Software Testing

- Unit Testing
- Integration Testing
- Validation Testing
- System Testing
- Testing techniques:
  - White Box Testing
    - Basis Path Testing
    - Control Structure Testing
  - Black Box Testing
    - Graph Based
    - Equivalence
    - Boundary Value
- Types of Software Maintenance
  - Re-Engineering
  - Reverse Engineering

Software Testing

- Software testing is a process of identifying the correctness of software by considering its attributes (Reliability, Scalability, Portability, Re-usability, Usability) and evaluating the execution of software components to find the software bugs or errors or defects.
- It is a critical element of software quality assurance and represents the ultimate review of specification, design, and code generation.
  - It is not unusual for a software development organization to pay between 30 and 40 percent of total project effort on testing.
  - The engineer creates a series of test cases that are intended to "defeat" the software that has been built.
- In fact, testing is one step in the software process that could be viewed as destructive rather than constructive.

Principles of Software Testing

❖ There are seven (7) testing principles of testing:

1. ***Complete testing of software is not possible***
   - Complete testing of software is not possible.
     - Rather, we require performing the best possible amount of testing with the help of risk assessment of the application.
   - Risk assessment is the process of guessing module where defects may occur and test that module to find out those defects.
     - Because we have not much amount of time to test each and every module from our software, exhaustive or complete testing of software is not possible.

Principles of Software Testing ….

2. ***Defect clustering***
   - This principle states that about eighty percent of defects are found in twenty percent of modules.
   - Using experience, we can find out such risky modules.
   - But this approach has its own demerits such as if we run similar test cases repetitively, such test cases do not find any new defects.

3. ***Pesticide Paradox***
   - If we use same test cases repetitively for testing, then after a particular point we cannot find new defects.
   - To solve this problem, the test cases required to be reviewed and revised on regular basis and include new test cases to help in finding new defects.

Principles of Software Testing ….

- Testers cannot use only existing testing mechanism.
  - They should try to improve the existing methods to find out more defects.
  - But, after performing testing, you can never claim your product is bug free.

5. ***Testing shows presence of defects***

- Testing process can show that the defects are present in the system under test, but it cannot prove that there are no defects.
- Even after testing we cannot guarantee that the system is 100% error free.
- Objectives of testing process are as follows:
  - Finding defects which are created by developer while developing software.

Principles of Software Testing ….

- Providing information about quality of software under test.
- To ensure that system meets the business and user requirements.
- To gain customer's confidence by providing them with a quantitative product.

6. ***Absence of error***

- The software which is 99% bug free can be still unsafe if it is tested for wrong requirements

  i.e. absence of error does not help if the system does not satisfy the user's needs and requirements.

Principles of Software Testing ….

6. ***Early Testing***
   - Testing must be started as early as possible in the Software Development Life Cycle (SDLC).
   - Hence, any bugs in the requirements or design phase are found out in early phases of software testing.
   - It is less costly to correct that bug in early stages of testing.
   - It is suggested that we can/ should start detecting the bug at the time when requirements are gathered.

7. ***Testing is context dependent***
   - "Testing is context dependent" specifies that the way a tester tests an online shopping site will be different from the way he/she tests a stand alone application.

Importance of Software Testing

- Software Testing is important because if there are any bugs or errors in the software, it can be identified early and can be solved before delivery of the software product.
    - Software bugs could be expensive or even dangerous.
    - Software bugs can potentially cause monetary and human loss.
- Properly tested software product ensures reliability, security and high performance which further results in time saving, cost effectiveness and customer satisfaction.

- Benefits of using software testing:
    1. Cost-Effective:
        - It is one of the important advantages of software testing. Testing any IT project on time helps you to save your money for the long term.

Importance of Software Testing ..

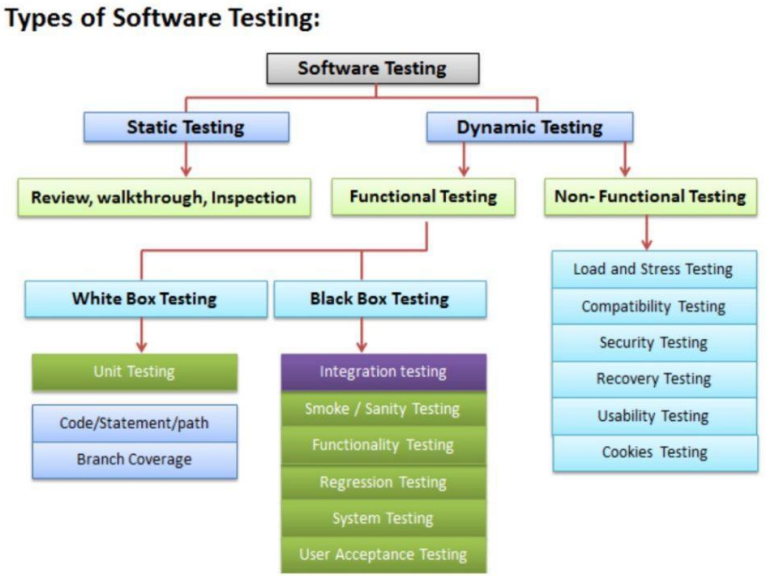
- If the bugs caught in the earlier stage of software testing, it costs less to fix.

3. Security:
   - It is the most vulnerable and sensitive benefit of software testing.
   - People are looking for trusted products.
   - It helps in removing risks and problems earlier.
4. Product quality:
   - It is an essential requirement of any software product.
   - Testing ensures a quality product is delivered to customers.
5. Customer Satisfaction:
   - The main aim of any product is to give satisfaction to their customers.
   - Testing ensures the best user experience.

Types of Software Testing

- The main objective of software testing is to design the tests in such a way that it systematically finds different types of errors without taking much time and effort so that less time is required for the development of the software.

- Testing types in general:
  - Manual testing
  - Automation testing



Types of Software Testing

Parameter	Manual Testing	Automation Testing
Human Interaction	To perform manual testing, human interaction is required for test execution	To perform automation testing, human interaction is not required. In automation testing, we use tools to run the test cases.
Requirements	To perform manual testing, we need skilled employees, more time and high costs.	To perform automation testing, we require automation tools. Automation testing saves time, cost and manpower. Once test suite is recorded in automation, then it can easily run test suite.
Application to Test	Any application can be first tested manually. Informal testing like ad-hoc and monkey testing are performed manually.	Automation testing is used to test an application whose requirements are stable at some extent and automation testing mainly used to perform Regression Testing.
Execution of Same test cases	We cannot use manual testing while executing same test suite. Repetitive execution of test cases become boring and error prone.	The most boring part which is of running same test cases repetitively can be performed with the help of automation testing.

Types of Software Testing …. Contd.

- **Static Testing**:-
  - It is a type of software testing in which software application is tested without code execution.
  - It checks the code, requirement documents, and design documents to find errors.
  - Static testing is about the prevention of defects.
  - Static testing does the verification process.
  - It is performed before compilation.
  - Static testing techniques are structural and statement coverage.

- **Static Testing Techniques:**
  - **Informal Reviews:**
    - This is one of the type of review which does not follow any process to find errors in the document.
    - Under this technique, you just review the document and give informal comments on it.

Static Testing

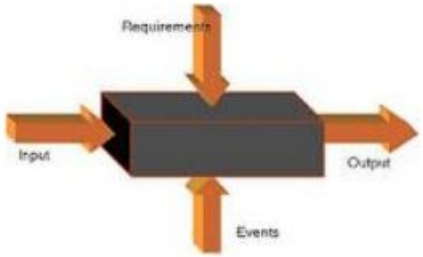
- **Technical Reviews:**
  - A team consisting of your peers, review the technical specification of the software product and checks whether it is suitable for the project.
  - They try to find any discrepancies in the specifications and standards followed.
  - This review concentrates mainly on the technical documentation related to the software such as Test Strategy, Test Plan and requirement specification documents.
- **Walkthrough:**
  - The author of the work product explains the product to his team.
  - Participants can ask questions if any.
  - A meeting is led by the author.
  - Scribe makes note of review comments

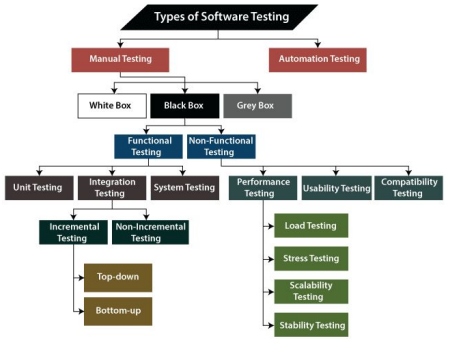
Static Testing …. Contd.

- **Inspection:**
  - The main purpose is to find defects and meeting is led by a trained moderator.
  - This review is a formal type of review where it follows a strict process to find the defects.
  - Reviewers have a checklist to review the work products.
  - They record the defect and inform the participants to rectify those errors.
- **Static code Review:**
  - This is a systematic review of the software source code without executing the code.
  - It checks the syntax of the code, coding standards, code optimization, etc.
  - This is also termed as white box testing.
  - This review can be done at any point during development.

Dynamic Testing

- **Dynamic Testing**:-
  - It is done by executing the program.
  - It checks the functional behavior of software system, memory/CPU usage and overall performance of the system.
  - Dynamic testing is about finding and fixing the defects.
  - Dynamic testing does the validation process.
  - It is performed after compilation.
  - Dynamic testing techniques are Boundary Value Analysis and Equivalence Partitioning.





Dynamic Testing …. Contd.

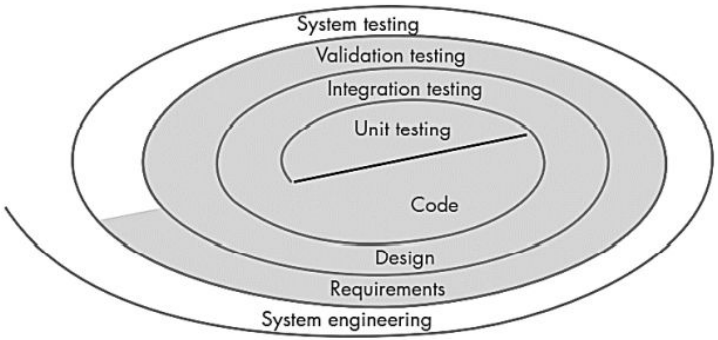
- **Dynamic Testing Techniques**:
  - Unit Testing: This software testing basic approach is followed by the programmer to test the unit of the program. Individual units or modules are tested by the developers. It involves testing of source code by developers.
  - Integration testing: Individual modules are grouped together and tested by the developers. The purpose is to determine what modules are working as expected once they are integrated.
  - System testing: It is performed on the whole system by checking whether the system or application meets the requirement specification document. This testing strategy checks the functionality, security, portability, amongst others.

Static Vs Dynamic Testing

Static Testing	Dynamic Testing
In static testing, we check the code or the application without executing the code.	In dynamic testing, we check the code/application by executing the code.
Static testing includes activities like code Review, Walkthrough, etc.	Dynamic testing includes activities like functional and non-functional testing such as UT (usability testing), IT (integration testing), ST (System testing) and UAT (user acceptance testing).
Static testing is a **Verification** Process.	Dynamic testing is a **Validation** Process.
Static testing is used to prevent defects.	Dynamic testing is used to find and fix the defects.
Static testing is a more cost-effective process.	Dynamic testing is a less cost-effective process.
This type of testing can be performed before the compilation of code.	Dynamic testing can be done only after the executables are prepared.
Under static testing, we can perform the statement coverage testing and structural testing.	Equivalence Partitioning and Boundary Value Analysis technique are performed under dynamic testing.
It involves the checklist and process which has been followed by the test engineer.	This type of testing requires the test case for the execution of the code.

Testing Strategies

- Initially, system engineering defines the role of software and leads to software requirement analysis, where the information domain, function, behavior, performance, constraints and validation criteria for software are established.
- Moving inward along the spiral, you come to design and finally to coding.
- To develop computer software, you spiral inward (counter clockwise) along streamlines that decrease the level of abstraction on each turn.

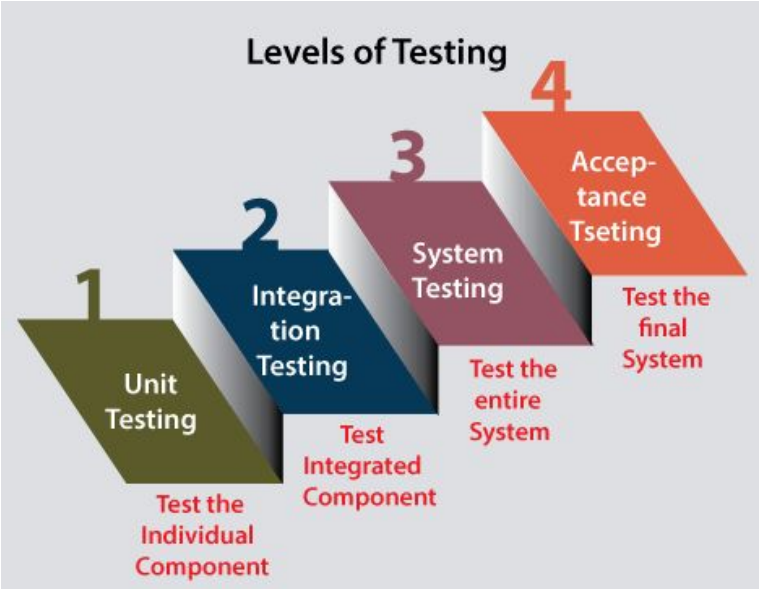


Levels of Testing

- The levels of software testing involve the different methodologies, which can be used while we are performing the software testing.

- In software testing, we have four different levels of testing, which are as discussed below:

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing



Unit Testing

- Unit is the smallest part of a software system, which is testable.
  - It may include code files, classes and methods which can be tested individually for correctness.
- Unit is a process of validating such small building blocks of a complex system, much before testing an integrated large module or the system as a whole.
- It 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.

Unit Testing …. Contd.

- Unit testing is a strategy that utilizes the white-box method and concentrates on testing individual programming units.
- Unit Testing Example
  - You are testing a function; whether loop or statement in a program is working properly or not than this is called as unit testing.
  - A beneficial example of a framework that allows automated unit testing is JUNIT (a unit testing framework for java).
  - XUnit is a more general framework which supports other languages like C#, ASP, C++, Delphi and Python to name a few.

- Tests that are performed during the unit testing are:

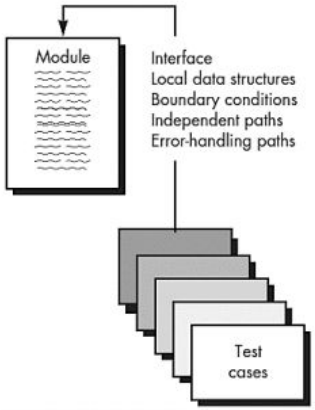
  1) Module Interface test:
    - In module interface test, it is checked whether the information is properly flowing in to the program unit (or module) and properly happen out of it or not.

Unit Testing …. Contd.

2) **Local data structures:**
   - These are tested to inquiry if the local data within the module is stored properly or not.
3) **Boundary conditions:**
   - It is observed that much software often fails at boundary related conditions.
   - That is why boundary related conditions are always tested to make safe that the program is properly working at its boundary conditions.
4) **Independent paths:**
   - All independent paths are tested to see that they are properly executing their task and terminating at the end of the program.
5) **Error handling paths:**
   - These are tested to review if errors are handled properly by them or not.

Unit Testing …. Contd.

- Driver and/or stub software must be developed for each unit test.
  - A driver is nothing more than "a main program" that accepts test case data, passes such data to the component, and prints relevant results.
  - Stubs serve to replace modules that are subordinate (called by) the component to be tested.
  - A stub or a "dummy subprogram" uses the subordinate module's interface.



Unit Testing …. Contd.

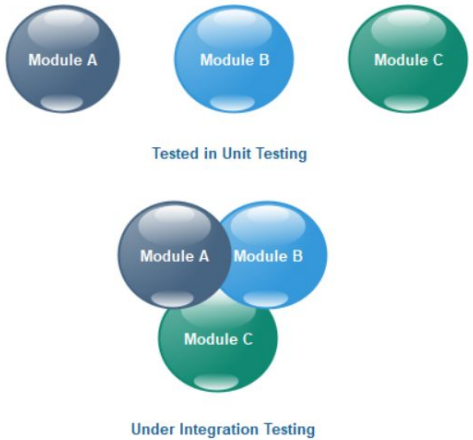
- **Advantages:-**
  - Developers looking to learn what functionality is provided by a unit and how to use it can look at the unit tests to gain a basic understanding of the unit API.
  - Unit testing allows the programmer to refactor code at a later date, and make sure the module still works correctly (i.e. Regression testing).
    - The procedure is to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified and fixed.
  - Due to the modular nature of the unit testing, we can test parts of the project without waiting for others to be completed.

Unit Testing …. Contd.

- **Disadvantages:-**
  - Unit testing can not be expected to catch every error in a program.
    - It is not possible to evaluate all execution paths even in the most trivial programs
  - Unit testing by its very nature focuses on a unit of code.
    - Hence it can not catch integration errors or broad system level errors.

Integration Testing

- Integration testing is the second level of the software testing process comes after unit testing.
- It is defined as a type of testing where software modules are integrated logically and tested as a group (set of integration among component).
- Testing the interactions between the module interactions with other system externally is called Integration Testing.
- A typical software project consists of multiple software modules, coded by different programmers.

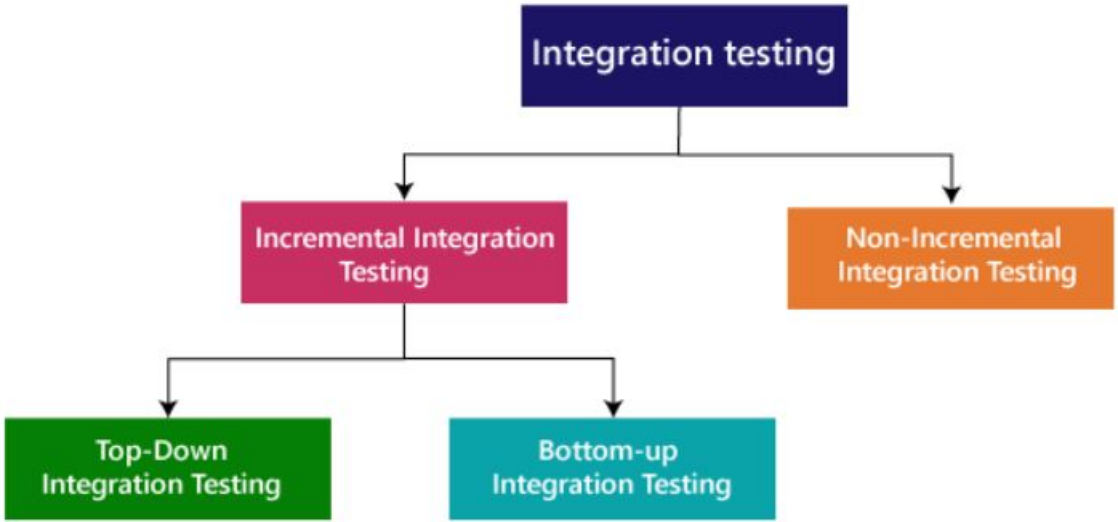


Integration Testing

- The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated.
- It addresses the issues associated with the dual problems of verification and program construction.
- Modules are typically code modules, individual applications, client and server applications on a network, etc.
  - This type of testing is especially relevant to client/server and distributed systems.
- It 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’.

Integration Testing ….

- Integration testing can be classified into two parts:
  - Incremental integration testing
  - Non-incremental integration testing



Integration Testing ….

- **Incremental integration testing:-**
  - In the Incremental Approach, modules are added in ascending order one by one or according to need.
  - The selected modules must be logically related.
  - Generally, two or more than two modules are added and tested to determine the correctness of functions.
  - The process continues until the successful testing of all the modules.
  - Ex.
    - Suppose we have a Flipkart application, we will perform incremental integration testing, and the flow of the application would like this:

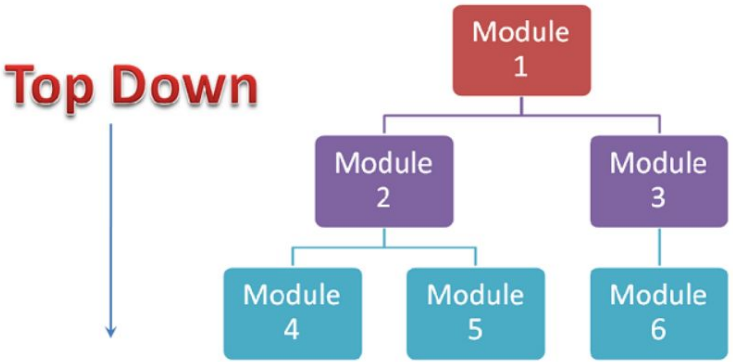
Flipkart→ Login→ Home → Search→ Add cart→ Payment → Logout

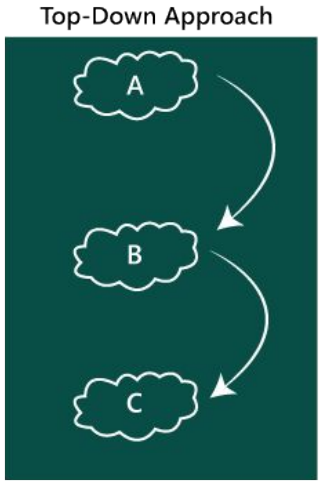
Integration Testing ….

- Incremental integration testing is carried out by further methods:
  - **a. 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.
      - Major design flaws can be detected and fixed early because critical modules tested first.
    - Stubs are used for testing if some modules are not ready.
      - Stubs and Drivers are the dummy programs in Integration testing used to facilitate the software testing activity.

Integration Testing ….

- These programs act as a substitutes for the missing models in the testing.
- They do not implement the entire programming logic of the software module but they simulate data communication with the calling module while testing.
  - » Stub: Is called by the Module under Test.
  - » Driver: Calls the Module to be tested.





Integration Testing ….

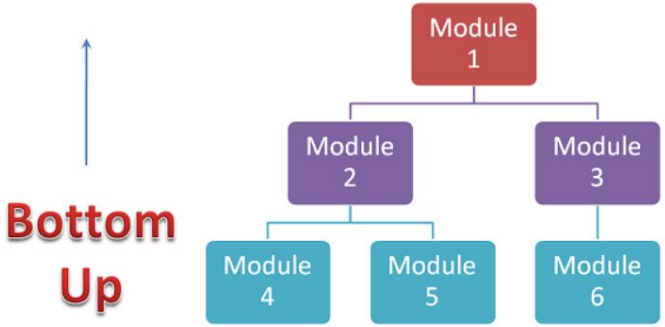
- **Advantages**:-
  - Fault Localization is easier.
  - Possibility to obtain an early prototype.
  - Critical Modules are tested on priority; major design flaws could be found and fixed first.

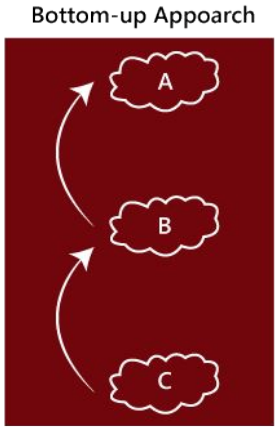
- **Disadvantages**:-
  - Needs many Stubs.
  - Modules at a lower level are tested inadequately.

Integration Testing ….

b. 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.





Integration Testing ….

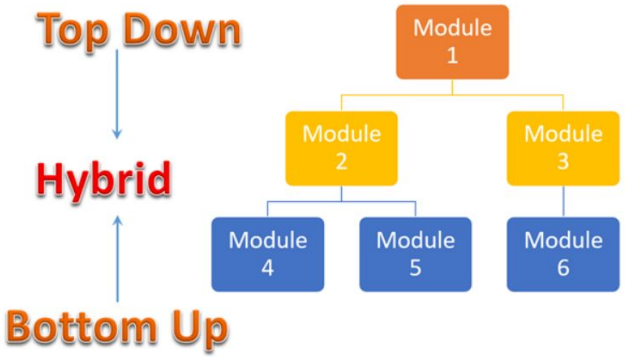
- **Advantages**:-
  - Fault Localization is easier.
  - No time is wasted waiting for all modules to be developed unlike Big-bang approach

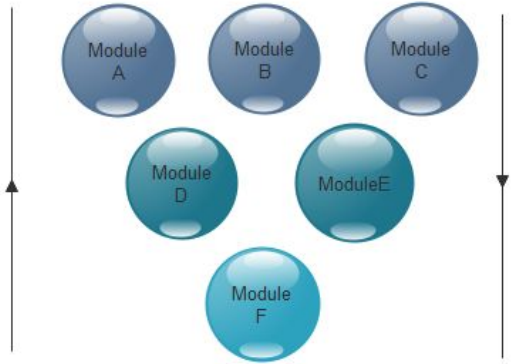
- **Disadvantages**:-
  - Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.
  - An early prototype is not possible

Integration Testing ….

c. Sandwich/ Hybrid Integration 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.





Hybrid Method

Integration Testing ….

- **Advantages**:-
  - The hybrid method provides features of both Bottom Up and Top Down methods.
  - It is most time reducing method.
  - It provides complete testing of all modules.

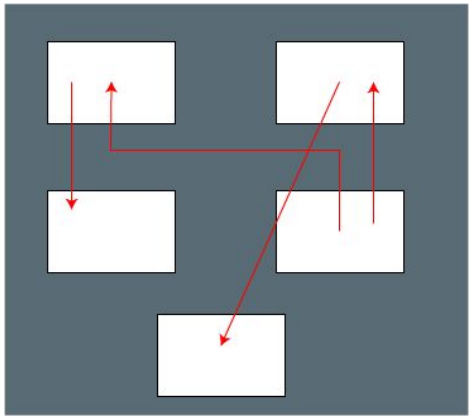
- **Disadvantages**:-
  - This method needs a higher level of concentration as the process carried out in both directions simultaneously.
  - Complicated method.

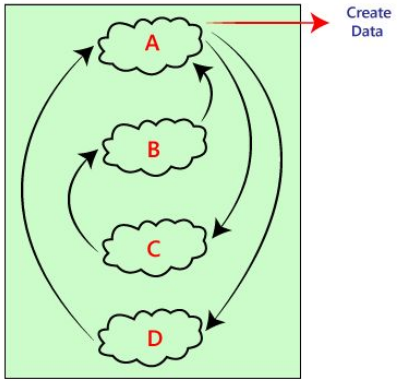
Integration Testing ….

- **Non-Incremental integration testing:-**
  - We will go for this method, when the data flow is very complex and when it is difficult to find who is a parent and who is a child.
  - In such case, we will create the data in any module bang on all other existing modules and check if the data is present.
  - Hence, it is also known as the Big bang method.
    - 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.

Integration Testing ….

- **Big-Bang Non-Incremental integration testing**
  - In this approach, testing is done via integration of all modules at once.
  - It is convenient for small software systems, if used for large software systems identification of defects is difficult.
  - Since this testing can be done after completion of all modules due to that testing team has less time for execution of this process so that internally linked interfaces and high-risk critical modules can be missed easily.





Integration Testing ….

- **Advantages**:-
  - It is convenient for small size software systems.

- **Disadvantages**:-
  - Fault Localization is difficult.
    - Identification of defects is difficult because finding the error where it came from is a problem, and the source of the bug is not known.
  - Small modules missed easily.
  - Time provided for testing is very less.
    - Since the Integration testing can commence only after “all” the modules are designed, the testing team will have less time for execution in the testing phase.
  - We may miss to test some of the interfaces.

Integration Testing ….

- Given the sheer number of interfaces that need to be tested in this approach, some interfaces link to be tested could be missed easily.
- Since all modules are tested at once, high-risk critical modules are not isolated and tested on priority. Peripheral modules which deal with user interfaces are also not isolated and tested on priority.

Validation Testing

- Validation testing in software engineering is in place to determine if the existing system complies with the system requirements and performs the dedicated functions for which it is designed along with meeting the goals and needs of the organization.
  - Validation Testing ensures that the product actually meets the client's needs.
  - It can also be defined as to demonstrate that the product fulfills its intended use when deployed on appropriate environment.
  - It answers to the question, Are we building the right product?
- The process of evaluating software during the development process or at the end of the development process to determine whether it satisfies specified business requirements.
- This mode of testing is extremely important especially if you want to be the best software testing company.

Validation Testing…. Contd.

- Validation Testing is Important –
  - To ensure customer satisfaction
  - To be confident about the product
  - To fulfill the client's requirement until the optimum capacity
  - Software acceptance from the end-user

- Verification testing:
  - Verification testing includes different activities such as business requirements, system requirements, design review, and code walkthrough while developing a product.
  - It is also known as static testing, where we are ensuring that "**we are developing the right product or not**".
  - And it also checks that the developed application fulfilling all the requirements given by the client.

Verification Vs Validation Testing

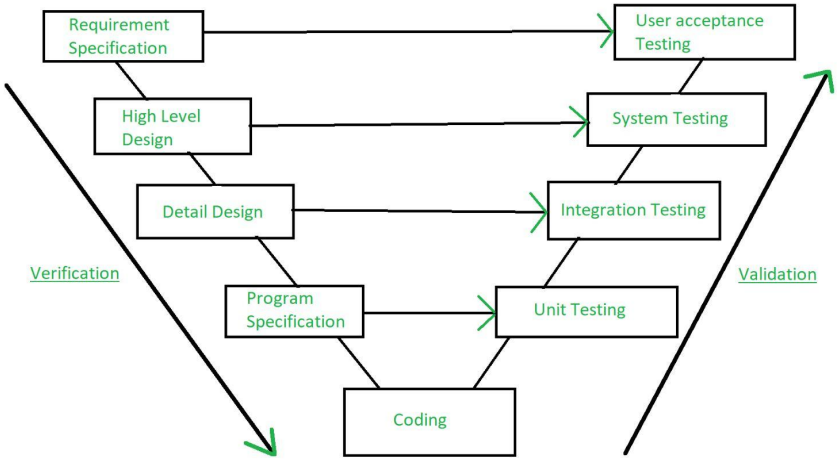
Verification	Validation
We check whether we are developing the right product or not.	We check whether the developed product is right.
Verification is also known as **static testing**.	Validation is also known as **dynamic testing**.
Verification includes different methods like Inspections, Reviews, and Walkthroughs.	Validation includes testing like functional testing, system testing, integration, and User acceptance testing.
It is a process of checking the work-products (not the final product) of a development cycle to decide whether the product meets the specified requirements.	It is a process of checking the software during or at the end of the development cycle to decide whether the software follow the specified business requirements.
Quality assurance comes under verification testing.	**Quality control** comes under validation testing.
The execution of code does not happen in the verification testing.	In validation testing, the execution of code happens.

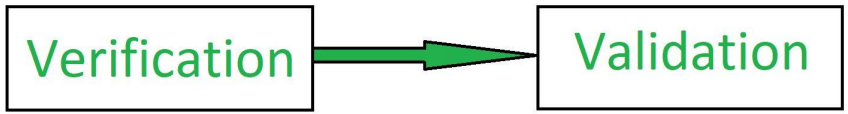
Verification Vs Validation ….

Verification	Validation
In verification testing, we can find the bugs early in the development phase of the product.	In the validation testing, we can find those bugs, which are not caught in the verification process.
Verification testing is executed by the Quality assurance team to make sure that the product is developed according to customers' requirements.	Validation testing is executed by the testing team to test the application.
Verification is done before the validation testing.	After verification testing, validation testing takes place.
In this type of testing, we can verify that the inputs follow the outputs or not.	In this type of testing, we can validate that the user accepts the product or not.

Validation Testing…. Contd.

- Validation testing provides final assurance that software meets all informational, functional, behavioral, and performance requirements.
- Verification is followed by Validation





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 actually a series of different tests whose sole purpose is to exercise the full computer-based system.

- 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.

System Testing …. Contd.

- Verify thorough testing of every input in the application to check for desired outputs.
- Testing of the user's experience with the application.

- Types of System Testing:
  1. **Usability Testing**:
     - Usability Testing is a type of software testing in which a group of end-users of a software system use the software to check user friendliness.
       - It is non-functional testing.
       - This testing basically concentrates on how easily user handles the application.
       - It is verified that after few hours of training, end user can use/ handle the system comfortably.

System Testing …. Contd.

- Icon used in user interface is not confusing i.e. for cut option, we use scissor picture, not CD picture.

- This type of testing is also known as User Experience testing.
- This testing is suggested to be performed at the initial design phase of SDLC because it finds out bugs (errors) in user interface and gives chance to improve them.

3. **Performance Testing**:
   - Performance Testing is a type of software testing that is carried out to test the speed, scalability, stability and reliability of the software product or application.

System Testing …. Contd.

3. **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.

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

5. **Scalability Testing:**
   - Scalability Testing is a type of software testing which is carried out to check the performance of a software application or system in terms of its capability to scale up or scale down the number of user request load.

System Testing …. Contd.

6. **Regression Testing**:
   - Regression Testing is a type of software testing which is used to check whether changes have been made in code due to some error or change in requirement affects the existing working functionality.
   - In Regression Testing, we execute already executed test cases to give assurance that old functionalities work well after performing changes in code.
   - This testing is performed to give guarantee that new code added in the software does not disturb working of the existing functionalities.

System Testing …. Contd.

7. **Recovery Testing**:
   - Recovery testing is done to specify whether system recovers itself after it crashes due to disasters such as power failure, or network not available, etc.
   - In recovery testing, system performs rollback
     i.e. identify the point where the behavior of the system was correct and then from that point, again perform the operations up to the point where the system got crashed.

8. **Migration Testing:**
   - Migration testing is performed to give assurance that the software can be moved from older system infrastructures to current system infrastructure without any problem.
   - In Migration testing, we check data from old system for the compatibility with the new system.

System Testing …. Contd.

9. **Functional Testing**:
   - Functional testing is a type of software testing which checks that every function present in the software application works as per requirements of the user.
   - This testing includes black box testing.
     - It does not focus on the source code of the software application.
   - Every functionality of the system is verified by the tester using appropriate test data and actual result is compared with the expected result.
     - If there is difference between actual result and expected result, then bug is considered.
   - Functional testing is done using Requirement Specification document.

System Testing …. Contd.

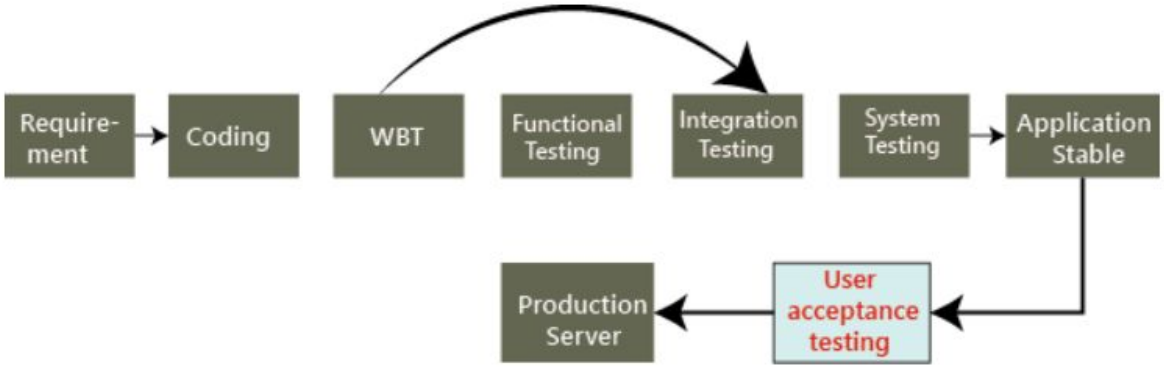
10. **Hardware/ Software Testing**:
    - In hardware/ software testing, one performs testing of communication between the hardware and the software used in the system.
    - IBM called the Hardware/ Software Testing as “HW/SW Testing”.

Acceptance Testing

- User Acceptance Testing (UAT) is a type of testing performed by the end user or the client to verify/accept the software system before moving the software application to the production environment.
- UAT is done in the final phase of testing after functional, integration and system testing is done.
- The main Purpose of UAT is to validate end to end business flow.
  - It does not focus on cosmetic errors, spelling mistakes or system testing.
  - User Acceptance Testing is carried out in a separate testing environment with production-like data setup.
  - It is kind of black box testing where two or more end-users will be involved.

Acceptance Testing …. Contd.

- Who Performs UAT?
  - Client
  - End users



Acceptance Testing ….Contd.

- **Need of User Acceptance Testing** arises once software has undergone Unit, Integration and System testing because developers might have built software based on requirements document by their own understanding and further required changes during development may not be effectively communicated to them.
  - So for testing whether the final product is accepted by client/end-user, user acceptance testing is needed.
    - Developers code software based on requirements document which is their “own” understanding of the requirements and may not actually be what the client needs from the software.
    - Requirements changes during the course of the project may not be communicated effectively to the developers.

Acceptance Testing ….Contd.

- **Use of Acceptance Testing:**
  - To find the defects missed during the functional testing phase.
  - To check how well the product is developed.
  - A product developed is what actually the customers need.
  - Feedbacks help in improving the product performance and user experience.
  - Minimize or eliminate the issues arising from the production.

Acceptance Testing…. Contd.

- It is further classified into following types:
  - User Acceptance Testing (UAT):
  - Business Acceptance Testing (BAT):
  - Contract Acceptance Testing (CAT):
  - Regulations Acceptance Testing (RAT):
  - Operational Acceptance Testing (OAT):

i. Alpha Testing

- ✔ The alpha test is conducted at the developer's site by a representative group of end users.
- ✔ The software is used in a natural setting with the developer "looking over the shoulder" of the users and recording error and usage problems.
- ✔ Alpha tests are conducted in a controlled environment.

Acceptance Testing…. Contd.

- ✔ The main aim of alpha testing is to identify and fix those bugs that were not found in the previous tests.
- ✔ Another aim is to get an understanding of how it feels to use the software in its entirety before releasing the software to the public for beta testing.
- ✔ Alpha testing refines the applications by removing such errors and identifying where improvements can be made.

- ✔ **Advantages**:
  - Ensures that the user gets a smooth flow using the system.
  - Refines the applications by identifying bugs and errors before actual deployment.
  - Real-time management of the product.

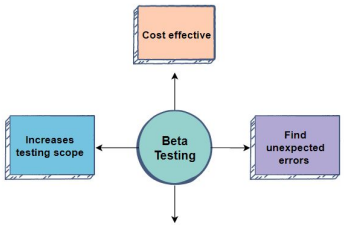
Acceptance Testing…. Contd.

- Ensures the quality of the software product.
- Analyzes and validates the software product's quality, functionality, performance, effectiveness, etc.

Acceptance Testing…. Contd.

ii. Beta Testing

- ✔ The beta test is conducted at one or more end-user sites.
- ✔ Unlike alpha testing, the developer generally is not present.
- ✔ Therefore, the beta test is a “live application of the software in an environment that cannot be controlled by the developer.
- ✔ Beta testing is used to assess the product by exposing it to the real end-users, usually called beta testers in their environment.
- ✔ Feedback is collected from the users and the defects are fixed.
- ✔ Also, this helps in enhancing the product to give a rich user experience.
- ✔ The common features evaluated during beta testing include the following:
  - reliability
  - security
  - Robustness



Acceptance Testing…. Contd.

- ✔ Although there is no set standard for beta testing, the following conditions must be fulfilled for successful beta testing:
  - The product should be in the completion stage.
  - The product should be free from any unpredictable bugs or crashes.
  - The testers should be real end-users of the product.
  - One should do the testing based on real-life scenarios and environments and not the testing labs.
- ✔ **Advantages**:
  - By gathering and incorporating customer feedback, beta testing increases product quality, reduces risks of product failures, and increases overall customer satisfaction.
- ✔ **Disadvantages**:
  - Bug reports are not systematic

Acceptance Testing…. Contd.

Alpha Testing	Beta Testing
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-users in a real-time environment, who is not an employee of the organization.
Alpha testing performed at the developer's site; it always needs a testing environment or lab environment.	Beta testing doesn't need any lab environment or the testing environment; it is performed at a client's location or end-user of the product.
Reliability or security testing not performed in-depth in alpha testing.	Reliability, security, and robustness checked during beta testing.
Alpha testing involves both white box and black-box techniques.	Beta testing uses only black-box testing.
Long execution cycles maybe require for alpha testing.	Only a few weeks are required for the execution of beta testing.

Acceptance Testing…. Contd.

Alpha Testing	Beta Testing
Critical issues or fixes can be identified by developers immediately in alpha testing.	Most of the issues or feedback is collecting from the beta testing will be implemented for the future versions of the product.
Alpha testing performed before the launch of the product into the market.	At the time of software product marketing.
Alpha testing focuses on the product's quality before going to beta testing.	Beta testing concentrates on the quality of the product, but gathers users input on the product and ensures that the product is ready for real-time users.
Alpha testing performed nearly the end of the software development.	Beta testing is a final test before shipping a product to the customers.
Alpha testing is conducting in the presence of developers and the absence of end-users.	Beta testing reversed of alpha testing.

Testing Techniques

- ✔ Based on the requirements of the software, a suitable testing technique is employed.
- ✔ Each testing technique offers various features and benefits to serve the purpose better.
- ✔ Though there are several types of testing techniques available, we shall focus on Black box testing and White box testing.

A. White Box Testing:-

- ✔ White Box Testing is software testing technique in which internal structure, design and coding of software are tested to verify flow of input-output and to 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.

White Box 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.
- ✔ White box testing involves the testing of the software code for the following:
  - Internal security holes
  - Broken or poorly structured paths in the coding processes
  - The flow of specific inputs through the code
  - Expected output
  - The functionality of conditional loops
  - Testing of each statement, object, and function on an individual basis

White Box Testing …. Contd.

- Using white box testing methods, one can derive test cases that
  a) Guarantee that all independent paths within a module have been exercised at least once.
  b) Exercise all logical decisions on their true and false sides.
  c) Execute all loops at their boundaries and within their operational bounds.
  d) Exercise internal data structures to ensure their validity.

- White box testing method is applicable to the following levels of software testing:
  - It is mainly applied to Unit testing and Integration testing.
    - ✔ Unit Testing:- For testing paths within a unit.
    - ✔ Integration Testing:- For testing paths between units.
    - ✔ System Testing:- For testing paths between subsystems.

White Box Testing …. Contd.

- **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 which 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.

White Box Testing Methods

- Following are the white box testing methods:-
  - Basis Path Testing
  - Control Structure Testing

- **Basis Path Testing**
  - Any software program includes, multiple entry and exit points.
  - Testing each of these points is a challenging as well as time-consuming.
  - In order to reduce the redundant tests and to achieve maximum test coverage, basis path testing is used.
    - Path testing is a structural testing method that involves using the source code of a program in order to find every possible executable path.
    - It helps to determine all faults lying within a piece of code.
    - This method is designed to execute all or selected path through a computer program.

Basis Path Testing

- Basis Path Testing in software engineering is a White Box Testing method in which test cases are defined based on flows or logical paths that can be taken through the program.
- The objective of basis path testing is to define the number of independent paths, so the number of test cases needed can be defined explicitly to maximize test coverage.
- In software engineering, Basis path testing involves execution of all possible blocks in a program and achieves maximum path coverage with the least number of test cases.
- It is a hybrid method of branch testing and path testing methods (statement coverage and branch coverage).

Basis Path Testing …. Contd.

- Basis Path Testing is based on the control structure of a program or a module.
- Using this structure, a control flow graph is prepared and the various possible paths present in the graph are executed as a part of testing.
- Thus, Basis path testing is a technique of selecting the paths in the control flow graph, that provide a basis set of execution paths through the program or module.
- Since this testing is based on the control structure of the program, it requires complete knowledge of the program's structure.
- To design test cases using this technique, four steps are followed :
  a) Construct the Control Flow Graph
  b) Compute the Cyclomatic Complexity of the Graph
  c) Identify the Independent Paths
  d) Design Test cases from Independent Paths
  e) Generate Graph Matrices

Basis Path Testing …. Contd.

1. **Control Flow Graph** –
   - A control flow graph (or simply, flow graph) is a directed graph which represents the control structure of a program or module.
   - A control flow graph (V, E) has V number of nodes/vertices and E number of edges in it.
   - Basic Terminologies associated with the Flow Graph:
     - Node
       - Each flow graph node represents one or more procedural statements.
       - Each node that contains a condition is called a predicate node.
       - A single node encompasses a sequence of process boxes and a decision diamond.

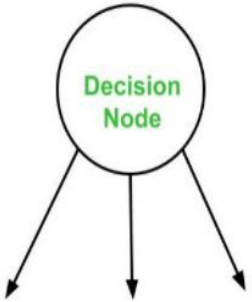
Basis Path Testing …. Contd.

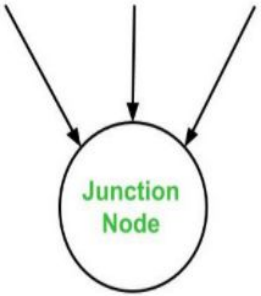
1. **Control Flow Graph** –
    - Edge/ Link
        - Edge is the connection between two nodes.
        - The edges between nodes represent flow of control.
        - An edge must terminate at a node, even if the node does not represent any useful procedural statements.
        - It is denoted by arrows on the flow graph. These arrows are similar to flowchart arrows.
    - Region
        - Region is the area surrounded/ bounded by edges and nodes.
        - When we consider regions, we should include the area outside the graph as a region.

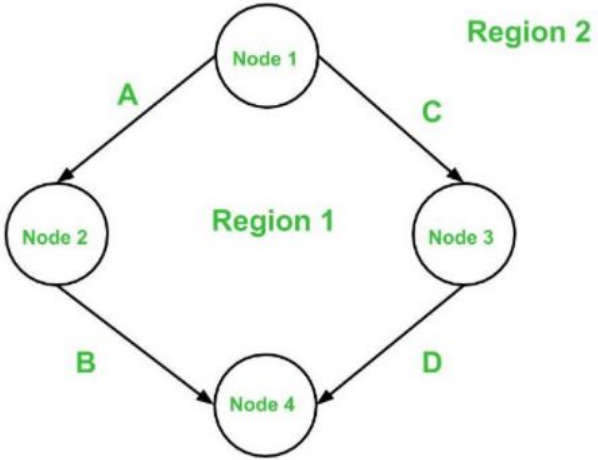
Basis Path Testing …. Contd.

1. **Control Flow Graph** –

- A control graph can also have :
  - ✔ Junction Node – a node with more than one arrow entering it.
  - ✔ Decision Node – a node with more then one arrow leaving it.
  - ✔ Region – area bounded by edges and nodes (area outside the graph is also counted as a region).

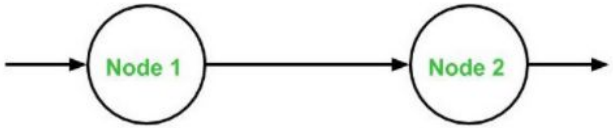




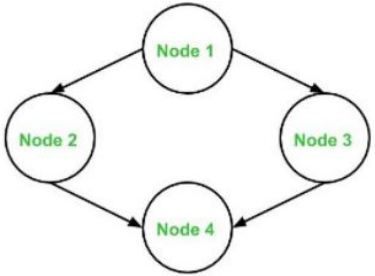


Basis Path Testing …. Contd.

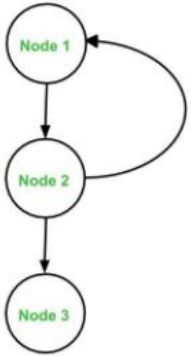
- The **notations** used while constructing a flow graph are:
  - Sequential Statements



  - If – Then – Else

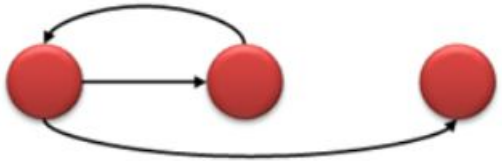


  - Do – While

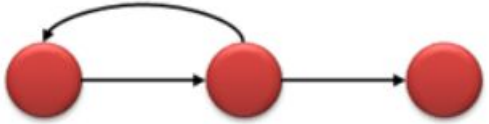


Basis Path Testing …. Contd.

- While

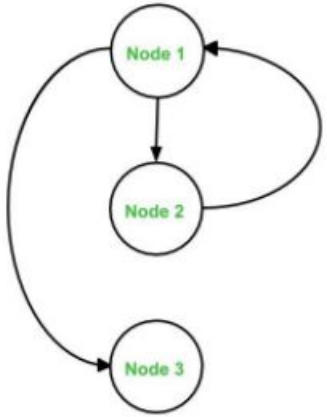


- Until

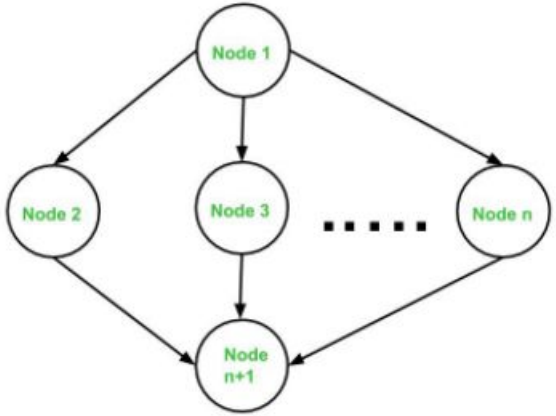


Basis Path Testing …. Contd.

- While – Do



- Switch – Case



Basis Path Testing …. Contd.

2. Cyclomatic Complexity –

- The cyclomatic complexity V(G) is said to be a measure of the logical complexity of a program.
- It can be calculated using three different formulae :

a. *Formula based on edges and nodes* :

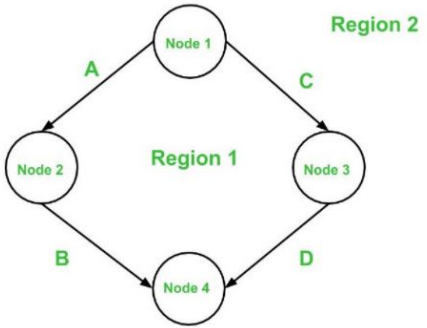
$$V(G) = e - n + 2*P$$

Where,

e is number of edges,

n is number of vertices,

P is number of connected components.



Ex: e = 4, n = 4 and p = 1

So, Cyclomatic complexity V(G)

= 4 - 4 + 2 * 1

= 2

It is a sub graph in which each pair of nodes is connected with each other via a path. Here, P is 1 for all the 3 examples

Basis Path Testing …. Contd.

2. Cyclomatic Complexity –

b. *Formula based on decision nodes* :

$$V(G) = d + P$$

Where,

d is number of decision nodes,

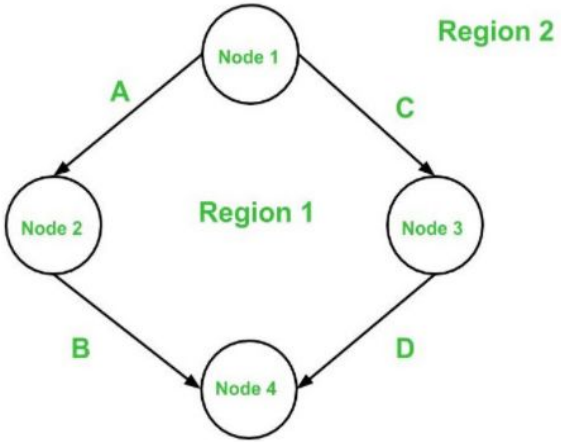
P is number of connected nodes.

Ex: d = 1 and p = 1

So, Cyclomatic complexity V(G)

= 1 + 1

= 2



Basis Path Testing …. Contd.

2. Cyclomatic Complexity –

c. *Formula based on Regions* :

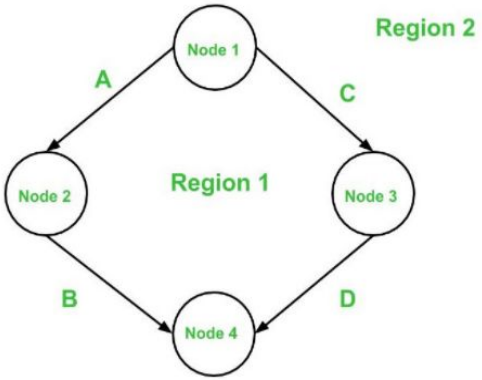
V(G) = number of regions in the graph

Ex:

So, Cyclomatic complexity V(G)

= 1 (for Region 1) + 1 (for Region 2)

= 2



- Hence, using all the three above formulae, the cyclomatic complexity obtained remains same.
- All these three formulae can be used to compute and verify the cyclomatic complexity of the flow graph.

Basis Path Testing …. Contd.

2. Cyclomatic Complexity –

c. *Formula based on Regions* :

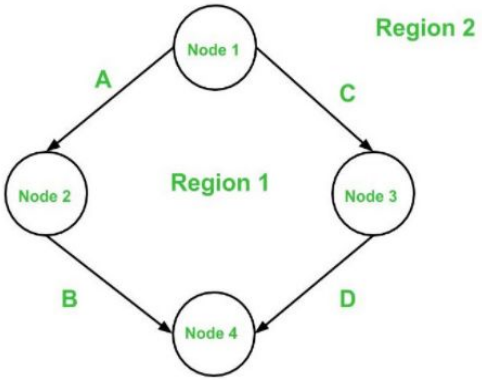
V(G) = number of regions in the graph

Ex:

So, Cyclomatic complexity V(G)

= 1 (for Region 1) + 1 (for Region 2)

= 2



- Hence, using all the three above formulae, the cyclomatic complexity obtained remains same.
- All these three formulae can be used to compute and verify the cyclomatic complexity of the flow graph.

Basis Path Testing …. Contd.

Note:

i. For one function [e.g. Main( ) or Factorial( )], only one flow graph is constructed.

- If in a program, there are multiple functions, then a separate flow graph is constructed for each one of them.
- Also, in the cyclomatic complexity formula, the value of 'p' is set depending of the number of graphs present in total.

ii. If a decision node has exactly two arrows leaving it, then it is counted as one decision node.

- However, if there are more than 2 arrows leaving a decision node, it is computed using this formula :

  $d = k - 1$

  where, k is number of arrows leaving the decision node.

Basis Path Testing …. Contd.

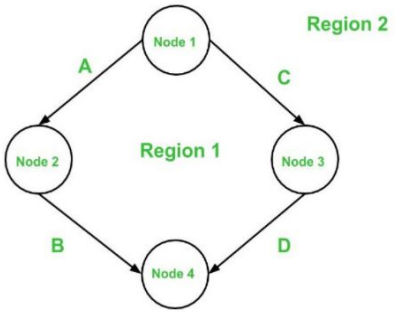
Uses of Cyclomatic Complexity:

- Helps developers and testers to determine independent path executions
- Developers can assure that all the paths have been tested at least once
- Helps us to focus more on the uncovered paths
- Improve code coverage in Software Engineering
- Evaluate the risk associated with the application or program
- Using these metrics early in the cycle reduces more risk of the program

Basis Path Testing …. Contd.

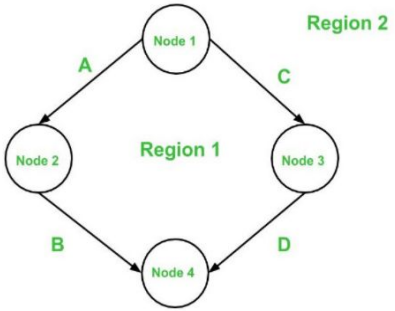
3. Independent Paths –

- An independent path in the control flow graph is the one which introduces at least one new edge that has not been traversed before the path is defined.
- The cyclomatic complexity gives the number of independent paths present in a flow graph.
- This is because the cyclomatic complexity is used as an upper-bound for the number of tests that should be executed in order to make sure that all the statements in the program have been executed at least once.
- Ex.
  - ✔ In this graph, the independent paths would be 2 because number of independent paths is equal to the cyclomatic complexity.



Basis Path Testing …. Contd.

3. Independent Paths –



✔ So, the independent paths will be:

- Path 1:

  A -> B

- Path 2:

  C -> D

Note:

- Independent paths are not unique.
- In other words, if for a graph the cyclomatic complexity comes out be N, then there is a possibility of obtaining two different sets of paths which are independent in nature.

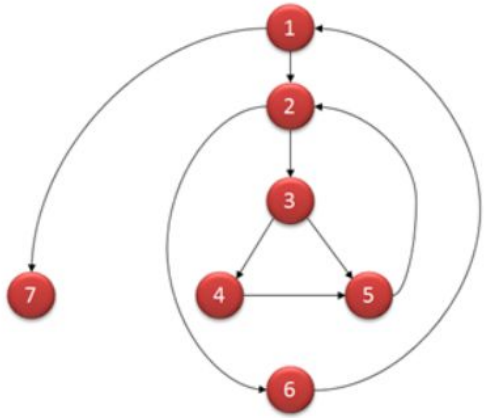
Basis Path Testing …. Contd.

4. Design Test Cases –

- Finally, after obtaining the independent paths, test cases can be designed where each test case represents one or more independent paths.

Ex:

```
i = 0;
n=4;  //N-Number of nodes present in the graph
while (i<n-1) do
j = i + 1;
while (j<n) do
if A[i]<A[j] then
swap(A[i], A[j]);
end do;
i=i+1;
end do;
```



Basis Path Testing …. Contd.

Computing mathematically,

V(G) = 9 – 7 + 2 = 4

V(G) = 3 + 1 = 4 (Condition nodes are 1, 2 and 3 nodes)

Basis Set – A set of possible execution path of a program

Path 1: 1, 7

Path 2: 1, 2, 6, 1, 7

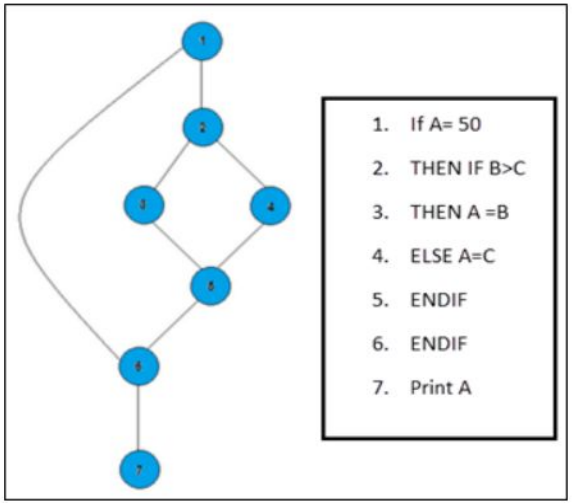
Path 3: 1, 2, 3, 4, 5, 2, 6, 1, 7

Path 4: 1, 2, 3, 5, 2, 6, 1, 7

Following are the properties of Cyclomatic complexity:

1. V (G) is the maximum number of independent paths in the graph
2. V (G) >=1
3. G will have one path if V (G) = 1
4. Minimize complexity to 10

Basis Path Testing …. Contd.



We can see there are few conditional statements that are executed depending on what condition they suffice.

Here, there are 3 paths or condition that need to be tested to get the output,

Path 1: 1,2,3,5,6, 7

Path 2: 1,2,4,5,6, 7

Path 3: 1, 6, 7

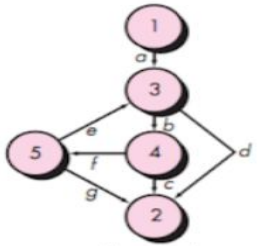
Basis Path Testing …. Contd.

5. Graph Matrices –

- A graph matrix is a tool that supports basis path testing.
- It is a square matrix whose rows and columns are equal to the number of nodes in the flow graph.
- Each row and column identifies a particular node and matrix entries represent a connection between the nodes.
- The following points describe a graph matrix:
  - ✔ Each cell in the matrix can be a direct connection or link between one node to another node.
  - ✔ If there is a connection from node 'a' to node 'b', then it does not mean that there is connection from node 'b' to node 'a'.
  - ✔ Conventionally, to represent a graph matrix, digits are used for nodes and letter symbols for edges or connections.

Basis Path Testing …. Contd.

- The graph matrix is actually a tabular representation of a flow graph.
- If we add link weights to each cell entry, then graph matrix can be used as a powerful tool in testing.
- The links between two nodes are assigned a link weight, which becomes the entry in the cell of matrix.
- The link weight provides information about control flow.
- In the simplest form, when the connection exists, then the link weight is 1 , otherwise 0 (but 0 is not) entered in the cell entry of matrix to reduce the complexity).
- A matrix defined with link weights is called a connection matrix.



Flow graph

Node \ Connected to node	1	2	3	4	5
1			a		
2					
3		d		b	
4		c			f
5		g	e		

Graph matrix

Basis Path Testing …. Contd.

- Connection matrix is used to see the control flow of the program.
- Further, it is used to find the cyclomatic complexity number of the flow graph.
- The procedure to find the cyclomatic number from the connection matrix is as follows:
  i. For each row, count the number of 1s and write it in front of that row.
  ii. Subtract 1 from that count. Ignore the blank rows, if any.
  iii. Add the final count of each row.
  iv. Add 1 to the sum calculated in Step 3.
  v. The final sum in Step 4 is the cyclomatic number of the graph.

Basis Path Testing …. Contd.

Advantages :

- It helps to reduce the redundant tests
- It focuses attention on program logic
- It helps facilitates analytical versus arbitrary case design
- Test cases which exercise basis set will execute every statement in a program at least once.

Basis Path Testing can be applicable in the following cases:

b. More Coverage –

- Basis path testing provides the best code coverage as it aims to achieve maximum logic coverage instead of maximum path coverage.
- This results in an overall thorough testing of the code.

Basis Path Testing …. Contd.

b. Maintenance Testing –
   - When a software is modified, it is still necessary to test the changes made in the software which as a result, requires path testing.

c. Unit Testing –
   - When a developer writes the code, he or she tests the structure of the program or module themselves first.
   - This is why basis path testing requires enough knowledge about the structure of the code.

d. Integration Testing –
   - When one module calls other modules, there are high chances of Interface errors.
   - In order to avoid the case of such errors, path testing is performed to test all the paths on the interfaces of the modules.

Basis Path Testing …. Contd.

e. Testing Effort –

- The basis path testing technique takes into account the complexity of the software (i.e., program or module) while computing the cyclomatic complexity.
- Therefore it is intuitive to note that testing effort in case of basis path testing is directly proportional to the complexity of the software or program.

Control Structure Testing

- **Control Structure Testing**
  - Control structure testing is used to increase the coverage area by testing various control structures present in the program.
  - The different types of testing performed under control structure testing are as follows-
    1) Branch Testing
    2) Condition Testing
    3) Data Flow Testing
    4) Loop Testing

1) Branch Testing

- For every decision, each branch needs to be executed at least once.
- It also called decision testing.
- However, it ignores implicit paths that result from compound conditionals.

Control Structure Testing

- It treats a compound conditional as a single statement. (We count each branch taken out of the decision, regardless which condition lead to the branch.)
- Ex
  - This example has two branches to be executed:

    IF (a equals b) THEN statement 1
    ELSE statement 2
    END IF

  - This examples also has just two branches to be executed, despite the compound conditional:

    IF (a equals b AND c less than d) THEN statement 1
    ELSE statement 2
    END IF

Control Structure Testing

- This example has four branches to be executed:

```
IF ( a equals b) THEN statement 1
ELSE
IF ( c equals d) THEN statement 2
ELSE statement 3
END IF
END IF
```

- Obvious decision statements are if, for, while, switch.
- Subtle decisions are return Boolean expression, ternary expressions, try-catch.
- One does not need to write test cases for IO Exception and Out Of Memory exception.

Control Structure Testing

2) Condition Testing

- Condition testing is a test cased design method, which ensures that the logical condition and decision statements are free from errors.
  - ✔ It is a test construction method that focuses on exercising the logical conditions in a program module.
- The errors present in logical conditions can be incorrect Boolean operators, missing parenthesis in a Boolean expression, error in relational operators, arithmetic expressions, and so on.
- The common types of logical conditions that are tested using condition testing are-
  a. A relation expression, like E1 op E2 where ‘E1’ and ‘E2’ are arithmetic expressions and ‘OP’ is an operator.
  b. A simple condition like any relational expression preceded by a NOT (~) operator.

Control Structure Testing

- Ex:

(~E1) where ‘E1’ is an arithmetic expression and a ‘~’ denotes NOT operator.

d. A compound condition consists of two or more simple conditions, Boolean operator, and parenthesis.

- Ex:

(E1 & E2)|(E2 & E3) where E1, E2, E3 denote arithmetic expression and ‘&’ and ‘|’ denote AND or OR operators.

e. A Boolean expression consists of operands and a Boolean operator like ‘AND’, OR, NOT.

Ex:

‘A|B’ is a Boolean expression where ‘A’ and ‘B’ denote operands and | denotes OR operator.

Control Structure Testing

- Errors in conditions can be due to:
  i. Boolean operator error
  ii. Boolean variable error
  iii. Boolean parenthesis error
  iv. Relational operator error
  v. Arithmetic operation error

- Thus, "For a compound condition C, the true and false branches of C and every simple condition in C need to be executed at least once".
- Multiple-condition testing requires that all true-false combinations of simple conditions be exercised at least once.
- Therefore, all statements, branches, and conditions are necessarily covered.

Control Structure Testing

3) Data Flow Testing

- Data flow testing method is effective for error protection because it is based on the relationship between statements in the program according to the definition and uses of variables.
- Test paths are selected according to the location of definitions and uses of variables in the program.
- It is unrealistic to assume that data flow testing will be used extensively when testing a large system.
  - ✔ However, it can be used in a targeted fashion for areas of software that are suspect.
- This is a somewhat sophisticated technique and is not practical for extensive use.
  - ✔ Its use should be targeted to modules with nested if and loop statements.

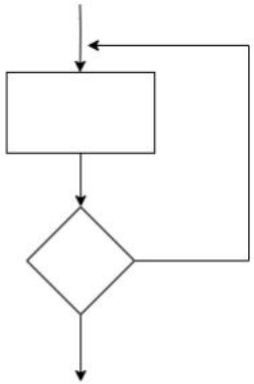
Control Structure Testing

4) Loop Testing

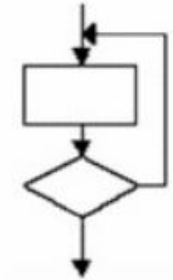
- Loop testing method concentrates on validity of the loop structures.
- Loops are fundamental to many algorithms and need thorough testing.
- Loops can be defined as simple, concatenated, nested, and unstructured.

A. *Simple loop*:

– The following set of test can be applied to simple loops, where the maximum allowable number through the loop is n.



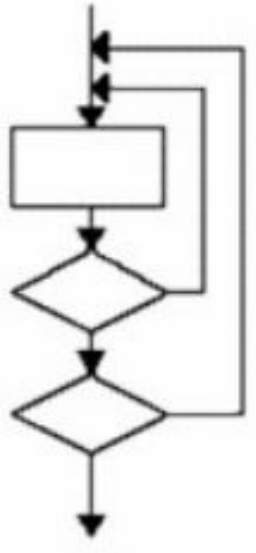
» Skip the entire loop.
» Traverse the loop only once.
» Traverse the loop two times.
» Make p passes through the loop where p<n.
» Traverse the loop n-1, n, n+1 times.



Control Structure Testing

B. *Nested loop*:

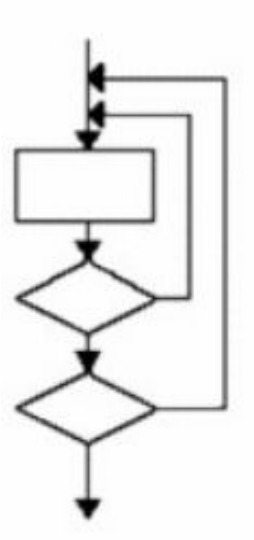
- Loops within loops are called as nested loops.
- When testing nested loops, the number of tested increases as level nesting increases.
- The following steps for testing nested loops are:
  - » Start with inner loop.
  - » Set all other loops to minimum values.
  - » Conduct simple loop testing on inner loop.
  - » Work outwards.
  - » Continue until all loops tested.



Control Structure Testing

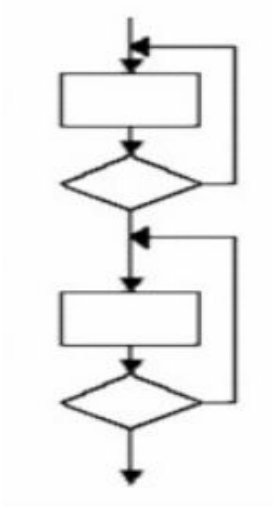
C. *Nested loop*:

- Loops within loops are called as nested loops.
- When testing nested loops, the number of tested increases as level nesting increases.
- The following steps for testing nested loops are:
  - » Start with inner loop.
  - » Set all other loops to minimum values.
  - » Conduct simple loop testing on inner loop.
  - » Work outwards.
  - » Continue until all loops tested.

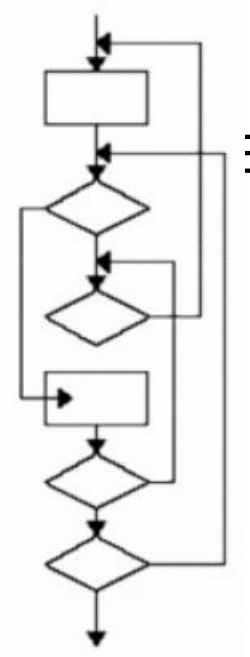


Control Structure Testing

D. *Concatenated loop*:
- If loops are not dependent on each other, contact loops can be tested using the approach used in simple loops.
- If the loops are interdependent, treat as nested loops.



E. *Unstructured loop*:
- This type of loop should be not be tested.
- Rather, redesign, whenever possible, to reflect the use of unstructured the structured programming constructs.



Control Structure Testing

- **Advantages:**
  - It provides thorough testing of the software.
  - It helps in finding out defects at an early stage.
  - It helps in elimination of dead code.
  - It is not time consuming as it is mostly automated.

- **Disadvantages:**
  - It requires knowledge of the code to perform test.
  - It requires training in the tool used for testing.
  - Sometimes it is expensive.

Testing Techniques

B. Black Box Testing:-

- ✔ The black box is a powerful technique to check the application under test from the user's perspective.
- ✔ Black box testing is used to test the system against external factors responsible for software failures.
- ✔ This testing approach focuses on the input that goes into the software, and the output that is produced.
- ✔ The testing team does not cover the inside details such as code, server logic, and development method.
  - It is a type of software testing in which the functionality of the software is not known.
  - The testing is done without the internal knowledge of the products.

Black Box Testing

- ✔ The black box testing method is mainly used to find missing functions, performance errors, initialization errors, and errors while accessing the external database.
- ✔ Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications.
  - The primary source of black box testing is a specification of requirements that is stated by the customer.
- ✔ It is also known as Behavioral Testing.

Black Box Testing …. Contd.

- ✔ Various parameters checked in black box testing are:
  - Accurate actions performed by users
  - System's interaction with the inputs
  - The response time of the system
  - Use of data structures Issues in the user interface
  - Usability issues
  - Performance issues
  - Abrupt application failure, unable to start or finish

Black Box Testing …. Contd.

- Following are the generic steps that are carried out for any type of Black Box Testing:
  - ✔ Initially, the requirements and specifications of the system are examined.
  - ✔ Tester chooses valid inputs (positive test scenario) to check whether software under test (SUT) processes them correctly.
  - ✔ Also, some invalid inputs (negative test scenario) are chosen to verify that the SUT is able to detect them.
  - ✔ Tester determines expected outputs for all those inputs.
  - ✔ Software tester constructs test cases with the selected inputs.
  - ✔ The test cases are executed.
  - ✔ Software tester compares the actual outputs with the expected outputs.
  - ✔ Defects if any are fixed and re-tested.

Black Box Testing …. Contd.

- ✔ Types of Black Box Testing
  - a) Functional testing –
    - It is concerned only with the functional requirements of a system and covers how well the system executes its functions.
    - It is done by software testers.
  - b) Non-functional testing –
    - This type of black box testing is not related to testing of specific functionality, but non-functional requirements such as performance, scalability, usability.
    - It is designed specifically to evaluate the readiness of a system according to the various criteria which are not covered by functional testing.

Black Box Testing …. Contd.

c) Regression testing –

- Regression Testing is done after code fixes, upgrades or any other system maintenance to check the new code has not affected the existing code.

Black Box Testing …. Contd.

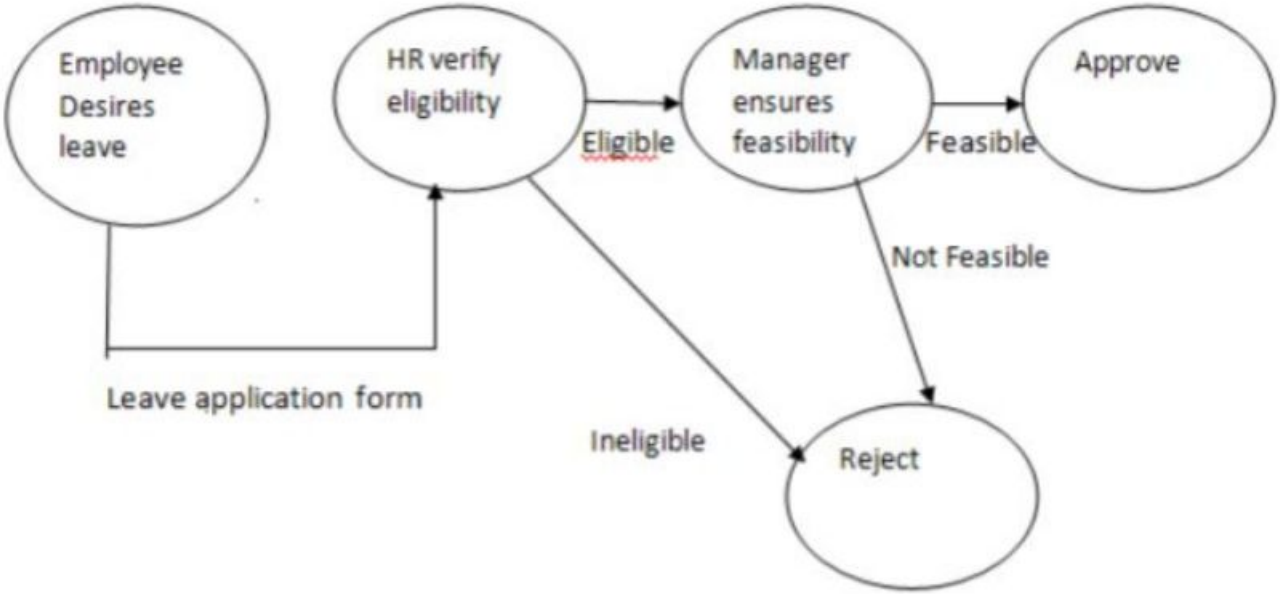
- ✔ Black Box Testing Techniques:-
  - **a) Graph Based Testing**:
    - It is also called as State Based Testing.
    - It provides a framework for model based testing.
    - Black-box methods based on the nature of the relationships (links) among the program objects (nodes), test cases are designed to traverse the entire graph.
    - Transaction flow testing – nodes represent steps in some transaction and links represent logical connections between steps that need to be validated.
    - Finite state modeling – nodes represent user observable states of the software and links represent transitions between states.
    - Data flow modeling – nodes are data objects and links are transformations from one data object to another.

Black Box Testing …. Contd.

- Timing modeling – nodes are program objects and links are sequential connections between these objects, link weights are required execution times.
- **Steps in graph testing:**
  a. Build a graph model.
  b. Identify the test requirements.
  c. Select test paths to cover those requirements.
- In order to design test design cases following steps are used:
  - Understanding the system
  - Identifying states, inputs and guards.
  - Create a state graph model of the application.
  - Verify whether State graph that we modeled is correct in all details.
  - Generate sequence of test actions.
  - Derive test data so that those test paths can be executed.

Black Box Testing …. Contd.

- Derive test data so that those test paths can be executed.
- Ex:



Black Box Testing …. Contd.

b) Equivalence Class Testing:

- It is also called as Equivalence Class Partitioning Testing (ECP).
- It is used to minimize the number of possible test cases to an optimum level while maintaining reasonable test coverage.
- The input data is divided into partitions of valid and invalid values, and it is mandatory that all partitions must exhibit the same behavior.
- To cover maximum requirements, Equivalence Partitioning uses the minimum test cases.
- This technique ensures to cover each partition at least once.

Black Box Testing …. Contd.

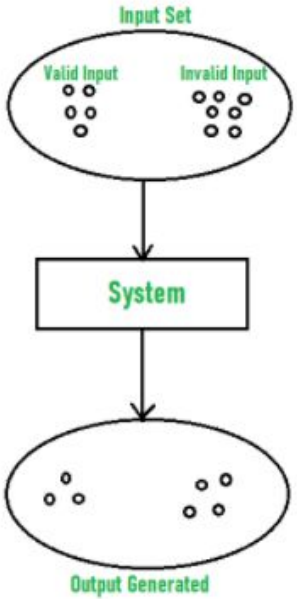
- And each value of every equal partition must display the same behavior as the other.
- In equivalence portioning, equivalence classes are evaluated for given input conditions.
- Whenever any input is given, then type of input condition is checked, then for this input conditions, Equivalence class represents or describes set of valid or invalid states.

- Guidelines for Equivalence Partitioning :
  i. If the range condition is given as an input, then one valid and two invalid equivalence classes are defined.
  ii. If a specific value is given as input, then one valid and two invalid equivalence classes are defined.

Black Box Testing …. Contd.

ii. If a member of set is given as an input, then one valid and one invalid equivalence class is defined.

iii. If Boolean no. is given as an input condition, then one valid and one invalid equivalence class is defined.



Black Box Testing …. Contd.

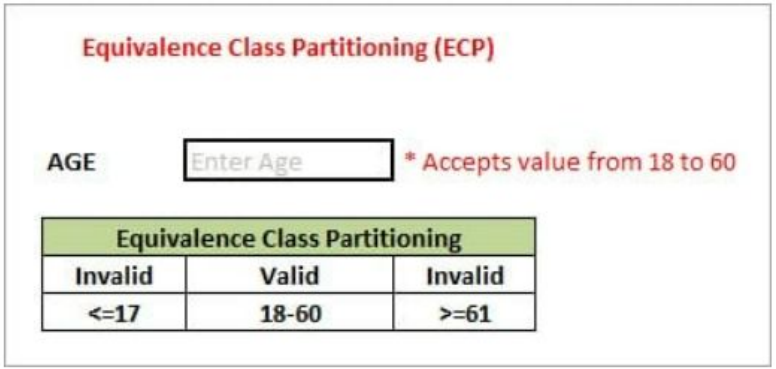
- Ex. College admission process that gives admissions to students based upon their percentage.
  - Let us consider that percentage field will accept percentage only between 50 to 90 %, more and even less than not be accepted, and application will redirect user to error page.
  - If percentage entered by user is less than 50 % or more than 90 %, it will show an invalid percentage.
  - If percentage entered is between 50 to 90 %, then equivalence partitioning method will show valid percentage.

Equivalence Partitioning		
Invalid	Valid	Invalid
<=50	50-90	>=90

Black Box Testing …. Contd.

In this image, the "AGE" text field accepts only numbers from 18 to 60. There will be three sets of classes or groups.

- Two invalid classes will be:
  a) Less than or equal to 17.
  b) Greater than or equal to 61.
- A valid class will be anything between 18 and 60.
- We have thus reduced the test cases to only 3 test cases based on the formed classes thereby covering all the possibilities.
- So, testing with any one value from each set of the class is sufficient to test the above scenario.



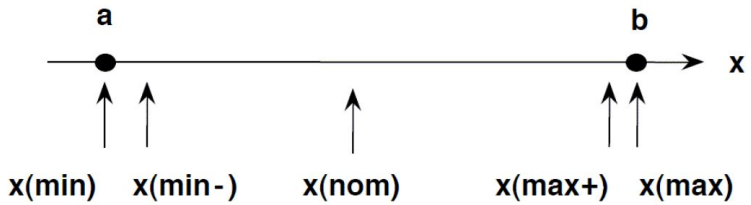
Black Box Testing …. Contd.

c) **Boundary Value Testing**:

- The name itself defines that in this technique, we focus on the values at boundaries as it is found that many applications have a high amount of issues on the boundaries.
- Boundary refers to values near the limit where the behavior of the system changes.
- In boundary value analysis, both valid and invalid inputs are being tested to verify the issues.
- It determines whether a certain range of values are acceptable by the system or not.
- It is very useful in reducing the number of test cases.
- It is most suitable for the systems where an input is within certain ranges.

Black Box Testing …. Contd.

- Boundary testing is the process of testing between extreme ends or boundaries between partitions of the input values.
- So these extreme ends like Start- End, Lower- Upper, Maximum-Minimum, Just Inside-Just Outside values are called boundary values and the testing is called “boundary testing”.
- The basic idea in normal boundary value testing is to select input variable values at their:
  - Minimum
  - Just above the minimum
  - A nominal value
  - Just below the maximum
  - Maximum



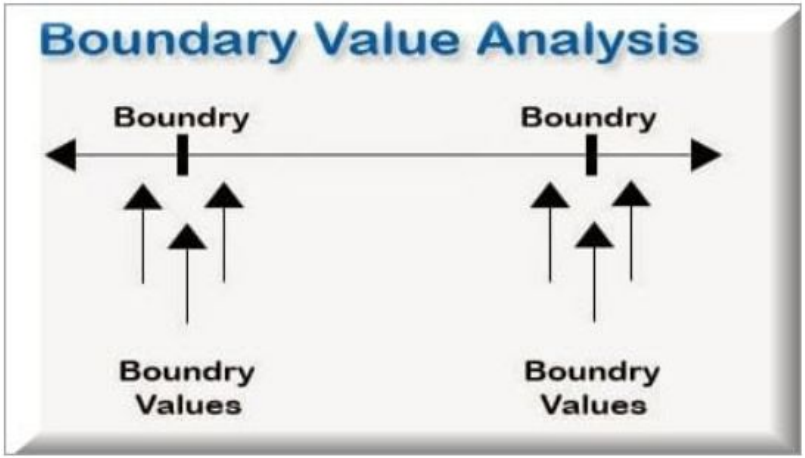
- It comes after Equivalence Class Partitioning.

Black Box Testing …. Contd.

Ex.

If we want to test a field where values from 1 to 100 should be accepted, then we choose the boundary values: 1-1, 1, 1+1, 100-1, 100, and 100+1.

Instead of using all the values from 1 to 100, we just use 0, 1, 2, 99, 100, and 101.



AGE Enter Age * Accepts Value 21 to 65

Boundary Value Test Case		
Invalid Test Case (Min value - 1)	Valid Test Case (Min + Min, Max, -Max	Invalid Test Case (Max value - 1)
20	21,22,65,64	66

Equivalence Testing Vs Boundary Value Testing

Boundary value analysis	Equivalence partitioning
It is a technique where we identify the errors at the boundaries of input data to discover those errors in the input center.	It is a technique where the input data is divided into partitions of valid and invalid values.
Boundary values are those that contain the upper and lower limit of a variable.	In this, the inputs to the software or the application are separated into groups expected to show similar behavior.
Boundary value analysis is testing the boundaries between partitions.	It allows us to divide a set of test conditions into a partition that should be considered the same.
It will help decrease testing time due to a lesser number of test cases from infinite to finite.	The Equivalence partitioning will reduce the number of test cases to a finite list of testable test cases covering maximum possibilities.
The Boundary Value Analysis is often called a part of the Stress and Negative Testing.	The Equivalence partitioning can be suitable for all the software testing levels such as unit, integration, system.
Sometimes, it is also known as Range Checking.	It is also known as Equivalence class partitioning.

Software Maintenance

- Software does not wear out or get tired.
- However, it needs to be upgraded and enhanced to meet new user requirements.
- For such modifications in the software system, software maintenance is performed.
  - Software maintenance is the process of changing, modifying, and updating software to keep up with customer needs.
  - It is done after the product has launched for several reasons including improving the overall software, correcting issues or bugs, to boost performance, and more.
  - The objective is to ensure that the software is able to accommodate changes after the system has been delivered and deployed.

Need for Software Maintenance

- As technology is changing at the speed of light, software must keep up with the market changes and demands.
- Software Maintenance must be performed in order to:
  a. Correct faults.
  b. Accommodate change in user requirement with time
  c. Improve the design.
  d. Implement enhancements.
  e. Adapt to change in hardware/software requirements.
  f. Improve system efficiency.
  g. Reduce any unwanted side effects.
  h. Accommodate programs so that different hardware, software, system features, and telecommunications facilities can be used.
  i. Optimize the code to run faster
  j. Migrate legacy software.

Types of Software Maintenance

- The four different types of software maintenance are each performed for different reasons and purposes.
- A given piece of software may have to undergo one, two, or all types of maintenance throughout its lifespan.
- The four types are:
  I. Corrective Software Maintenance
  II. Preventive Software Maintenance
  III. Perfective Software Maintenance
  IV. Adaptive Software Maintenance

I. **Corrective Software Maintenance:**
   - Corrective software maintenance is associated with the maintenance of any kind.

Corrective Maintenance

- Defect in the software arises due to errors and faults in design, logic, and code of the software.
- Corrective maintenance action (commonly referred to as “bug fixing”) addresses these errors and faults in your software system.
- Corrective changes in the software are required when:
  - Software is not working the way it is expected due to some acute issues, such as faulty logic flow, incorrect implementation, invalid or incomplete tests, etc.
  - The issues in the software are affecting users after you have released the software.
- Corrective software maintenance addresses the errors and faults within software applications that could impact various parts of your software, including the design, logic, and code.

Corrective Maintenance ….

- These corrections usually come from bug reports that were created by users or customers – but corrective software maintenance can help to spot them before your customers do, which can help your brand's reputation.
- Most commonly, bug reports are created by users and sent as feedback to the company that designed the software.
- Then the company's developers and testers review the code and make corrective changes to the software accordingly.
- Its purpose is to enhance and improve the software.

- If you detect and resolve the flaws in the software before users discover it, then the maintenance action is preventive or adaptive.
- However, if you fix the problem after getting bug reports from the user's side, then it is corrective maintenance action.

Preventive Maintenance

II. **Preventive Software Maintenance:**

- It is the process by which we prevent our system from being obsolete.
- It involves the concept of reengineering and reverse engineering in which an old system with old technology is re-engineered using new technology.
- This maintenance prevents the system from dying out.
- This type of maintenance is about looking into the future so that your software can keep working as desired for as long as possible.
- This service helps in preventing the system from any forthcoming vulnerabilities.

Preventive Maintenance …..

- Preventive maintenance defines improvements of the software, which is done to safeguard the software for the future.
- It is carried out to prevent the product from any potential software alteration.
- Preventive maintenance also makes it easier to scale or maintain your code and handle your legacy system.
- Preventive maintenance also makes it easier to scale or maintain your code and handle your legacy system.
- Ex.
  - A vaccine to prevent you from getting the flu.
  - A vaccine to prevent severe infection from corona virus

Perfective Maintenance

III. Perfective Software Maintenance:

- Perfective software maintenance focuses on the evolution of requirements and features that existing in your system.
- As users interact with your applications, they may notice things that you did not or suggest new features that they would like as part of the software, which could become future projects or enhancements.
- It takes over some of the work, both adding features that can enhance user experience and removing features that are not effective and functional.
  - ✔ This can include features that are not used or those that do not help you to meet your end goals.
- It defines improving processing efficiency or performance or restricting the software to enhance changeability.

Perfective Maintenance …..

- ✔ This may contain enhancement of existing system functionality, improvement in computational efficiency, etc.
- Perfective software maintenance aims to adjust software by adding new features as necessary and removing features that are irrelevant or not effective in the given software.
- This process keeps software relevant as the market, and user needs, change.
- Perfective maintenance is important in order to improve a system efficiency, reliability or maintainability.

V. Adaptive Software Maintenance:

- Adaptive software maintenance has to do with the changing technologies as well as policies and rules regarding your software.

Adaptive Maintenance

- ✔ These include operating system changes, cloud storage, hardware, etc.
- When these changes are performed, your software must adapt in order to properly meet new requirements and continue to run well.
- It is the process of conversion in the system to keep the software compatible with changing business needs and technical evolution.
- This type of software maintenance primarily focuses on software frameworks.
- Adaptive maintenance is made in response to new operating systems, platforms, and hardware to retain continuity with the software.

Adaptive Maintenance …..

- This type of maintenance often occurs as a result of external influences or strategic changes within the company.
- The system is being adapted to remain up to date.
- Ex.
  1. The Government recently changed the VAT rate from 17.5% to 20%. This change meant that many organizations had to make alterations to their systems.
  2. A bank decides to offer a new mortgage product. This will have to be included in the system so that mortgage interest and payments can be calculated.
  3. A company has introduced an online system for customers to place orders. The online system needs to be integrated into their normal ordering system.

Software Engineering

- Software engineering covers not only the technical aspects of building software systems, but also management issues, such as directing programming teams, scheduling, and budgeting.
- It is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures.
  - The outcome of software engineering is an efficient and reliable software product.
- Software project management has wider scope than software engineering process as it involves communication, pre and post-delivery support etc.
- The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working:-

Software Engineering …. Contd.

- Large software-
  - It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.

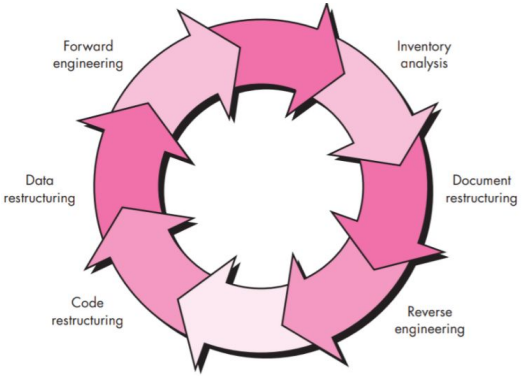
- Scalability-
  - If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.

- Cost-
  - As hardware industry has shown its skills and huge manufacturing has lower down the price of computer and electronic hardware.
  - But the cost of software remains high if proper process is not adapted.

Software Engineering …. Contd.

- Dynamic Nature-
  - The always growing and adapting nature of software hugely depends upon the environment in which user works.
  - If the nature of software is always changing, new enhancements need to be done in the existing one.
  - This is where software engineering plays a good role.

- Quality Management-
  - Better process of software development provides better and quality software product.



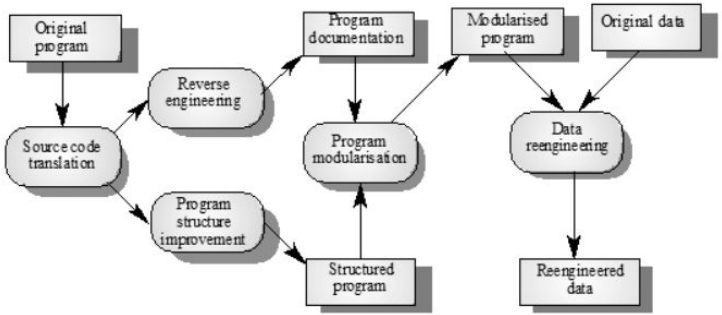
Software Re-Engineering

- Software Reengineering is a process of software development which is done to improve the maintainability of a software system.
  - It is the examination and alteration of a system to reconstitute it in a new form.
  - The principles of Re-Engineering when applied to the software development process is called software re-engineering.
  - It affects positively at software cost, quality, service to the customer and speed of delivery.
  - In Software Re-engineering, we are improving the software to make it more efficient and effective.

- Software Re-engineering is restructuring or rewriting part or all of a system without changing its functionality.
- It is applicable when some (but not all) subsystems of a larger system require frequent maintenance.

Software Re-Engineering …..

- Reengineering involves putting in the effort to make it easier to maintain.
- The reengineered system may also be restructured and should be re-documented
- When do you decide to reengineer?
  - When system changes are confined to one subsystem, the subsystem needs to be reengineered.
  - When hardware or software support becomes obsolete.
  - When tools to support restructuring are readily available.



Software Re-Engineering …..

- **Economics of Reengineering:**
  - Cost of maintenance:
    - Annual cost of operation and maintenance over application lifetime.
  - Cost of reengineering:
    - Predicted return on investment reduced by cost of implementing changes and engineering risk factors
  - Cost benefit:
    - Cost of re engineering - Cost of maintenance

- **Re-engineering advantages**:
  - Reduced risk.
  - There is a high risk in new software development. There may be development problems, staffing problems and specification problems.

Software Re-Engineering …..

- Reduced cost.
- The cost of re-engineering is often significantly less than the costs of developing new software.

- The complete Software Re-Engineering lifecycle includes:
  - Product Management:
    - Risks analysis, root cause analysis, business analysis, requirements elicitation and management, product planning and scoping, competitive analysis.
  - Research and Innovation:
    - Definition of a problem, data gathering and analysis, identifying a solution and developing best-of-breed or innovative algorithms, verification of quality for data and results, patent preparation.

Software Re-Engineering …..

- Product Development:
  - Technology analysis and selection, software architecture and design, data architecture, deployment architecture, prototyping and production code development, comprehensive software testing, data quality testing, and product packaging and deployment preparation

- Product Delivery and Support:
  - Hardware/Platform analysis and selection, deployment and release procedures definition, installations and upgrades, tracking support issues, organizing maintenance releases.

- Project Management:
  - Brings efficiency and productivity to your software re-engineering project by utilizing modern, practical software project management, software quality assurance, data quality assurance, and advanced risk management techniques.

Software Reverse Engineering

- Reverse engineering is taking apart an object to see how it works in order to duplicate or enhance the object.
- The practice, taken from older industries, is now frequently used on computer hardware and software.
- Software reverse engineering involves reversing a program's machine code (the string of 0s and 1s that are sent to the logic processor) back into the source code that it was written in, using program language statements.

- Reverse-engineering is used for many purposes:
  - as a learning tool;
  - as a way to make new, compatible products that are cheaper than what i currently on the market;
  - for making software interoperate more effectively or to bridge data between different operating systems or databases; and
  - to uncover the undocumented features of commercial products.

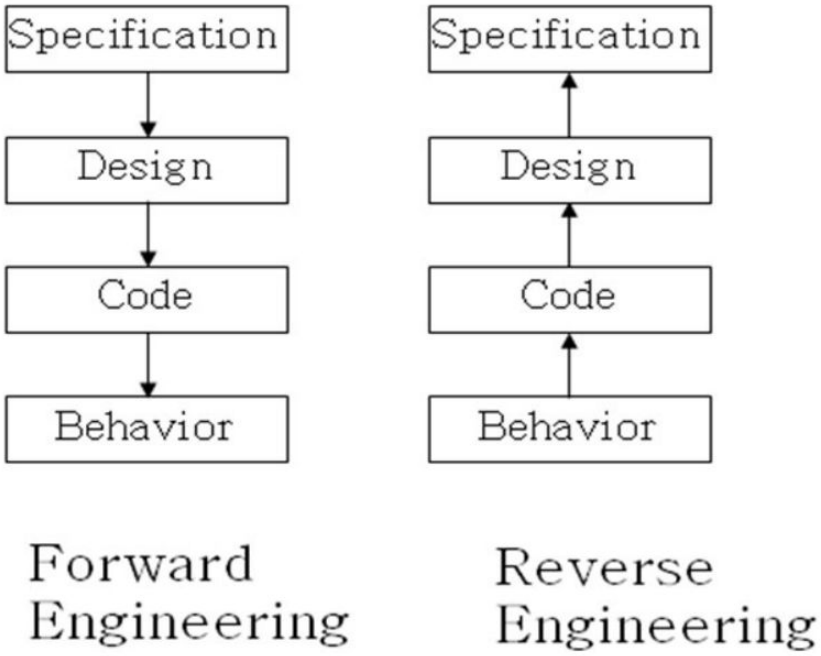
Software Reverse Engineering ….

- It is a process to achieve system specification by thoroughly analyzing, understanding the existing system.
  - This process can be seen as reverse SDLC model, i.e. we try to get higher abstraction level by analyzing lower abstraction levels.
- An existing system is previously implemented design, about which we know nothing.
  - Designers then do reverse engineering by looking at the code and try to get the design.
- With design in hand, they try to conclude the specifications. Thus, going in reverse from code to system specification.

- Reverse engineering, in computer programming, is a technique used to analyze software in order to identify and understand the parts it is composed of.

Software Reverse Engineering ….

- The usual reasons for reverse engineering a piece of software are to recreate the program, to build something similar to it, to exploit its weaknesses or strengthen its defenses.



Software Reverse Engineering

- **Program Restructuring**:-
  - It is a process to re-structure and re-construct the existing software.
  - It is all about re-arranging the source code, either in same programming language or from one programming language to a different one.
  - Restructuring can have either source code-restructuring and data-restructuring or both.

- Re-structuring does not impact the functionality of the software but enhance reliability and maintainability.
- Program components, which cause errors very frequently can be changed, or updated with re-structuring.
- The dependability of software on obsolete hardware platform can be removed via re-structuring.

Software Reverse Engineering

- **Reasons for reverse engineering:-**
  - Interfacing
    - Reverse engineering can be used when a system is required to interface to another system and how both systems would negotiate is to be established.
  - Military or commercial espionage
    - Learning about an enemy's or competitor's latest research by stealing or capturing a prototype and dismantling it.
  - Improve documentation shortcomings
    - Reverse engineering can be done when documentation of a system for its design, production, operation or maintenance have shortcomings and original designers are not available.

Software Reverse Engineering

- Obsolescence
  - Integrated circuits are often designed on proprietary systems, and built on production lines which become obsolete in only a few years.

- Software modernization
  - Often knowledge is lost over time, which can prevent updates and improvements.
  - Reverse engineering is generally needed in order to understand the 'as is' state of existing or legacy software in order to properly estimate the effort required to migrate system knowledge into a 'to be' state.

Software Reverse Engineering

- Product security analysis
  - To examine how a product works, what are specifications of its components, estimate costs and identify potential patent infringement.

- Bug fixing
  - To fix (or sometimes to enhance) legacy software which is no longer supported by its creators (e.g. abandonware).

Software Reverse Engineering

- Following are reasons for reverse engineering a part or product:
  1. The original manufacturer of a product no longer produces a product.
  2. There is inadequate documentation of the original design.
  3. The original manufacturer no longer exists, but a customer needs the product.
  4. The original design documentation has been lost or never existed.
  5. Some bad features of a product need to be designed out.
     - For example, excessive wear might indicate where a product should be improved.
  6. To strengthen the good features of a product based on long-term usage of the product.
  7. To analyze the good and bad features of competitors' product.

Software Reverse Engineering

8. To explore new avenues to improve product performance and features.
9. To gain competitive benchmarking methods to understand competitor's products and develop better products.
10. The original Computer Aided Design (CAD) model is not sufficient to support modifications or current manufacturing methods.
11. The original supplier is unable or unwilling to provide additional parts.
12. The original equipment manufacturers are either unwilling or unable to supply replacement parts, or demand inflated costs for sole-source parts.
13. To update obsolete materials or antiquated manufacturing processes with more current, less-expensive technologies.