UNIT IV SOFTWARE TESTING AND MAINTENANCE

Testing – Unit testing – Black box testing– White box testing – Integration and System testing– Regression testing – Debugging - Program analysis – Symbolic execution – Model Checking-Case Study

---

Basic Terminologies involved in Testing

- **Testing**

Testing is the process of executing a program with the intent of finding faults. It also includes

- ✓ Testing is the process of demonstrating that errors are not present.
- ✓ The purpose of testing is to show that a program performs its intended functions correctly.
- ✓ Testing is the process of establishing confidence that a program does what it is supposed to do.

- **Verification and Validation**

Verification is related to static testing which is performed manually, only inspect and review the document.

Validation is dynamic in nature and requires the execution of the program**.**

- ✓ **Verification -** *"It is the process of evaluating the system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase."* Hence, it is the process of reviewing the requirement document, design document, source code and other related documents of the project.
- ✓ **Validation -** *"It is the process of evaluating a system or component during or at the end of development process to determine whether it satisfies the specified requirements."* It requires the actual execution of the program. It is dynamic testing and requires a computer for execution of the program.

Hence, testing includes both verification and validation. Thus

Testing = Verification + Validation

- **Testing Methods**

- ✓ ***Static Testing*** - refers to testing activities without executing the source code. All verification activities like inspections, walkthroughs, reviews, etc. come under this category of testing.

- ✓ ***Dynamic Testing*** - refers to executing the source code and seeing how it performs with specific inputs. All validation activities come in this category where execution of the program is essential.

Static Testing	**Dynamic Testing**
✓ It is also known as *Verification in Software Testing.* ✓ Verification is a *static method* of checking documents and files. ✓ Verification is the process, to ensure that whether *we are building the product right.*	✓ It is also known as *Validation in Software Testing.* ✓ Validation is a *dynamic process* of testing the real product. ✓ Validation is the process, whether *we are building the right product.*

- **Testing Approaches**

There are three types of software testing approaches.

- ✓ White Box Testing
- ✓ Black Box Testing
- ✓ Grey Box Testing

White Box Testing - *It is also called Glass Box, Clear Box, Structural Testing.*

- ✓ White Box Testing is based on the application's internal code structure.
- ✓ In white-box testing, an internal perspective of the system, as well as programming skills, are used to design test cases. This testing is usually done at the unit level.

Black Box Testing - *It is also called Behavioral/Specification-Based/Input-Output Testing.*

- ✓ Black Box Testing is a software testing method in which testers evaluate the functionality of the software under test without looking at the internal code structure.

Grey Box Testing - *Grey box is the combination of both White Box and Black Box Testing.*

- ✓ The tester who works on this type of testing needs to have access to design documents.
- ✓ This helps to create better test cases in this process.

- **Testing Levels**
  - ✓ Unit Testing
  - ✓ Integration Testing
  - ✓ System Testing
  - ✓ Acceptance Testing

Unit Testing

- ✓ A software testing where individual units or components of a software are tested.
- ✓ The purpose is to validate that each unit of the software code performs as expected.
- ✓ Unit Testing is done during the development (coding phase) of an application by the developers.

Integration testing

- ✓ The second level of the software testing process, following unit testing.

- ✓ Integration testing involves checking individual components or units of a software project to expose defects and problems to verify that they work together as designed.
- ✓ Integration testing runs unit tests on one or a few integrated modules to verify if the integrated modules work as designed or not.

System testing

- ✓ *System Testing means testing the system as a whole. All the modules/components are integrated in order to verify if the system works as expected or not.*
- ✓ It tests the entire system, seeing if the system works in harmony with all the integrated modules and components. System Testing is done after Integration Testing.
- ✓ This plays an important role in delivering a high-quality product.

Acceptance Testing

- ✓ It is a black-box testing process where the functionality is verified to ensure the software product meets the acceptance criteria.
- ✓ Acceptance testing is a software testing approach where the system is tested for acceptability. It's the last phase of the software testing process, and it's important before making the software available for actual use.
- ✓ After the system testing, fixed most of the bugs, and verified and closed them, it's time for acceptance testing.

- **Testing and Debugging**

- ✓ **Testing -** The purpose of testing is to find faults and find them as early as possible.
- ✓ **Debugging** - Finding any such fault, the process used to determine the cause of this fault and to remove it is known as debugging.

- **Debugging Concept**

Fault, Error, Bug and Failure

Bug

- ✓ Making an error during coding, is called a 'bug'. Hence, error / mistake / defect in coding is called a bug.

Fault

- ✓ A fault is the representation of an error where representation is the mode of expression such as data flow diagrams, ER diagrams, source code, use cases, etc.
- ✓ If fault is in the source code, call it a bug.

Failure

- ✓ A failure is the result of execution of a fault and is dynamic in nature. When the expected output does not match with the observed output, we experience a failure.
- ✓ The program has to execute for a failure to occur.
- ✓ A fault may lead to many failures.

Error

- ✓ An error is a mistake, misconception, or misunderstanding on the part of a software developer**.**
- ✓ Errors are generated due to wrong logic, syntax, or loop that can impact the end-user experience.
- ✓ It is calculated by differentiating between the expected results and the actual results

- **Test Case and Test Suite**

Test Case

- ✓ A test case consists of inputs given to the program and its expected outputs. Every test case will have a unique identification number.
- ✓ A good test case has a high probability of showing a failure condition. Hence, test case designers should identify weak areas of the program and design test cases accordingly.

Test Suite

- ✓ The set of test cases is called a test suite. (i.e.) test suite of all successful test cases and test suite of all unsuccessful test cases.
- ✓ Any combination of test cases will generate a test suite.

Software Testing

- ✓ Software testing is the process of confirming and assessing whether software performs as anticipated. Before the program is used, bugs, errors, or defects are found.
- ✓ To test a software application or product, run a program using fictitious data and check the findings for flaws, variations, or details about the software's non-functional characteristics.
- ✓ Teams can spot potential issues using software testing and fix them before they have an impact on real users.
- ✓ To guarantee the software's performance, effectiveness, and dependability, the QA professionals should test it using the appropriate software testing strategy.
- ✓ Software testing is a component of a broader procedure of verification and validation.

Principles of Software Testing

Test principles will help to draft error-catching test cases and create an effective Test Strategy.

Software testing is governed by the following seven principles:

- ✓ 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

Testing Shows the Presence of Defects

- ✓ As stated in this testing principle, "Testing talks about the presence of defects and doesn't talk about the absence of defects".
- ✓ Despite this, the testing process does not guarantee that the software is 100% error-free. It is true that testing greatly reduces the number of defects buried in software, however discovering and repairing these problems does not guarantee a bug-free product or system.
- ✓ Even if testers cannot find defects after repeating regression testing, it does not mean the software is 100 % bug-free.

Exhaustive Testing is Impossible

- ✓ Exhaustive testing usually tests and verifies all functionality of a software application while using both valid and invalid inputs and pre-conditions.
- ✓ No matter how hard you try, testing EVERYTHING is pretty much impossible.
- ✓ The inputs and outputs alone have an infinite number of combinations, so it is 100% not possible to test an application from every angle.

Early Testing

- ✓ In software development, early testing means incorporating testing as early as possible in the development process. It plays a critical role in the software development lifecycle (SDLC).
- ✓ For instance, testing the requirements before coding begins.
  - ✓ **Example:-** In the first case, an incorrect requirement in the requirement gathering phase. In the second case, you found a defect in a fully developed functionality. It is less expensive to fix the incorrect requirement than fully developed functionality that isn't working the way it should.
- ✓ Therefore, to improve software performance, software testing should begin at the initial phase, that is, during requirement analysis.

Defect Clustering

- ✓ In software testing, defect clustering refers to a small module or feature that has the most bugs or operation issues. This is because defects are not evenly distributed within a system but are clustered.

- ✓ ***Pareto Principle*** (80-20 Rule) states that 80% of issues originate from 20% of modules, while the remaining 20% originate from the remaining 80% of modules. Thus, we prioritize testing on 20% of modules where we experience 80% of bugs.

Pesticide Paradox

- ✓ In software testing, the Pesticide Paradox generally refers to the practice of repeating the exact same test cases over and over again. As time passes, these test cases will cease to find new bugs. Developers will create tests which are passing so they can forget about negative or edge cases.
- ✓ Therefore, in order to overcome the Pesticide Paradox, it is imperative to regularly review and update the test cases so that more defects can be found.

Testing is Context-Dependent

- ✓ Each type of software system is tested differently. According to this principle, testing depends on the context of the software developed, and this is entirely true.
- ✓ Various methodologies, techniques, and types of testing are used depending on the nature of an application. For example, health industry applications require more testing than gaming applications.

Absence of Error – Fallacy

- ✓ The software which we built not only must be 99% bug-free software but also it must fulfill the business, as well as user requirements otherwise it will become unusable software.
- ✓ Even bug-free software may still be unusable if incorrect requirements are incorporated into the software, or if the software fails to meet the business needs.

Software Testing Life Cycle

Software Testing Life Cycle (STLC) is a process used to test software and ensure that quality standards are met. Tests are carried out systematically over several phases. During product development, phases of the STLC may be performed multiple times until a product is deemed suitable for release.

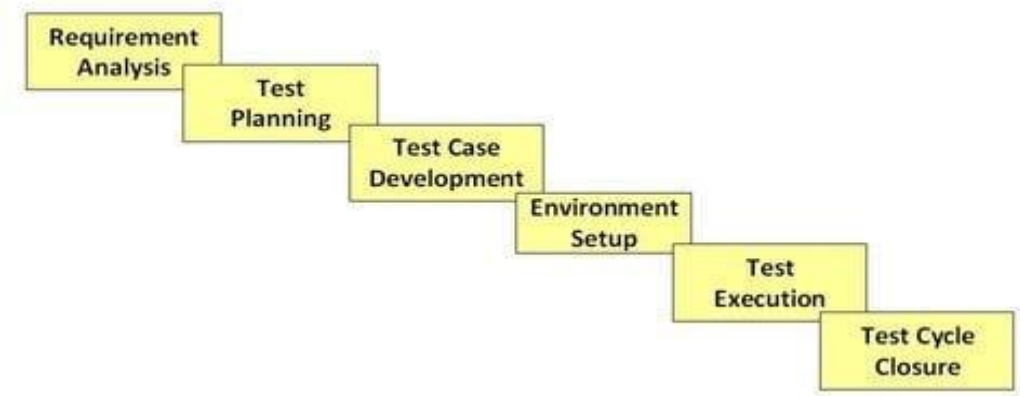
Entry and Exit Criteria in STLC

- ➢ **Entry Criteria:** Entry Criteria gives the prerequisite items that must be completed before testing can begin.
- ➢ **Exit Criteria:** Exit Criteria defines the items that must be completed before testing can be concluded

The 6 STLC Phases In-Depth

Software Testing Life Cycle consists of 6 phases, *each of these stages has a definite Entry and Exit criteria, Activities & Deliverables associated with it.*

Entry and exit criteria - define when a phase can start and when it can end.
Activities and deliverables - define what actions are performed and what the expected result is.



STLC Model Phases

1. Requirement Analysis
2. Test Planning
3. Test case development
4. Test Environment setup
5. Test Execution
6. Test Cycle closure

Requirement Analysis

In this phase, feature requirements collected in the SDLC process are evaluated to identify testable aspects. If necessary, testing teams may need to consult with stakeholders to clarify requirements. These requirements can either be functional or non-functional, defining what a feature can do or it's characteristics respectively. The ability to automate testing is also evaluated during this phase.

- ✓ ***Entry Criteria***—documented requirements, acceptance criteria, and intended product architecture.
- ✓ ***Exit Criteria***—approved requirement traceability matrix (RTM) and automation feasibility report.
- ✓ ***Activities***
  - ✓ Identify types of tests to be performed.
  - ✓ Gather details about testing priorities and focus.
  - ✓ Prepare Requirement Traceability Matrix (RTM).
- ✓ ***Deliverables***
  - ✓ RTM
  - ✓ Automation feasibility report

Test Planning

In this phase, the test strategy is outlined in a test plan document. This strategy includes tools needed, testing steps, and roles and responsibilities. Part of determining this strategy is a risk and cost analysis and an estimated timeline for testing.

- ✓ ***Entry Criteria***—requirement analysis, RTM, and automation feasibility report.
- ✓ ***Exit Criteria***—approved test plan including timelines and risk/cost analysis.
- ✓ ***Activities***
  - ✓ Preparation of test plan/strategy
  - ✓ Test effort estimation
  - ✓ Resource planning and determining roles and responsibilities.
- ✓ ***Deliverables***
  - ✓ Test plan /strategy document.
  - ✓ Effort estimation document.

Test Case Development

In this phase, test cases are created. Each case defines test inputs, procedures, execution conditions, and anticipated results. Test cases should be transparent, efficient, and adaptable. Once all test cases are created, test coverage should be 100%.

- ✓ ***Entry Criteria***—approved test plan including timelines and risk/cost analysis.
- ✓ ***Exit Criteria***—approved test cases and automation scripts.
- ✓ ***Activities***
  - ✓ Create test cases, automation scripts (if applicable)
  - ✓ Review and baseline test cases and scripts
  - ✓ Create test data
- ✓ ***Deliverables***
  - ✓ Test cases/scripts and Test data

Test Environment Setup

In this phase, testing environments are configured and deployed. This phase may include a variety of testing tools, including TestComplete, Selenium, Appium, or Katalon Studio. Sometimes, this phase also includes setting up test servers. Once environments are deployed, smoke tests are performed to ensure that environments are working as expected with all intended functionality.

- ✓ ***Entry Criteria***: system design and project architecture definitions.
- ✓ ***Exit Criteria***: a fully functional test environment and approved test cases.
- ✓ ***Activities***
  - ✓ Understand the required architecture, environment set-up and prepare hardware/software requirement list for the Test Environment.
  - ✓ Setup test Environment and test data
  - ✓ Perform smoke test on the build

- ✓ ***Deliverables***
  - ✓ Environment ready with test data set up
  - ✓ Smoke Test Results.

Test Execution

The features are tested in the deployed environment, using the established test cases. Expected test results are compared to actual and results are gathered to report back to development teams.

- ✓ ***Entry Criteria***—all exit criteria from previous steps.
- ✓ ***Exit Criteria***—all tests are performed and results are documented.
- ✓ ***Activities***
  - ✓ Execute tests as per plan
  - ✓ Document test results, and log defects for failed cases
  - ✓ Retest the Defect fixes and Track the defects to closure
- ✓ ***Deliverables***
  - ✓ Test cases updated with results
  - ✓ Defect reports

Test Cycle Closure

This is the last phase of the STLC, during which a test result report is prepared. This report should summarize the entire testing process and provide comparisons between expected results and actual. These comparisons include objectives met, time taken, total costs, test coverage, and any defects found.

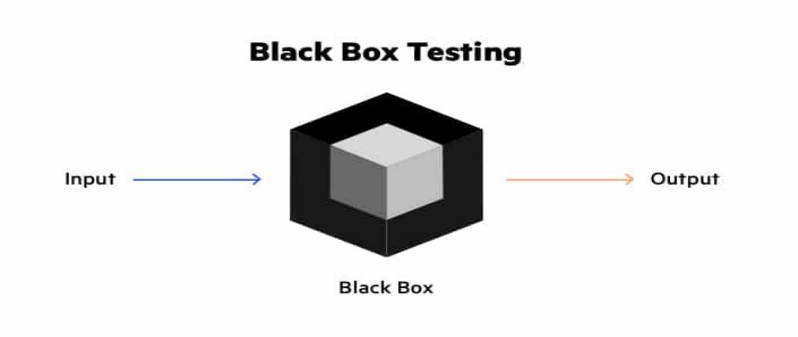
- ✓ ***Entry Criteria***—test results and logging from all previous phases.
- ✓ ***Exit Criteria***—delivered and approved test closure report.
- ✓ ***Activities***
  - ✓ Evaluate cycle completion criteria based on Time, Test coverage, Cost, Software, Critical Business Objectives, Quality
  - ✓ Prepare Test metrics and Test closure report
  - ✓ Qualitative and quantitative reporting of the work product to the customer.
  - ✓ Test result analysis
- ✓ ***Deliverables***
  - ✓ Test Closure report and Test metrics

Black Box Testing

- ✓ Black box testing involves testing a system with no prior knowledge of its internal workings. A tester provides an input, and observes the output generated by the system under test.
- ✓ This makes it possible to identify how the system responds to expected and unexpected user actions, its response time, usability issues and reliability issues.

✓ Black box testing is a powerful testing technique because it exercises a system end-to-end.

Black box testing checks that the system as a whole is working as expected.



Black Box Testing Techniques

- **Equivalence Partitioning**

✓ Testers can divide possible inputs into groups or "partitions", and test only one example input from each group.

✓ For example, if a system requires a user's birth date and provides the same response for all users under the age of 18, and a different response for users over 18, it is sufficient for testers to check one birth date in the "under 18" group and one date in the "over 18" group.

- **Decision Table Testing**

✓ Many systems provide outputs based on a set of conditions. Testers can then identify "rules" which are a combination of conditions, identify the outcome of each rule, and design a test case for each rule.

✓ For example, a health insurance company may provide different premium based on the age of the insured person (under 40 or over 40) and whether they are a smoker or not. This generates a decision table with four rules and up to four outcomes—below is an example with three possible outcomes.

Conditions	**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**
Under 40	False	False	True	True
Smoker	False	True	False	True

- **State Transition Testing**

✓ In some systems, significant responses are generated when the system transitions from one state to another.

✓ A common example is a login mechanism which allows users to authenticate, but after a specific number of login attempts, transition to a different state, locking the account.

- **Error Guessing**

✓ This technique involves testing for common mistakes developers make when building similar systems.

✓ For example, testers can check if the developer handled null values in a field, text in a numeric field or numbers in a text-only field, and sanitization of inputs—whether it is possible to submit user inputs that contain executable code, which has security significance.

✓ A specific type of error guessing is testing for known software vulnerabilities that can affect the system under test.

Black Box Testing Pros and Cons

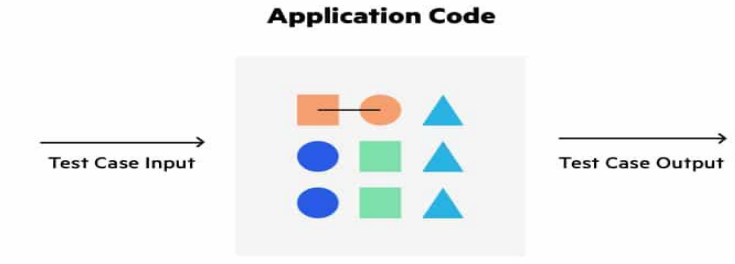
Pros	**Cons**
✓ Testers do not require technical knowledge, programming or IT skills. ✓ Testers do not need to learn implementation details of the system. ✓ Tests can be executed by crowd sourced or outsourced testers. ✓ Low chance of false positives. ✓ Tests have lower complexity, since they simply model common user behavior.	✓ Difficult to automate. ✓ Requires prioritization, typically infeasible to test all user paths. ✓ Difficult to calculate test coverage. ✓ If a test fails, it can be difficult to understand the root cause of the issue. ✓ Tests may be conducted at low scale or on a non-production-like environment.

White Box Testing

✓ White box testing is an approach that allows testers to inspect and verify the inner workings of a software system—its code, infrastructure, and integrations with external systems.

✓ White box testing provides inputs and examines outputs, considering the inner workings of the code.

White box testing can uncover structural problems, hidden errors and problems with specific components.



Process of performing White Box Testing

The process of performing White box Testing is divided into the following steps;

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.

Step 2 - Creating and Executing Test Cases

- ✓ The second step includes the real development of test cases based on Statement/Decision/Condition/Branch coverage, as well as the actual execution of test cases to ensure that the software has been tested completely.
  - ➢ ***Statement coverage*** will include those statements that are executed at least once during the execution of the program.
  - ➢ ***Branch coverage*** will include the outcome for every code module (statement or loop).
  - ➢ ***Conditional coverage*** is used to test the variables used in different types of conditional statements like IF / ELSE, SWITCH etc.
  - ➢ ***Decision coverage*** will include reports for each Boolean expression present in the source code. An expression is said to be Boolean if it evaluates to either TRUE or FALSE.

Testing Techniques

Code Coverage

- ✓ One of the main goals of white box testing is to cover the source code as comprehensively as possible. Code coverage is a metric that shows how much of an application's code has unit tests checking its functionality.
- ✓ Within code coverage, it is possible to verify how much of an application's logic is actually executed and tested by the unit test suite, using concepts like statement coverage, branch coverage, and path coverage. These concepts are discussed in more detail below.

Statement Coverage

- ✓ Statement coverage is a white box testing technique that ensures all executable statements in the code are run and tested at least once.
- ✓ For example, if there are several conditions in a block of code, each of which is used for a certain range of inputs, the test should execute each and every range of inputs, to ensure all lines of code are actually executed.

Branch Coverage

- ✓ Branch coverage maps the code into branches of conditional logic, and ensures that each and every branch is covered by unit tests.
- ✓ In a branch coverage approach, the tester identifies all conditional and unconditional branches and writes code to execute as many branches as possible.
- ✓ For example, if there are several nested conditional statements:

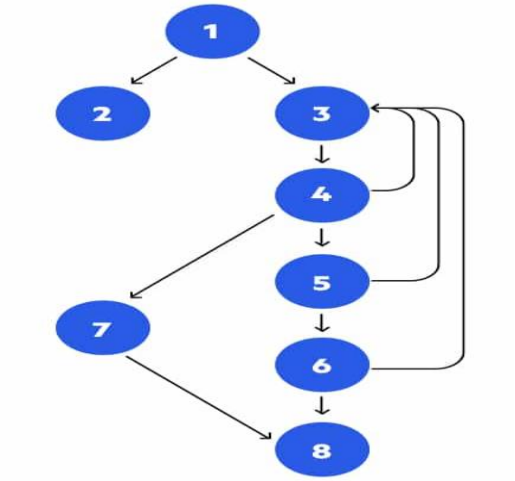
if X then..
if Y then..
A
B
else
if Z then..
C
else..
D

A, C, and D are **conditional branches**, because they occur only if a condition is satisfied.

B is an **unconditional branch**, because it is always executed after A.

Path Coverage

- ✓ Path coverage is concerned with linearly independent paths through the code.
- ✓ Control flow diagram used to design tests in a path coverage approach.
- ✓ Testers draw a control flow diagram of the code, such as the example below.



In this example, there are several possible paths through the code:

- 1, 2
- 1, 3, 4, 5, 6, 8
- 1, 3, 4, 7, 6, 8 ,….etc.

In a path coverage approach, the tester writers unit tests to execute as many as possible of the paths through the program's control flow. The objective is to identify paths that are broken, redundant, or inefficient

White Box Testing Pros and Cons

Pros	Cons
✓ Ability to achieve complete code coverage ✓ Easy to automate ✓ Reduces communication overhead between testers and developers ✓ Reduces communication overhead between testers and developers	✓ Requires a large effort to automate ✓ Sensitive to changes in code base, automation requires expensive maintenance ✓ Cannot test expected functionality that does not exist in the codebase ✓ Cannot test from the user's perspective

---

Boundary Value Analysis & Equivalence Class Partitioning

Black Box Test Design Technique

- ✓ Black Box Test Design is defined as a testing technique in which functionality of the Application Under Test (AUT) is tested without looking at the internal code structure, implementation details and knowledge of internal paths of the software.
- ✓ This type of testing is based entirely on software requirements and specifications.
- ✓ In Black Box Testing, it focuses on input and output of the software system without the knowledge of inner working of the software.
- ✓ By using this technique, a lot of testing time and get good test case coverage could be saved.

Test Techniques are generally categorized into:

- ✓ *Boundary Value Analysis (BVA)*
- ✓ *Equivalence Class Partitioning (ECP)*
- ✓ *Decision Table based testing*
- ✓ *State Transition*
- ✓ *Error Guessing*

Boundary Value Analysis (BVA)

- ✓ BVA is another Black Box Test Design Technique, which is used to find the errors at boundaries of input domain (tests the behavior of a program at the input boundaries) rather than finding those errors in the centre of input.
- ✓ The boundary value testing is to select input variable values at their:
    - Minimum,
    - Just above the minimum,
    - Just below the minimum,
    - A nominal value,
    - Just below the maximum,

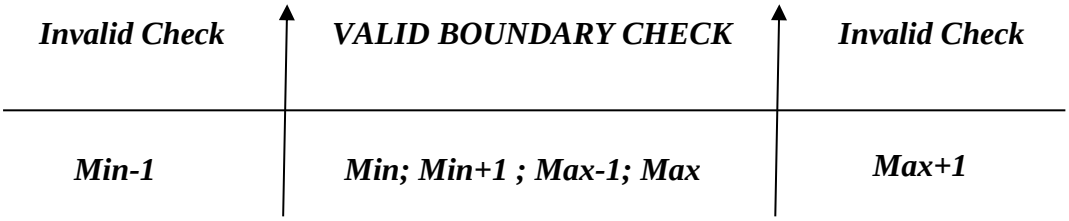
- Maximum and
- Just above the maximum.

✓ That is, for each range, there are two boundaries, ***the lower boundary*** (start of the range) and ***the upper boundary*** (end of the range) and ***the boundaries are the beginning and end of each valid partition.***

✓ Test cases are designed based on the program functionality at the boundaries, and ***with values just inside and outside the boundaries***.

✓ Boundary value analysis is also a part of stress and negative testing.

Boundary Value Analysis



Example of BVA

✓ A developer writes code for an amount text field which will accept and transfer values only from 100 to 5000.

✓ The valid and invalid test cases are listed below.

Boundary Value Analysis		
Invalid Boundary Check	***Valid Boundary Check***	***Invalid Boundary Check***
(Min-1) 99	***(Min; Min+1 ; Max-1; Max)*** 100, 101, 4999, 5000	***(Max+1)*** 5001

Invalid Test Cases

✓ Enter the value 99 which is min-1 value.

✓ Enter the value 5001 which is max+1 value

Valid Test Cases

✓ Enter the value 100 which is min value.

✓ Enter the value 101 which is min+1 value.

✓ Enter the value 4999 which is max-1 value.

✓ Enter the value 5000 which is max value.

Equivalence Class Partitioning (ECP)

- ✓ Equivalence partitioning is also known as "Equivalence Class Partitioning".
- ✓ In this method, the input domain data is divided into different equivalence data classes – which are generally termed as ***'Valid' and 'Invalid'***.
- ✓ The inputs to the software or system are divided into groups that are expected to exhibit similar behavior.
- ✓ Thus, ***it reduces the number of test cases to a finite list of testable test cases covering maximum possibilities.***
- ✓ During test case execution, each value of ***every equivalence partition must display the same output behavior as the other one***.

- ✓ That is, if one of the *partition's conditions pass/valid*, *the entire partition's condition/value will pass/valid*.
- ✓ Similarly, if one of the *partition's conditions fails/invalid, the partition's condition/value will likewise fail/invalid.*

Partition's Conditions in EVP	Output
For ***all Valid*** inputs	**Pass/valid** Output
For ***all invalid*** inputs	**Fail/invalid** Output
For ***both valid and invalid input*** in the same partition	**Pass/valid** Output

Equivalence Partitioning has been categorized into two parts:

- ✓ Pressman Rule
- ✓ Practice Method

1. Pressman Rule:

- ✓ ***Rule 1:*** If input is a ***range of values,*** then design test cases ***for one valid and two invalid values.***
- ✓ ***Rule 2:*** If input is a ***set of values***, then design test cases ***for all valid value sets and two invalid values.***
- ✓ ***Rule 3:*** If input is **Boolean,** then design test cases ***for both true and false values***.

Rules	***Input Values***	***Test Cases Designed***
Rules 1	***Range Of Values*** *Eg: Age 0-9* *Age 13-19, 30-40* *(i.e Particular ranges)*	*For **1 Valid** Values* *For **2 Invalid** Values*
Rule 2	***Set Of Values*** *Eg: Roll No: 101 to 150* *Roll No: 201to 250*	*For **All Valid** Values* *For **2 Invalid** Values*

	(i.e Continues number)	
Rule 3	***Boolean*** *Eg: Login success – True* *Login Failed - False*	*For **True** Values* *For **False** Values*

2. Practice Method

- ✓ If the input is a range of values, then divide the range into equivalent parts.
- ✓ Then test for all the valid values and ensure that 2 invalid values are being tested for.

Rules	***Input Values***	***Test Cases Designed***
Rules 1	*range of values, then divide that into equivalent parts*	*For **all valid** values* *For **2 invalid** values*

Example of ECP

The password of length 8-12 numbers (min 8 and max 12 numbers)
Let's consider some password values for valid and invalid class

1. 1234 is of length 4 which is an invalid password as 4<8.
2. 567890234 is of length 9 which is a valid password as 9 lies between 8-12
3. 4536278654329 is of length 13 which is an invalid password as 13>12.

Invalid Equivalence Class	**Valid Equivalence Class**	**Invalid Equivalence Class**
<8	8-12	>12

Advantages and Disadvantages of EVP and BVA

Advantages/Benefits of EVP and BVA

- ✓ These techniques are *used to break down a huge number of test cases into smaller, more manageable test data sets.*
- ✓ These techniques will further help in *saving time and resources.*
- ✓ These techniques facilitate *increasing test efficiency and effectiveness by testing with the most likely and critical input data values,* which are likely to detect more bugs in the application.
- ✓ Test design and execution are more simplified, as these *techniques provide a systematic and logical approach* to the use of the input domain.
- ✓ It *enables easier identification and selection of test cases with less complexity and ambiguity.*
- ✓ Testing of applications that *consist of numerous calculations or those with lots of combinations of input data* is supported by using these testing techniques.

Disadvantages/Challenges of Equivalence Partitioning

- ✓ One of the main challenges of using equivalence class testing is that it *requires a very good understanding of the system specifications (requirements), functionality, and the expected behavior of the application under test*.
- ✓ It also requires *good domain knowledge*.
- ✓ It is essential to create a formal method to specify equivalence classes, which *can be quite difficult for large and complex features.*
- ✓ This technique is most appropriate while creating new test cases/test data sets from scratch, as *converting existing test cases could be time-consuming and difficult.*
- ✓ Equivalence class partitioning *may not be suitable for non-functional testing, such as performance, stress, or usability testing,* where different variations of the system are to be tested.

Disadvantages/Challenges of Boundary Value Analysis

- ✓ This testing technique may not be useful for *testing input domains that are complex and have multiple boundaries, interactions, or dependencies.*
- ✓ It may necessitate some assumptions about the input data domain, *which may or may not be accurate or consistent.*
- ✓ As with equivalence partitioning, the boundary value analysis technique also requires a proper understanding of system specifications.
- ✓ Many times, system requirements *are not quite clear or the quality and completeness of the specifications are not good enough*.
- ✓ The boundary value analysis technique is *not suitable for Boolean variables, Boolean data type has values 0 and 1*.

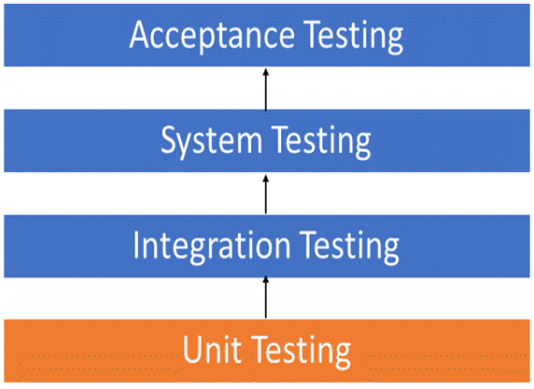
Boundary Value Analysis Vs Equivalence Partitioning

	Boundary Value Analysis	**Equivalence Partitioning**
Uses	BVA considers the input data values from the defined boundaries.	ECP examines input data values from the range of equivalence class intervals.
Testing Values	It considers min+1, min, min-1, nominal, max+1, max, and max-1 values as input test data values.	It considers valid and invalid ranges of equivalence classes are taken for testing developed applications.

	Boundary Value Analysis	Equivalence Partitioning
Bug Identification	It identifies bugs at boundary values only.	It helps in identifying bugs in-between the partitioned equivalence data class.
Application Areas	It is a part of stress and negative testing.	It can be performed at any stage of software testing like unit testing.
Usage Condition	It is restricted to applications with close boundary values.	Correctness of Equivalence Partitioning is dependent on how correctly the tester identifies equivalence class.

Unit Testing

- ✓ **Unit Testing** is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected.
- ✓ Unit Testing is done during the development (coding phase) of an application by the developers.
- ✓ Unit Tests isolate a section of code and verify its correctness.
- ✓ A unit may be an individual function, method, procedure, module, or object.



The key reasons to perform unit testing

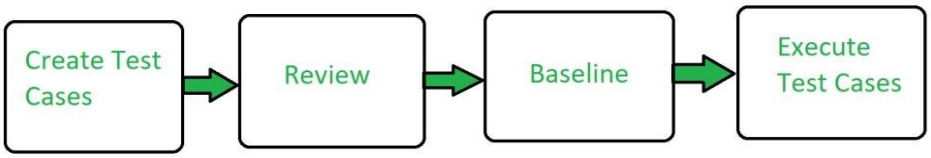
- ✓ Unit tests help to fix bugs early in the development cycle and save costs.
- ✓ It helps the developers to understand the testing code base and enables them to make changes quickly
- ✓ Good unit tests serve as project documentation

- ✓ Unit tests help with code re-use. Migrate both the code **and** tests to that new project. Tweak the code until the tests run again.

The workflow of Unit Testing

The workflow of Unit Testing is *1) Create Test Cases 2) Review/Rework 3) Baseline 4) Execute Test Cases.*

- ✓ **Review/Rework** - a review is a process of examining a software product or component by a group of individuals to detect defects, errors, or vulnerabilities.
- ✓ **Baseline** - Baseline testing refers to the validation of the documents and specifications on which test cases are designed. Baseline, in general, refers to a benchmark that forms the base of any new creation.



An effective unit test must fulfill certain features, which are

- ✓ The individual tests must be independent of each other. The order of execution does not matter and has no influence on the entire test result. Also, if one test fails, the failed test does not have an influence on other unit tests.
- ✓ Each unit test focuses on exactly one property of the code.
- ✓ Unit tests must be completely automated. Usually, the tests are executed frequently to check for regression errors.
- ✓ The tests must be easy to understand to the developers and collaborators who are working on the same code.
- ✓ The quality of the unit tests must be as high as the quality of the production code.

Unit Testing Techniques

There are 3 types of Unit Testing Techniques. They are

1. **Black Box Testing** - This testing technique is used in covering the unit tests for input, user interface, and output parts.
2. **White Box Testing** - This technique is used in testing the functional behavior of the system by giving the input and checking the functionality output including the internal design structure and code of the modules.
3. **Gray Box Testing -** This technique is used in executing the relevant test cases, test methods, test functions, and analyzing the code performance for the modules.

Benefits of Unit Testing

- ✓ Developers receive fast feedback on code quality through regular execution of unit tests.

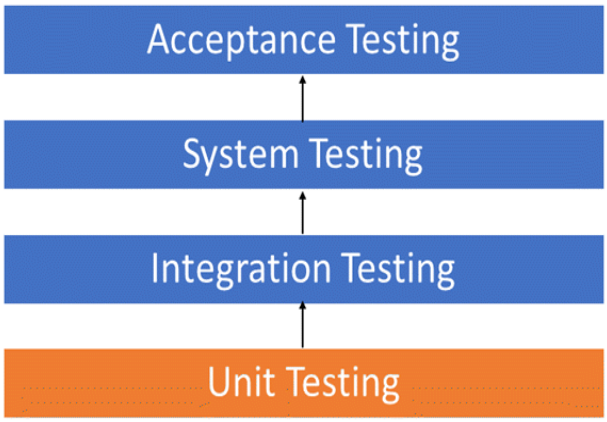
- ✓ Unit tests force developers to work on the code instead of just writing it. In other words, the developer must constantly rethink their own methodology and optimize the written code after receiving feedback from the unit test.
- ✓ Unit tests enable high test coverage.
- ✓ It's possible to perform automated, high-quality testing of the entire software unit by unit with speed and accuracy.

Disadvantages of Unit Testing:

- ✓ The process is time-consuming for writing the unit test cases.
- ✓ It requires more time for maintenance when the source code is changed frequently.
- ✓ It cannot cover the non-functional testing parameters such as scalability, the performance of the system, etc.
- ✓ Difficulty in Testing Complex Units
- ✓ Difficulty in Testing Interactions

Integration Testing

- ✓ Integration testing is known as the second level of the software testing process, following unit testing. Integration testing involves checking individual components or units of a software project to expose defects and problems to verify that they work together as designed.
- ✓ The main function or goal of this testing is to test the interfaces between the units/modules.
- ✓ The individual modules are first tested in isolation. Once the modules are unit tested, they are integrated one by one, till all the modules are integrated, to check the combinational behavior, and validate whether the requirements are implemented correctly or not.



Example of Integration Test Case

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.

Check its integration to the Delete Mails Module.

Test Case ID	Test Case Objective	Test Case Description	Expected Result
1	Check the interface link between the Login and Mailbox module	Enter login credentials and click on the Login button	To be directed to the Mail Box
2	Check the interface link between the Mailbox and Delete Mails Module	From Mailbox select the email and click a delete button	Selected email should appear in the Deleted/Trash folder

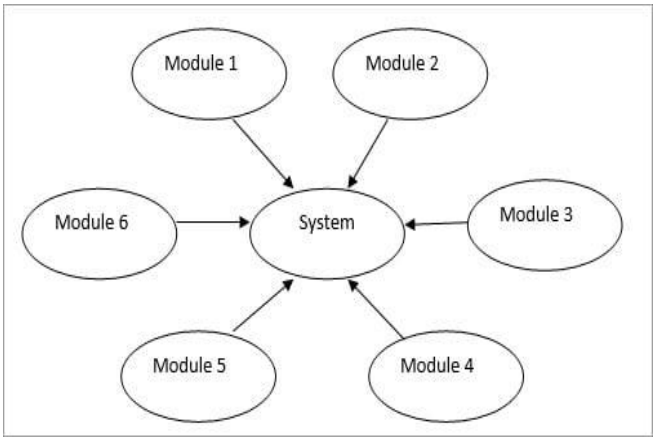
Types of Integration Testing

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

1. Big Bang Approach
2. Incremental Approach
   a) Top-Down Approach
   b) Bottom-Up Approach
3. Sandwich Approach – Combination of Top Down and Bottom Up

Big Bang Testing

- ✓ **Big Bang Testing** is an Integration testing approach in which all the components or modules are integrated together at once and then tested as a unit.
- ✓ This combined set of components is considered as an entity while testing. If all of the components in the unit are not completed, the integration process will not execute.



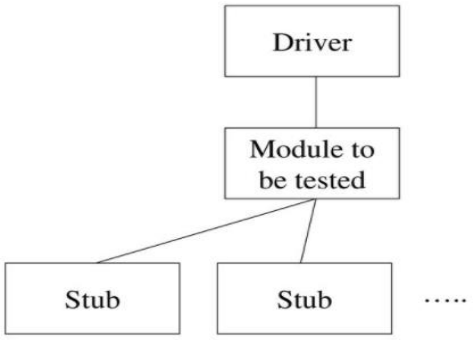
Incremental Testing

- ✓ In the **Incremental Testing** approach, testing is done by integrating two or more modules that are logically related to each other and then tested for proper functioning of the application.

- ✓ Then the other related modules are integrated incrementally and the process continues until all the logically related modules are integrated and tested successfully.
- ✓ Incremental Approach, in turn, is carried out by two different Methods:
    - Bottom Up
    - Top Down

Stubs and Drivers - Stubs and Drivers *are the dummy programs in Integration testing used to facilitate the software testing activity. These programs act as a substitute for the missing models in the testing.*

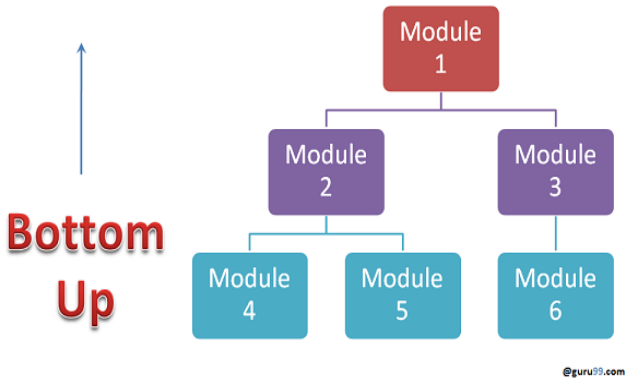
- ✓ **Stub**: Is called by the Module under Test.
- ✓ **Driver**: Calls the Module to be tested.



Stubs	Driver
✓ Used in Top down approach	✓ Used in Bottom up approach
✓ Top most module is tested first	✓ Lowest modules are tested first.
✓ Stimulates the lower level of components	✓ Stimulates the higher level of components
✓ Dummy program of lower level components	✓ Dummy program for Higher level component

Bottom-up Integration Testing

- ✓ **Bottom-up Integration Testing** is a strategy in which the lower level modules are tested first. These tested modules are then further used to facilitate the testing of higher level modules.
- ✓ The process continues until all modules at top level are tested. Once the lower level modules are tested and integrated, then the next level of modules are formed.



Advantages:

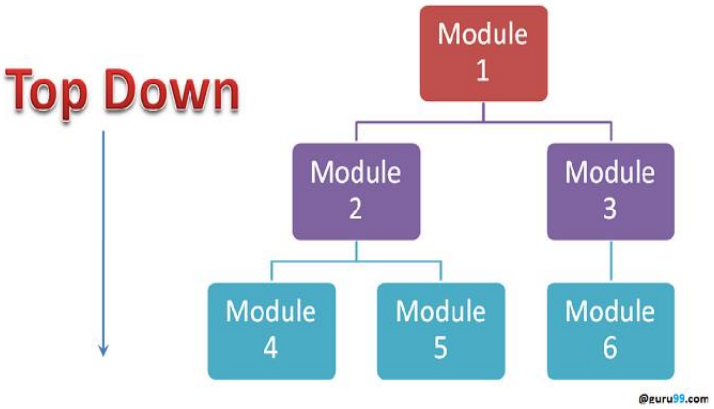
- ✓ Fault localization is easier.
- ✓ No time is wasted waiting for all modules to be developed unlike Big-bang approach

Disadvantages:

- ✓ Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.
- ✓ An early prototype is not possible

Top-down Integration Testing

- ✓ **Top-Down Integration Testing** is a method in which integration testing takes place from top to bottom following the control flow of software system.
- ✓ The higher-level modules are tested first and then lower-level modules are tested and integrated in order to check the software functionality. Stubs are used for testing if some modules are not ready.



Advantages:

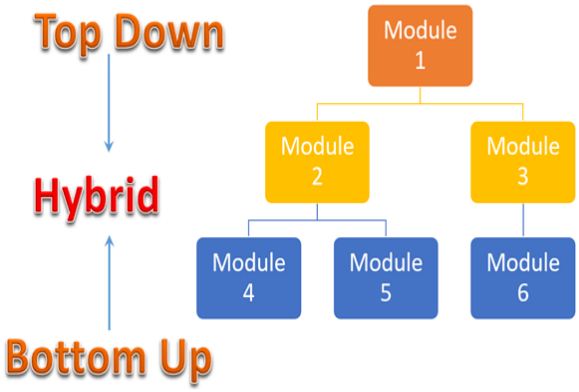
- ✓ Fault Localization is easier.
- ✓ Possibility to obtain an early prototype.
- ✓ Critical Modules are tested on priority; major design flaws could be found and fixed first.

Disadvantages:

- ✓ Needs many Stubs.
- ✓ Modules at a lower level are tested inadequately.

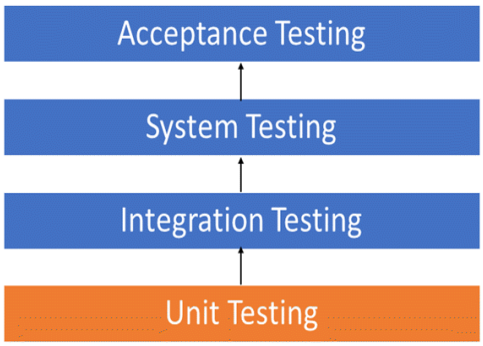
Sandwich Testing

- ✓ **Sandwich Testing** is a strategy in which top level modules are tested with lower level modules at the same time lower modules are integrated with top modules and tested as a system.
- ✓ It is a combination of Top-down and Bottom-up approaches therefore it is called **Hybrid Integration Testing**. It makes use of both stubs as well as drivers.



System Testing

- ✓ System testing, also referred to as *system-level testing* or *system integration testing*, is the process in which a quality assurance (QA) team evaluates how the various components of an application interact together in the full, integrated system or application.
- ✓ System testing verifies that an application performs tasks as designed. It's a type of black box testing that focuses on the functionality of an application rather than the inner workings of a system, which white box testing is concerned with.



- ✓ System testing is the third level of testing in the software development process. It's typically performed before acceptance testing and after integration testing.

Importance of system testing

- ✓ **Improved product quality** - A comprehensive system testing process ultimately boosts the product quality.

- ✓ **Error reduction -** System testing verifies a system's code and functionality against its requirements, so errors that aren't detected during integration and unit testing can be exposed during system testing.
- ✓ **Cost savings -** It can be more time-consuming to fix a system defect that's detected later in the project lifecycle.
- ✓ **Security -** Well-tested products are reliable. They ensure that the tested system doesn't contain vulnerabilities.
- ✓ **Customer satisfaction -** System testing offers visibility into the stability of a product at every stage of development. This builds customer confidence and improves the overall user experience.
- ✓ **Easier code modification -** System testing can identify code problems during software development.
- ✓ **Software performance -** Performance-based system tests can help understand changes in a system's performance and behavior, such as memory consumption, central processing unit utilization and latency.

System Testing Process

- ✓ **Test Planning**

The test plan is a document that includes the information of the testing, like objectives, strategies, entry-exit criteria, software requirements, testing tools, guidelines, etc.

- ✓ **Test Case Design and Execution**

Create a test case for every feature with the test scenarios, description, process, and more. Then perform the tests and execute them.

- ✓ **Defect Tracking and Defect Management**

Track and record the defects with the defect tracking tools like- JIRA, Bugzilla, Trello, etc. Also, write about them for the developers to resolve them.

- ✓ **Reporting and Communication**

Create bug reports about defect reports to send to the developers. Try to follow an organizational method for this.

System Testing Types

A. Functional Testing

1. Unit Testing

This testing process checks whether each system unit or component works accordingly. This is the first testing step performed at the development stage, and it helps to fix the bugs before the development cycle.

2. Integration Testing

This testing aims to validate the different software units working together to achieve the expected functionality. Thus it establishes a steady communication between several components and subsystems of software.

3. Regression Testing

It would be best to modify the software every time as per needs. So, regression testing ensures the system is working without default after very few changes are made, and the changes should not damage the system.

Teams will independently perform regression testing as part of SDLC

4. User Acceptance Testing

Generally, the end users perform this testing to verify the system meets the ultimate expectations. So, it goes through the users' perspective. This testing is done at the last stage of the SDLC.

B. Non-Functional Testing

1. Performance Testing

It checks the system's behavior instead of a certain workload. So it clarifies the system's speed, stability, responsiveness, memory usage, network usage, and more. It determines the system doesn't break when multiple users try to access it.

2. Load Testing

Load Testing is a type of software Testing which is carried out to determine the behavior of a system or software product under extreme load.

3. Stress Testing

Stress Testing is a type of software testing performed to check the robustness of the system under the varying loads.

4. Usability Testing

Simply, usability testing checks the product's usability. So, it aims to test whether or not the system is easy to use. This test ensures the application has a 'user manual' or 'help menu' to simplify the product's usability to the end users.

5. Compatibility Testing

This testing examines the compatibility of a system with different browsers, platforms, operating systems, hardware, network, mobile devices, etc. So, it ensures the system can behave accurately with the variable models and devices.

Regression Testing

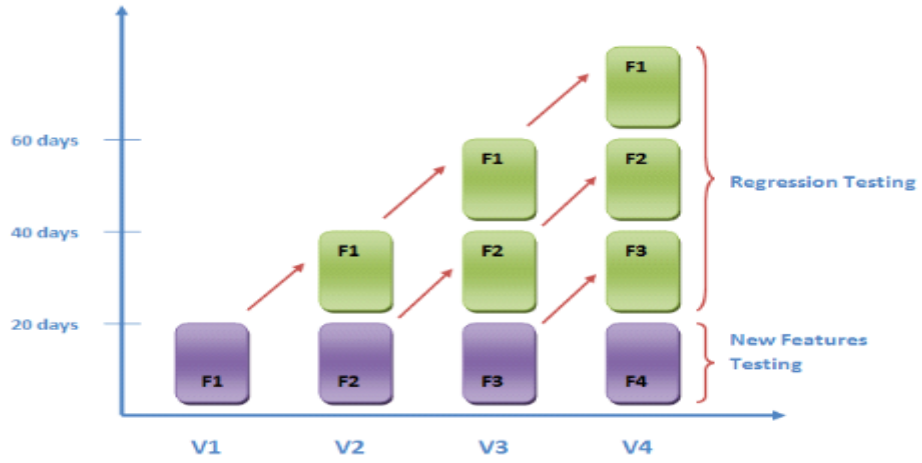
- ✓ Regression testing is a type of software testing conducted after a code update to ensure that the update introduced no new bugs. This is because new code may bring in new logic that conflicts with the existing code, leading to defects.
- ✓ Regression testing addresses a common issue that developers face (i.e.) the emergence of old bugs with the introduction of new changes.

Importance of Regression Testing

- ✓ It involves writing a test for a known bug and re-running this test after every change to the code base. This aims to immediately identify any change that reintroduces a bug.

- ✓ With a push for agility in software development, there is an emphasis on adopting an iterative process – push new code often and break things if necessary.
- ✓ Regression testing ensures that with frequent pushes, developers do not break things that already work.

Applying Regression Testing



Regression testing is applied under these circumstances:

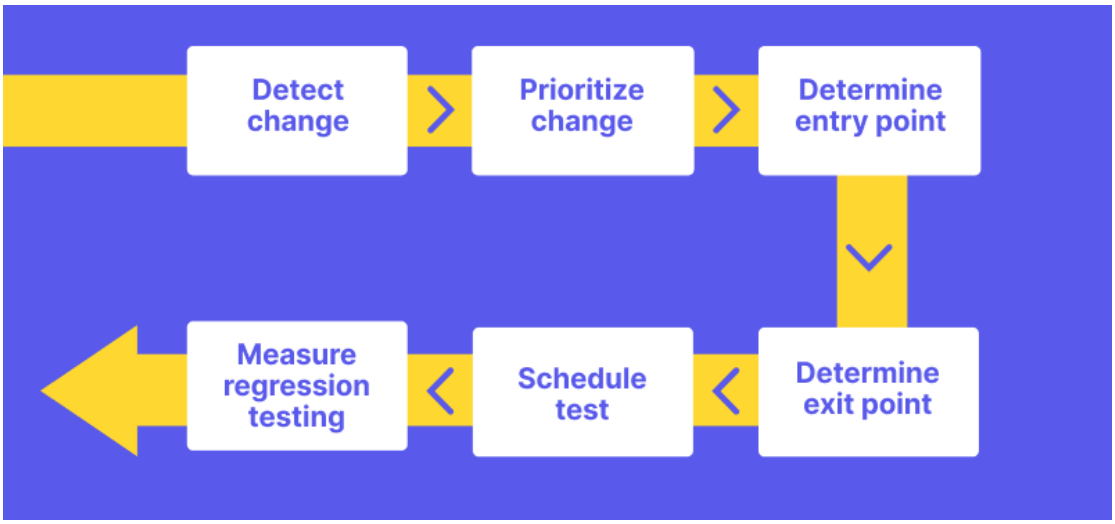
- ✓ A new requirement is added to an existing feature
- ✓ A new feature or functionality is added
- ✓ The codebase is fixed to solve defects
- ✓ The source code is optimized to improve performance
- ✓ Patch fixes are added
- ✓ A new version of the software is released
- ✓ When changes to the User Interface are made
- ✓ Changes in configuration
- ✓ A new third-party system is integrated with the current system

Regression Testing Techniques

- ✓ **Unit Regression Testing** approach uses a bird's-eye view philosophy to test code. This is a simple method in which the tester has a list of items to test every time a change occurs. This is the best way to start regression testing in an existing project.
- ✓ **Partial Regression Testing** approach divides the project into logical, coherent units that work together to form the whole application. Select the units that are most critical to the application and define specific test cases for them while performing unit regression testing for the rest of the modules.
- ✓ **Complete Regression Testing**, which is the most detailed form of regression testing. In this case, one takes a comprehensive view of the codebase to identify all functionalities that would affect usability on breaking and write detailed tests for

each of them. This technique is time-consuming but highly beneficial if applied from the early stages of project development.

Steps involved in Regression Testing



Detect source code changes

- ✓ Identifying the impact and risk of the latest code change is key to building a solid regression test.
- ✓ Conduct code review sessions to identify the components or modules that were changed, as well as their impacts on the existing features.

Prioritize impacted areas and test cases

- ✓ Modifications that have an impact on core features, or those that significantly alter how the application works, should always be the top priority.
- ✓ Defining priorities heightens in importance when the size of the codebase is bigger. The number of tests and time they take to complete could take up to months or an entire sprint.

Determine Entry Point and Exit Point

- ✓ Once the team agrees on which changes should be examined, they can select the tests that need to be performed.
- ✓ Generally, testing teams have a range of test suites ready to be executed, but they only have to execute only the relevant ones in each regression testing session.

Categorize regression test cases

- ✓ evaluate test cases based on urgency and significance and assign them a value of importance (high, medium, low). Other factors can also be brought into consideration based on business requirements.

Test Environment Preparation

- ✓ Having test environments at hand at all times is important for frequent regression testing. As new code is being developed constantly, environments need to be stable and ready-to-test to not interfere with the planned testing schedule.
- ✓ In additional, poor environment setup can give tests more reason to fail, missed defects and false positive/negatives.

Schedule And Execute Tests

- ✓ At this stage, all test cases are ready for execution. Teams can schedule test cases to run based on the plan. Certain test cases can even be scheduled to run periodically in intervals over the entire development cycle.
- ✓ Time-based test execution allows teams to have greater quality control over the constant changes of their application.

Measure regression testing success

- ✓ This stage gives important insights for future test runs. Analytics allows QA managers and other key stakeholders to quantify their testing efficiency and make data-driven decisions.
- ✓ Test reports can reveal weak points in the application for in-time adjustments for the development team.

Advantages

- ✓ It ensures that no new bugs have been introduced after adding new functionalities to the system.
- ✓ As most of the test cases used in Regression Testing are selected from the existing test suite, and we already know their expected outputs. Hence, it can be easily automated by the automated tools.
- ✓ It helps to maintain the quality of the source code.

Disadvantages

- ✓ It can be time and resource-consuming if automated tools are not used.
- ✓ It is required even after very small changes in the code.

Acceptance Testing

Acceptance testing is software testing that evaluates whether a system meets its business and user requirements. It is the final testing stage before a system is released to production.

The main purpose of acceptance testing is to verify that a system

- ✓ Meets all of its functional and non-functional requirements

- ✓ Easy to use and navigate
- ✓ Reliable and functions as expected
- ✓ Secure and comply with all applicable regulations

This testing is performed by end users or an expected group of users. This ensures that the system is tested from the perspective of the people using it daily.

Types of Acceptance Testing

- ✓ ***User Acceptance Testing*** - Performed by end-users or client representatives to validate whether the system meets their business requirements and expectations. It typically includes real-world scenarios and helps ensure the software is ready for production deployment.
- ✓ ***Alpha Testing*** - Conducted by the internal development team before releasing the software to a selected group of users. The goal is to identify any critical issues early on and make necessary improvements before moving to beta testing.
- ✓ ***Beta Testing*** - This involves releasing the software to a limited set of external users who provide feedback based on their experience using the application. This helps uncover potential issues and gather insights for further improvements before a broader release.
- ✓ ***Contract Acceptance Testing*** - Ensures the software development meets the contractual requirements and specifications outlined in the project contract. It involves checking if the delivered product aligns with what was agreed upon in the contract.
- ✓ ***Regulatory Acceptance Testing*** - Ensures the software complies with relevant industry-specific regulations, standards, or legal requirements. This is particularly important in healthcare, finance, or government industries, where strict regulatory compliance is necessary.
- ✓ ***Compliance Acceptance Testing*** - Focuses on confirming that the software complies with internal organizational policies, guidelines, and standards. It ensures the software aligns with the organization's rules and regulations.
- ✓ ***Performance Acceptance Testing*** - Evaluate the software's performance to ensure it meets predefined performance criteria, such as response time, scalability, and stability under various load conditions.
- ✓ ***Security Acceptance Testing*** - Focuses on validating the security features and controls of the software, ensuring that it is protected against potential security threats and vulnerabilities.

Steps to Perform Acceptance Testing

This acceptance testing process is divided into five stages

- ➢ Requirement Analysis
- ➢ Test Plan Creation
- ➢ Test Case Design

- Test Case Execution
- Result Analysis and Reporting

Step 1: Requirement Analysis

- ✓ During this phase, the testing team first analyzes the requirement document and then determines the objectives of the developed software based on these requirements.
- ✓ Requirement documents, flow diagrams, and business use cases are used. Completing the business requirement document, system requirement specification, project charter, and test planning.

Step 2: Test Plan Creation

- ✓ Test planning outlines the entire acceptance testing strategy. This strategy ensures and verifies whether the software meets the specified requirements.
- ✓ It catalogs the test strategy, objectives, schedule, estimations, deadlines, and the resources required for successful project completion.

Step 3: Test Case Design

- ✓ Create detailed test cases based on the identified requirements and acceptance criteria.
- ✓ These test cases define specific steps, input data, and expected outcomes for validating the software's behavior and functionalities against the defined criteria.

Step 4: Test Case Execution

- ✓ All acceptance test cases are implemented with the input values during this phase. The tester collects and executes all user input values to ensure that the software works properly in scenarios.
- ✓ During this phase, testers interact with the software, record results, and identify expected and actual outcomes discrepancies.

Step 5: Result Analysis and Reporting

- ✓ All acceptance test cases are implemented with the input values during this phase. The tester collects and executes all user input values to ensure that the software works properly in scenarios.
- ✓ Prepare a comprehensive test report summarizing the testing activities, results, and findings. This report is a valuable resource for stakeholders to make informed decisions about the software's deployment.

Advantages

- ✓ As the users perform the tests, it improves requirement definition and user satisfaction.
- ✓ Outlining the software's quality criteria at the start of the software development life cycle (SDLC) makes carrying out acceptance tests much easier.
- ✓ The user gains experience and knowledge about the application by running acceptance tests.

Disadvantages

- ✓ Users may refuse to participate in acceptance testing processes.
- ✓ The test cases defined by a software tester are incomprehensible to the users. As a result, a software tester must assist users in testing by translating for them.

Difference Between Alpha Testing and Beta Testing

Alpha Testing	Beta Testing
Alpha testing aims to make a preliminary assessment of the software's performance and reliability.	Beta testing aims to get opinions from a broader range of external testers or end-users.
The goal is to finding and correcting problems, ensuring fundamental functionalities are operational, and checking performance in various conditions.	The goal is to test the product's functionality, compatibility, and general user experience in its intended setting.
Often conducted in a *controlled environment* within the developer's premises or under their supervision.	Performed in the *real world* by users in their environments, simulating actual usage conditions.
More in depth, involving techniques like *white-box testing (understanding internal structures) and black box testing (user perspective).*	Primarily uses *black box testing,* focusing on how the product functions from a user's standpoint.
Primarily focuses *on identifying major bugs, system stability, and core functionalities.*	Focus on broader aspects *like usability, user experience, performance, compatibility, and overall appeal* to the target audience.
Helps *shape the core functionalities* and resolve major issues before wider exposure.	Provides insights *for refining the user experience and ensuring market fit* before official release.
Typically done by *internal testers* within the development team or company.	Done by *external users,* including potential customers, early adopters, or the general public.

Debugging

- ✓ Debugging is defined as the process of identifying errors or bugs in a program and fixing the identified bugs.
- ✓ The process of debugging involves the tracing of the program's execution for the identification of the source of the problem, analysis of the code to understand the cause of the error, and modifying the code to fix the bug.

- ✓ Debugging is an iterative process that involves the repeated testing of the code until the bug is fixed. It can be done manually as well as by using automated debugging tools.

Importance of Debugging

- ✓ **Improving software quality -** Debugging helps identify and fix errors, improving the software's quality and reducing the likelihood of bugs in future releases.
- ✓ **Saving time and resources -** It helps in catching errors or bugs at an earlier stage of the SDLC, thus saving time and resources for the organization.
- ✓ **Enhancing user satisfaction -** It helps in providing a better user experience as debugging helps in ensuring that the software is functioning as per requirements. This enhances user Satisfaction.

Steps involved in Debugging

Step 1: Identify the Error

This is the first stage of Debugging is identifying the actual Error in the code of the software.

Step 2: Find the error Location

Once the problem has been identified, you will need to thoroughly review the code several times to identify the precise location of the error. This stage, in general, focuses on locating the error rather than perceiving it.

Step 3: Analyze the error

The third step is error analysis, which is a bottom-up technique that starts with identifying the issue and then analyses the code. This stage helps in the understanding of the errors.

Error analysis has two major goals*: reevaluating errors to identify existing bugs and proposing the uncertainty of incoming collateral damage in a fix.*

Step 4: Prove the analysis

After examining the primary bugs, it is necessary to search for any additional issues that may show on the program. The fourth step is used to develop automated tests for such domains by integrating the test framework.

Step 5: Cover Lateral Damage

The fifth phase involves collecting all of the unit tests for the code that has to be modified. When you run these unit tests, they must pass.

Step 6: Fix & Validate

The final stage is fix and validation, which focuses on resolving defects before running all of the test scripts to determine if they pass.

Strategies for Debugging

- **Brute – Force Method**
  This is the most common method of debugging are least efficient method, program is loaded with print statements, print the intermediate values, hope that some of printed values will help identify the error.
- **Backtracking**
  The bug hunter begins at the statement where an error symptom is detected and works backward through the source code to the actual fault.
- **Cause Elimination Method**
  The software engineer generates a list of possible reasons for an error and performs tests to find the origin of a point of failure.
- **Program Slicing**
  Quality Assurance (QA) testers execute a group of program statements in the program (slice) under some specific conditions.
- **Rubber Duck Debugging**
  Rubber Duck Debugging is when a programmer explains code to a small rubber duck or another inanimate item line by line. The idea is to encourage flexible, rational thinking.
- **Pair Debugging**
  Instead of describing code line by line to an inanimate object, coders pair up explaining code line by line to each other.

Advantages

- ✓ The identification of the bugs at an earlier stage helps in saving the time of the developers.
- ✓ This identification and fixing of bugs also save money in the long run.
- ✓ This process of debugging helps in improving the code quality of the software.
- ✓ Debugging can result in better code optimization of the software/product.
- ✓ This process ensures that the software is working fine as per the expectations and plans.

Disadvantages

- ✓ The process of debugging is time-consuming.
- ✓ It can be stressful for developers if they are working hard to find bugs or errors.
- ✓ In some cases, Debugging may result in the introduction of new bugs in the software.
- ✓ The process of debugging requires specialized tools, which might not be directly available in the market.
- ✓ Developers might get distracted by debugging from other important tasks such as implementing a new feature.

Program Analysis

- ✓ **Program Analysis** is the process of automatically analyzing the behavior of computer programs regarding a property such as correctness, robustness, safety and liveness.
- ✓ Program analysis focuses on two major areas: ***program optimization and program correctness.***
- ✓ The first focuses on improving the program's performance while reducing the resource usage while the latter focuses on ensuring that the program does what it is supposed to do.
- ✓ Program analysis can be performed without executing the program (static program analysis), during runtime (dynamic program analysis) or in a combination of both.

Program Analysis Tool

- ✓ Program Analysis Tool is an automated tool whose input is the source code or the executable code of a program and the output is the observation of characteristics of the program.
- ✓ It gives various characteristics of the program such as its size, complexity, adequacy of commenting, adherence to programming standards and many other characteristics. These tools are essential to software engineering because they help programmers comprehend, improve and maintain software systems over the course of the whole development life cycle.

Classification of Program Analysis Tools

1. Static Program Analysis Tools

Static Program Analysis Tool is such a program analysis tool that evaluates and computes various characteristics of a software product without executing it. Normally, static program analysis tools analyze some structural representation of a program to reach a certain analytical conclusion. Basically some structural properties are analyzed using static program analysis tools.

The structural properties that are usually analyzed are:

1. Whether the coding standards have been fulfilled or not.
2. Some programming errors such as uninitialized variables.
3. Mismatch between actual and formal parameters.
4. Variables that are declared but never used.

Code walkthroughs and code inspections are considered as static analysis methods but static program analysis tool is used to designate automated analysis tools. Hence, a compiler can be considered as a static program analysis tool.

Types of static analysis

- ✓ ***Control analysis*** -- focuses on the control flow in a calling structure. For example, a control flow could be a process, function, method or in a subroutine.
- ✓ ***Data analysis*** -- makes sure defined data is properly used while also making sure data objects are properly operating.
- ✓ ***Fault/failure analysis*** -- analyzes faults and failures in model components.
- ✓ ***Interface analysis*** -- verifies simulations to check the code and makes sure the interface fits into the model and simulation.

Static code analysis advantages

- ✓ It can find weaknesses in the code at the exact location.
- ✓ It can be conducted by trained software assurance developers who fully understand the code.
- ✓ It allows a quicker turn around for fixes.
- ✓ It is relatively fast if automated tools are used.
- ✓ Automated tools can scan the entire code base.
- ✓ Automated tools can provide mitigation recommendations, reducing the research time.
- ✓ It permits weaknesses to be found earlier in the development life cycle, reducing the cost to fix.

2. Dynamic Program Analysis Tools

- ✓ Dynamic Program Analysis Tool is such type of program analysis tool that require the program to be executed and its actual behavior to be observed. A dynamic program analyzer basically implements the code. It adds additional statements in the source code to collect the traces of program execution.
- ✓ When the code is executed, it allows us to observe the behavior of the software for different test cases. Once the software is tested and its behavior is observed, the dynamic program analysis tool performs a post execution analysis and produces reports which describe the structural coverage that has been achieved by the complete testing process for the program.
- ✓ The result of dynamic analysis is the extent of testing performed as white box testing. If the testing result is not satisfactory then more test cases are designed and added to the test scenario. Also dynamic analysis helps in elimination of redundant test cases.

Dynamic code analysis limitations

- ✓ Automated tools provide a false sense of security that everything is being addressed.
- ✓ Automated tools produce false positives and false negatives.

- ✓ Automated tools are only as good as the rules they are using to scan with.
- ✓ There are not enough trained personnel to thoroughly conduct dynamic code analysis [as with static analysis].
- ✓ It is more difficult to trace the vulnerability back to the exact location in the code, taking longer to fix the problem.

Model Checking

Model Checking is part of the Formal Methods which address formal specification, formal development, formal verification and theorem provers. Model Checking belongs to the formal verification and provides a complete mathematical proof to verify your system-under-test.

Symbolic Model Checking

- ✓ A formal verification technique called symbolic model checking is employed to confirm the accuracy and security of software and hardware systems.
- ✓ It is widely used in many different sectors to guarantee the dependability of complex systems. It entails symbolically modeling systems and automating the verification process.

Importance of Symbolic Model Checking

- ✓ Symbolic model checking is a conventional technique utilized in the field of software engineering and the hypothesis of calculation to confirm the rightness of equipment and programming situations. The methodology includes addressing the framework under examination as a numerical model, normally as a limited state machine or progress framework, and afterward utilizing robotized calculations to dissect the model and check for blunders or properties.
- ✓ Symbolic model checking has been applied to confirm a large number of frameworks, including equipment plans, correspondence conventions, and programming programs.

Attributes of Symbolic Model Checking

- ✓ **Emblematic control**: Symbolic model checking utilizes emblematic control methods, for example, Twofold Choice Charts (BDDs) and Satisfiability Modulo Speculations (SMT), to address and control enormous and complex state spaces. These methods can fundamentally lessen the computational intricacy of confirmation issues, making it conceivable to actually take a look at frameworks with billions of potential states.
- ✓ **Robotization**: symbolic model checking is a completely computerized method that requires practically zero human intercession. When the framework under

examination and its determination are encoded in a reasonable conventional language, the confirmation cycle can be performed naturally utilizing a model really looking at the device.

- ✓ **Culmination**: symbolic model checking can give total confirmation, implying that it can really take a look at all potential conditions of a framework and check that the framework fulfills its determination.
- ✓ **Adaptability**: Emblematic model checking scales well with framework size, making it appropriate for confirming enormous and complex frameworks. It can deal with frameworks with many cooperating parts, complex information designs, and simultaneousness.

Methods Associated with Symbolic Model Checking

- ✓ **Emblematic portrayal**: The framework under investigation is addressed emblematically utilizing numerical designs, for example, Boolean equations, Paired Choice Outlines (BDDs), and propositional rationale. This permits the framework to be dissected all the more effectively by staying away from unequivocal list of states.
- ✓ **Model development**: A model of the framework under investigation is built utilizing formal strategies, for example, automata, Petri nets, or change frameworks. The model is then changed into an emblematic portrayal utilizing strategies, for example, BDDs or other choice graphs.
- ✓ **State-space investigation**: symbolic model checking investigates the state space of the framework under examination to confirm in the event that it fulfills a given detail. State-space investigation methods include utilizing calculations that cross the state space of the framework and check assuming the determination is fulfilled.
- ✓ **Choice strategies**: symbolic model checking frequently involves choice techniques for satisfiability modulo speculations (SMT) or satisfiability (SAT) issues to check regardless of whether a bunch of consistent recipes is satisfiable. This assists with distinguishing counterexamples and check the accuracy of the framework.
- ✓ **Property detail**: The determination that the framework needs to fulfill is regularly indicated by utilizing a transient rationale, like Straight Worldly Rationale (LTL) or Calculation Tree Rationale (CTL). These rationales take into consideration the declaration of transient connections between framework ways of behaving.
- ✓ **Deliberation**: Symbolic model checking might utilize reflection methods to lessen the intricacy of the framework under examination, for example, by eliminating unimportant subtleties or amassing comparative states. This can assist with making the confirmation cycle more proficient.