goal - Integration strategies for procedures and functions - Integration strategies for classes - Designing integration test - System test - The different types - Regression testing - Alpha, beta and acceptance test - Test planning - Test plan components - Test plan attachments - Reporting test results.

UNIT IV

Software quality: Defining Quality: importance of quality - Quality control v/s quality assurance - Quality assurance at each phase of SDLC - Quality assurance in software support projects - SQA function - Quality management system in an organization - Software quality assurance plans - Product quality.

UNIT V

Software metrics and models: Walkthroughs and Inspections - Software Configuration Management - ISO:9001 Model - CMM Model - CMM and ISO comparative analysis - CMM-I .

Text Books

1. IIene Burnstein, "Practical Software Testing", Springer International Edition, First Indian reprint,
   2004.
2. Nina S Godbole, "Software Quality Assurance, Principles and Practice", Narosa Publishing House, 2004.

Reference Books

1. P.C. Jorgensen, "Software Testing - A Craftman's Approach", CRC press, 1995.
2. Boris Beizer, van Nostrand Reinhold, "Software Testing Techniques", 2nd Edition, 1990.

LECTURE NOTES

UNIT-1

Role of process in software quality

The need for software products of high quality has pressured those in the profession to identify and quantify quality factors such as usability, testability, maintainability, and reliability, and to identify engineering practices that support the production of quality products having these favorable attributes. Among the practices identified that contribute to the development of high-quality software are project planning, requirements management, development of formal specifications, structured design with use of information hiding and encapsulation, design and code reuse, inspections and reviews, product and process measures, education and training of software professionals, development and application of CASE tools, use of effective testing techniques, and integration of testing activities into the entire life cycle. In addition to identifying these individual best technical and managerial practices, software researchers realized that it was important to integrate them within the context of a high-quality software development process.

Process, in the software engineering domain, is the set of methods, practices, standards, documents, activities, policies, and procedures that software engineers use to develop and maintain a software system and its associated artifacts, such as project and test plans, design documents, code, and m It also was clear that adding individual practices to an existing software development process in an ad hoc way was not satisfactory. The software development process, like most engineering artifacts, must be engineered. That is, it must be designed, implemented, evaluated, and maintained. As in other engineering disciplines, a software development process must evolve in a consistent and predictable manner, and the best technical and managerial practices must be integrated in a systematic way. These models allow an organization to evaluate its current software process and to capture an understanding of its state. Strong support for incremental process improvement is provided by the models, consistent with historical process evolution and the application of quality principles. The models have received much attention from industry, and resources have been invested in process improvement efforts with many successes recorded.

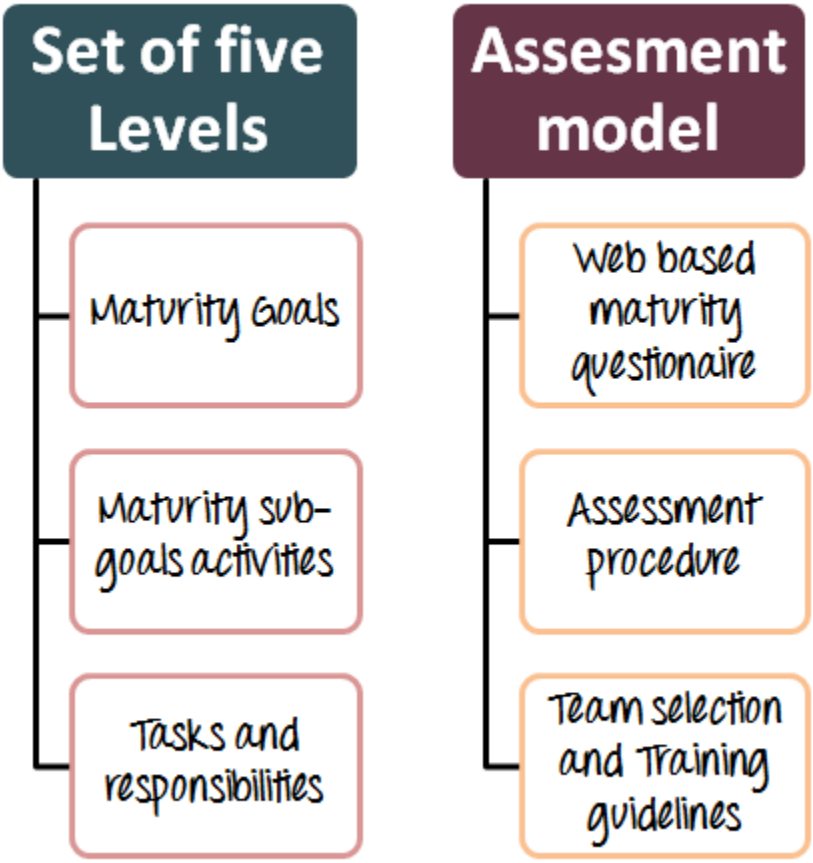
All the software process improvement models that have had wide acceptance in industry are high-level models, in the sense that they focus on the software process as a whole and do not offer adequate support to evaluate and improve specific software development sub processes such as design and testing. Most software engineers would agree that testing is a vital component of a quality software process, and is one of the most challenging and costly activities carried out during software development and maintenance.

Testing Maturity Model in Software Testing

Testing Maturity Model (TMM) in Software Testing is a framework for evaluating the maturity of software testing processes. The purpose of using testing maturity model is identifying maturity and providing targets to improve the software testing process to achieve progress. It can be complemented with any process improvement model or can be used as a stand alone model.
The Test Maturity Model (TMM) is based on the Capability Maturity Model (CMM) and it was first developed by the Illinois Institute of Technology. It is a detailed model for test process improvement.

TMM model has major two components

1. A set of 5 levels that define testing capability
2. An Assessment Model



Different Levels of Maturity Model

The five levels of the TMM helps the organization to determine the maturity of its process and to identify the next improvement steps that are essential to achieving a higher level of test maturity.

TMM Levels	Goals	An objective of TMM levels
Level 1: Initial	Software should run successfully	• At this level, no process areas are identifie • An objective of testing is to ensure that so • This level lacks resources, tools, and traine • No Quality Assurance checks before softw
Level 2: Defined	Develop testing and debugging goals and policies	• This level distinguish testing from debuggi considered distinct activities • Testing phase comes after coding • A primary goal of testing is to show softwa • Basic testing methods and techniques are
Level 3: Integrated	Integration of testing into the	• Testing gets integrated into an entire life • Based on requirements test objectives are

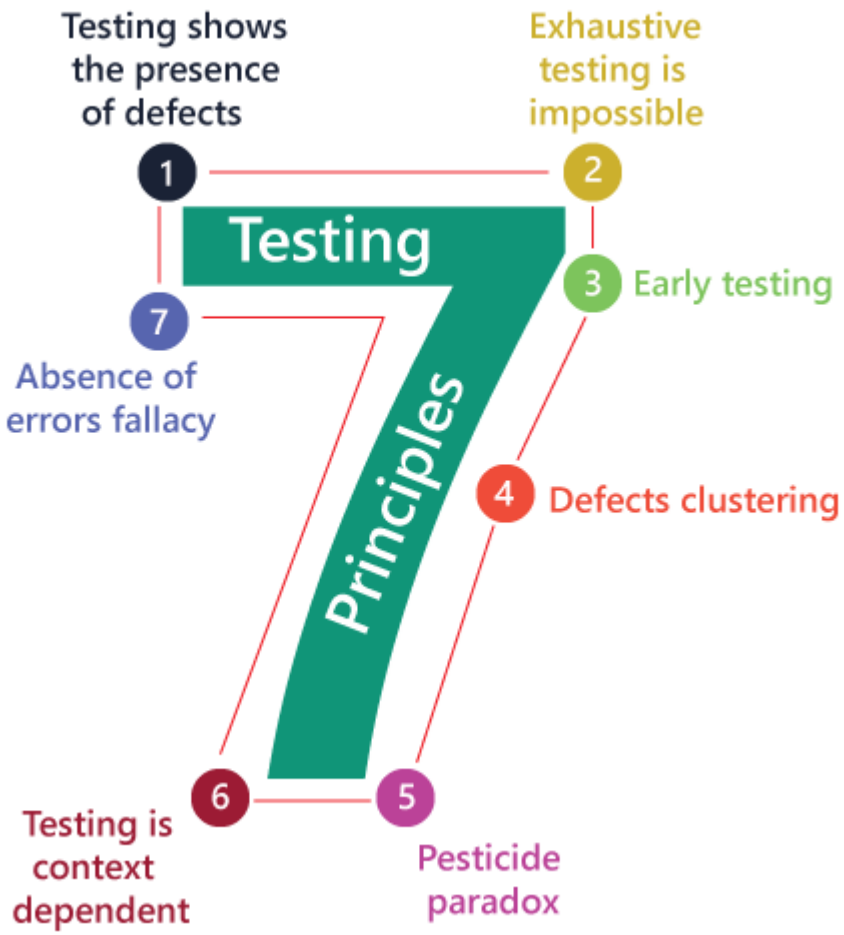
TMM Levels	Goals	An objective of TMM levels
	software lifecycle	• Test organization exists • Testing recognized as a professional activi
Level 4: Management and Measurement	Establish a test measurement program	• Testing is a measured and quantified proc • Review at all development phases are rec • For reuse and Regression Testing, test cas recorded in a test database • Defects are logged and given severity leve
Level 5: Optimized	Test process optimization	• Testing is managed and defined • Testing effectiveness and costs can be mo • Testing can be fine-tuned and continuous • Quality control and Defect prevention are • Process reuse is practiced • Test related metrics also have tool suppor • Tools provide support for Test Case desig

Software Testing Principles

Software testing is a procedure of implementing software or the application to identify the defects or bugs. For testing an application or software, we need to follow some principles to make our product defects free, and that also helps the test engineers to test the software with their effort and time. Here, in this section, we are going to learn about the seven essential principles of software testing.

Let us see the seven different testing principles, one by one:

- Testing shows the presence of defects
- Exhaustive Testing is not possible
- Early Testing
- Defect Clustering
- Pesticide Paradox
- Testing is context-dependent
- Absence of errors fallacy

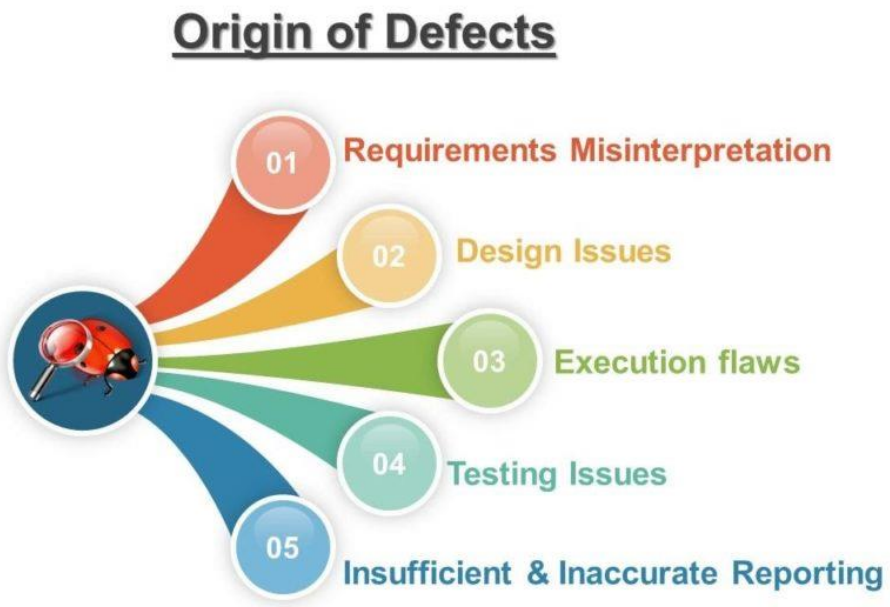


ORIGIN OF DEFECTS

Debugging is a painful exercise, therefore identifying the origin of the software defects is extremely important in order to take timely corrective measures. The rate at which the new features and modifications are introduced, it becomes even more important to ensure that the builds are defect-free before any new changes are incorporated.

Traditionally, the focus has been more on code defects, but the fact is that the defects can originate as early as requirements gathering/analysis, and can be introduced at the time of software design too. If the base is defective, how can an end product be of high quality? The worst is to realize this much later and then start firefighting, which obviously is an expensive exercise.

In this article, we will explore the possible origins of defects and how we can help you in handling them.



Requirements misinterpretation

Understanding customer needs and translating them to technical requirements lays the foundation of an application. If done incorrectly, the resulting design will be faulty. The code may be defect-free, but it was built on the wrong premise. Hence, it is a good idea to introduce thorough requirement-related reviews to nip any potential defect at the onset only.

Requirement understanding and misinterpretation can be due to either **poor communication** between the teams and clients or due to **immature requirement gathering processes**.

Defect Examples: The Coin Problem

The following examples illustrate some instances of the defect classes that were discussed in the previous sections. A simple specification, a detailed design description, and the resulting code are shown, and defects in each are described. Note that these defects could be injected via one or more of the five defect sources discussed at the beginning of this chapter. Also note that there may be more than one category that fits a given defect. Figure 3.3 shown a sample informal specification for a simple program that calculates the total monetary value of a set of coins. The program could be a component of an interactive cash register system to support retail store clerks. This simple example shows requirements/ specification defects, functional description defects, and interface description defects.

The functional description defects arise because the functional description is ambiguous and incomplete. It does not state that the input, number_of_coins, and the output, number_of_dollars and number _of_cents, should all have values of zero or greater. The number_of_coins cannot be negative, and the values in dollars and cents cannot be negative in the real-world domain. As a consequence of these ambiguities and specification incompleteness, a checking routine may be omitted from the design, allowing the final program to accept negative values for the input

number_of_coins for each of the denominations, and consequently it may calculate an invalid value for the results. A more formally stated set of preconditions and postconditions would be helpful here, and would address some of the problems with the specification. These are also useful for designing black box tests.

A precondition is a condition that must be true in order for a software component to operate properly.

In this case a useful precondition would be one that states for example:number_of_coins __0

A postcondition is a condition that must be true when a software component completes its operation properly.

A useful postcondition would be: number_of_dollars, number_of_cents _0.

In addition, the functional description is unclear about the largest number of coins of each denomination allowed, and the largest number of dollars and cents allowed as output values. Interface description defects relate to the ambiguous and incomplete description of user-software interaction. It is not clear from the specification how the user interacts with the program to provide input, and how the output is to be reported. Because of ambiguities in the user interaction description the software may be difficult to use. Likely origins for these types of specification defects lie in the nature of the development process, and lack of proper education and training. A poor-quality development process may not be allocating the proper time and resources to specification development and review. In addition, software engineers may not have the proper education and training to develop a quality specification. All of these specification defects, if not detected and repaired, will propagate to the design and coding phases. Black box testing techniques, which we will study in Chapter 4, will help to reveal many of these functional weaknesses. Figure 3.4 shows the specification transformed in to a design description. There are numerous design defects, some due to the ambiguous and incomplete nature of the specification; others are newly introduced. Design defects include the following:

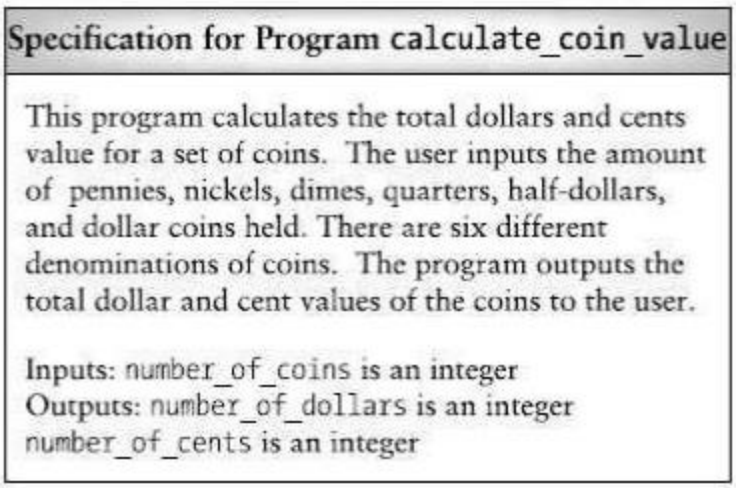


FIG. 3.3

A sample specification with defects.

Control, logic, and sequencing defects. The defect in this subclass arises from an incorrect while loop condition (should be less than or equal to six)

Algorithmic, and processing defects. These arise from the lack of error checks for incorrect and/or invalid inputs, lack of a path where users can correct erroneous inputs, lack of a path for recovery from input errors. The lack of an error check could also be counted as a functional design

defect since the design does not adequately describe the proper functionality for the program. **Data defects.** This defect relates to an incorrect value for one of the elements of the integer array, coin_values, which should read 1,5,10,25,50,100.

External interface description defects. These are defects arising from the absence of input messages or prompts that introduce the program to the user and request inputs. The user has no way of knowing in which order the number of coins for each denomination must be input, and when to stop inputting values. There is an absence of help messages, and feedback for user if he wishes to change an input or learn the correct format and order for inputting the number of coins. The output description and output formatting is incomplete. There is no description of what the outputs means in terms of the problem domain. The user will note that two values are output, but has no clue as to their meaning. The control and logic design defects are best addressed by white box- based tests, (condition/branch testing, loop testing). These other design defects will need a combination of white and black box testing techniques for detection.

Figure shows the code for the coin problem in a —C-like programming language. Without effective reviews the specification and

```
Design Description for Program calculate_coin_values

Program calculate_coin_values
number_of_coins is  integer
total_coin_value is integer
number_of_dollars is integer
number_of_cents is integer
coin_values is array of  six integers representing
each coin value in cents
initialized to: 1,5,10,25,25,100
begin

initialize total_coin_value to zero
initialize loop_counter to one
while loop_counter is less then six
begin
       output "enter number of coins"
       read (number_of_coins )
       total_coin_value = total_coin_value +
       number_of_coins * coin_value[loop_counter]
       increment loop_counter
end
number_dollars =  total_coin_value/100
number_of_cents = total_coin_value - 100 * number_of_dollars
output (number_of_dollars, number_of_cents)
end
```

FIG. 3.4
A sample design specification with defects.

design defects could propagate to the code. Here additional defects have been introduced in the coding phase.

Control, logic, and sequence defects. These include the loop variable increment step which is outof the scope of the loop. Note that incorrect loop condition (*i* _ 6) is carried over from design and should be counted as a design defect.

Algorithmic and processin g defects. The division operator may cause problems if negative values are divided, although this problem could be eliminated with an input check.

Data Flow defects. The variable total_coin_value is not initialized. It is used before it is defined. (This might also be considered a data defect.)

Data Defects. The error in initializing the array coin_values is carried over from design and should be counted as a design defect.

External Hardware, Software Interface Defects. The call to the external func tion ―scanf‖ is incorrect. The address of the variable must be provided (&number_of_coins).

Code Documentation Defects. The documentation that accompanies this code is incomplete and ambiguous. It reflects the deficiencies in the external interface description and other defects that occurred during speci fication and design. Vital information is missing for anyone who will need to repair, maintain or reuse this code.

The poor quality of this small program is due to defects injected during several of the life cycle phases with probable causes ranging from lack of education, a poor process, to oversight on the part of the designers and developers. Even though it implements a simple function the program is unusable because of the nature of the defects it contains. Such software is not acceptable to users; as testers we must make use of all our static and dynamic testing tools as described in subsequent chapters to ensure that such poor-quality software is not delivered to our user/client group. We must work with analysts, designers and code developers to ensure that quality issues are addressed early the software life cycle. We must also catalog defects and try to eliminate them by improving education, training, communication, and process.

SHORT QUESTIONS

1.Define software quality?

2.Explain testing maturity model(TMM)?

3.Define any two software testing principles?

4.Write about origin of defects?

5.Explain test design in software testing?

LONG QUESTIONS

1.Explain in detail about the role of process in software quality?

2.List out all the software testing principles ?

3.Descibe the overview of testing maturity model(TMM)?

4.Explain the terms of defect classes and defect repository?

5.Define the defect example-coin problem?

UNIT-2

Test Case Design Strategies

A smart tester who wants to maximize use of time and resources knows that she needs to develop what we will call effective test cases for execution-based testing. By an effective test case we mean one that has a good possibility of revealing a defect (see Principle 2 in Chapter 2). The ability to develop effective test cases is important to an organization evolving toward a higher-quality testing process. It has many positive consequences. For example, if test cases are effective there is (i) a greater probability of detecting defects, (ii) a more efficient use of organizational resources, (iii) a higher probability for test reuse, (iv) closer adherence to testing and project schedules and budgets, and, (v) the possibility for delivery of a higher -quality software product. What are the approaches a tester should use to design effective test cases? To answer the question we must adopt the view that software is an engineered product. Given this view there are two basic strategies that can be used to design test cases. These are called the black box (sometimes called functional or specification) and white box (sometimes called clear or glassbox) test strategies. The approaches are summarized in Figure

Using the black box approach, a tester considers the software-under test to be an opaque box. There is no knowledge of its inner structure (i.e., how it works). The tester only has knowledge of what it does. The size of the software-under-test using this approach can vary from a simple module, member function, or object cluster to a subsystem or a complete Software system. The description of behavior or functionality for the software-under-test may come from a formal specification, an Input/Process/Output Diagram (IPO), or a well-defined set of

pre and post conditions. Another source for information is a requirements specification document that usually describes the functionality of the software-under-test and its inputs and expected outputs. The tester provides the specified inputs to the software-under-test, runs the test and then determines if the outputs produced are equivalent to those in the specification. Because the black box approach only considers software behavior and functionality, it is often called functional or specification-based testing. This approach is especially useful for revealing requirements and specification defects.

The white box approach focuses on the inner structure of the software to be tested. To design test cases using this strategy the tester must have knowledge of that structure. The code, or a suitable pseudo codelike representation must be available. The tester selects test cases to exercise specific internal structural elements to determine if they are working properly. For example, test cases are often designed to exercise all statements or true/false branches that occur in a module or member function. Since designing, executing, and analyzing the results of white

box testing is very time consuming, this strategy is usually applied to smaller-sized pieces of software such as a module or member function. The reasons for the size restriction will become more apparent in Chapter 5 where the white box strategy is described in more detail. White box testing methods are especially useful for revealing design and code-based control, logic and sequence defects, initialization defects, and data flow defects.

The smart tester knows that to achieve the goal of providing users with low-defect, high-quality software, both of these strategies should be used to design test cases. Both support the tester with the task of selecting the finite number of test cases that will be applied during test. Neither approach by itself is guaranteed to reveal all defects types we have studied in Chapter 3. The approaches complement each other; each may be useful for revealing certain types of defects. With a suite of test cases designed using both strategies the tester increases the chances of revealing the many different type of defects in the software under test. The tester will also have an effective set of reusable test cases for regression testing (re-test after changes), and for testing new releases of the software.

There is a great deal of material to introduce to the reader relating to both of these strategies. To facilitate the learning process, the material has been

partitioned into two chapters. This chapter focuses on black box methods, and Chapter 5 will describe white box methods and how to apply them to design test cases.

Black box testing

Black box testing is a technique of software testing which examines the functionality of software without peering into its internal structure or coding. The primary source of black box testing is a specification of requirements that is stated by the customer.

In this method, tester selects a function and gives input value to examine its functionality, and checks whether the function is giving expected output or not. If the function produces correct output, then it is passed in testing, otherwise failed. The test team reports the result to the development team and then tests the next function. After completing testing of all functions if there are severe problems, then it is given back to the development team for correction.

Generic steps of black box testing

- The black box test is based on the specification of requirements, so it is examined in the beginning.
- In the second step, the tester creates a positive test scenario and an adverse test scenario by selecting valid and invalid input values to check that the software is processing them correctly or incorrectly.
- In the third step, the tester develops various test cases such as decision table, all pairs test, equivalent division, error estimation, cause-effect graph, etc.
- The fourth phase includes the execution of all test cases.
- In the fifth step, the tester compares the expected output against the actual output.
- In the sixth and final step, if there is any flaw in the software, then it is cured and tested again.

Test procedure

The test procedure of black box testing is a kind of process in which the tester has specific knowledge about the software's work, and it develops test cases to check the accuracy of the software's functionality.

It does not require programming knowledge of the software. All test cases are designed by considering the input and output of a particular function.A tester knows about the definite output of a particular input, but not about how the result is arising. There are various techniques used in black box testing for testing like decision table technique, boundary value analysis technique, state transition, All-pair testing, cause-effect graph technique, equivalence partitioning technique, error guessing technique, use case technique and user story technique. All these techniques have been explained in detail within the tutorial.

Random Testing

Random Testing, also known as monkey testing, is a form of functional black box testing that is performed when there is not enough time to write and execute the tests.

Random Testing Characteristics:

- Random testing is performed where the defects are NOT identified in regular intervals.
- Random input is used to test the system's reliability and performance.
- Saves time and effort than actual test efforts.
- Other Testing methods Cannot be used to.

Random Testing Steps:

- Random Inputs are identified to be evaluated against the system.
- Test Inputs are selected independently from test domain.
- Tests are Executed using those random inputs.
- Record the results and compare against the expected outcomes.
- Reproduce/Replicate the issue and raise defects, fix and retest.

Boundary Value Analysis

Boundary value analysis is one of the widely used case design technique for black box testin boundary values because the input values near the boundary have higher chances of error.

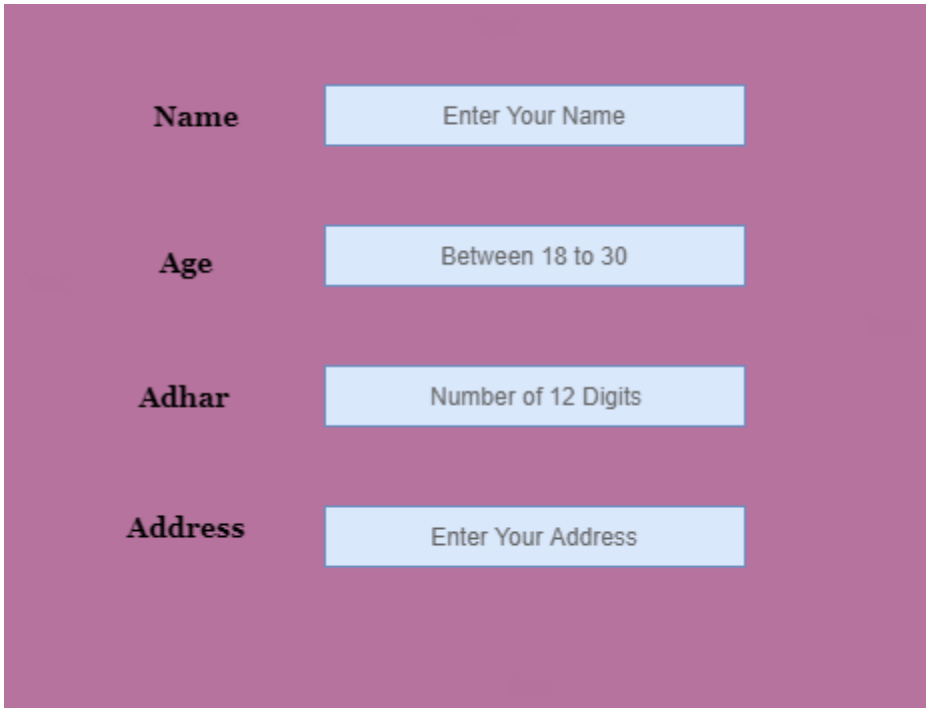
Whenever we do the testing by boundary value analysis, the tester focuses on, while entering bou the software is producing correct output or not.

Boundary values are those that contain the upper and lower limit of a variable. Assume that, age function, and its minimum value is 18 and the maximum value is 30, both 18 and 30 will be con

values.

The basic assumption of boundary value analysis is, the test cases that are created using bound
likely to cause an error.

There is 18 and 30 are the boundary values that's why tester pays more attention to these values, b
that the middle values like 19, 20, 21, 27, 29 are ignored. Test cases are developed for each an
range.



Testing of boundary values is done by making valid and invalid partitions. Invalid partitions are te
of output in adverse condition is also essential.

Let's understand via practical:

Imagine, there is a function that accepts a number between 18 to 30, where 18 is the minimum an
value of valid partition, the other values of this partition are 19, 20, 21, 22, 23, 24, 25, 26, 27, 28
partition consists of the numbers which are less than 18 such as 12, 14, 15, 16 and 17, and more th
34, 36 and 40. Tester develops test cases for both valid and invalid partitions to capture the behav
different input conditions.

Invalid test cases	Valid test cases	Invalid test cases
11, 13, 14, 15, 16, 17	18, 19, 24, 27, 28, 30	31, 32, 36, 37, 38, 39

The software system will be passed in the test if it accepts a valid number and gives the desired ou it is unsuccessful. In another scenario, the software system should not accept invalid numbers number is invalid, then it should display error massage.

If the software which is under test, follows all the testing guidelines and specifications then it is team otherwise to the development team to fix the defects.

White Box Testing

White Box Testing is a testing technique in which software's internal structure, design, and coding are tested to verify input-output flow and improve design, usability, and security. In white box testing, code is visible to testers, so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing, and Glass box testing.
It is one of two parts of the Box Testing approach to software testing. Its counterpart, Blackbox testing, involves testing from an external or end-user perspective. On the other hand, White box testing in software engineering is based on the inner workings of an application and revolves around internal testing.

The term "WhiteBox" was used because of the see-through box concept. The clear box or WhiteBox name symbolizes the ability to see through the software's outer shell (or "box") into its inner workings. Likewise, the "black box" in "Black Box Testing" symbolizes not being able to see the inner workings of the software so that only the end-user experience can be tested.

In this white box testing tutorial, you will learn:

-
-
-
-
-
-
-
-

-
-

We have divided it into two basic steps to give you a simplified explanation of white box testing. This is what testers do when testing an application using the white box testing technique:

STEP 1) UNDERSTAND THE SOURCE CODE

The first thing a tester will often do is learn and understand the source code of the application. Since white box testing involves the testing of the inner workings of an application, the tester must be very knowledgeable in the programming languages used in the applications they are testing. Also, the testing person must be highly aware of secure coding practices. Security is often one of the primary objectives of testing software. The tester should be able to find security issues and prevent attacks from hackers and naive users who might inject malicious code into the application either knowingly or unknowingly.

STEP 2) CREATE TEST CASES AND EXECUTE

The second basic step to white box testing involves testing the application's source code for proper flow and structure. One way is by writing more code to test the application's source code. The tester will develop little tests for each process or series of processes in the application. This method requires that the tester must have intimate knowledge of the code and is often done by the developer. Other methods include Manual Testing, trial, and error testing and the use of testing tools as we will explain further on in this article.

WhiteBox Testing Example

Consider the following piece of code

```
Printme (int a, int b) {                          ------------  Printme is a
function
    int result = a+ b;
    If (result> 0)
        Print ("Positive", result)
    Else
        Print ("Negative", result)
    }                                            -----------   End of the
source code
```

The goal of WhiteBox testing in software engineering is to verify all the decision branches, loops, and statements in the code.

To exercise the statements in the above white box testing example, WhiteBox test cases would be

- A = 1, B = 1
- A = -1, B = -3

White Box Testing Techniques

A major White box testing technique is Code Coverage analysis. Code Coverage analysis eliminates gaps in a Test Case suite. It identifies areas of a program that are not exercised by a set of test cases. Once gaps are identified, you create test cases to verify untested parts of the code, thereby increasing the quality of the software product

There are automated tools available to perform Code coverage analysis. Below are a few coverage analysis techniques a box tester can use:

Statement Coverage:- This technique requires every possible statement in the code to be tested at least once during the testing process of software engineering.

Branch Coverage – This technique checks every possible path (if-else and other conditional loops) of a software application.

Apart from above, there are numerous coverage types such as Condition Coverage, Multiple Condition Coverage, Path Coverage, Function Coverage etc. Each technique has its own merits and attempts to test (cover) all parts of software code. **Using Statement and Branch coverage you generally attain 80-90% code coverage which is sufficient.**

Following are important WhiteBox Testing Techniques:

- Statement Coverage
- Decision Coverage
- Branch Coverage
- Condition Coverage

- Multiple Condition Coverage
- Finite State Machine Coverage
- Path Coverage
- Control flow testing
- Data flow testing

Types of White Box Testing

White box testing encompasses several testing types used to evaluate the usability of an application, block of code or specific software package. There are listed below —

- **Unit Testing:** It is often the first type of testing done on an application. Unit Testing is performed on each unit or block of code as it is developed. Unit Testing is essentially done by the programmer. As a software developer, you develop a few lines of code, a single function or an object and test it to make sure it works before continuing Unit Testing helps identify a majority of bugs, early in the software development lifecycle. Bugs identified in this stage are cheaper and easy to fix.
- **Testing for Memory Leaks**: Memory leaks are leading causes of slower running applications. A QA specialist who is experienced at detecting memory leaks is essential in cases where you have a slow running software application.

Apart from the above, a few testing types are part of both black box and white box testing. They are listed below

- **White Box Penetration Testing:** In this testing, the tester/developer has full information of the application's source code, detailed network information, IP addresses involved and all server information the application runs on. The aim is to attack the code from several angles to expose security threats.
- **White Box Mutation Testing**: Mutation testing is often used to discover the best coding techniques to use for expanding a software solution.

White Box Testing Tools

Below is a list of top white box testing tools.

- EclEmma
- NUnit
- PyUnit
- HTMLUnit
- CppUnit

Advantages of White Box Testing

- Code optimization by finding hidden errors.
- White box tests cases can be easily automated.
- Testing is more thorough as all code paths are usually covered.
- Testing can start early in SDLC even if GUI is not available.

Disadvantages of WhiteBox Testing

- White box testing can be quite complex and expensive.
- Developers who usually execute white box test cases detest it. The white box testing by developers is not detailed and can lead to production errors.
- White box testing requires professional resources with a detailed understanding of programming and implementation.
- White-box testing is time-consuming, bigger programming applications take the time to test fully.

Conclusion:

- White box testing can be quite complex. The complexity involved has a lot to do with the application being tested. A small application that performs a single simple operation could be white box tested in few minutes, while larger programming applications take days, weeks, and even longer to fully test.
- White box testing in software testing should be done on a software application as it is being developed after it is written and again after each modification.

Test Coverage

Test coverage is defined as a metric in Software Testing that measures the amount of testing performed by a set of test. It will include gathering information about which parts of a program are executed when running the test suite to determine which branches of conditional statements have been taken.

In simple terms, it is a technique to ensure that your tests are testing your code or how much of your code you exercised by running the test.

Test Coverage does

- Finding the area of a requirement not implemented by a set of test cases
- Helps to create additional test cases to increase coverage
- Identifying a quantitative measure of test coverage, which is an indirect method for quality check
- Identifying meaningless test cases that do not increase coverage

Test Coverage can be accomplished

- Test coverage can be done by exercising the static review techniques like peer reviews, inspections, and walkthrough
- By transforming the ad-hoc defects into executable test cases
- At code level or unit test level, test coverage can be achieved by availing the automated code coverage or unit test coverage tools
- Functional test coverage can be done with the help of proper test management tools

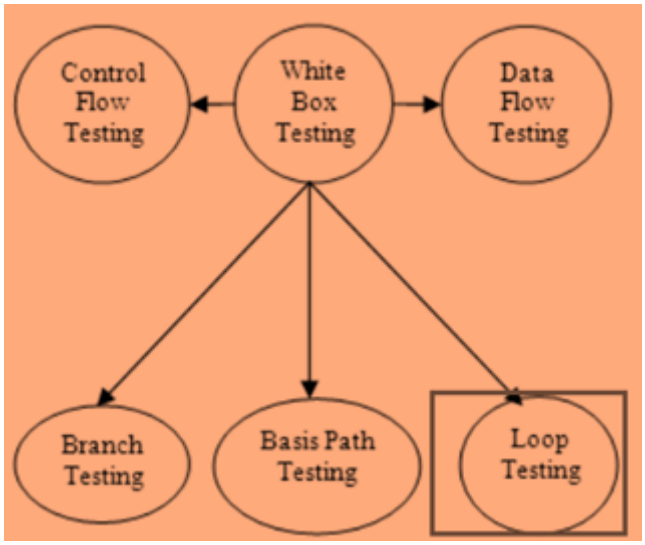
Benefits of Test Coverage in Software Engineering

- It can assure the quality of the test
- It can help identify what portions of the code were actually touched for the release or fix
- It can help to determine the paths in your application that were not tested
- Prevent Defect leakage
- Time, scope and cost can be kept under control
- Defect prevention at an early stage of the project lifecycle
- It can determine all the decision points and paths used in the application, which allows you to increase test coverage
- Gaps in requirements, test cases and defects at the unit level and code level can be found in an easy way

Loop Testing

Loop Testing is defined as a software testing type, that completely focuses on the validity of the loop constructs. It is one of the parts of Control Structure Testing (path testing, data validation testing, condition testing).

Loop testing is a White box testing. This technique is used to test loops in the program.



Types of loop Tested

Examples of types of loop tested are,

- Simple loop
- Nested loop
- Concatenated loop
- Unstructured loop

Loop Testing is done for the following reasons

- Testing can fix the loop repetition issues
- Loops testing can reveal performance/capacity bottlenecks
- By testing loops, the uninitialized variables in the loop can be determined
- It helps to identify loops initialization problems.

SHORT QUESTIONS

1.Define about test case design stratigies?

2.write a short note on black box?

3.Explain random testing?

4.Define white box?

5.write about loop testing?

LONG QUESTIONS

1.Discuss in detail about black box approach?

2.Explain about boundary value analysis?

3.Discuss about state transition testing and error guessing?

4.explain about data flow and white box testing?

5.discuss in detail about white box test design?

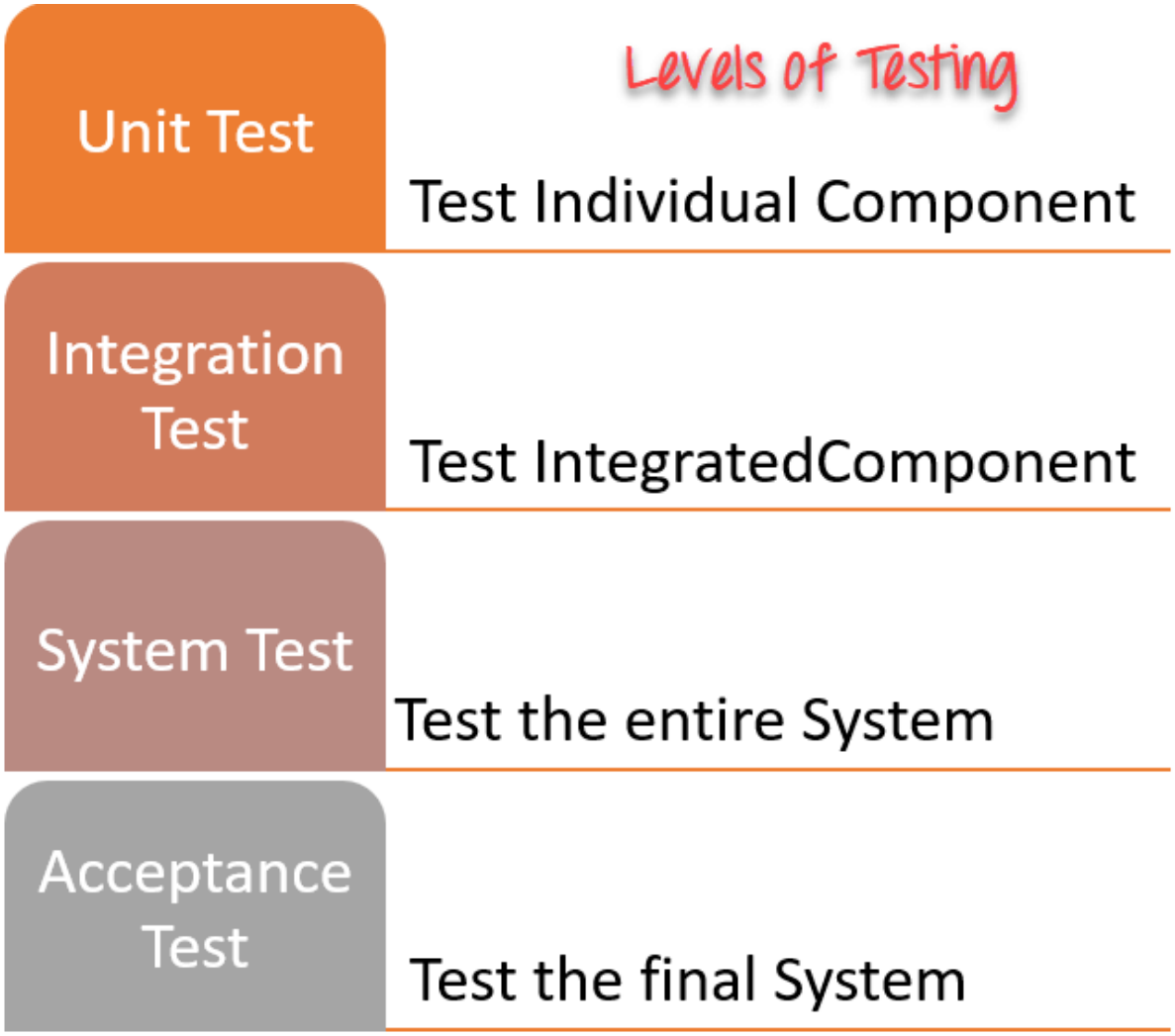
UNIT-3

Levels of Testing in Software Testing

4 Levels of Testing

There are mainly four **Levels of Testing** in software testing :

1. **Unit Testing** : checks if software components are fulfilling functionalities or not.
2. **Integration Testing** : checks the data flow from one module to other modules.
3. **System Testing** : evaluates both functional and non-functional needs for the testing.
4. **Acceptance Testing** : checks the requirements of a specification or contract are met as per its delivery.



Each of these testing levels has a specific purpose. These testing level provide value to the software development lifecycle.

In this Testing Level tutorial, you will learn:

-
-
-

Each Testing Level Details

Unit testing:

A Unit is a smallest testable portion of system or application which can be compiled, liked, loaded, and executed. This kind of testing helps to test each module separately.

The aim is to test each part of the software by separating it. It checks that component are fulfilling functionalities or not. This kind of testing is performed by developers.

Integration testing:

Integration means combining. For Example, In this testing phase, different software modules are combined and tested as a group to make sure that integrated system is ready for system testing.

System Testing

System testing is performed on a complete, integrated system. It allows checking system's compliance as per the requirements. It tests the overall interaction of components. It involves load, performance, reliability and security testing.

System testing most often the final test to verify that the system meets the specification. It evaluates both functional and non-functional need for the testing.

Acceptance testing:

Acceptance testing is a test conducted to find if the requirements of a specification or contract are met as per its delivery. Acceptance testing is basically done by the user or customer. However, other stockholders can be involved in this process.

Conclusion:

- A level of software testing is a process where every unit or component of a software/system is tested.
- The primary goal of system testing is to evaluate the system's compliance with the specified needs.
- In Software Engineering, four main levels of testing are Unit Testing, Integration Testing, System Testing and Acceptance Testing.

DESIGNING TEST UNITS

Let's open the post with some fundamentals on test design.

Test Design

Test design means determining how tests will work. When you do test design, you're essentially defining the details of your test cases. What will their steps be? How will they be structured?

Among other things, designing a test might involve:

- Thinking about the data the test will use, including its realism, amount, and boundaries
- Expressing detailed test scenarios in a visual—e.g. a diagram—or a written way—e.g. a table.
- Trying to predict possible errors and edge cases based on one's previous experience and knowledge of the application.

Purpose of Test Design

Software testing—both automated and manual—isn't free. You should treat it as an investment, which includes tracking its ROI. Thus, you want to make sure your software testing strategy is as efficient as possible.

That's where test design comes in handy. With a proper test design process in place, your team will create tests that generate value for the organization, ensuring you can ship high-quality code as fast as the market demands it.

How Is Test Design Done? Who Does It? When?

You already know what test design at a high level is and why it matters. Let's talk about how to get it done.

Who actually does test design? It depends on the type of organization. In more traditional companies with a more stark divide between roles, testers and QA staff would be responsible for test design—and probably for all other activities regarding tests.

However, "the times, they are a-changing." Due to the rise of automation, agile, and DevOps, the lines between roles inside tech organizations are becoming thinner and thinner. Today, we dare to say that everyone performs testing. This, by extension, means everyone does test design.

Here's what this might look like in practice:

- Engineers write unit tests, which involve designing the necessary assertions, doing setups and teardowns for classes, and coming up with test data.
- Testers with no coding skills can use record-and-playback tools to record UI or end-to-end tests, which they or other QA staff designed.
- If the organization adopts BDD (Behavior-driven development), they might have the customer—or someone acting as their proxy—collaborating in the creation of automated specifications.

The list above isn't exhaustive. But you can see how test design can be the work of many different actors.

When does test design take place? Like the previous question, the answer to this one depends on how things are done in your organization.

In organizations that still follow a more traditional, phase-based development methodology, you'll probably also find a more formal software testing life cycle in which test design is a phase with a well-defined place.

However, nowadays most organizations are following an agile-based approach to software development. Such approaches favor development in short, iterative cycles. In such scenarios, testing is no longer a phase but an activity you carry out as early in the process as possible. If that's the case, test design is done several times in the context of a single iteration.

Test Design Techniques

Having covered the "what," "why," "who," and "when" of test design, the only major question left for us to tackle is the "how." Now we'll cover three techniques for test design.

Equivalence Class Testing

One of the hardest things when designing test cases is knowing when it's enough. How many examples do you need? How comprehensive do your test data need to be?

Equivalence class testing—aka equivalence class partitioning—is a solution to this dilemma. This technique tells us to partition our test data into big "buckets" we can consider equal. Then, only a few values inside each group—and at their boundaries—are enough to ensure our test is comprehensive.

For instance, let's say you need to test a feature that asks for the user age and does something different according to these age groups:

- 0 to 12 years
- 13 to 18 years
- 19 to 40
- 41 to 65
- more than 65

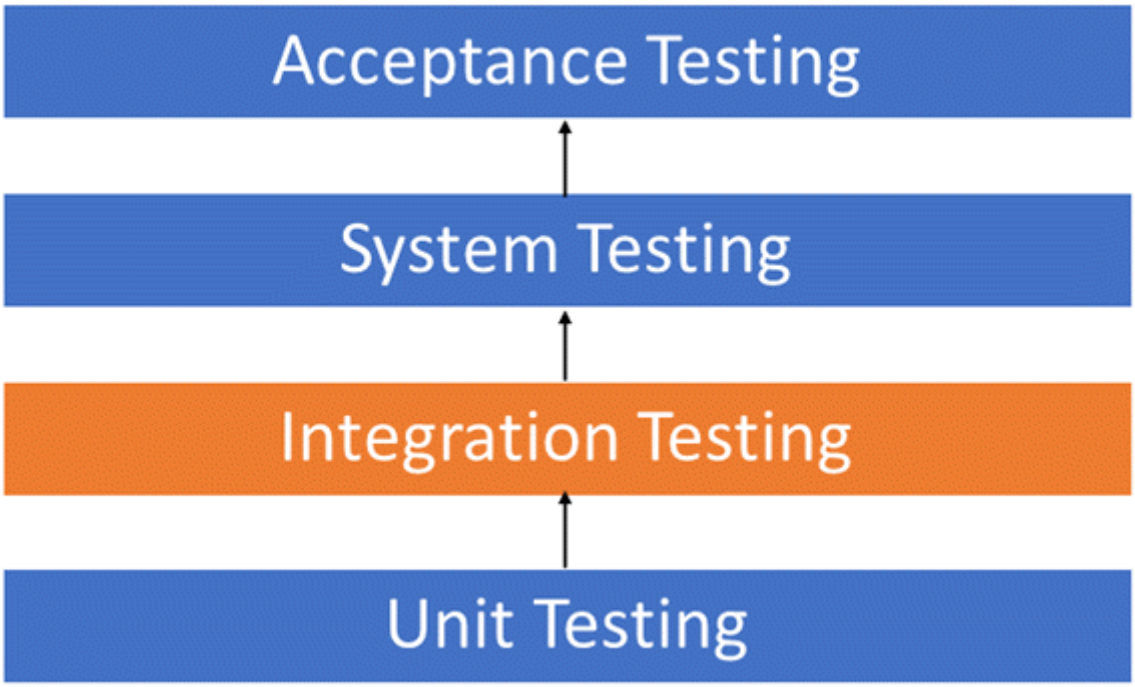
For each of these "buckets," you need to have a test case:

- at each boundary
- immediately below and above each boundary
- for one value inside the group

So, for the "19 to 40" group, the values 18, 19, 20, 39, 40, 41, and 23 should be enough as test cases.

Integration Testing

Integration Testing is defined as a type of testing where software modules are integrated logically and tested as a group. A typical software project consists of multiple software modules, coded by different programmers. The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated
Integration Testing focuses on checking data communication amongst these modules. Hence it is also termed as **'I & T'** (Integration and Testing), **'String Testing'** and sometimes **'Thread Testing'**.



Although each software module is unit tested, defects still exist for various reasons like

- A Module, in general, is designed by an individual software developer whose understanding and programming logic may differ from other programmers. Integration Testing becomes necessary to verify the software modules work in unity
- At the time of module development, there are wide chances of change in requirements by the clients. These new requirements may not be unit tested and hence system integration Testing becomes necessary.
- Interfaces of the software modules with the database could be erroneous
- External Hardware interfaces, if any, could be erroneous

Inadequate exception handling could cause issues.

Example of Integration Test Case

-

Integration Test Case differs from other test cases in the sense it **focuses mainly on the interfaces & flow of data/information between the modules**. Here priority is to be given for the **integrating links** rather than the unit functions which are already tested.

Sample Integration Test Cases for the following scenario: Application has 3 modules say 'Login Page', 'Mailbox' and 'Delete emails' and each of them is integrated logically.

Here do not concentrate much on the Login Page testing as it's already been done in Unit Testing. But check how it's linked to the Mail Box Page.

Similarly Mail Box: Check its integration to the Delete Mails Module.

Test Case ID	Test Case Objective	Test Case Description	Expected Resul
1	Check the interface link between the Login and Mailbox module	Enter login credentials and click on the Login button	To be directed
2	Check the interface link between the Mailbox and Delete Mails Module	From Mailbox select the email and click a delete button	Selected email the Deleted/Tr

Types of Integration Testing

Software Engineering defines variety of strategies to execute Integration testing, viz.

- Big Bang Approach :
- Incremental Approach: which is further divided into the following
  - Top Down Approach
  - Bottom Up Approach
  - Sandwich Approach – Combination of Top Down and Bottom Up

Below are the different strategies, the way they are executed and their limitations as well advantages.

System Testing

System Testing is a level of testing that validates the complete and fully integrated software product. The purpose of a system test is to evaluate the end-to-end system specifications. Usually, the software is only one element of a larger computer-based system. Ultimately, the software is interfaced with other software/hardware systems. System Testing is defined as a series of different tests whose sole purpose is to exercise the full computer-based system.

System Testing Video Explanation

System Testing is Blackbox

Two Category of Software Testing

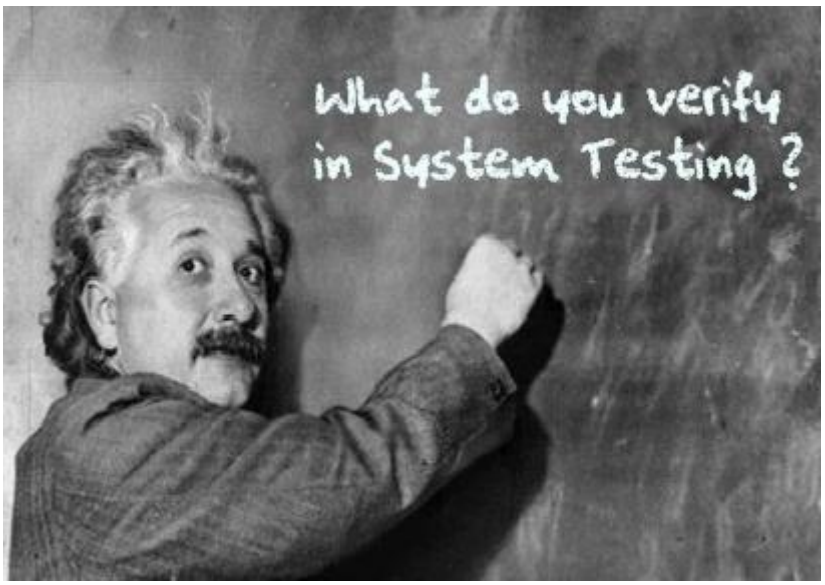
- Black Box Testing
- White Box Testing

System test falls under the **black box testing** category of software testing.

White box testing is the testing of the internal workings or code of a software application. In contrast, black box or System Testing is the opposite. System test involves the external workings of the software from the user's perspective.

What do you verify in System Testing?

System Testing involves testing the software code for following

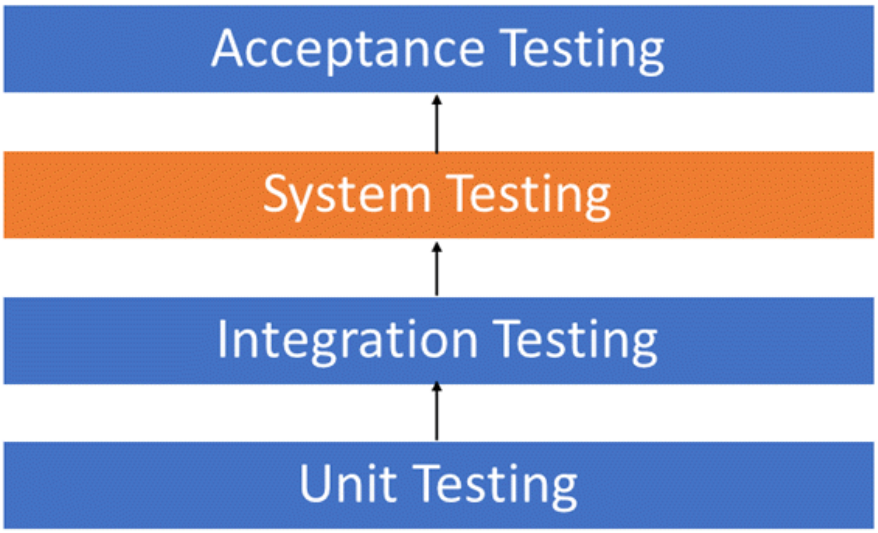


- Testing the fully integrated applications including external peripherals in order to check how components interact with one another and with the system as a whole. This is also called End to End testing scenario.
- Verify thorough testing of every input in the application to check for desired outputs.
- Testing of the user's experience with the application

That is a very basic description of what is involved in system testing. You need to build detailed test cases and test suites that test each aspect of the

application as seen from the outside without looking at the actual source code.

Software Testing Hierarchy



As with almost any software engineering process, software testing has a prescribed order in which things should be done. The following is a list of software testing categories arranged in chronological order. These are the steps taken to fully test new software in preparation for marketing it:

- Unit testing performed on each module or block of code during development. Unit Testing is normally done by the programmer who writes the code.
- Integration testing done before, during and after integration of a new module into the main software package. This involves testing of each individual code module. One piece of software can contain several modules which are often created by several different programmers. It is crucial to test each module's effect on the entire program model.
- System testing done by a professional testing agent on the completed software product before it is introduced to the market.
- Acceptance testing – beta testing of the product done by the actual end users.

Types of System Testing

There are more than 50 types of System Testing. For an exhaustive list of software testing types click here. Below we have listed types of system testing a large software development company would typically use

1. Usability Testing – mainly focuses on the user's ease to use the application, flexibility in handling controls and ability of the system to meet its objectives
2. Load Testing – is necessary to know that a software solution will perform under real-life loads.
3. Regression Testing – involves testing done to make sure none of the changes made over the course of the development process have caused new bugs. It also makes sure no old bugs appear from the addition of new software modules over time.
4. Recovery Testing – is done to demonstrate a software solution is reliable, trustworthy and can successfully recoup from possible crashes.
5. Migration Testing – is done to ensure that the software can be moved from older system infrastructures to current system infrastructures without any issues.
6. Functional Testing – Also known as functional completeness testing, Functional Testing involves trying to think of any possible missing functions. Testers might make a list of additional functionalities that a product could have to improve it during functional testing.
7. Hardware/Software Testing – IBM refers to Hardware/Software testing as "HW/SW Testing". This is when the tester focuses his/her attention on the

Alpha Testing and Beta Testing

Alpha testing is a type of acceptance testing, which is performed to identify all possible bugs/issues before releasing the product to the end-user. Alpha test is a preliminary software field test carried out by a team of users to find out the bugs that were not found previously by other tests. Alpha testing is to simulate a real user environment by carrying out tasks and operations that actual user might perform. Alpha testing implies a meeting with a software vendor and client to ensure that the developers appropriately meet the client's requirements in terms of the performance, functionality, and durability of the software.

Alpha testing needs lab environment, and usually, the testers are an internal employee of the organization. This testing is called alpha because it is done early on, near the end of the software development, but before beta testing.

Beta Testing is a type of acceptance testing; it is the final test before shipping a product to the customers. Beta testing of a product is implemented by "real users "of the software application in a "real environment." In this phase of testing, the software is released to a limited number of end-users of the product to obtain feedback on

the product quality. It allows the real customers an opportunity to provide inputs into the design, functionality, and usability of the product. These inputs are essential for the success of the product. Beta testing reduces product failure risks and increases the quality of the product through customer validation. Direct feedback from customers is a significant advantage of beta testing. This testing helps to tests the software in a real environment. The experiences of the previous users are forwarded back to the developers who make final changes before releasing the software product.

Differences between the Alpha testing and Beta testing are:

Sr. No.	Alpha Testing	Beta Testing
1.	Alpha testing performed by a team of highly skilled testers who are usually the internal employee of the organization.	Beta testing performed by clients or end-u environment, who is not an employee of the
2.	Alpha testing performed at the developer's site; it always needs a testing environment or lab environment.	Beta testing doesn't need any lab environ environment; it is performed at a client's loc the product.
3.	Reliability or security testing not performed in-depth in alpha testing.	Reliability, security, and robustness checked o
4.	Alpha testing involves both white box and black-box techniques.	Beta testing uses only black-box testing.
5.	Long execution cycles maybe require for alpha testing.	Only a few weeks are required for the executi
6.	Critical issues or fixes can be identified by developers immediately in alpha testing.	Most of the issues or feedback is collecting fr will be implemented for the future versions o
7.	Alpha testing performed before the launch of the product into the market.	At the time of software product marketing.
8.	Alpha testing focuses on the product's quality before going to beta testing.	Beta testing concentrates on the quality o gathers users input on the product and ensu

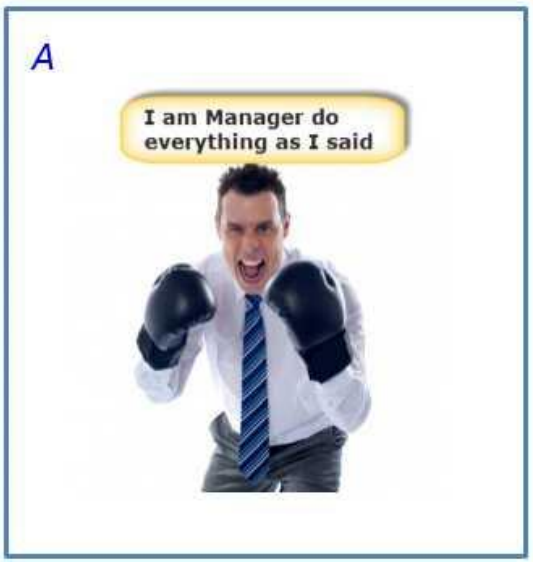
		is ready for real-time users.
9.	Alpha testing performed nearly the end of the software development.	Beta testing is a final test before shipping customers.
10.	Alpha testing is conducting in the presence of developers and the absence of end-users.	Beta testing reversed of alpha testing.

Test Plan

A **Test Plan** is a detailed document that describes the test strategy, objectives, schedule, estimation, deliverables, and resources required to perform testing for a software product. Test Plan helps us determine the effort needed to validate the quality of the application under test. The test plan serves as a blueprint to conduct software testing activities as a defined process, which is minutely monitored and controlled by the test manager.
As per ISTQB definition: "Test Plan is A document describing the scope, approach, resources, and schedule of intended test activities."

Let's start with following Test Plan example/scenario: In a meeting, you want to discuss the Test Plan with the team members, but they are not interested – .

In such case, what will you do? Select your answer as following figure



Importance of Test Plan

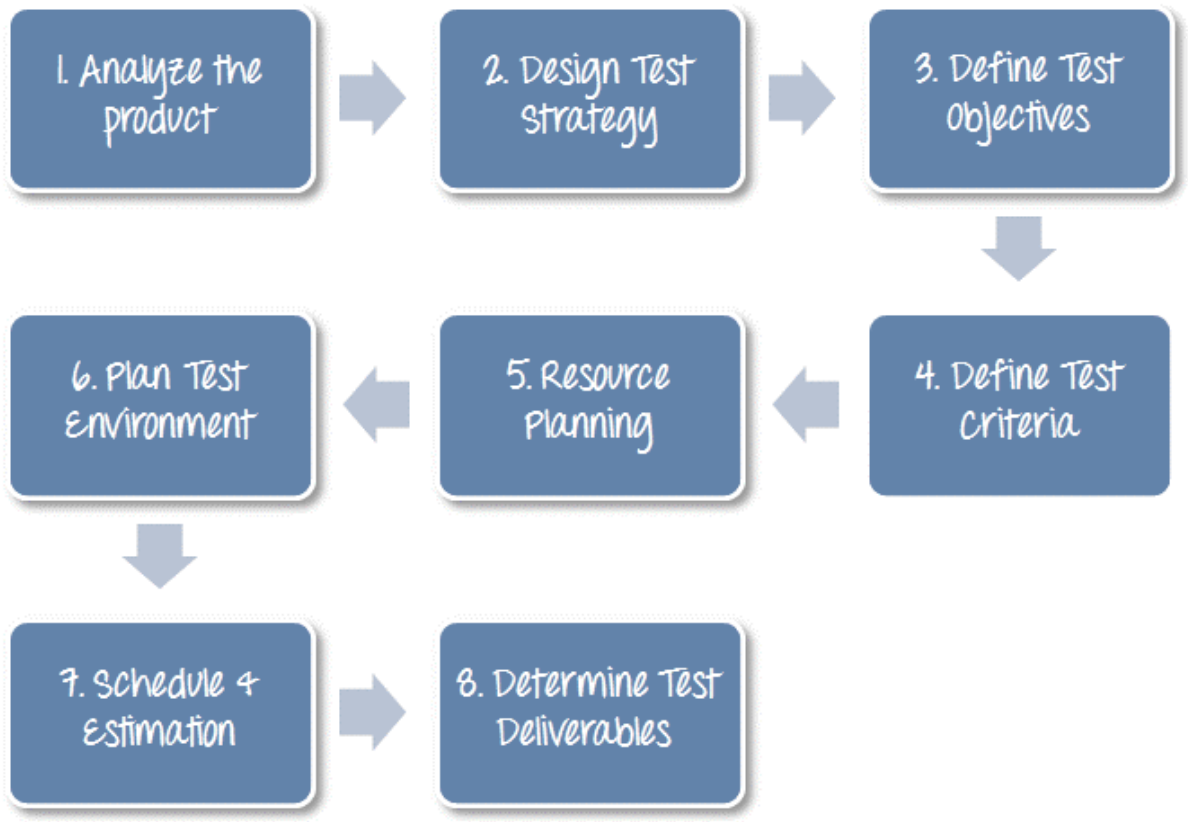
Making Test Plan document has multiple benefits

- Help people outside the test team such as developers, business managers, customers **understand** the details of testing.
- Test Plan **guides** our thinking. It is like a rule book, which needs to be followed.
- Important aspects like test estimation, test scope, Test Strategy are **documented** in Test Plan, so it can be reviewed by Management Team and re-used for other projects.

How to write a Test Plan

You already know that making a **Test Plan** is the most important task of Test Management Process. Follow the seven steps below to create a test plan as per IEEE 829

1. Analyze the product
2. Design the Test Strategy
3. Define the Test Objectives
4. Define Test Criteria
5. Resource Planning
6. Plan Test Environment
7. Schedule & Estimation
8. Determine Test Deliverables



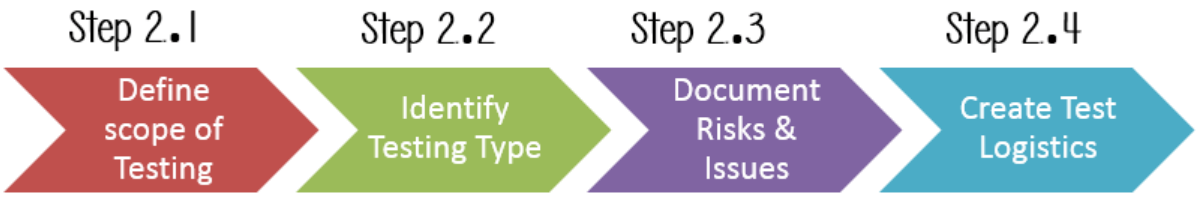
Step 1) Analyze the product

Step 2) Develop Test Strategy

Test Strategy is a **critical step** in making a Test Plan in Software Testing. A Test Strategy document, is a high-level document, which is usually developed by Test Manager. This document defines:

- The project's **testing objectives** and the means to achieve them
- Determines testing **effort** and **costs**

Back to your project, you need to develop Test Strategy for testing that banking website. You should follow steps below



Step 2.1) Define Scope of Testing

Before the start of any test activity, scope of the testing should be known. You must think hard about it.

- The components of the system to be tested (hardware, software, middleware, etc.) are defined as "**in scope**"
- The components of the system that will not be tested also need to be clearly defined as being "**out of scope**."

Defining the scope of your testing project is very important for all stakeholders. A precise scope helps you

- Give everyone a **confidence & accurate information** of the testing you are doing
- All project members will have a **clear** understanding about what is tested and what is not

How do you determine scope your project?

To determine scope, you must –

- Precise customer requirement
- Project Budget
- Product Specification
- Skills & talent of your test team

Now should clearly define the "in scope" and "out of scope" of the testing.

- As the software requirement specs, the project Guru99 Bank only focus on testing all the **functions** and external interface of website **Guru99** Bank (**in scope** testing)
- Nonfunctional testing such as **stress**, **performance** or **logical database** currently will not be tested. (**out of** scope)

Problem Scenario

The customer wants you to test his API. But the project budget does not permit to do so. In such a case what will you do?

Well, in such case you need to convince the customer that Api Testing is extra work and will consume significant resources. Give him data supporting your facts. Tell him if Api Testing is included in-scope the budget will increase by XYZ amount.

The customer agrees and accordingly the new scopes, out of scope items are

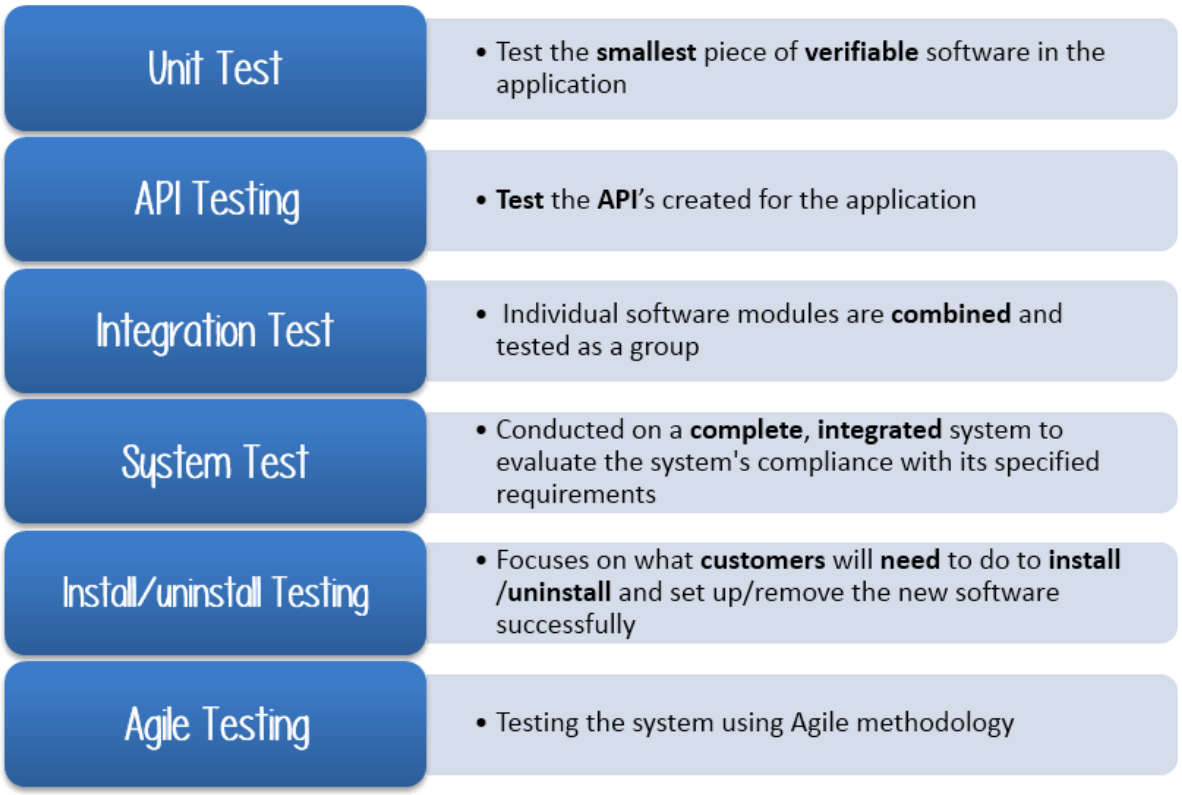
- In-scope items: Functional Testing, Api Testing
- Out of scope items: Database Testing, hardware & any other external interfaces

Step 2.2) Identify Testing Type

A **Testing Type** is a standard test procedure that gives an expected test outcome.

Each testing type is formulated to identify a specific type of product bugs. But, all Testing Types are aimed at achieving one common goal "**Early detection of** all the defects before releasing the product to the customer"

The **commonly used** testing types are described as following figure



Commonly Used Testing Types

There are **tons of Testing Types** for testing software product. Your team **cannot have** enough efforts to handle all kind of testing. As Test Manager, you must set **priority** of the Testing Types

- Which Testing Types should be **focused** for web application testing?
- Which Testing Types should be **ignored** for saving cost?

Step 2.3) Document Risk & Issues

Risk is future's **uncertain event** with a probability of **occurrence** and a **potential** for loss. When the risk actually happens, it becomes the '**issue'.**

In the article Risk Analysis and Solution, you have already learned about the 'Risk' analysis in detail and identified potential risks in the project.

In the QA Test Plan, you will document those risks

Risk	Mitigation
Team member lack the required skills for website testing.	Plan **training course** to skill up your members
The project schedule is too tight; it's hard to complete this project on time	Set **Test Priority** for each of the test activity.
Test Manager has poor management skill	Plan **leadership training** for manager
A lack of cooperation negatively affects your employees' productivity	**Encourage** each team member in his task, **and insp** greater efforts.
Wrong budget estimate and cost overruns	Establish the **scope** before beginning work, pay a l project planning and constantly track and measur

Step 2.4) Create Test Logistics

In Test Logistics, the Test Manager should answer the following questions:

- **Who** will test?
- **When** will the test occur?

Who will test?

You may not know exact names of the tester who will test, but the **type of tester** can be defined.

To select the right member for specified task, you have to consider if his skill is qualified for the task or not, also estimate the project budget. Selecting wrong member for the task may cause the project to **fail** or **delay**.

Person having the following skills is most ideal for performing software testing:

- Ability to **understand** customers point of view
- Strong **desire** for quality
- **Attention** to detail
- Good **cooperation**

In your project, the member who will take in charge for the test execution is the **tester.** Base on the project budget, you can choose in-source or outsource member as the tester.

When will the test occur?

Test activities must be matched with associated development activities.

You will start to test when you have **all required items** shown in following figure

Step 3) Define Test Objective

Test Objective is the overall goal and achievement of the test execution. The objective of the testing is finding as many software defects as possible; ensure that the software under test is **bug free** before release.

To define the test objectives, you should do 2 following steps

1. List all the software features (functionality, performance, GUI…) which may need to test.
2. Define the **target** or the **goal** of the test based on above features

Let's apply these steps to find the test objective of your Guru99 Bank testing project

You can choose the '**TOP-DOWN**' method to find the website's features which may need to test. In this method, you break down the application under test to **component** and **sub-component**.

In the previous topic, you have already analyzed the requirement specs and walk through the website, so you can create a **Mind-Map** to find the website features as following

Test plan attachments

The previous components of the test plan were principally managerial in nature: tasks, schedules, risks, and so on. A general discussion of technical issues such as test designs and test cases for the items under test appears in Section 5 of the test plan, __Approach.'' The reader may be puzzled as to

where in the test plan are the details needed for organizing and executing the tests.

For example, what are the required inputs, outputs, and procedural steps for each test; where will the tests be stored for each item or feature; will it be tested using a black box, white box, or functional approach? The following components of the test plan contain this detailed information. These documents are generally attached to the test plan.

Requirement identifier	Requirement description	Priority (scale 1–10)	Review status	Test ID
SR-25-13.5	Displays opening screens	8	Yes	TC-25-2 TC-25-5
SR-25-52.2	Checks the validity of user password	9	Yes	TC-25-18 TC-25-23

TABLE 7.3
Example of entries in a requirements traceability matrix.

Test Design Specifications

The IEEE standard for software test documentation describes a test design specification as a test deliverable that specifies the requirements of the test approach . It is used to identity the features covered by this design and associated tests for the features. The test design specification also has links to the associated test cases and test procedures needed to test the features, and also describes in detail pass/fail criteria for the features. The test design specification helps to organize the tests and provides the connection to the actual test inputs/outputs and test steps.

To develop test design specifications many documents such as the requirements, design documents, and user manual are useful. For requirements-based test, developing a requirements traceability matrix is valuable. This helps to insure all requirements are covered by tests, and connects the requirements to the tests. Examples of entries in such a matrix are shown in Table 7.3. Tools called requirements tracers can help to automate traceability tasks . These will be described in Chapter 14. A test design

specification should have the following components according to the IEEE standard . They are listed in the order in which the IEEE recommends they appear in the document. The test planner should be sure to list any related documents that may also contain some of this material.

Test Design Specification Identifier

Give each test design specification a unique identifier and a reference to its associated test plan.

Features to Be Tested

Test items, features, and combination of features covered by this test design specification are listed. References to the items in the requirements and/or design document should be included.

Approach Refinements

In the test plan a general description of the approach to be used to test each item was described. In this document the necessary details are added. For example, the specific test techniques to be used to generate test cases are described, and the rational is given for the choices. The test planner also describes how test results will be analyzed. For example, will an automated comparator be used to compare actual and expected results? The relationships among the associated test cases are discussed. This includes any shared constraints and procedural requirements.

Test Case Identification

Each test design specification is associated with a set of test cases and a set of set procedures. The test cases contain input/output information, and the test procedures contain the steps necessary to execute the tests. A test case may be associated with more than one test design specification.

Pass/Fail Criteria

In this section the specific criteria to be used for determining whether the item has passed/failed a test is given.

Test Case Specifications

This series of documents attached to the test plan defines the test cases required to execute the test items named in the associated test design specification. There are several components in this document. IEEE standards require the components to appear in the order shown here, and references should be provided if some of the contents of the test case specification appear in other documents .

Much attention should be placed on developing a quality set of test case specifications. Strategies and techniques, as described in Chapters 4 and 5 of this text, should be applied to accomplish this task. Each test case must be specified correctly so that time is not wasted in analyzing the results of an erroneous test. In addition, the development of test software and test documentation represent a considerable investment of resources for an organization. They should be considered organizational assets and stored in a test repository. Ideally, the test-related deliverables may be recovered from the test repository and reused by different groups for testing and regression testing in subsequent releases of a particular product or for related products. Careful design and referencing to the appropriate test design specification is important to support testing in the current project and for reuse in future projects.

Test Case Specification Identifier

Each test case specification should be assigned a unique identifier.

Test Items

This component names the test items and features to be tested by this test case specification. References to related documents that describe the items and features, and how they are used should be listed: for example the requirements, and design documents, the user manual.

Input Specifications

This component of the test design specification contains the actual inputs needed to execute the test. Inputs may be described as specific values, or as file names, tables, databases, parameters passed by the operating system,and so on. Any special relationships between the inputs should be identified.

Output Specifications

All outputs expected from the test should be identified. If an output is to be a specific value it should be stated. If the output is a specific feature such as a level of performance it also should be stated. The output specifications are necessary to determine whether the item has passed/failed the test.

Special Environmental Needs

Any specific hardware and specific hardware configurations needed to execute this test case should be identified. Special software required to execute the test such as compilers, simulators, and test coverage tools should be described, as well as needed laboratory space and equipment.

Special Procedural Requirements

Describe any special conditions or constraints that apply to the test procedures associated with this test.

Intercase Dependencies

In this section the test planner should describe any relationships between this test case and others, and the nature of the relationship. The test case identifiers of all related tests should be given.

Test Procedure Specifications

A procedure in general is a sequence of steps required to carry out a specific task.

In this attachment to the test plan the planner specifies the steps required to execute a set of test cases. Another way of describing the test procedure specification is that it specifies the steps necessary to analyze a software item in order to evaluate a set of features. The test procedure specification has several subcomponents that the IEEE recommends being included in the order shown below. As noted previously, reference to documents where parts of these components are described must be provided.

Test Procedure Specification Identifier

Each test procedure specification should be assigned a unique identifier.

Purpose

Describe the purpose of this test procedure and reference any test cases it executes.

Specific Requirements

List any special requirements for this procedure, like software, hardware, and special training.

Procedure Steps

Here the actual steps of the procedure are described. Include methods, documents for recording (logging) results, and recording incidents. These will have associations with the test logs and test incident reports that result from a test run. A test incident report is only required when an unexpected output is observed. Steps include

(i) setup: to prepare for execution of the procedure;

(ii) start: to begin execution of the procedure;

(iii) proceed: to continue the execution of the procedure;

(iv) measure: to describe how test measurements related to outputs will be made;

(v) shut down: to describe actions needed to suspend the test when unexpected events occur;

(vi) restart: to describe restart points and actions needed to restart the procedure from these points;

(vii) stop: to describe actions needed to bring the procedure to an orderly halt;

(viii) wrap up: to describe actions necessary to restore the environment;

(ix) contingencies: plans for handling anomalous events if they occur during execution of this procedure.

Based on above features, you can define the Test Objective of the project as following

- Check that whether website Guru99 **functionality**(Account, Deposit…) is working as expected without any error or bugs in real business environment
- Check that the external interface of the website such as **UI** is working as expected and & meet the customer need
- Verify the **usability** of the website. Are those functionalities convenient for user or not?

SHORT QUESTIONS

1.Define levels of testing?

2.Explain classes and methods as units?

3.Define test planning?

4.Explain integration testing?

5.Define test plan attachments?

LONG QUESTIONS

1.Explain functions and procedures,classes,and methods as units?

2.write about unit test planning and designing test units?

3.Define integration test and system test?

4.Explain in detail about different types of testing?

5.Discuss about test plan components and attachments?

UNIT-4

SOFTWARE QUALITY

Software quality is defined as a field of study and practice that describes the desirable attributes of software products. There are two main approaches to software quality: defect management and quality attributes.

SOFTWARE QUALITY DEFECT MANAGEMENT APPROACH

A software defect can be regarded as any failure to address end-user requirements. Common defects include missed or misunderstood requirements and errors in design, functional logic, data relationships, process timing, validity checking, and coding errors.

The software defect management approach is based on counting and managing defects. Defects are commonly categorized by severity, and the numbers in each category are used for planning. More mature software development organizations use tools, such as defect leakage matrices (for counting the numbers of defects that pass through development phases prior to detection) and control charts, to measure and improve development process capability.

SOFTWARE QUALITY ATTRIBUTES APPROACH

This approach to software quality is best exemplified by fixed quality models, such as ISO/IEC 25010:2011. This standard describes a hierarchy of eight quality characteristics, each composed of sub-characteristics:

1. Functional suitability
2. Reliability
3. Operability
4. Performance efficiency
5. Security
6. Compatibility
7. Maintainability
8. Transferability

ISO/IEC 25010:2011 Software Quality Model

Additionally, the standard defines a quality-in-use model composed of five characteristics:

1. Effectiveness
2. Efficiency
3. Satisfaction
4. Safety
5. Usability

A fixed software quality model is often helpful for considering an overall understanding of software quality. In practice, the relative importance of particular software characteristics typically depends on software domain, product type, and intended usage. Thus, software characteristics should be defined for, and used to guide the development of, each product.

Quality Assurance vs Quality Control – Difference Between Them

ByThomas HamiltonUpdatedJanuary 28, 2023

Key Difference between Quality Assurance and Quality Control

- Quality Assurance is aimed to avoid the defect, whereas Quality control is aimed to identify and fix the defects.
- Quality Assurance provides assurance that the quality requested will be achieved, whereas Quality Control is a procedure that focuses on fulfilling the quality requested.
- Quality Assurance is done in the software development life cycle, whereas Quality Control is done in the software testing life cycle.
- Quality Assurance is a proactive measure, whereas Quality Control is a Reactive measure.
- Quality Assurance requires the involvement of all team members, whereas Quality Control needs only a testing team.
- Quality Assurance is performed before Quality Control.

What is Quality Assurance (QA)?

Quality Assurance is popularly known as QA Testing, is defined as an activity to ensure that an organization is providing the best possible product or service to customers.

What is Quality Control (QC)?

Quality Control in Software Testing is a systematic set of processes used to ensure the quality of software products or services. The main purpose of the quality control process is ensuring that the software product meets the actual requirements by testing and reviewing its functional and non-functional requirements. Quality control is popularly abbreviated as QC. This tutorial gives the difference between Quality Assurance and Quality Control

Difference between Quality Assurance and Quality Control

Quality Assurance (QA)	Quality Control (QC)
It is a procedure that focuses on providing assurance that quality requested will be achieved	It is a procedure that focuses on fulfilling the qua
QA aims to prevent the defect	QC aims to identify and fix defects
It is a method to manage the quality- Verification	It is a method to verify the quality-Validation

It does not involve executing the program	It always involves executing a program
It's a Preventive technique	It's a Corrective technique
It's a Proactive measure	It's a Reactive measure
It is the procedure to create the deliverables	It is the procedure to verify that deliverables
QA involves in full software development life cycle	QC involves in full software testing life cycle
In order to meet the customer requirements, QA defines standards and methodologies	QC confirms that the standards are followed whil product
It is performed before Quality Control	It is performed only after QA activity is done
It is a Low-Level Activity, it can identify an error and mistakes which QC cannot	It is a High-Level Activity, it can identify an error t
Its main motive is to prevent defects in the system. It is a less time-consuming activity	Its main motive is to identify defects or bugs in th time-consuming activity
QA ensures that everything is executed in the right way, and that is why it falls under verification activity	QC ensures that whatever we have done is as per that is why it falls under validation activity
It requires the involvement of the whole team	It requires the involvement of the Testing team
The statistical technique applied on QA is known as SPC or Statistical Process Control (SPC)	The statistical technique applied to QC is known a Quality Control

Software Development Life Cycle (SDLC) Phases & Models

What is SDLC?

SDLC is a systematic process for building software that ensures the quality and correctness of the software built. SDLC process aims to produce high-quality software that meets customer expectations. The system development should be complete in the pre-defined time frame and cost. SDLC consists of a detailed plan which explains how to plan, build, and maintain specific software. Every phase of the SDLC life Cycle has its own process and deliverables that feed into the next phase. SDLC stands for **Software Development Life Cycle** and is also referred to as the Application Development life-cycle.
In this Software Development Life Cycle tutorial, you will learn

Here, are prime reasons why SDLC is important for developing a software system.

- It offers a basis for project planning, scheduling, and estimating
- Provides a framework for a standard set of activities and deliverables

- It is a mechanism for project tracking and control
- Increases visibility of project planning to all involved stakeholders of the development process
- Increased and enhance development speed
- Improved client relations
- Helps you to decrease project risk and project management plan overhead

SDLC Phases

The entire SDLC process divided into the following SDLC steps:

SDLC Phases

- Phase 1: Requirement collection and analysis
- Phase 2: Feasibility study
- Phase 3: Design
- Phase 4: Coding
- Phase 5: Testing
- Phase 6: Installation/Deployment
- Phase 7: Maintenance

In this tutorial, I have explained all these Software Development Life Cycle Phases

Phase 1: Requirement collection and analysis

The requirement is the first stage in the SDLC process. It is conducted by the senior team members with inputs from all the stakeholders and domain experts in the industry. Planning for the quality assurance requirements and recognization of the risks involved is also done at this stage.

This stage gives a clearer picture of the scope of the entire project and the anticipated issues, opportunities, and directives which triggered the project.

Requirements Gathering stage need teams to get detailed and precise requirements. This helps companies to finalize the necessary timeline to finish the work of that system.

Phase 2: Feasibility study

Once the requirement analysis phase is completed the next sdlc step is to define and document software needs. This process conducted with the help of 'Software Requirement Specification' document also known as 'SRS' document. It includes everything which should be designed and developed during the project life cycle.

There are mainly five types of feasibilities checks:

- **Economic:** Can we complete the project within the budget or not?

- **Legal:** Can we handle this project as cyber law and other regulatory framework/compliances.
- **Operation feasibility:** Can we create operations which is expected by the client?
- **Technical:** Need to check whether the current computer system can support the software
- **Schedule:** Decide that the project can be completed within the given schedule or not.

Phase 3: Design

In this third phase, the system and software design documents are prepared as per the requirement specification document. This helps define overall system architecture.

This design phase serves as input for the next phase of the model.

There are two kinds of design documents developed in this phase:

High-Level Design (HLD)

- Brief description and name of each module
- An outline about the functionality of every module
- Interface relationship and dependencies between modules
- Database tables identified along with their key elements
- Complete architecture diagrams along with technology details

Low-Level Design (LLD)

- Functional logic of the modules
- Database tables, which include type and size
- Complete detail of the interface
- Addresses all types of dependency issues
- Listing of error messages
- Complete input and outputs for every module

Phase 4: Coding

Once the system design phase is over, the next phase is coding. In this phase, developers start build the entire system by writing code using the chosen programming language. In the coding phase, tasks are divided into units or modules and assigned to the various developers. It is the longest phase of the Software Development Life Cycle process.

In this phase, Developer needs to follow certain predefined coding guidelines. They also need to use programming tools like compiler, interpreters, debugger to generate and implement the code.

Phase 5: Testing

Once the software is complete, and it is deployed in the testing environment. The testing team starts testing the functionality of the entire system. This is done to verify that the entire application works according to the customer requirement.

During this phase, QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and send back to QA for a re-test. This

process continues until the software is bug-free, stable, and working according to the business needs of that system.

Phase 6: Installation/Deployment

Once the software testing phase is over and no bugs or errors left in the system then the final deployment process starts. Based on the feedback given by the project manager, the final software is released and checked for deployment issues if any.

Phase 7: Maintenance

Once the system is deployed, and customers start using the developed system, following 3 activities occur

- Bug fixing – bugs are reported because of some scenarios which are not tested at all
- Upgrade – Upgrading the application to the newer versions of the Software
- Enhancement – Adding some new features into the existing software

The main focus of this SDLC phase is to ensure that needs continue to be met and that the system continues to perform as per the specification mentioned in the first phase.

Popular SDLC Models

Here, are some of the most important models of Software Development Life Cycle (SDLC):

Waterfall model in SDLC

The waterfall is a widely accepted SDLC model. In this approach, the whole process of the software development is divided into various phases of SDLC. In this SDLC model, the outcome of one phase acts as the input for the next phase.

This SDLC model is documentation-intensive, with earlier phases documenting what need be performed in the subsequent phases.

Incremental Model in SDLC

The incremental model is not a separate model. It is essentially a series of waterfall cycles. The requirements are divided into groups at the start of the project. For each group, the SDLC model is followed to develop software. The SDLC life cycle process is repeated, with each release adding more functionality until all requirements are met. In this method, every cycle act as the maintenance phase for the previous software release. Modification to the incremental model allows development cycles to overlap. After that subsequent cycle may begin before the previous cycle is complete.

V-Model in SDLC

In this type of SDLC model testing and the development, the phase is planned in parallel. So, there are verification phases of SDLC on the side and the validation phase on the other side. V-Model joins by Coding phase.

Agile Model in SDLC

Agile methodology is a practice which promotes continue interaction of development and testing during the SDLC process of any project. In the Agile method, the entire project is divided into small incremental builds. All of these builds are provided in iterations, and each iteration lasts from one to three weeks.

Spiral Model

The spiral model is a risk-driven process model. This SDLC testing model helps the team to adopt elements of one or more process models like a waterfall, incremental, waterfall, etc.

This model adopts the best features of the prototyping model and the waterfall model. The spiral methodology is a combination of rapid prototyping and concurrency in design and development activities.

Big bang model

Big bang model is focusing on all types of resources in software development and coding, with no or very little planning. The requirements are understood and implemented when they come.

This model works best for small projects with smaller size development team which are working together. It is also useful for academic software development projects. It is an ideal model where requirements is either unknown or final release date is not given.

Summary

- The Software Development Life Cycle (SDLC) is a systematic process for building software that ensures the quality and correctness of the software built
- The full form SDLC is Software Development Life Cycle or Systems Development Life Cycle.
- SDLC in software engineering provides a framework for a standard set of activities and deliverables
- Seven different SDLC stages are 1) Requirement collection and analysis 2) Feasibility study: 3) Design 4) Coding 5) Testing: 6) Installation/Deployment and 7) Maintenance
- The senior team members conduct the requirement analysis phase
- Feasibility Study stage includes everything which should be designed and developed during the project life cycle
- In the Design phase, the system and software design documents are prepared as per the requirement specification document
- In the coding phase, developers start build the entire system by writing code using the chosen programming language
- Testing is the next phase which is conducted to verify that the entire application works according to the customer requirement.
- Installation and deployment face begins when the software testing phase is over, and no bugs or errors left in the system
- Bug fixing, upgrade, and engagement actions covered in the maintenance face
- Waterfall, Incremental, Agile, V model, Spiral, Big Bang are some of the popular SDLC models in software engineering
- SDLC in software testing consists of a detailed plan which explains how to plan, build, and maintain specific software

Software Development Life Cycle (SDLC) Phases & Models

What is SDLC?

SDLC is a systematic process for building software that ensures the quality and correctness of the software built. SDLC process aims to produce high-quality software that meets customer expectations. The system development should be complete in the pre-defined time frame and cost. SDLC consists of a detailed plan which explains how to plan, build, and maintain specific software. Every phase of the SDLC life Cycle has its own process and deliverables that feed into the next phase. SDLC stands for **Software Development Life Cycle** and is also referred to as the Application Development life-cycle.
In this Software Development Life Cycle tutorial, you will learn

Here, are prime reasons why SDLC is important for developing a software system.

- It offers a basis for project planning, scheduling, and estimating
- Provides a framework for a standard set of activities and deliverables
- It is a mechanism for project tracking and control
- Increases visibility of project planning to all involved stakeholders of the development process
- Increased and enhance development speed
- Improved client relations
- Helps you to decrease project risk and project management plan overhead

SDLC Phases

The entire SDLC process divided into the following SDLC steps:

SDLC Phases

- Phase 1: Requirement collection and analysis
- Phase 2: Feasibility study
- Phase 3: Design
- Phase 4: Coding
- Phase 5: Testing
- Phase 6: Installation/Deployment
- Phase 7: Maintenance

In this tutorial, I have explained all these Software Development Life Cycle Phases

Phase 1: Requirement collection and analysis

The requirement is the first stage in the SDLC process. It is conducted by the senior team members with inputs from all the stakeholders and domain experts in the industry. Planning for the quality assurance requirements and recognization of the risks involved is also done at this stage.

This stage gives a clearer picture of the scope of the entire project and the anticipated issues, opportunities, and directives which triggered the project.

Requirements Gathering stage need teams to get detailed and precise requirements. This helps companies to finalize the necessary timeline to finish the work of that system.

Phase 2: Feasibility study

Once the requirement analysis phase is completed the next sdlc step is to define and document software needs. This process conducted with the help of 'Software Requirement Specification' document also known as 'SRS' document. It includes everything which should be designed and developed during the project life cycle.

There are mainly five types of feasibilities checks:

- **Economic:** Can we complete the project within the budget or not?
- **Legal:** Can we handle this project as cyber law and other regulatory framework/compliances.
- **Operation feasibility:** Can we create operations which is expected by the client?
- **Technical:** Need to check whether the current computer system can support the software
- **Schedule:** Decide that the project can be completed within the given schedule or not.

Phase 3: Design

In this third phase, the system and software design documents are prepared as per the requirement specification document. This helps define overall system architecture.

This design phase serves as input for the next phase of the model.

There are two kinds of design documents developed in this phase:

High-Level Design (HLD)

- Brief description and name of each module
- An outline about the functionality of every module
- Interface relationship and dependencies between modules
- Database tables identified along with their key elements
- Complete architecture diagrams along with technology details

Low-Level Design (LLD)

- Functional logic of the modules
- Database tables, which include type and size
- Complete detail of the interface
- Addresses all types of dependency issues

- Listing of error messages
- Complete input and outputs for every module

Phase 4: Coding

Once the system design phase is over, the next phase is coding. In this phase, developers start build the entire system by writing code using the chosen programming language. In the coding phase, tasks are divided into units or modules and assigned to the various developers. It is the longest phase of the Software Development Life Cycle process.

In this phase, Developer needs to follow certain predefined coding guidelines. They also need to use programming tools like compiler, interpreters, debugger to generate and implement the code.

Phase 5: Testing

Once the software is complete, and it is deployed in the testing environment. The testing team starts testing the functionality of the entire system. This is done to verify that the entire application works according to the customer requirement.

During this phase, QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and send back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the business needs of that system.

Phase 6: Installation/Deployment

Once the software testing phase is over and no bugs or errors left in the system then the final deployment process starts. Based on the feedback given by the project manager, the final software is released and checked for deployment issues if any.

Phase 7: Maintenance

Once the system is deployed, and customers start using the developed system, following 3 activities occur

- Bug fixing – bugs are reported because of some scenarios which are not tested at all
- Upgrade – Upgrading the application to the newer versions of the Software
- Enhancement – Adding some new features into the existing software

Software Quality Assurance

Quality defines to any measurable characteristics such as correctness, maintainability, portability, testability, usability, reliability, efficiency, integrity, reusability, and interoperability.

There are two kinds of Quality:

Quality of Design: Quality of Design refers to the characteristics that designers specify for an item. The grade of materials, tolerances, and performance specifications that all contribute to the quality of design.

Quality of conformance: Quality of conformance is the degree to which the design specifications are followed during manufacturing. Greater the degree of conformance, the higher is the level of quality of conformance.

Quality Control: Quality Control involves a series of inspections, reviews, and tests used throughout the software process to ensure each work product meets the requirements place upon it. Quality control includes a feedback loop to the process that created the work product.

Quality Assurance: Quality Assurance is the preventive set of activities that provide greater confidence that the project will be completed successfully.

Quality Assurance focuses on how the engineering and management activity will be done?

As anyone is interested in the quality of the final product, it should be assured that we are building the right product.

It can be assured only when we do inspection & review of intermediate products, if there are any bugs, then it is debugged. This quality can be enhanced.

Importance of Quality

We would expect the quality to be a concern of all producers of goods and services. However, the distinctive characteristics of software and in particular its intangibility and complexity, make special demands.

Increasing criticality of software: The final customer or user is naturally concerned about the general quality of software, especially its reliability. This is increasing in the case as organizations become more dependent on their computer systems and software is used more and more in safety-critical areas. For example, to control aircraft.

The intangibility of software: This makes it challenging to know that a particular task in a project has been completed satisfactorily. The results of these tasks can be made tangible by demanding that the developers produce 'deliverables' that can be examined for quality.

Accumulating errors during software development: As computer system development is made up of several steps where the output from one level is input to the next, the errors in the earlier ?deliverables? will be added to those in the later stages leading to accumulated determinable effects. In general the later in a project that an error is found, the more expensive it will be to fix. In addition, because the number of errors in the system is unknown, the debugging phases of a project are particularly challenging to control.

Software Quality Assurance

Software quality assurance is a planned and systematic plan of all actions necessary to provide adequate confidence that an item or product conforms to establish technical requirements.

A set of activities designed to calculate the process by which the products are developed or manufactured.

SQA Encompasses

- A quality management approach
- Effective Software engineering technology (methods and tools)
- Formal technical reviews that are tested throughout the software process
- A multitier testing strategy
- Control of software documentation and the changes made to it.
- A procedure to ensure compliances with software development standards
- Measuring and reporting mechanisms.

SQA Activities

Software quality assurance is composed of a variety of functions associated with two different constituencies ? the software engineers who do technical work and an SQA

group that has responsibility for quality assurance planning, record keeping, analysis, and reporting.

Following activities are performed by an independent SQA group:

1. **Prepares an SQA plan for a project:** The program is developed during project planning and is reviewed by all stakeholders. The plan governs quality assurance activities performed by the software engineering team and the SQA group. The plan identifies calculation to be performed, audits and reviews to be performed, standards that apply to the project, techniques for error reporting and tracking, documents to be produced by the SQA team, and amount of feedback provided to the software project team.
2. **Participates in the development of the project's software process description:** The software team selects a process for the work to be performed. The SQA group reviews the process description for compliance with organizational policy, internal software standards, externally imposed standards (e.g. ISO-9001), and other parts of the software project plan.
3. **Reviews software engineering activities to verify compliance with the defined software process:** The SQA group identifies, reports, and tracks deviations from the process and verifies that corrections have been made.
4. **Audits designated software work products to verify compliance with those defined as a part of the software process:** The SQA group reviews selected work products, identifies, documents and tracks deviations, verify that corrections have been made, and periodically reports the results of its work to the project manager.
5. **Ensures that deviations in software work and work products are documented and handled according to a documented procedure:** Deviations may be encountered in the project method, process description, applicable standards, or technical work products.
6. **Records any noncompliance and reports to senior management:** Non-compliance items are tracked until they are resolved.

Quality Assurance v/s Quality control

Quality Assurance	Quality Control
Quality Assurance (QA) is the set of actions including facilitation, training, measurement, and analysis needed to provide adequate	**Quality Control (QC)** is described a methods used to compare p

confidence that processes are established and continuously improved to produce products or services that conform to specifications and are fit for use.	requirements and applicable standa are taken when a nonconformance i
QA is an activity that establishes and calculates the processes that produce the product. If there is no process, there is no role for QA.	**QC** is an activity that demonstrates product produced met standards.
QA helps establish process	**QC** relates to a particular product or
QA sets up a measurement program to evaluate processes	**QC** verified whether particular attrib exist, in a explicit product or service.
QA identifies weakness in processes and improves them	**QC** identifies defects for the primary errors.
Quality Assurance is a managerial tool.	Quality Control is a corrective tool.
Verification is an example of QA.	Validation is an example of QC.

SHORT QUESTIONS

1.Define software quality?

2.Differentiate between quality control and quality assurance?

3.Write a short note on sdlc?

4.Explain sqa function?

5.Define quality management system in organization?

LONG QUESTIONS

1.Define software quality and Explain importance of quality?

2.Explain in detail about quality control and quality assurance?

3.Discuss about quality assurance at each phase of sdlc?

4.Write about software support projects on quality assurance?

5.Explain in detail about software quality and product quality?

UNIT-5

Software Metrics

A software metric is a measure of software characteristics which are measurable or countable. Software metrics are valuable for many reasons, including measuring software performance, planning work items, measuring productivity, and many other uses.

Within the software development process, many metrics are that are all connected. Software metrics are similar to the four functions of management: Planning, Organization, Control, or Improvement.

Classification of Software Metrics

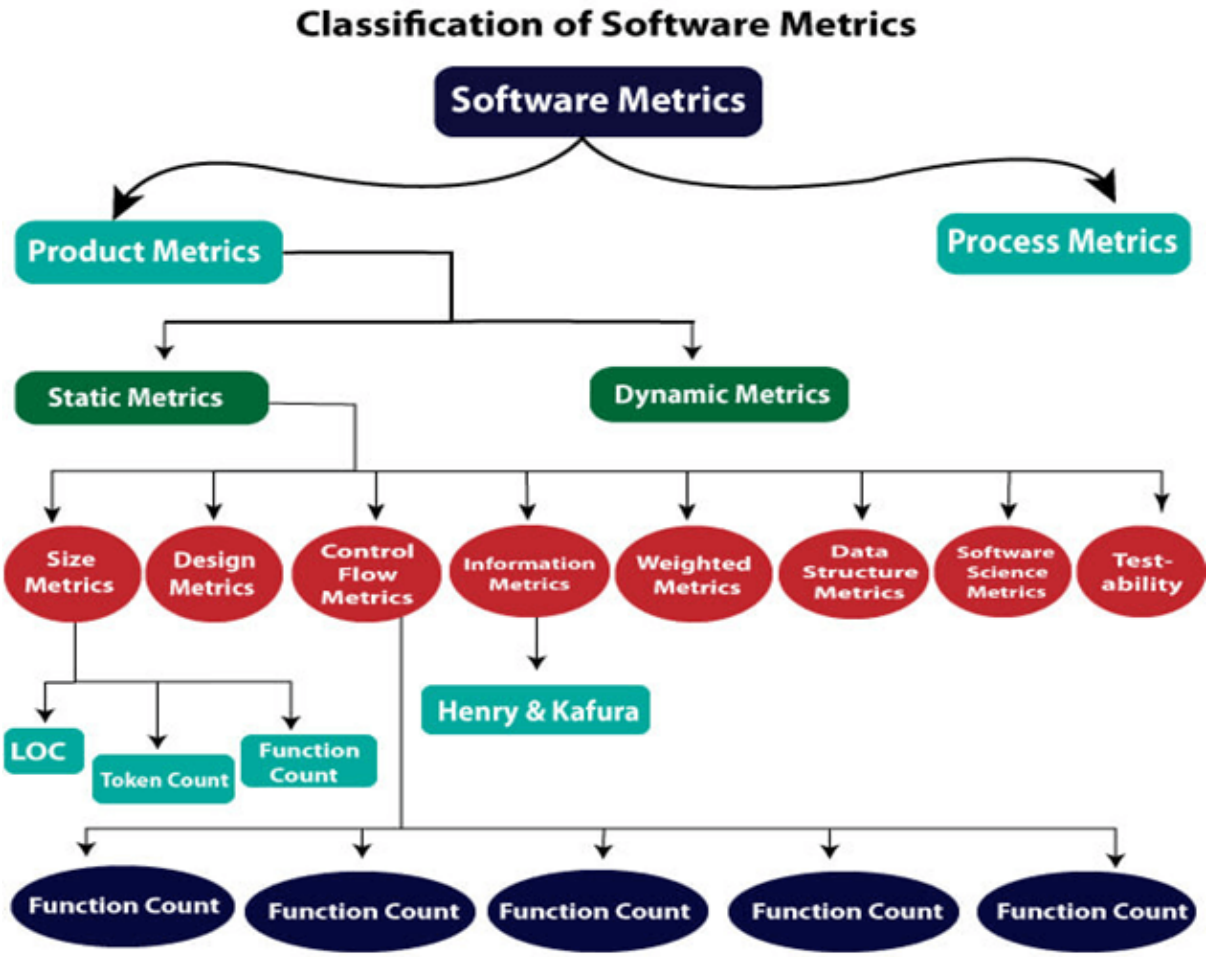
Software metrics can be classified into two types as follows:

1. Product Metrics: These are the measures of various characteristics of the software product. The two important software characteristics are:

1. Size and complexity of software.
2. Quality and reliability of software.

These metrics can be computed for different stages of SDLC.

2. Process Metrics: These are the measures of various characteristics of the software development process. For example, the efficiency of fault detection. They are used to measure the characteristics of methods, techniques, and tools that are used for developing software.



Types of Metrics

Internal metrics: Internal metrics are the metrics used for measuring properties that are viewed to be of greater importance to a software developer. For example, Lines of Code (LOC) measure.

External metrics: External metrics are the metrics used for measuring properties that are viewed to be of greater importance to the user, e.g., portability, reliability, functionality, usability, etc.

Hybrid metrics: Hybrid metrics are the metrics that combine product, process, and resource metrics. For example, cost per FP where FP stands for Function Point Metric.

Project metrics: Project metrics are the metrics used by the project manager to check the project's progress. Data from the past projects are used to collect various metrics, like time and cost; these estimates are used as a base of new software. Note that as the project proceeds, the project manager will check its progress from time-to-time and will compare the effort, cost, and time with the original effort, cost and time. Also understand that these metrics are used to decrease the development

costs, time efforts and risks. The project quality can also be improved. As quality improves, the number of errors and time, as well as cost required, is also reduced.

Advantage of Software Metrics

Comparative study of various design methodology of software systems.

For analysis, comparison, and critical study of different programming language concerning their characteristics.

In comparing and evaluating the capabilities and productivity of people involved in software development.

In the preparation of software quality specifications.

In the verification of compliance of software systems requirements and specifications.

In making inference about the effort to be put in the design and development of the software systems.

In getting an idea about the complexity of the code.

In taking decisions regarding further division of a complex module is to be done or not.

In guiding resource manager for their proper utilization.

In comparison and making design tradeoffs between software development and maintenance cost.

In providing feedback to software managers about the progress and quality during various phases of the software development life cycle.

In the allocation of testing resources for testing the code.

Disadvantage of Software Metrics

The application of software metrics is not always easy, and in some cases, it is difficult and costly.

The verification and justification of software metrics are based on historical/empirical data whose validity is difficult to verify.

These are useful for managing software products but not for evaluating the performance of the technical staff.

The definition and derivation of Software metrics are usually based on assuming which are not standardized and may depend upon tools available and working environment.

Most of the predictive models rely on estimates of certain variables which are often not known precisely.

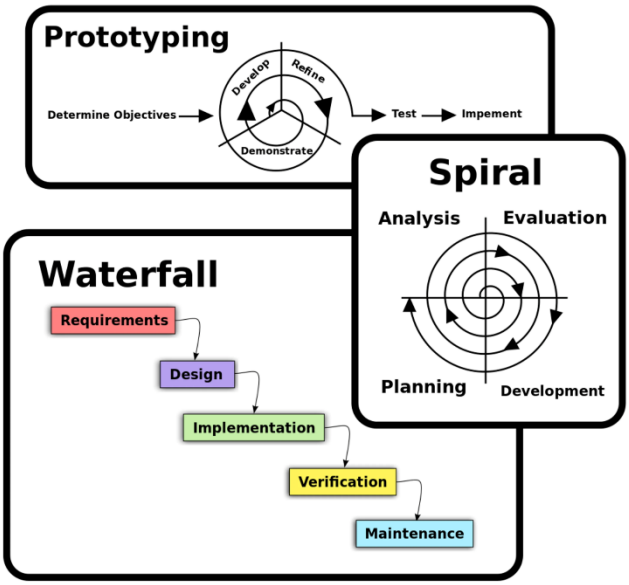
Software process model

A software process model is an abstraction of the software development process. The models specify the stages and order of a process. So, think of this as a representation of the **order of activities** of the process and the **sequence** in which they are performed.

A model will define the following:

- The tasks to be performed
- The input and output of each task
- The pre and post-conditions for each task
- The flow and sequence of each task

The goal of a software process model is to provide guidance for controlling and coordinating the tasks to achieve the end product and objectives as effectively as possible.



Source: Omar Elgabry

There are many kinds of process models for meeting different requirements. We refer to these as **SDLC models** (Software Development Life Cycle models). The most popular and important SDLC models are as follows:

- Waterfall model
- V model
- Incremental model

- RAD model

- Agile model
- Iterative model
- Prototype model
- Spiral model

Factors in choosing a software process

Choosing the right software process model for your project can be difficult. If you know your requirements well, it will be easier to select a model that best matches your needs. You need to keep the following factors in mind when selecting your software process model:

Project requirements

Before you choose a model, take some time to go through the project requirements and clarify them alongside your organization's or team's expectations. Will the user need to specify requirements in detail after each iterative session? Will the requirements *change* during the development process?

Project size

Consider the size of the project you will be working on. Larger projects mean bigger teams, so you'll need more extensive and elaborate project management plans.

Project complexity

Complex projects may not have clear requirements. The requirements may change often, and the cost of delay is high. Ask yourself if the project requires constant monitoring or feedback from the client.

Cost of delay

Is the project highly time-bound with a huge cost of delay, or are the timelines flexible?

Customer involvement

Do you need to consult the customers during the process? Does the user need to participate in all phases?

Familiarity with technology

This involves the developers' knowledge and experience with the project domain, software tools, language, and methods needed for development.

Project resources

This involves the amount and availability of funds, staff

Types of software process models

As we mentioned before, there are multiple kinds of software process models that each meet different requirements below, we will look at the top seven types of software process models that you should know.

Waterfall Model

The waterfall model is a **sequential, plan driven-process** where you must plan and schedule all your activities before starting the project. Each activity in the waterfall model is represented as a separate phase arranged in linear order.

It has the following phases:

- Requirements
- Design
- Implementation
- Testing
- Deployment
- Maintenance

Each of these phases produces one or more documents that need to be approved before the next phase begins. However, in practice, these phases are very likely to overlap and may feed information to one another.

The software process **isn't linear**, so the documents produced may need to be modified to reflect changes.

The waterfall model is easy to understand and follow. It doesn't require a lot of customer involvement after the specification is done. Since it's inflexible, it can't adapt to changes. There is no way to see or try the software until the last phase.

The waterfall model has a rigid structure, so it should be used in cases where the requirements are understood completely and unlikely to radically change.

V Model

The V model (Verification and Validation model) is an extension of the waterfall model. All the requirements are gathered at the start and cannot be changed. You have a corresponding testing activity for each stage. For every phase in the development cycle, there is an **associated testing phase.**

The corresponding testing phase of the development phase is planned in parallel, as you can see above.

The V model is highly disciplined, easy to understand, and makes project management easier. But it isn't good for complex projects or projects that have unclear or changing requirements. This makes the V model a good choice for software where downtimes and failures are unacceptable.

Incremental Model

The incremental model divides the system's functionality into **small increments** that are delivered one after the other in quick succession. The most important functionality is implemented in the initial increments.

The subsequent increments expand on the previous ones until everything has been updated and implemented.

Incremental development is based on developing an initial implementation, exposing it to user feedback, and evolving it through new versions. The process' activities are interwoven by feedback.

Each iteration passes through the requirements, design, coding, and testing stages.

The incremental model lets stakeholders and developers see results with the first increment. If the stakeholders don't like anything, everyone finds out a lot sooner. It is efficient as the developers only focus on what is important and bugs are fixed as they arise, but you need a **clear and complete definition** of the whole system before you start.

The incremental model is great for projects that have loosely coupled parts and projects with complete and clear requirements.

Iterative Model

The iterative development model develops a system by **building small portions** of all the features. This helps to meet the initial scope quickly and release it for feedback.

In the iterative model, you start off by implementing a small set of software requirements. These are then **enhanced iteratively** in the evolving versions until the system is completed. This process model starts with part of the software, which is then implemented and reviewed to identify further requirements.

Like the incremental model, the iterative model allows you to see the results at the early stages of development. This makes it easy to identify and **fix any functional or design flaws**. It also makes it easier to manage risk and change requirements.

The deadline and budget may change throughout the development process, especially for large complex projects. The iterative model is a good choice for large software that can be **easily broken down into modules**.

RAD Model

The Rapid Application Development (RAD model) is based on iterative development and prototyping with **little planning involved**. You develop functional modules in parallel for faster product delivery. It involves the following phases:

1. Business modeling
2. Data modeling
3. Process modeling
4. Application generation

5. Testing and turnover

The RAD concept focuses on gathering requirements using focus groups and workshops, reusing software components, and informal communication.

The RAD model accommodates changing requirements, reduces development time, and increases the reusability of components. But it can be complex to manage. Therefore, the RAD model is great for systems that need to be produced in a short time and have known requirements.

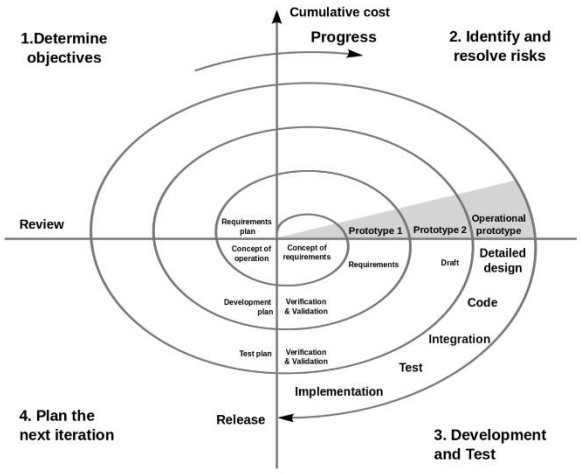
Spiral Model

The spiral model is a risk driven iterative software process model. The spiral model delivers projects in loops. Unlike other process models, its steps aren't activities but **phases** for addressing whatever problem has the greatest risk of causing a failure.

It was designed to include the best features from the waterfall and introduces risk-assessment.

You have the following phases for each cycle:

1. Address the highest-risk problem and determine the objective and alternate solutions
2. Evaluate the alternatives and identify the risks involved and possible solutions
3. Develop a solution and verify if it's acceptable
4. Plan for the next cycle



You develop the concept in the first few cycles, and then it evolves into an implementation. Though this model is great for managing uncertainty, it can be difficult to have stable documentation. The spiral model can be used for projects with **unclear needs** or projects still in research and development.

Agile model

The agile process model encourages **continuous iterations of development** and testing. Each incremental part is developed over an

iteration, and each iteration is designed to be small and manageable so it can be completed within a few weeks.

Each iteration focuses on implementing a small set of features completely. It involves customers in the development process and minimizes documentation by using informal communication.

Agile development considers the following:

- Requirements are assumed to change
- The system evolves over a series of short iterations
- Customers are involved during each iteration
- Documentation is done only when needed

ISO 9001-MODEL

ISO 9001 is defined as the international standard that specifies requirements for a quality management system (QMS). Organizations use the standard to demonstrate the ability to consistently provide products and services that meet customer and regulatory requirements. It is the most popular standard in the ISO 9000 series and the only standard in the series to which organizations can certify.

ISO 9001 was first published in 1987 by the International Organization for Standardization (ISO), an international agency composed of the national standards bodies of more than 160 countries. The current version of ISO 9001 was released in September 2015.

- Who should use the 9001:2015 revision?
- What are the benefits of ISO 9001?
- ISO 9001 certification
- **ISO 9001**

ISO 9001:2015 applies to any organization, regardless of size or industry. More than one million organizations from more than 160 countries have applied the ISO 9001 standard requirements to their quality management systems.

Organizations of all types and sizes find that using the ISO 9001 standard helps them:

- Organize processes
- Improve the efficiency of processes
- Continually improve

All organizations that use ISO 9001 are encouraged to transition to ISO 9001:2015 as soon as possible. This includes not only organizations that are certified to ISO 9001:2008, but also any organizations involved in training or certifying others.

ISO 9001 is based on the plan-do-check-act methodology and provides a process-oriented approach to documenting and reviewing the structure, responsibilities, and procedures required to achieve effective quality management in an organization. Specific sections of the standard contain information on many topics, such as:

- Requirements for a QMS, including documented information, planning and determining process interactions
- Responsibilities of management
- Management of resources, including human resources and an organization's work environment
- Product realization, including the steps from design to delivery
- Measurement, analysis, and improvement of the QMS through activities like internal audits and corrective and preventive action

Changes introduced in the 2015 ISO 9001 revision are intended to ensure that ISO 9001 continues to adapt to the changing environments in which organizations operate. Some of the key updates in ISO 9001:2015 include:

- The introduction of new terminology
- Restructuring some of the information
- An emphasis on risk-based thinking to enhance the application of the process approach
- Improved applicability for services
- Increased leadership requirements

The Seven Principles of ISO 9001:2015

.

Previous versions of ISO 9001

Originally published in 1987, ISO 9001 underwent revisions in 1994, 2000, and again in 2008. The latest revision was published in September 2015.

- ISO 9001:1994 included changes to improve the control of design and development clause, as well as provide other clarifications. The 1994 series also slightly modified the role of ISO 9002 and 9003.
- The ISO 9001:2008 revision sought to clarify issues raised during the application of ISO 9001:2000.

Because ISO 9001 specifies the requirements for an effective quality management system, organizations find that using the standard helps them:

- Organize a QMS
- Create satisfied customers, management, and employees
- Continually improve their processes
- Save costs

In Nevada, the Clark County School District used ISO 9001 to save $174 million over 10 years in actual expenditures and cost avoidance. More than 3,000 employees were trained to the standard, enabling three critical components of the system's success: training, communication and respect, and efficiency.

ISO 9001 CERTIFICATION

ISO 9001 is the only standard in the ISO 9000 series to which organizations can certify. Achieving ISO 9001:2015 certification means that an organization has demonstrated the following:

- Follows the guidelines of the ISO 9001 standard
- Fulfills its own requirements
- Meets customer requirements and statutory and regulatory requirements
- Maintains documentation

Certification to the ISO 9001 standard can enhance an organization's credibility by showing customers that its products and services meet expectations. In some instances or in some industries, certification is required or legally mandated. The certification process includes implementing the requirements of ISO 9001:2015 and then completing a successful registrar's audit confirming the organization meets those requirements.

Organizations should consider the following as they begin preparing for an ISO 9001 quality management system certification:

- Registrar's costs for ISO 9001 registration, surveillance, and recertification audits
- Current level of conformance with ISO 9001 requirements
- Amount of resources that the company will dedicate to this project for development and implementation
- Amount of support that will be required from a consultant and the associated costs

Courses in the ISO 9001 Standard

Training can provide an opportunity to review the ISO 9001:2015 standard and apply quality management principles in a practice environment.

Professionals responsible for developing, implementing, auditing, and managing an ISO quality management system or quality professionals interested in updating their documented ISO 9001-based QMS can take ISO 9000 training courses, which include courses focused on ISO 9001 and quality management systems. Additionally, organizations looking to improve employee performance and employees looking to continually improve will also find ISO 9000 training relevant.

ISO 9001 RESOURCES

You can also search articles, case studies, and publications for ISO 9001 resources.

ISO 9001:2015 Standard

The 2015 revision to ISO 9001, the international standard specifying requirements for quality management systems, is available for purchase in these formats:

- Published hard copy
- PDF e-standard for immediate download (please note that e-standards cannot be printed)
- Site license for posting an electronic version to your Local Area Network or Intranet
- Request for Interpretation (RFI) for ISO 9001:2015

Capability Maturity Model (CMM)

The Capability Maturity Model (CMM) is a methodology used to develop and refine an organization's software development process. The model describes a five-level evolutionary path of increasingly organized and systematically more mature processes.

CMM was developed and is promoted by the Software Engineering Institute (SEI), a research and development center sponsored by the U.S. Department of Defense (DOD) and now part of Carnegie Mellon University. SEI was founded in 1984 to address software engineering issues and, in a broad sense, to advance software engineering methodologies. More specifically, SEI was established to optimize the process of developing,

acquiring and maintaining heavily software-reliant systems for the DOD. SEI advocates industry-wide adoption of the CMM Integration (CMMI), which is an evolution of CMM. The CMM model is still widely used as well.

CMM is similar to ISO 9001, one of the ISO 9000 series of standards specified by the International Organization for Standardization. The ISO 9000 standards specify an effective quality system for manufacturing and service industries; ISO 9001 deals specifically with software development and maintenance.

The main difference between CMM and ISO 9001 lies in their respective purposes: ISO 9001 specifies a minimal acceptable quality level for software processes, while CMM establishes a framework for continuous process improvement. It is more explicit than the ISO standard in defining the means to be employed to that end.

CMM's five levels of maturity for software processes

There are five levels to the CMM development process. They are the following:

1. **Initial.** At the initial level, processes are disorganized, ad hoc and even chaotic. Success likely depends on individual efforts and is not considered to be repeatable. This is because processes are not sufficiently defined and documented to enable them to be replicated.
2. **Repeatable.** At the repeatable level, requisite processes are established, defined and documented. As a result, basic project management techniques are established, and successes in key process areas are able to be repeated.
3. **Defined.** At the defined level, an organization develops its own standard software development process. These defined processes enable greater attention to documentation, standardization and integration.

4. **Managed.** At the managed level, an organization monitors and controls its own processes through data collection and analysis.
5. **Optimizing.** At the optimizing level, processes are constantly improved through monitoring feedback from processes and introducing innovative processes and functionality.

The Capability Maturity Model takes software development processes from disorganized and chaotic to predictable and constantly improving.

CMM vs. CMMI: What's the difference?

CMMI is a newer, updated model of CMM. SEI developed CMMI to integrate and standardize CMM, which has different models for each function it covers. These models were not always in sync; integrating them made the process more efficient and flexible.

CMMI includes additional guidance on how to improve key processes. It also incorporates ideas from Agile development, such as continuous improvement.

SEI released the first version of CMMI in 2002. In 2013, Carnegie Mellon formed the CMMI Institute to oversee CMMI services and future model development. ISACA, a professional organization for IT governance, assurance and cybersecurity professionals, acquired CMMI Institute in 2016. The most recent version -- CMMI V2.0 -- came out in 2018. It focuses on establishing business objectives and tracking those objectives at every level of business maturity.

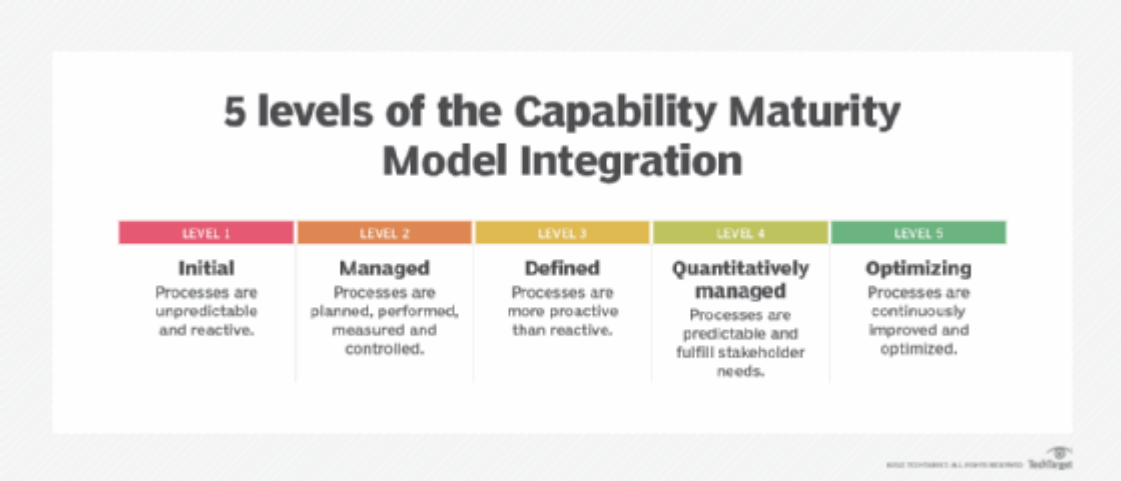
CMMI adds Agile principles to CMM to help improve development processes, software configuration management and software quality management. It does this, in part, by incorporating continuous feedback and continuous improvement into the software development process. Under CMMI, organizations are expected to continually optimize processes, record feedback and use that feedback to further improve processes in a cycle of improvement.

One criticism of CMM is that it is too process-oriented and not goal-oriented enough. Organizations have found it difficult to tailor CMM to specific goals and needs. One of CMMI's improvements is to focus on strategic goals. CMMI is designed to make it easier for businesses to apply the methodology to specific uses than with CMM.

Like CMM, CMMI consists of five process maturity levels. However, they are different from the levels in CMM.

The process performance levels of CMMI are the following:

1. **Initial.** Processes are unpredictable and reactive. They increase risk and decrease efficiency.
2. **Managed.** Processes are planned and managed, but they still have issues.
3. **Defined.** Processes become more proactive than reactive.
4. **Quantitatively managed.** Quantitative data is used to craft predictable processes that fulfill stakeholder needs based on more accurate measurement of adherence to business goals.
5. **Optimizing.** The organization has a set of consistent processes that are constantly being improved and optimized.



Comparative Analysis

Comparison testing is a sort of testing in which the strengths and weaknesses of newly generated software are compared to those of previously released software products on the market. It aids in determining how well a present software product performs in contrast to the competitors, as well as aiding in the creation of a high-quality software product with enhanced performance and functionality.

Actually, comparative testing helps you to discover the flaws in an existing software product and drives you to fix them in order to stay ahead of the competition. However, comparative testing does not attempt to create a competitive market; rather, it focuses on continuously improving software products. For comparison testing, any aspect of the software program might be evaluated. It may be the User Interface, the number of functions, the speed, the database, the security, and so on. These test criteria are mostly determined by the kind of software application is evaluated as well as business-specific use-cases.

Time to do Comparative Testing

There is no distinct phase for comparison testing, nor is there a set guideline for doing it, and it is not a step of software development. It may be done on its own or in conjunction with other types of software testing. However, it is usually done at three phases throughout software development, i.e.

- The software development process is in its early stages.
- The software development process is in its middle stages.
- The last step of the software development cycle

There are two types of testing criteria

When there is a disagreement over the test criteria, it goes through two separate comparison stages, i.e.

- Compare your software to industry standards or benchmarks.
- Compare the features of the software program to those of other software applications.

For example, if you're building a PDF Merge software application, you'll want to compare your product to other PDF Merge software apps to see how it compares in terms of merge speed, performance, and PDF quality of the combined file, among other things.

Example − a) If a Siebel CRM application is being evaluated, for example, we know that all CRM applications include modules that deal with acquiring customer information, processing client orders, handling customer requests, and dealing with customer difficulties..

We may evaluate the application's functionality against recognized standards and functionality as they exist in the market at the time of testing in the first round of testing.

We may pose queries such as −

- Is the program equipped with all of the features that a CRM application should have?
- Is the fundamental functioning of the modules as expected?

We will develop test scenarios in such a manner that the test results confirm the application's functioning in comparison to industry standards.

b) In the second step of testing, we may compare an application's features to those of other software products available on the market.

The following characteristics, for example, maybe compared to those of other software programs.

#1) Price

#2) Application performance

Example − Response time and network load

#3) User Interface (look and feel, ease of use)

Testing activities are planned in such a manner that possible areas of business interruption are discovered in both rounds of testing. To guide test design and execution, a suitable testing strategy is developed.

It is unavoidable to have a thorough understanding of business use cases and needs.

Examples of CRM Application Test Scenarios

Let's look at a CRM application for mobile phone purchases as an example of test scenarios.

We know that every CRM solution should provide the following functions in general −

- Creating a user profile for the purpose of doing business
- Before starting a transaction or placing an order, make sure all checks and criteria are met.
- Checking the item inventory
- Order fulfillment for things
- Customer concerns and requests are handled.

We may develop test scenarios or test conditions as follows, taking into consideration the aforementioned functionalities −

Template for comparing with established standards

Scenario ID	Description	Requirement ID	Business Use case ID

Scenario ID	Description	Requirement ID	Business Use case ID
Scenario###	Check to see whether the CRM program records customer information.	Req####	
Scenario###	Before starting a deal, see whether the CRM program verifies the customer's creditworthiness.	Req####	
Scenario###	Before starting a deal, see whether the CRM program verifies the customer's creditworthiness.	Req####	
Scenario###	Check to see whether the equipment you ordered is in stock.	Req####	
Scenario###	Check whether the customer's geographic location is covered by a mobile network.	Req####	
Scenario###	Check to see whether a problem ticket has been created for each client’s concern.	Req####	
Scenario###	Check to see whether the CRM software has dealt with and resolved the client problem.	Req####	

Comparison Testing's Benefits

- It may identify your application's flaws and strengths.
- It aids in determining the software product's quality.
- It indicates how competitive and beneficial your product is.
- It determines whether or not a software project is commercially viable.
- It indicates if the program has a reasonable probability of becoming lucrative or not.
- It aids with the thorough examination of all critical aspects of software prior to its commercial release.
- It aids in the comprehension of interior design structure.
- It aids in the product's competitiveness, allowing it to function successfully in the market.

Comparison Testing's Drawbacks

- Because it has already gone through a succession of development processes, it becomes very tough to adapt or change anything.
- When customers learn about the flaws or weaknesses in your product, they may develop a negative attitude about it.
- **comparative testing benefit a company**

Unambiguous comparison test criteria and precise test findings may assist the firm in making claims for software products such as,

- In terms of reaction speed, this app is the fastest.
- In terms of network load and other factors, the most durable product

The results of the tests may be used not only to promote the software product but also to uncover flaws and improve it.

An understanding of the testing's obstacles, constraints, and scope −

- Design, development, testing, sales and marketing tactics, investments, and accruing earnings all contribute to the success of any new enterprise or software product
- In this context, comparison testing helps in making crucial software product choices, but it cannot guarantee the product's success. Despite thorough testing, the company may still collapse due to poor business strategy and judgments. As a result, market research and assessment of alternative business strategies is a separate topic from comparison testing.

To get a sense of the extent of this testing, consider the following scenario

- The introduction of Disney mobile in the United States in 2005 is a case in point. Disney entered the cellular communications sector with no previous expertise in the industry. Despite the moniker "Disney," the new mobile enterprise failed miserably in the United States.
- The product failed, not because of terrible design or erroneous testing, but because of poor marketing and commercial choices, according to a postmortem.
- With the promise of delivering unique downloading and family management capabilities, Disney mobile targeted youngsters and sports fans as clients.
- In Japan, the identical Disney smartphone app that flopped spectacularly in the United States found traction. This time, the major target clients were women in their 20s and 30s, rather than children

Conclusion

Introducing a new software product is like venturing into an unknown land with a wide range of options.

Many successful products are the result of their inventors identifying a market need and assessing the feasibility of a novel concept.

Comparison testing may be a useful method for determining a software product's potential.

It gives critical business inputs for promoting the software product and exposing flaws before it is released to the public

SHORT QUESTIONS

1.Define software metrics?

2.Define software models?

3.Explain walk throughs and inspections?

4.Define ISO:9001 Model?

5.Explain ISO comparative analysis?

LONG QUESTIONS

1.Explain in detail about software models and software metrics?

2.Define ISO:9001 Model and CMM Model?

3.Discuss about software configuration management?

4.Explain about walkthroughs and inspections?

5.Discuss about CMM and ISO comparative analysis?

Question papers for Sample:

S.V.U. COLLEGE OF COMMERCE MANAGEMENT AND COMPUTER SCIENCE :: TIRUPATHI

DEPARTMENT OF COMPUTER SCIENCE

Time:**2hours** **INTERNAL EXAMINATIONS -I** MAX MARKS:30

Section-A

Answer any five from the following 5*2=10m

1.Define the role of process in software quality?

2.Explain the overview of TMM?

3.Define software testing principles?

4.Explain origin of defects?

5.Define testcase design strategies?

6.Discuss about blackbox approach?

7.Define loop testing?

8.Explain integration test?

Section-B

Answer any one Question from each unit 2*10=20marks

UNIT-1

9(a).Explain in detail about the overview of the testing maturity model(TMM)?

b).Discuss about software testing principles?

(or)

10(a).Define the role of process in software quality?

b).Explain about origin of defects and defect repository?

UNIT-2

11(a).Define about testcase design strategies?

b).Explain about in detail Black Box approach?

(or)

12a).Discuss about random testing and boundary value analysis?

S.V.U COLLEGE OF COMMERCE MANAGEMENT AND COMPUTER SCIENCE :: TIRUPATHI

DEPARTMENT OF COMPUTER SCIENCE

TIME:2 Hours **INTERNAL EXAMINATIONS-2** Max.Marks:30

SECTION-A

Answer any five of the following 5*2=10M

1.Define levels of testing?

2.Explain unit test planning?

3.Define regression testing?

4.Explain test plan components?

5.Define software quality?

6.Discuss about SDLC?

7.Explain software metrics?

8.Define CMM model?

SECTION-B

Answer any one question from each unit 2*10=20M

UNIT-1

9a).Explain in detail about levels of testing?

b).Discuss about regression testing and alpha,beta and acceptance test?

(or)

10a).Differentiate between quality control and quality assurance?

b).Explain about test paln components and test palin attachments?

UNIT-2

11a).Explain about the quality assurance at each phase of SDLC?

b).Define software quality assurance plan and product quality?

(or)

12a).Explain about software metrics and software models?

b).Differentiate between the ISO:9001 Model and CMM model?

MASTER OF COMPUTER APPLICATIONS DEGREE EXAMINATION

FOURTH SEMESTER

Paper MCA 401C: SOFTWARE TESTING

(Under C.B.S.C Revised Regulations w.e.f.2021-2023)

(Common paper to University and all Affliated Colleges)

Time:3 hours Max.Marks:70

PART-A

(Compulsory)

Answer any five of the following questions each question carries 2 marks(5*2=10)

1a).Define TMM model?

b).Explain origin of defects?

c).Define loop testing?

d).Explain White box approach?

e).Define integration tset?

f).Discuss about levels of testing?

g).Define quality control?

h).Define SQA?

i).Explain software models?

j).Discuss about software configuaration management?

PART-B

Answer any ONE full question from each unit

Each question carries 12 Marks (5*12=60)

UNIT-1

2.Explain about test design and defect repository?

(or)

3.Discuss about the role of process in software quality?

UNIT-2

4.Explain test adequacy criteria and Mutation testing?

(or)

5.Define data flow and white box approach?

UNIT-3

6.Explain in detail about levels of testing in software?

(or)

7.Discuss about the system testing and reporting test results?

UNIT-4

8.Define quality assurance at each phase of SDLC?

(or)

9.Differentiate between the quality control and quality assurance?

UNIT-5

10.Explain software models and software metrics?

(or)

11.Discuss in detail about CMM model and ISO:9001 Model?