Unit-6
Software Testing and Quality Assurance

Software Testing

- Testing is intended to show that a program does what it is intended to do and to discover program defects before it is put into use.
- When you test software, you execute a program using artificial data.
- You check the results of the test run for errors, anomalies or information about the program’s non-functional attributes.
- Can reveal the presence of errors NOT their absence.
- Testing is part of a more general verification and validation process, which also includes static validation techniques.
- **Software Testing** is a method to check whether the actual software product matches expected requirements and to ensure that software product is Defect free.
- According to **ANSI/IEEE 1059** standard – Software Testing is a process of analyzing a software item to detect the differences between existing and required conditions (i.e., defects) and to evaluate the features of the software item.

Why testing is needed?

- Testing is important because software bugs could be expensive or even dangerous. Software bugs can potentially cause monetary and human loss, and history is full of such examples.
  - In April 2015, Bloomberg terminal in London crashed due to software glitch affected more than 300,000 traders on financial markets. It forced the government to postpone a 3bn pound debt sale.
  - Nissan cars recalled over 1 million cars from the market due to software failure in the airbag sensory detectors. There has been reported two accident due to this software failure.
  - Starbucks was forced to close about 60 percent of stores in the U.S and Canada due to software failure in its POS system. At one point, the store served coffee for free as they were unable to process the transaction.
  - Some of Amazon's third-party retailers saw their product price is reduced to 1p due to a software glitch. They were left with heavy losses.
  - Vulnerability in Windows 10. This bug enables users to escape from security sandboxes through a flaw in the win32k system.
  - In 2015 fighter plane F-35 fell victim to a software bug, making it unable to detect targets correctly.
  - China Airlines Airbus A300 crashed due to a software bug on April 26, 1994, killing 264 innocents live
  - In 1985, Canada's Therac-25 radiation therapy machine malfunctioned due to software bug and delivered lethal radiation doses to patients, leaving 3 people dead and critically injuring 3 others.
  - In April of 1999, a software bug caused the failure of a $1.2 billion military satellite launch, the costliest accident in history
  - In May of 1996, a software bug caused the bank accounts of 823 customers of a major U.S. bank to be credited with 920 million US dollars.
- Testing provides cost effectiveness.
- Testing ensure performance, trust, quality, security and reliability.
- Testing increases the customer satisfaction
- Testing ensure correctness of the output.
- Testing gives assurance of the actual or correct functionality of the software and hardware

Objectives of Testing

- Testing is the process of executing a program with the intent of finding an error.
- Software quality improvement, verification and validation and software reliability estimation are the major objectives of testing.
- Testing objective is to find the uncover or undiscovered fault, defect and inconsistencies.

Software Testing Principles

- All tests should be based on customer requirements
- Test should be planned long before testing begins
- Testing should begin “in small” and progress towards testing “in large”
- Testing shows presence of defects
- Exhaustive testing is not possible i.e It is impossible to test everything
- Testing time and resources are limited .So use best resources for the test and avoid the redundant tests.
- Keep software static during testing.
- Document test cases and test results.
- Early testing
- Use Defect clustering and modular testing approaches
- Testing should be context dependent

- **Verification and Validation(V&V)**
- *Verification* refers to the set of tasks that ensure that software correctly implements a specific function.
- *Validation* refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements. Boehm [Boe81] states this another way:
  - *Verification:* "Are we building the product right?" i.e The software should conform to its specification.
  - *Validation:* "Are we building the right product?“ i. e. The software should do what the user really requires.
- Verification and validation should establish confidence that the software is fit for purpose.
- This does NOT mean completely free of defects.
- Rather, it must be good enough for its intended use and the type of use will determine the degree of confidence that is needed.
- Is a whole life-cycle process - V & V must be applied at each stage in the software process.
- Has two principal objectives
  - The discovery of defects in a system;
  - The assessment of whether or not the system is useful and useable in an operational situation.

Difference between Verification and Validation

Verification	**Validation**
Verification is the process to find whether the software meets the specified requirements for particular phase.	The validation process is checked whether the software meets requirements and expectation of the customer.
Verification: "Are we building the product right?"	*Validation:* "Are we building the right product?“
The objectives of verification is to check whether software is constructed according to requirement and design specification.	The objectives of the validation is to check whether the specifications are correct and satisfy the business need.
It describes whether the outputs are as per the inputs or not.	It explains whether they are accepted by the user or not.
Verification is done before the validation.	It is done after the verification.
Plans, requirement, specification, code are evaluated during the verifications.	Actual product or software is tested under validation.
It manually checks the files and document.	It is a computer software or developed program based checking of files and document.
It estimates an intermediate product.	It estimates the final product.

Validation Testing

- Fig :Validation testing is the process of ensuring if the tested and developed software satisfies the client /user needs. The business requirement logic or scenarios have to be tested in detail. All the critical functionalities of an application must be tested here.
- 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 organisation.
- **Why 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*

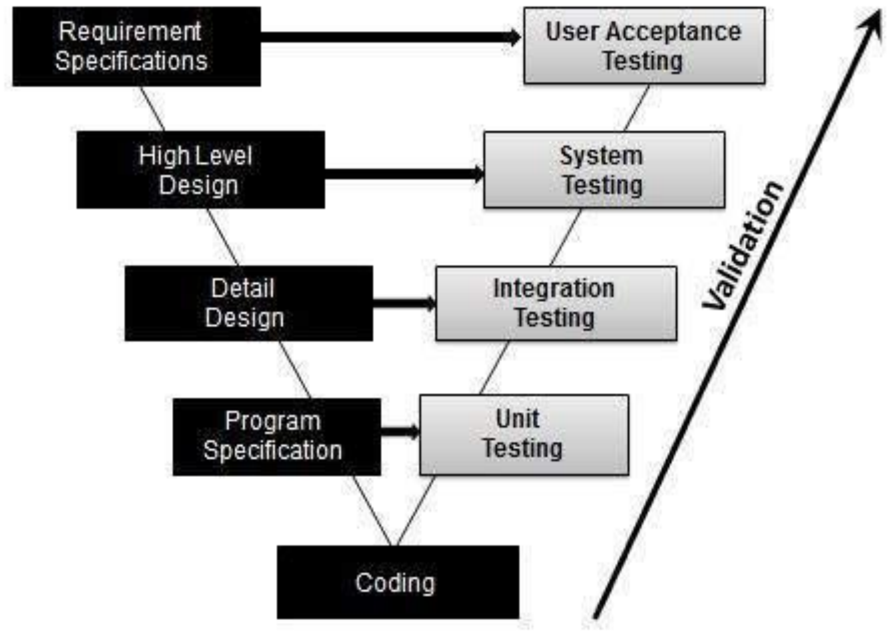


Fig :Validation testing using V and V model

Stages of Validation testing Process:

•**Validation Planning** – To plan all the activities that need to be included while testing.

•**Define Requirements** – To set goals and define the requirements for testing.

•**Selecting a Team** – To select a skilled and knowledgeable development team (the third party included).

•**Developing Documents** – To develop a user specification document describing the operating conditions.

•**Estimation/Evaluation** – To evaluate the software as per the specifications and submit a validation report.

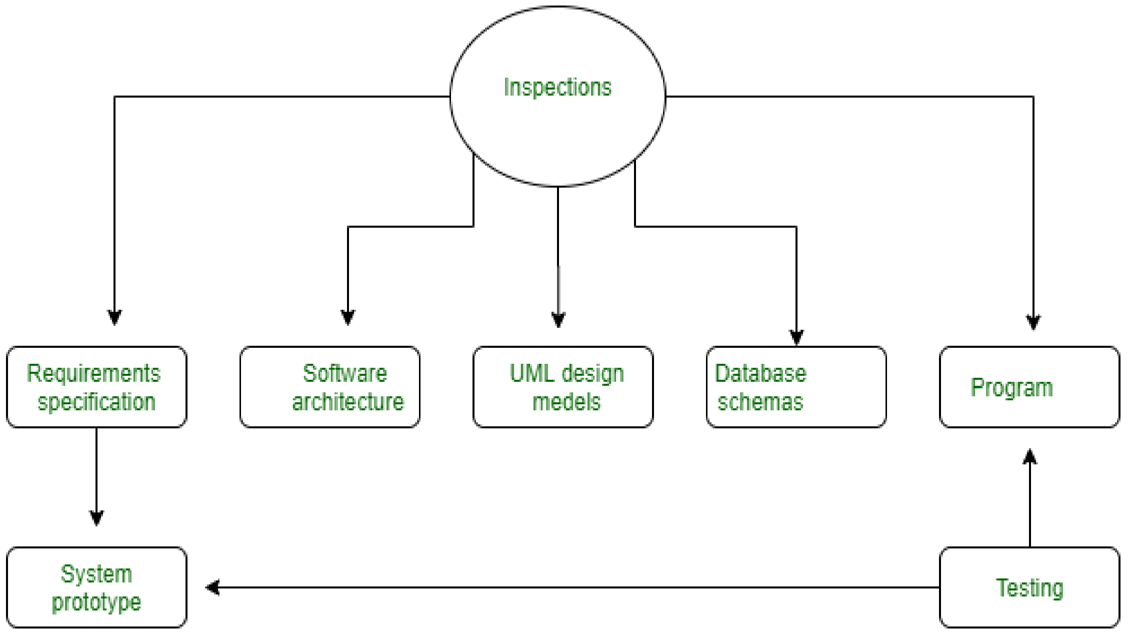
•**Fixing bugs or Incorporating Changes** – To change the software so as to remove any errors found during evaluation.

Activities involved in validation Testing:

1.Black box testing
2.White box testing
3.Unit testing
4.Integration testing
5.System Testing
6.Acceptance Testing

Software Inspection

- **Software Inspection**
  - The term software inspection was developed by IBM in the early 1970s, when it was noticed that the testing was not enough sufficient to attain high quality software for large applications.
  - Inspection is used to determine the defects in the code and remove it efficiently.
  - This prevents defects and enhances the quality of testing to remove defects.
  - This software inspection method achieved the highest level for efficiently removing defects and improving software quality.
  - Software Inspections refers to peer review of any work product by trained individuals who look for defects using a well defined process.
    - It is usually manual and a static technique that is applied in the early development cycle.
    - Software inspection is regarded as the most formal type of review.
    - It is led by the trained moderators and involves peers to examine the product.
    - The defects found during this process are documented in a issue log for convenience.



The software inspectoin and testing diagram

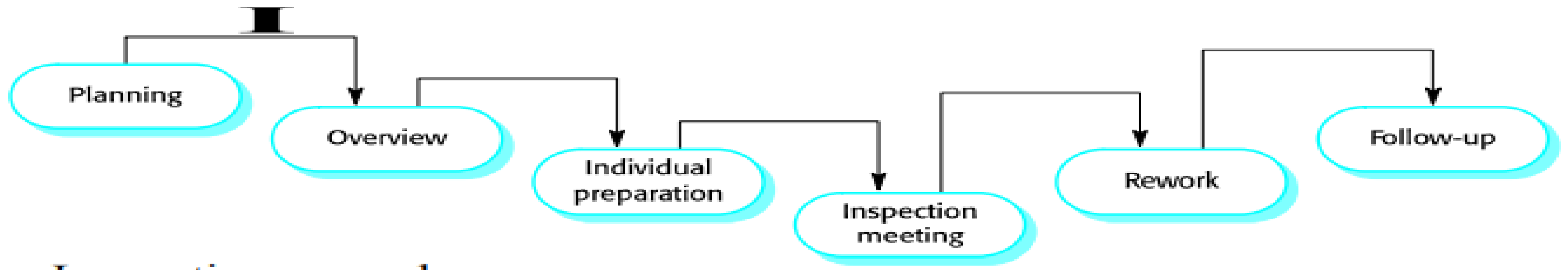
Software Inspection Process :-

The process must have an entry criterion that determines whether the inspection process is ready to begin. this prevents incomplete products from entering the inspection process. Entry criteria can be interstitial with items such as “The Spell-Document Check”.

There are some of the stages in the software inspection process such as-

1. **Planning –** The moderator plan the inspection.
2. **Overview Meeting –** The background of the work product is described by the author.
3. **Preparation –** The examination of the work product is done by inspector to identify the possible defects.
4. **Inspection Meeting –** The reader reads the work product part by part during this meeting and the inspectors the faults of each part.
5. **Rework –** after the inspection meeting, the writer changes the work product according to the work plans.
6. **Follow Up –** The changes done by the author are checked to make sure that everything is correct.

The inspection process



Inspection procedure

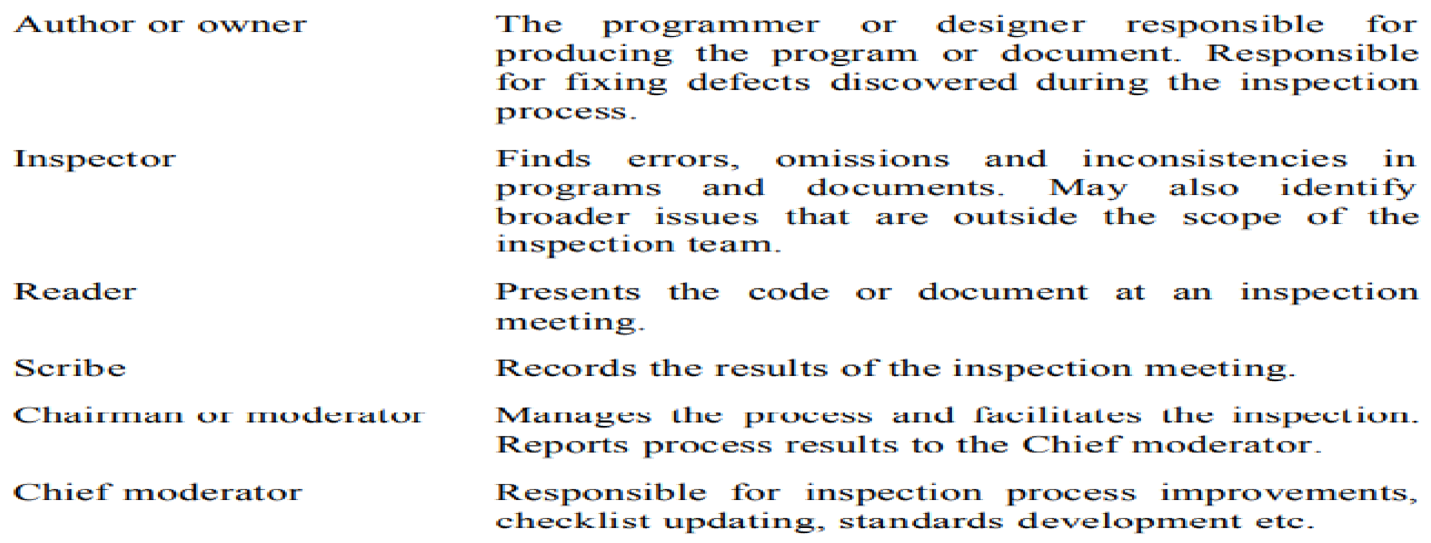
- System overview presented to inspection team.
- Code and associated documents are distributed to inspection team in advance.
- Inspection takes place and discovered errors are noted.
- Modifications are made to repair discovered errors.
- Re-inspection may or may not be required.

Role of Inspection Team:

An inspection team consists of **three to eight** members who play the roles of moderator, author, reader, recorder, and inspector.

- **Moderator** leads the inspection, schedules meetings, controls meetings, reports inspection results, and follows up on rework issues.
- **Author** creates or maintains the work product being inspected.
- **Reader** describes the sections of the work product to the team as they proceed through inspection.
- **Recorder** classifies and records defects and issues raised during the inspection. The moderator might perform this role in a small inspection team.
- **Inspector** finds errors in the product. All participants play the role of inspectors. However, good inspectors are those who have created the specification for the work product being inspected.
    - For example, the designer can act as an inspector during code inspection while a quality assurance representative can act as standard enforcer.

It also helps to have a client representative participate in requirements specification inspections.



Author or owner	The programmer or designer responsible for producing the program or document. Responsible for fixing defects discovered during the inspection process.
Inspector	Finds errors, omissions and inconsistencies in programs and documents. May also identify broader issues that are outside the scope of the inspection team.
Reader	Presents the code or document at an inspection meeting.
Scribe	Records the results of the inspection meeting.
Chairman or moderator	Manages the process and facilitates the inspection. Reports process results to the Chief moderator.
Chief moderator	Responsible for inspection process improvements, checklist updating, standards development etc.

An inspection checklist

Fault class	Inspection check
Data faults	• Are all program variables initialized before their values are used? • Have all constants been named? • Should the upper bound of arrays be equal to the size of the array or Size -1? • If character strings are used, is a delimiter explicitly assigned? • Is there any possibility of buffer overflow?
Control faults	• For each conditional statement, is the condition correct? • Is each loop certain to terminate? • Are compound statements correctly bracketed? • In case statements, are all possible cases accounted for? • If a break is required after each case in case statements, has it been included?
Input/output faults	• Are all input variables used? • Are all output variables assigned a value before they are output? • Can unexpected inputs cause corruption?

Fault class	Inspection check
Interface faults	• Do all function and method calls have the correct number of parameters? • Do formal and actual parameter types match? • Are the parameters in the right order? • If components access shared memory, do they have the same model of the shared memory structure?
Storage management faults	• If a linked structure is modified, have all links been correctly reassigned? • If dynamic storage is used, has space been allocated correctly? • Is space explicitly deallocated after it is no longer required?
Exception management faults	• Have all possible error conditions been taken into account?

Test Case and Test Case Design

- **A *test case*** is a document which has *a set of conditions or actions* that are performed on the software application in order to verify the expected functionality of the feature.
- A Test Case contains test steps, test data, precondition, post condition developed for specific test scenario to verify any requirement.
- The test case includes specific variables or conditions, using which a testing engineer can compare expected and actual results to determine whether a software product is functioning as per the requirements of the customer.
- A test case is a set of instructions on "HOW" to validate a particular test objective/target, which when followed will tell us if the expected behavior of the system is satisfied or not.
- ***Test scenarios*** are rather vague and cover a wide range of possibilities
- A single test scenario may contains various test cases.
- **A basic example of test case design:**
  - **Title:** Login to the website or app
  - **Description:** User should be able to successfully log in to their account on the website/app
  - **Preconditions:** User must already be registered and use their correct login details
  - **Assumptions:** They are using a supported device or browser to log in
  - **Test Steps:**
    - Open website or app
    - Enter the username and password in the appropriate fields
    - Click "login"
  - **Expected Result:** The user should log in successfully.
  - **Post conditions**
- Example of test cases
- **Benefits or Writing test cases**
- The key purpose of a test case is to ensure if different features within an application are working as expected. It helps tester, validate if the software is free of defects and if it is working as per the expectations of the end users. Other benefits of test cases include:
    - Test cases ensure good test coverage
    - Help improve the quality of software,
    - Decreases the maintenance and software support costs
    - Help verify that the software meets the end user requirements
    - Allows the tester to think thoroughly and approach the tests from as many angles as possible
    - Test cases are reusable for the future – anyone can reference them and execute the test.

Techniques of Testing:Black Box Testing

- This type of testing is based entirely on software requirements and specifications not the code.
- Black box testing is a high level of testing that focuses on the behavior of the software.
- It involves testing from an external or end-user perspective.
- emphasis on parameters, inputs/outputs (and their validity)
- Black box testing can be applied to virtually every level of software testing: unit, integration, system, and acceptance.
- Generally black box testing attempts to uncover the following:
  - Incorrect and missing functions
  - Data structure error, external database access error, initializing and termination errors and performance error
- Black Box testing is also called functional testing because the tester is only concerned with the functionality and not the implementation of software.

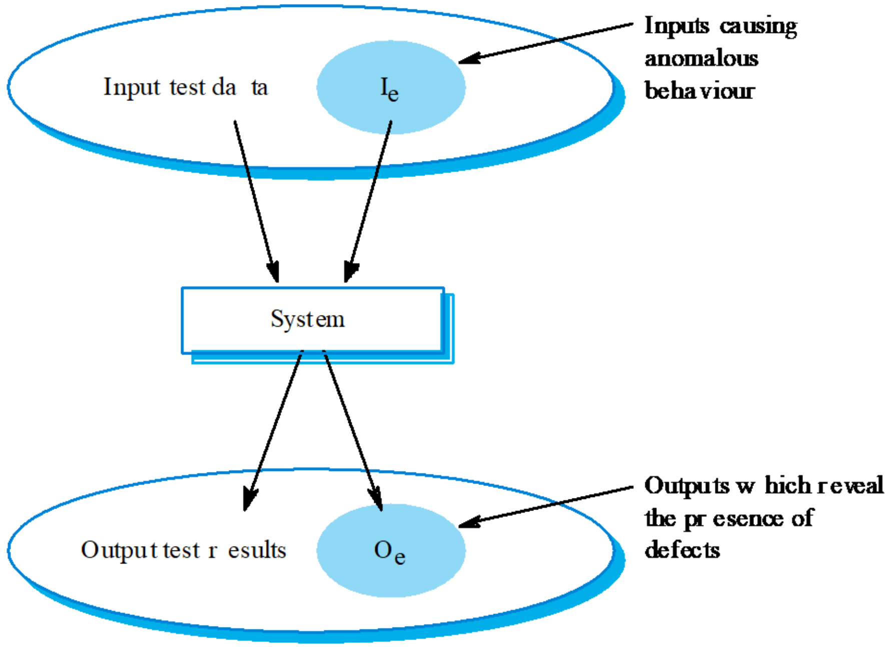

Input test da ta
I_e
Inputs causing anomalous behaviour
System
Output test r esults
O_e
Outputs w hich reveal the pr esence of defects

Fig:Black Box Testing

Types of Black Box testing

• requirements based	-validating the requirements given in SRS
• positive/negative	- checks both good/bad results
• boundary value analysis	-test cases are designed for boundary values of input domain
• decision tables	-based on decision point (conditions)
• equivalence partitioning	- group related inputs/outputs
• state-based	- based on object state diagrams
• compatibility testing	-based on system compatibility

Equivalence Class Partitioning

- The domain of input values to a program is partitioned into set of equivalence class
- The main idea behind defining equivalence class is that testing the code with any one value belonging to an equivalence class is as good as testing the software with any other value belonging to that equivalence class.
- Equivalence class for software can be designed by examining the input data and output data.
- Test case design for equivalence partitioning is based on an evaluation of equivalence classes for an input condition.
- An input condition is either a specific numeric value, a range of values, a set of related values, or Boolean condition.
- Equivalence classes may be defined according to the following guidelines:
  - If an input condition specifies a range, one valid and two invalid equivalence classes are defined.
  - If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.
  - If an input condition specifies a member of a set, one valid and one invalid equivalence classes are defined.
  - If an input condition is Boolean, one valid and one invalid classes are defined.
- Example
- **Consider a numerical input variable, i, whose values may range from -200 through +200. Then a possible partitioning of testing input variable by 4 people may be:**
  - **-200 to -100**
  - **-101 to 0**
  - **1 to 100**
  - **101 to 200**

- **Define "same sign" as the equivalence relation, R, defined over the input variable's value set, i = {-200 - -,0, - -, +200}. Then one partitioning will be:**
  - **-200 to -1 (negative sign)**
  - **0 (no sign)**
  - **1 to 200 (positive sign)**

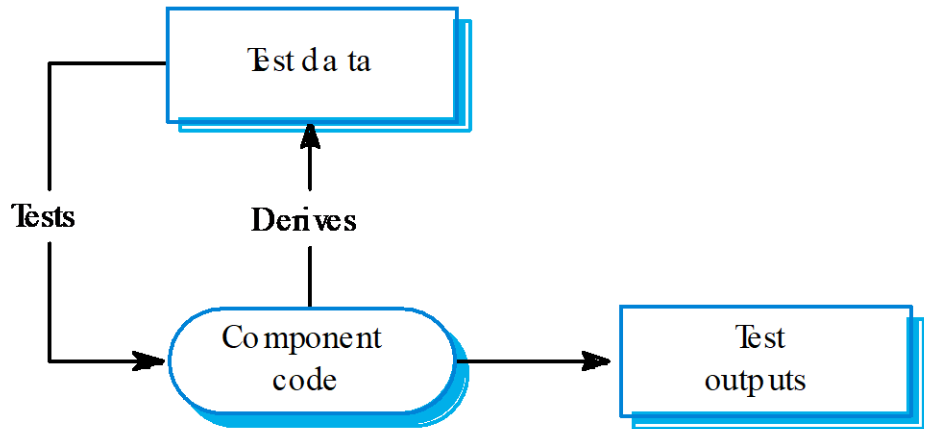
Boundary Value Analysis(BVA)

- BVA is a method useful for arriving at tests that are effective in catching defects that happen at boundaries. i.e.BVA believes and extends the concept that density of defect is more towards the boundaries.
- A type of programming errors frequently occurs at boundaries of different equivalent classes of inputs and The reason behind such error might purely be due to the psychological factor.
- Programmers often fail to see the special processing required by input values that lie at the boundary of different equivalence classes.
- To determine BVA, we can consider **the Range, maximum and minimum number, output conditions and the internal data structure** etc.
- is used when a great number of errors tends to occur at the boundaries of input domain rather than in the "center".
- It leads to a selection of test cases that exercise boundary values.
- If a test case design technique that complements equivalence partitioning.
- Leads to the selection of test case at the "edges" of the class.
- **Guidelines for boundary value analysis** are similar in many respects to those provided for equivalence partitioning:
    - If an input condition specifies a number of values, test cases should be developed that exercise the minimum and maximum numbers.
    - If an input condition specifies a range bounded by values a and b, test cases should be designed with values a and b and just above and below a and b.
    - Apply guidelines 1 and 2 to output conditions.
    - If internal program data structures have prescribed boundaries, be certain to design a test case to exercise the data structure at its boundary.
- By applying these guidelines, boundary testing will be more complete, thereby having a higher likelihood for error detection.
- For example: if we have an input variable x in the range of 1 to 10 then the boundary values can be 1,2,9 and 10.
- Imagine we are testing a `Date` class with a `daysInMonth(month, year)` method.
    - What are some conditions and boundary tests for this method?
- Possible answers:
    - check for leap years (every 4th yr, no 100s, yes 400s)
    - try years such as: even 100s, 101s, 4s, 5s
    - try months such as: June, July, Feb, invalid values

- Advantages of black box testing
  - Unbiased tests because the designer and tester work independently
  - Tester is free from any pressure of knowledge of specific programming languages
  - Facilitates identification of contradictions and vagueness in functional specifications
  - Test is performed from a user’s point-of-view and not of the designer’s
  - Test cases can be designed immediately after the completion of specifications
- Disadvantages of black box testing
  - Tests can be redundant if already run by the software designer and are extremely difficult to be designed without clear and concise specifications
  - Testing every possible input stream is not possible because it is time-consuming
  - Results might be overestimated at times
  - Cannot be used for testing complex segments of code

Techniques of Testing: White Box Testing

- White Box Testing is defined as the testing of a software solution's internal structure, design, and coding. In this type of testing, the code is visible to the tester.
- It focuses primarily on verifying the flow of inputs and outputs through the application, improving design and usability, strengthening security.
- White box testing is also known as Clear Box testing, Open Box testing, Structural testing, Transparent Box testing, Code-Based testing, and Glass Box testing. It is usually performed by developers.
- 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



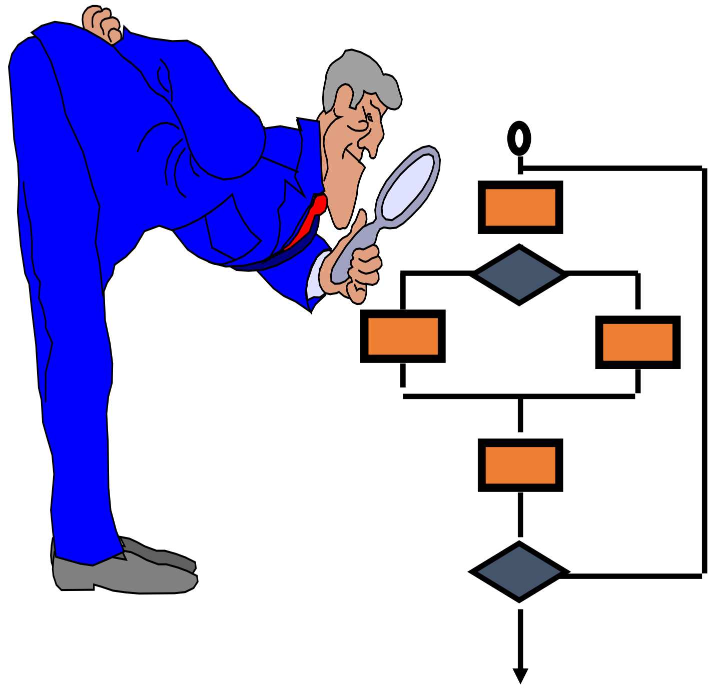


Fig:White Box Testing

- The following approaches are used for white box testing.
  - **Basic path testing**: an attempt to use test input that will pass once over each path in the code. Basic 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. Cyclomatic complexity is used to compute the complexity of the different paths.
  - **Code coverage testing:** Examines what fraction of the code under test is reached by existing unit tests.
  - **statement coverage -** tries to reach every line (impractical)
  - **path coverage** - follow every distinct branch through code
  - **condition coverage** - every condition that leads to a branch
  - **function coverage** - treat every behavior / end goal separately

Flow Graph Notations:

- Before basis path method can be introduced, a simple notation for the representation of control flow, called flow graph must be introduced.
- It enables you to trace program paths more readily.
- Have to draw a flow graph when the logical control structure of a module is complex.

Statement Coverage:

- It is a measure of percentage of statement that have been executed by test cases.
- The main objective of statement coverage is to achieve 100% statement coverage through testing.

Path Coverage:

- In path coverage we write test cases to ensure that each and every path that have been traversed at once. In this testing the path coverage is understand by:
- Drawing the flow graph to indicate corresponding logic in the program.
- By calculating individual paths in the flow graph.
- Running the program in all possible ways to cover every possible statement.

Cyclomatic Complexity:

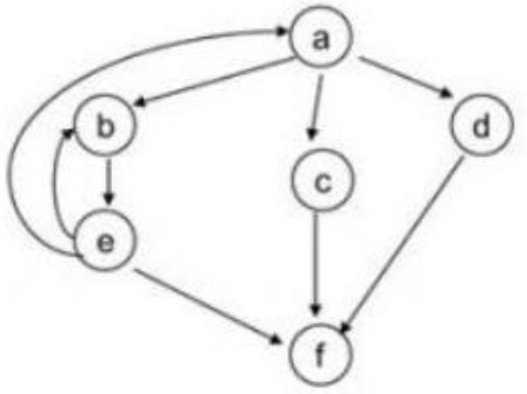
- is a software metric that provides a quantitative measure of the logical complexity of a program.
- The value computed for the cyclomatic complexity defines the number of independent paths in the basis set of a program and provides us with an upper-bound for the number of tests that must be conducted to ensure that all statements have been executed at least once.
- An independent path is any path through the program that introduces at least one new set of processing statements or a new condition.
- Cyclomatic complexity is a useful metric for predicting those modules that are likely to be error prone.
- It can be used for test planning as well as test case design.
- Cyclomatic complexity is computed in one of three ways:

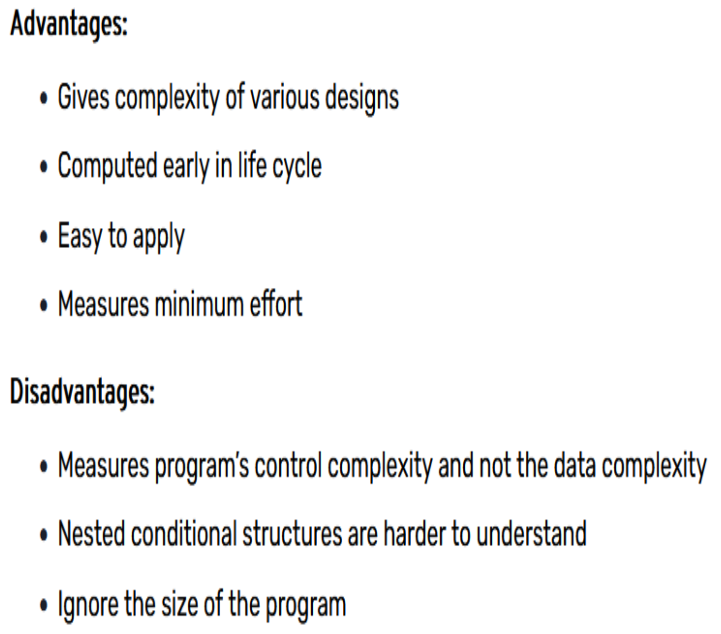
1. The number of regions of the flow graph corresponds to the cyclomatic complexity.
2. Cyclomatic complexity, V (G), for a flow graph, G, is defined as:

   V(G) = E – N + 2P; Where E is the number of flow graph edges, N is the number o flow graph nodes and P=number of strongly connected component in graph
3. Cyclomatic complexity, V (G), for a flow graph, G, is also defined as:

   $$\mathbf{V(G)} = \pi + 1$$

   Where π is the number of predicate nodes contained in the flow graph G.




Advantages:

- Gives complexity of various designs
- Computed early in life cycle
- Easy to apply
- Measures minimum effort

Disadvantages:

- Measures program's control complexity and not the data complexity
- Nested conditional structures are harder to understand
- Ignore the size of the program

- Advantage 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 white box 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 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 vs Black Box Testing

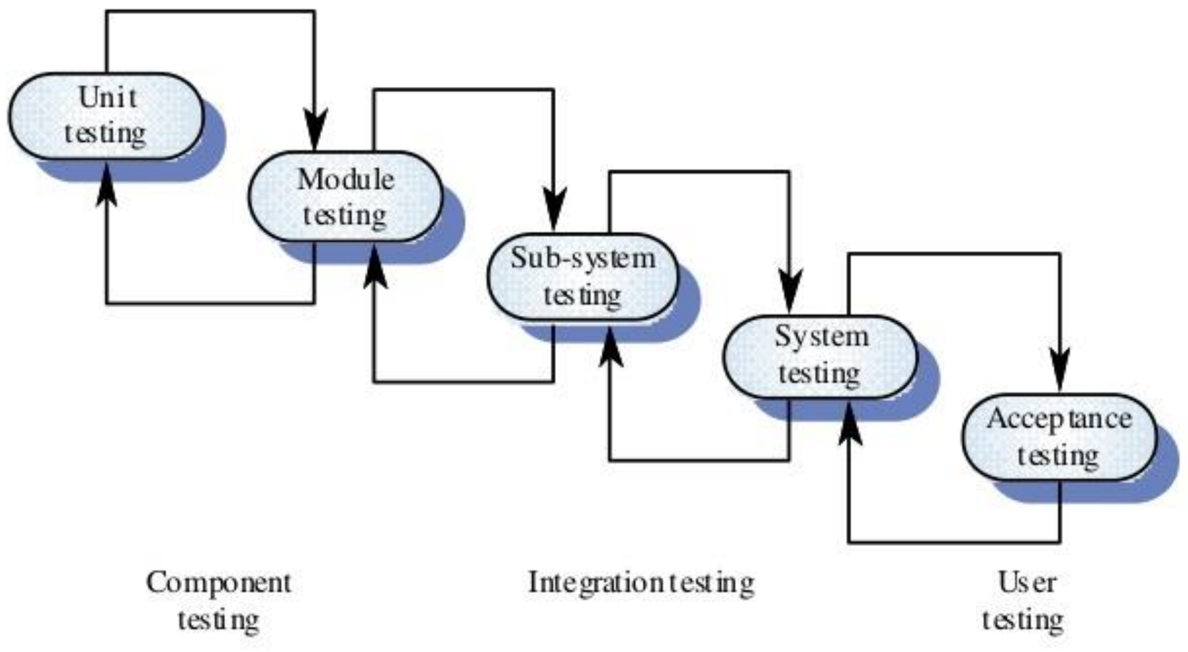
Criteria	**Black Box Testing**	**White Box Testing**
Definition	Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested *is NOT known* to the tester.	White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested *is known* to the tester.
Levels Applicable To	Mainly applicable to higher levels of testing: Acceptance Testing & System Testing	Mainly applicable to lower levels of testing: Unit Testing & Integration Testing
Responsibility	Generally, independent Software Testers	Generally, Software Developers
Also known as	Functional Testing	Structural Testing
Implementation Knowledge	Not Required	Required
Basis for Test Cases	Requirement Specifications	Detail Design
Testing Techniques	Requirement based testing,positive and negative testing,boundary value analysis,equivalence partitioning,compatibility testing etc.	Basic Path testing,structural testing ,fault based testing,logic based testing etc.
Programming Knowledge	Not Required	Required

Software Testing Process

Stages of Software testing

1. **Unit Testing:**
   - Individual components are tested to ensure that they operate correctly.
   - Each component is tested independently without other system components.
2. **Module Testing:**
   - This involves the testing of independent components such as procedures and functions.
   - A module encapsulates related components so it can be tested without other system modules.
3. **Subsystem Testing:**
   - Modules are integrated into sub-systems and tested.
   - The focus here should be on interface testing to detect module interface errors or mismatches.
   - The sub-system test process should therefore concentrate on the detection of interface errors by rigorously exercising the interfaces.

The stages of testing process



Testing Process contd…

4. **System Testing:**
    - Sub systems are integrated to make up the entire system.
    - The testing process is concerned with finding errors that result from unanticipated interactions between sub-systems and system components.
    - It is also concerned with validating that the system meets its functional and non-functional requirements.

5. **Acceptance Testing:**
    - This is the final stage in the testing process before the system is accepted for operational use.
    - The system is tested with data supplied by the system procurer rather than simulated test data.
    - Acceptance testing may reveal errors and omissions in the system requirements definition because the real data exercises the system in different ways from the test data.
    - It may also reveal requirements problems where the system's facilities do not really meet the user's needs or the system's performance is not acceptable.

Component testing

- Testing of individual program components
- Usually the responsibility of the component developer (except sometimes for critical systems)
- Tests are derived from the developer's experience

Integration testing

- Testing of groups of components integrated to create a system or sub-system
- The responsibility of an independent testing team Tests are based on a system specification

Acceptance Testing

- Making sure the software works correctly for intended user in his or her normal work environment.
    - Alpha test-version of the complete software is tested by customer under the supervision of the developer at the developer's site.
    - Beta test-version of the complete software is tested by customer at his or her own site without the developer being present

Level of testing

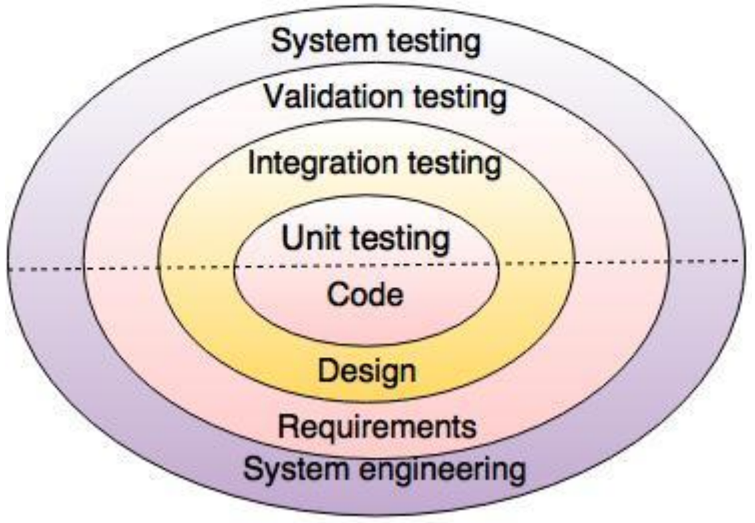


Fig. - Testing Strategy

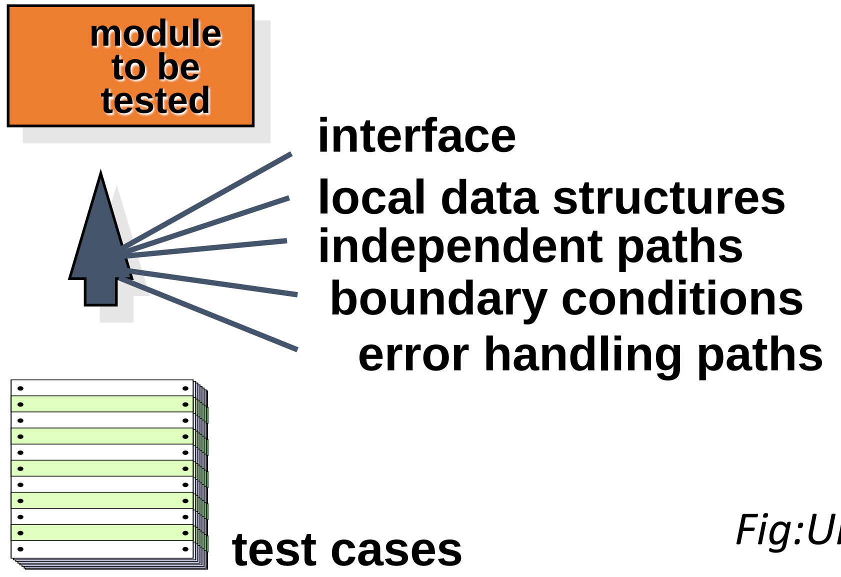


Fig:Unit Testing

Unit Testing

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

Why Unit Testing?

1. Unit tests help to fix bugs early in the development cycle and save costs.
2. It helps the developers to understand the testing code base and enables them to make changes quickly
3. Good unit tests serve as project documentation
4. Unit tests help with code re-use. Migrate both your code **and** your tests to your new project. Tweak the code until the tests run again.
5. Reduces Defects in the Newly developed features or reduces bugs when changing the existing functionality.
6. Reduces Cost of Testing as defects are captured in very early phase.
7. Improves design and allows better refactoring of code.
8. Unit Tests, when integrated with build gives the quality of the build as well.

Unit Testing tools

- NUnit
- JUnit
- PHPunit
- Parasoft Jtest
- EMMA

- Unit testing techniques
  - Data flow Testing
  - Control Flow Testing
  - Branch Coverage Testing
  - Statement Coverage Testing
  - Decision Coverage Testing

Unit Testing Advantage

•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 Disadvantages

•Unit testing can't 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't catch integration errors or broad system level errors.

Integration Testing

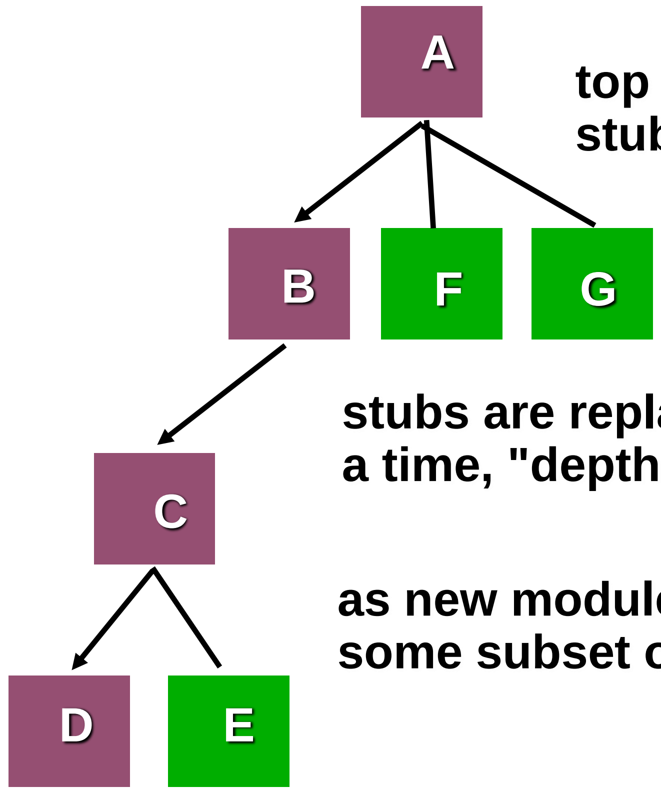
- **INTEGRATION TESTING** is defined as a type of testing where software modules are integrated logically and tested as a group.
- A typical software project consists of multiple software modules, coded by different programmers.
- The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated
- Upon completion of unit testing, the units or modules are to be integrated which gives raise to integration testing.
- The purpose of integration testing is to verify the functional, performance, and reliability between the modules that are integrated.
- The goal of integration testing is to check the correctness of communication among all the modules.
- **Why Integration Testing?**
  - Integration Testing becomes necessary to verify the software modules work in unity.
  - At the time of module development, there are wide chances of change in requirements by the clients. These new requirements may not be unit tested and hence system integration Testing becomes necessary.
  - Interfaces of the software modules with the database could be erroneous
  - External Hardware interfaces, if any, could be erroneous
  - Inadequate exception handling could cause issues.

- **Approaches/Strategy of Integration Testing**

1. **Big Bang 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.
   - **Advantages:**
     - Convenient for small systems.
   - **Disadvantages:**
     - Fault Localization is difficult.
     - Given the sheer number of interfaces that need to be tested in this approach, some interfaces link to be tested could be missed easily.
     - The testing team will have less time for execution in the testing phase.
     - Since all modules are tested at once, high-risk critical modules are not isolated and tested on priority

2. **Incremental Approach**: which is further divided into the following
   - Top Down Approach
   - Bottom Up Approach
   - Sandwich Approach - Combination of Top Down and Bottom Up

Top-down Integration

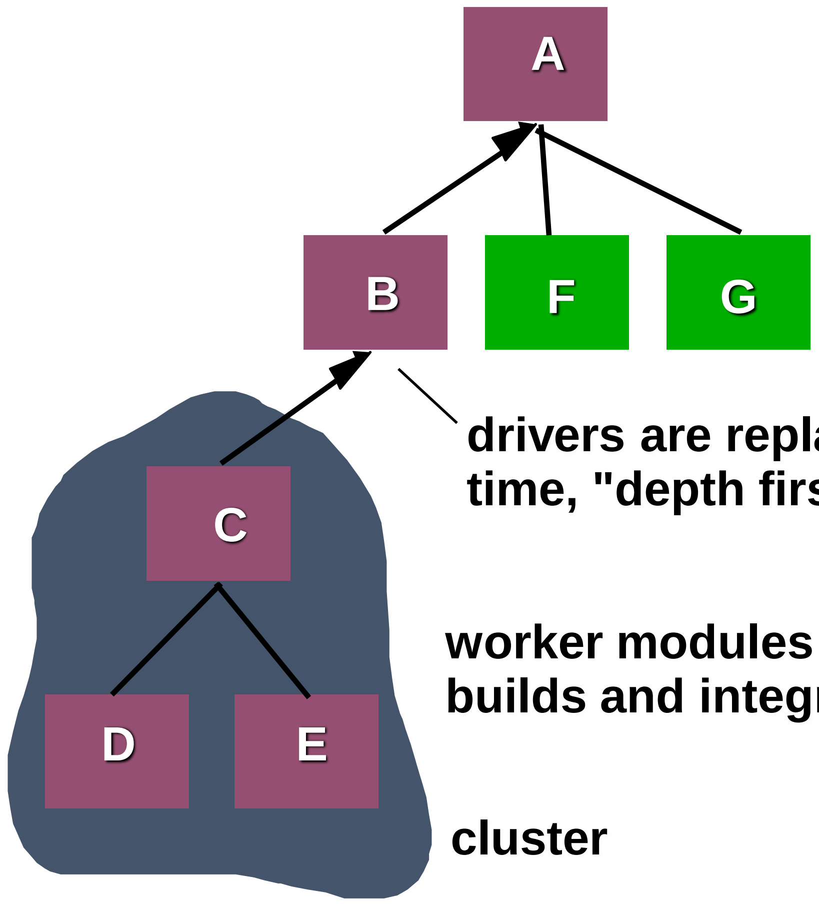


top module is tested with stubs

stubs are replaced one at a time, "depth first"

as new modules are integrated, some subset of tests is re-run

Bottom-Up Integration



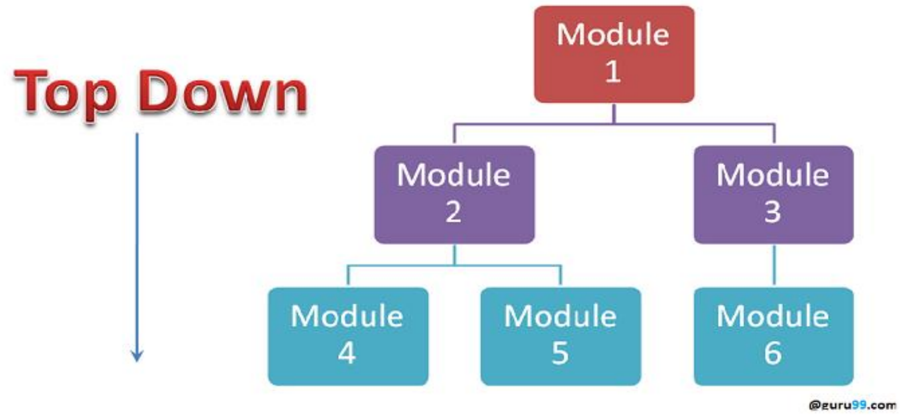
drivers are replaced one at a time, "depth first"

worker modules are grouped into builds and integrated

cluster

Integration Testing

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



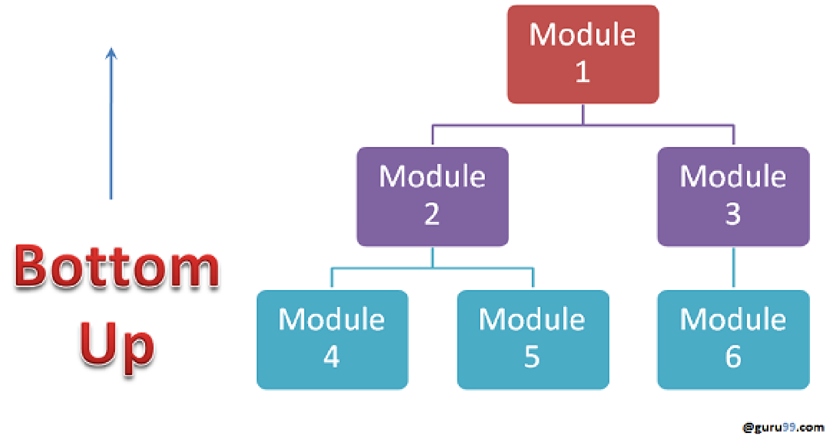
Advantages:

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

Disadvantages:

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

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



- **Advantages:**
  - Fault localization is easier.
  - No time is wasted waiting for all modules to be developed unlike Big-bang approach
- **Disadvantages:**
  - Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.
  - An early prototype is not possible

Integration Testing

- **Sandwich Integration Approach**
- **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.

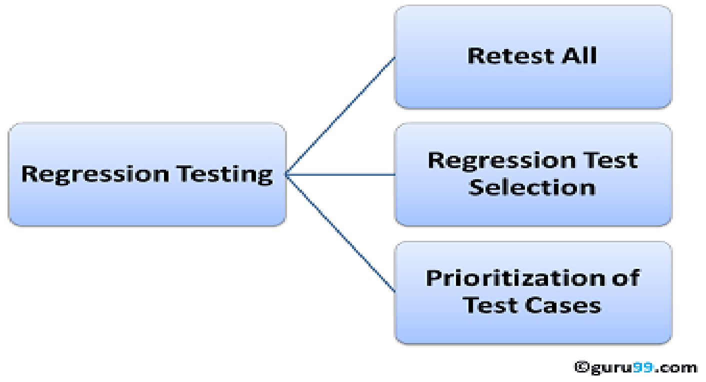
Regression Testing

- **REGRESSION TESTING** is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features.
- Regression Testing is nothing but a full or partial selection of already executed test cases which are re-executed to ensure existing functionalities work fine.
- This testing is done to make sure that new code changes should not have side effects on the existing functionalities. It ensures that the old code still works once the latest code changes are done.

Why Regression Testing?

- The **Need of Regression Testing** mainly arises whenever there is **requirement to change** the code and we need to test whether the modified code affects the other part of software application or not.
- Moreover, regression testing is needed, when **a new feature** is added to the software application and for defect fixing as well as performance issue fixing.

- **How to perform Regression Testing?**
  - Regression Testing can be carried out using the following techniques:

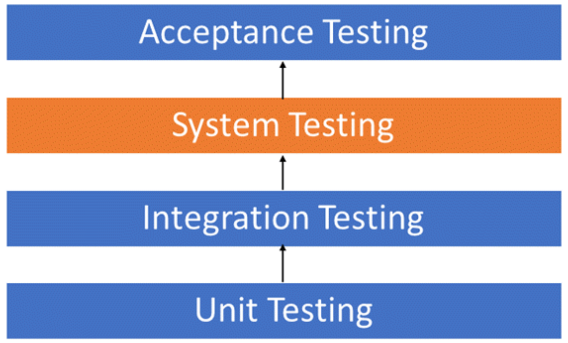


- **Reset All:**
  - All the tests in the existing test bucket or suite should be re-executed. This is very expensive as it requires huge time and resources.
- **Regression Test Selection**
  - **Regression Test Selection** is a technique in which some selected test cases from test suite are executed to test whether the modified code affects the software application or not.
  - Test cases are categorized into two parts, reusable test cases which can be used in further regression cycles and obsolete test cases which can not be used in succeeding cycles.
- **Prioritization of Test Cases**
  - Prioritize the test cases depending on business impact, critical & frequently used functionalities. Selection of test cases based on priority will greatly reduce the regression test suite.

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.
- System Testing includes testing of a fully integrated software system.
- System Testing is actually a series of different tests whose sole purpose is to exercise the full computer-based system.
- System testing tests the design and behavior of the system and also the expectations of the customer.
- System Testing is basically performed by a testing team that is independent of the development team that helps to test the quality of the system impartial. It has both functional and non-functional testing.

Testing Hierarchy



System Testing Process:

System Testing is performed in the following steps:

•**Test Environment Setup:**
Create testing environment for the better quality testing.

•**Create Test Case:**
Generate test case for the testing process.

•**Create Test Data:**
Generate the data that is to be tested.

•**Execute Test Case:**
After the generation of the test case and the test data, test cases are executed.

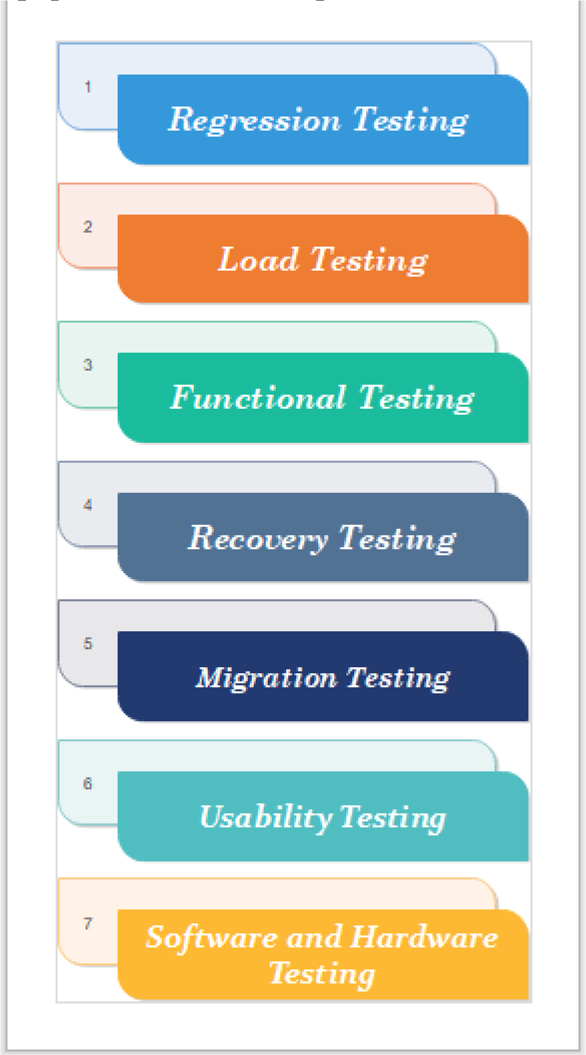
•**Defect Reporting:**
Defects in the system are detected.

•**Regression Testing:**
It is carried out to test the side effects of the testing process.

•**Log Defects:**
Defects are fixed in this step.

•**Retest:**
If the test is not successful then again test is performed.

Types of System Testing



Regression Testing

Regression testing is performed under system testing to confirm and identify that if there's any defect in the system due to modification in any other part of the system. It makes sure, any changes done during the development process have not introduced a new defect and also gives assurance; old defects will not exist on the addition of new software over the time.

Load Testing

Load testing is performed under system testing to clarify whether the system can work under real-time loads or not.

Functional Testing

Functional testing of a system is performed to find if there's any missing function in the system.

Recovery Testing

Recovery testing of a system is performed under system testing to confirm reliability, trustworthiness, accountability of the system and all are lying on recouping skills of the system. It should be able to recover from all the possible system crashes successfully.

Migration Testing

Migration testing is performed to ensure that if the system needs to be modified in new infrastructure so it should be modified without any issue.

Usability Testing

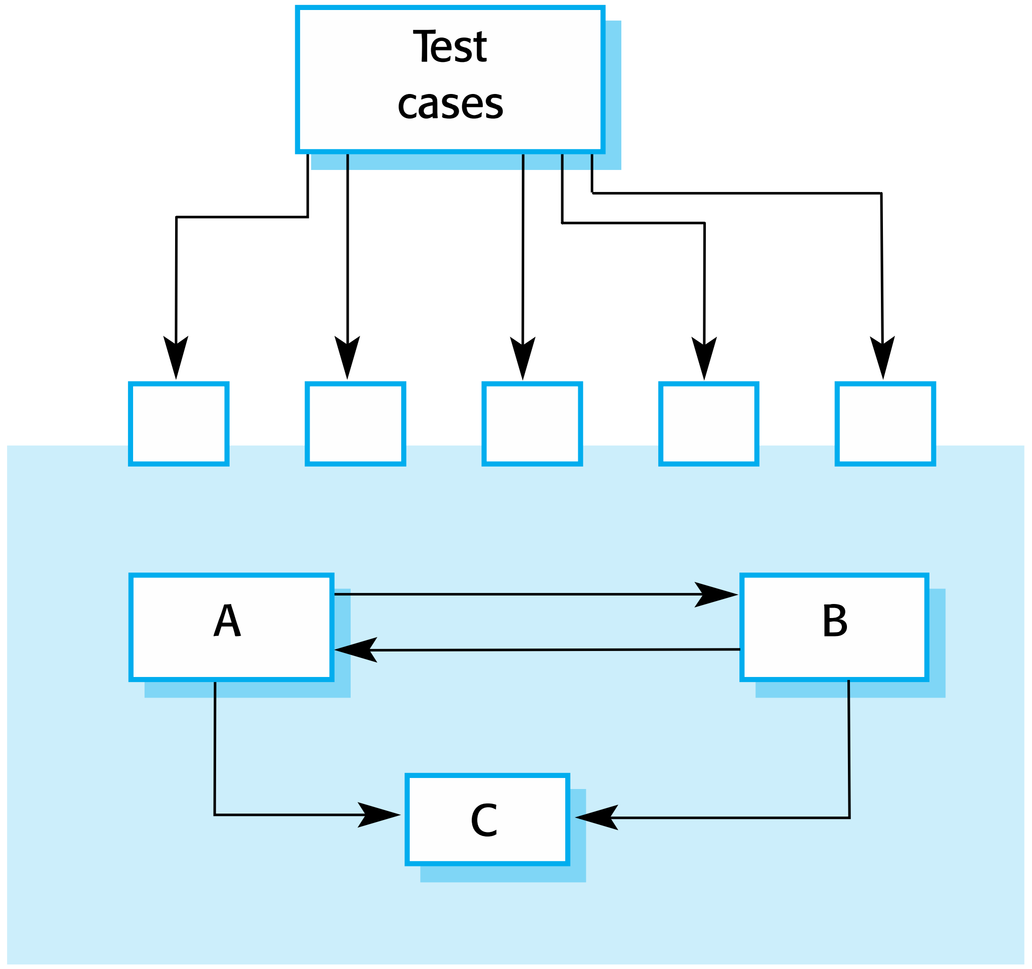
The purpose of this testing to make sure that the system is well familiar with the user and it meets its objective for what it supposed to do.

Software and Hardware Testing

This testing of the system intends to check hardware and software compatibility. The hardware configuration must be compatible with the software to run it without any issue. Compatibility provides flexibility by providing interactions between hardware and software.

Interface Testing

- Objectives are to detect faults due to interface errors or invalid assumptions about interfaces.
- Interface types
  - **Parameter interfaces:**
    - Data passed from one method or procedure to another.
  - **Shared memory interfaces:**
    - Block of memory is shared between procedures or functions.
  - **Procedural interfaces:**
    - Sub-system encapsulates a set of procedures to be called by other sub-systems.
  - **Message passing interfaces:**
    - Sub-systems request services from other sub-systems



- **Interface Errors:**
  - Interface misuse
    - A calling component calls another component and makes an error in its use of its interface e.g. parameters in the wrong order.
  - Interface misunderstanding
    - A calling component embeds assumptions about the behaviour of the called component which are incorrect.
  - Timing errors
    - The called and the calling component operate at different speeds and out-of-date information is accessed.
- **Interface Testing Guidelines**
  - Design tests so that parameters to a called procedure are at the extreme ends of their ranges.
  - Always test pointer parameters with null pointers.
  - Design tests which cause the component to fail.
  - Use stress testing in message passing systems.
  - In shared memory systems, vary the order in which components are activated.

Acceptance Testing

- It is a formal testing according to user needs, requirements and business processes conducted to determine whether a system satisfies the acceptance criteria or not and to enable the users, customers or other authorized entities to determine whether to accept the system or not.
- Acceptance testing, a testing technique performed to determine whether or not the software system has met the requirement specifications.
- The main purpose of this test is to evaluate the system's compliance with the business requirements and verify if it is has met the required criteria for delivery to end users.
- Acceptance Testing is the last phase of software testing performed after System Testing and before making the system available for actual use.
- There are various forms of acceptance testing:

1. **User acceptance Testing**
2. **Business acceptance Testing**
3. **Contract Acceptance Testing (CAT)**
4. **Operational Acceptance Testing(OAT)**
5. **Regulations Acceptance Testing (RAT)**
6. **Alpha Testing**
7. **Beta Testing**

Alpha testing

Users of the software work with the development team to test the software at the developer’s site.

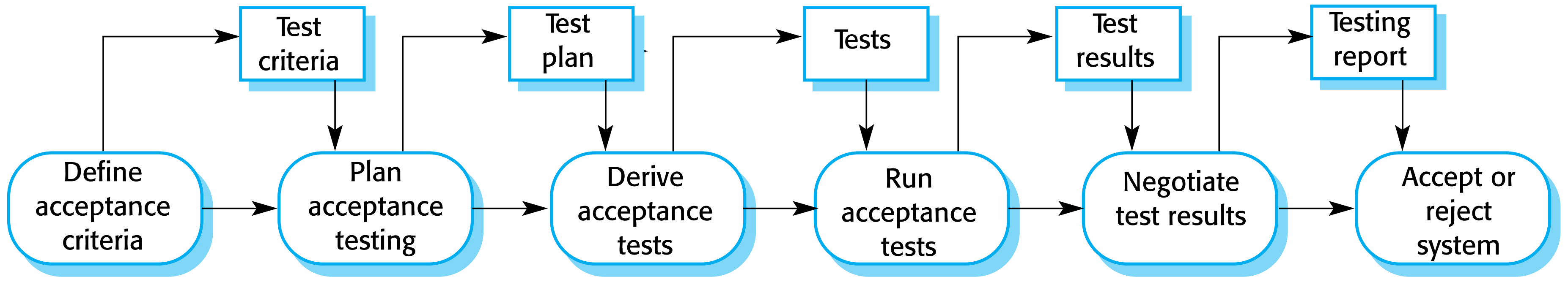
Beta testing

A release of the software is made available to users to allow them to experiment and to raise problems that they discover with the system developers.

Acceptance testing

Customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment. Primarily for custom systems.

Acceptance Testing Process



Steps in Acceptance Testing

1. **Define acceptance criteria**
2. **Plan acceptance testing**
3. **Derive acceptance tests**
4. **Run acceptance tests**
5. **Negotiate test results**
6. **Reject/accept system**

Use of Acceptance Testing:

•To find the defects missed during the functional testing phase.
•How well the product is developed.
•A product 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.

Alpha testing vs Beta Testing

Sr. No.	Alpha Testing	Beta Testing
1.	Alpha testing performed by a team of highly skilled testers who are usually the internal employee of the organization.	Beta testing performed by clients or end-users in a real-time environment, who is not an employee of the organization.
2.	Alpha testing performed at the developer's site; it always needs a testing environment or lab environment.	Beta testing doesn't need any lab environment or the testing environment; it is performed at a client's location or end-user of the product.
3.	Reliability or security testing not performed in-depth in alpha testing.	Reliability, security, and robustness checked during beta testing.
4.	Alpha testing involves both white box and black-box techniques.	Beta testing uses only black-box testing.
5.	Long execution cycles maybe require for alpha testing.	Only a few weeks are required for the execution of beta testing.
6.	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.
7.	Alpha testing performed before the launch of the product into the market.	At the time of software product marketing.
8.	Alpha testing focuses on the product's quality before going to beta testing.	Beta testing concentrates on the quality of the product, but gathers users input on the product and ensures that the product is ready for real-time users.
9.	Alpha testing performed nearly the end of the software development.	Beta testing is a final test before shipping a product to the customers.
10.	Alpha testing is conducting in the presence of developers and the absence of end-users.	Beta testing reversed of alpha testing.

Quality and Quality Management Activities

- **Software Quality:**
  - Quality, simplistically, means that a product should meet its specification.
  - This is problematical for software systems
    - There is a tension between customer quality requirements (efficiency, reliability, etc.) and developer quality requirements (maintainability, reusability, etc.);
    - Some quality requirements are difficult to specify in an unambiguous way;
    - Software specifications are usually incomplete and often inconsistent.
  - The focus may be ‘fitness for purpose’ rather than specification conformance.
- **There are two kinds of quality**
  - **Quality of Design:** Quality of Design refers to the characteristics that designers specify for an item. The grade of materials, tolerances, and performance specifications that all contribute to the quality of design.
  - **Quality of conformance:** Quality of conformance is the degree to which the design specifications are followed during manufacturing. Greater the degree of conformance, the higher is the level of quality of conformance.

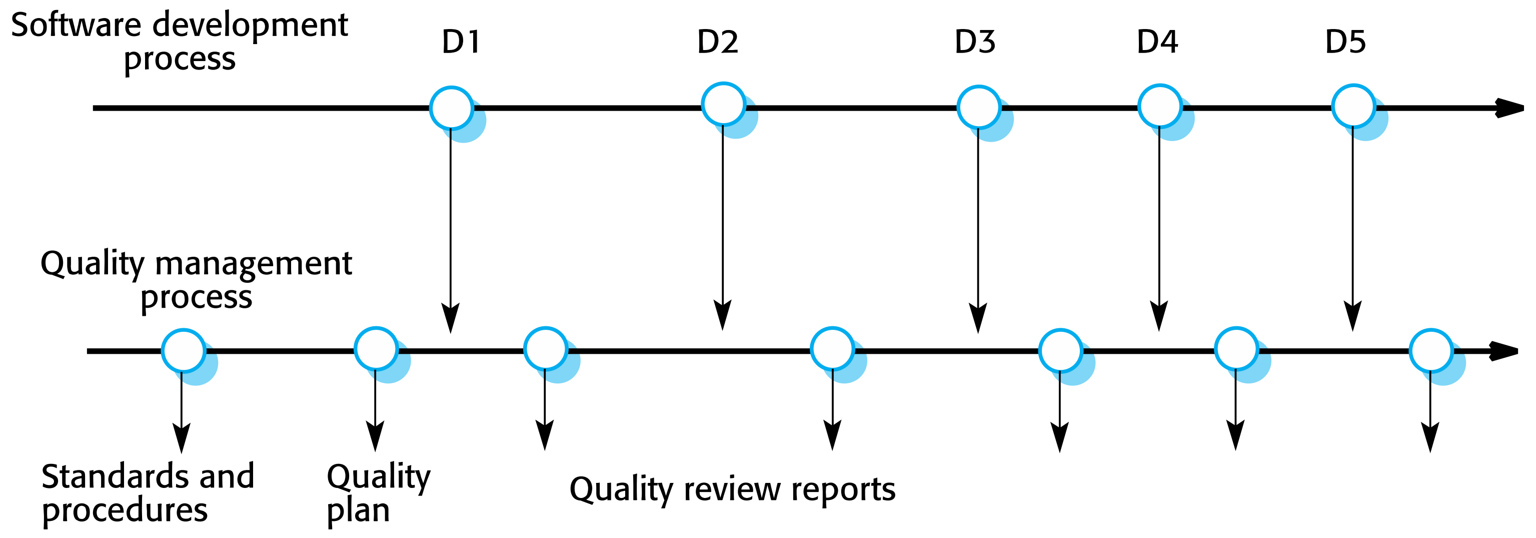
Software Quality Management (SQM)

- Software Quality Management (SQM) is the management aspects of developing quality software. SQM starts with an idea for a product and then continues on through the design, testing, and launch phases.
- Concerned with ensuring that the required level of quality is achieved in a software product.
- SQM is a management process that aims to develop and manage the **quality of software** in such a way so as to best ensure that the product meets the quality standards expected by the customer while also meeting any necessary regulatory and developer requirements.
- Three principal concerns:
  - At **the organizational level**, quality management is concerned with establishing a framework of organizational processes and standards that will lead to high-quality software.
  - At the **product level**, quality management involves the application of specific quality processes and checking that these planned processes have been followed.
  - At the **project level**, quality management is also concerned with establishing a quality plan for a project. The quality plan should set out the quality goals for the project and define what processes and standards are to be used.
- Generally speaking, SQM can be broken down into three phases:
  - Quality planning,
  - Quality assurance and
  - Quality control.

Quality management activities

- Quality management provides an independent check on the software development process.
- The quality management process checks the project deliverables to ensure that they are consistent with organizational standards and goals
- The quality team should be independent from the development team so that they can take an objective view of the software. This allows them to report on software quality without being influenced by software development issues.

Quality management and software development



Quality planning

- A quality plan sets out the desired product qualities and how these are assessed and defines the most significant quality attributes.
- The quality plan should define the quality assessment process.
- It should set out which organisational standards should be applied and, where necessary, define new standards to be used.
- Quality plan structure
  - Product introduction;
  - Product plans;
  - Process descriptions;
  - Quality goals;
  - Risks and risk management.
- Quality plans should be short, succinct documents
  - If they are too long, no-one will read them.

Quality Assurance

- **Quality Assurance** is defined as a procedure to ensure the quality of software products or services provided to the customers by an organization.
- Quality assurance focuses on improving the software development process and making it efficient and effective as per the quality standards defined for software products.
- Quality Assurance is popularly known as QA Testing.
- **Software quality assurance (SQA)** is a process which assures that all software engineering processes, methods, activities and work items are monitored and comply against the defined standards. These defined standards could be one or a combination of any like ISO 9000, CMMI model, ISO15504, etc
- This stage can include:
  - encouraging documentation process standards, such as the creation of well-defined engineering documents using standard templates
  - mentoring how to conduct standard processes, such as quality reviews
  - performing in-process test data recording procedures
  - identifying standards, if any, that should be used in software development processes

Quality Control

- Quality control popularly abbreviated as QC. It is a Software Engineering process used to ensure quality in a product or a service.
- It does not deal with the processes used to create a product; rather it examines the quality of the "end products" and the final outcome.
- The main aim of Quality control is to check whether the products meet the specifications and requirements of the customer. If an issue or problem is identified, it needs to be fixed before delivery to the customer.
- Walkthrough, Testing, inspection, checkpoint review are the example of quality control.
- Activities include:
    - release testing of software, including proper documentation of the testing process
    - examination of software and associated documentation for non-conformance with standards
    - follow-up review of software to ensure any required changes detailed in previous testing are addressed
    - application of software measurement and metrics for assessment

Software Product Quality Standards

- **ISO**
  - ISO(International Standards Organization) is a group or consortium of 63 countries established to plan and fosters standardization.
  - The ISO 9000 standard determines the guidelines for maintaining a quality system.
  - The ISO standard mainly addresses operational methods and organizational methods such as responsibilities, reporting, etc.
  - ISO 9000 defines a set of guidelines for the production process and is not directly concerned about the product itself.
- **Why ISO Certification required by Software Industry?**
  - There are several reasons why software industry must get an ISO certification. Some of reasons are as follows :
  - This certification has become a standards for international bidding.
  - It helps in designing high-quality repeatable software products.
  - It emphasis need for proper documentation.
  - It facilitates development of optimal processes and totally quality measurements.
- **ISO 9000 Software Quality Standards:**
- ISO 9000 is defined as a set of international standards on quality management and quality assurance developed to help companies effectively document the quality system elements needed to maintain an efficient quality system. They are not specific to any one industry and can be applied to organizations of any size.
- ISO 9000 can help a company satisfy its customers, meet regulatory requirements, and achieve continual improvement. It should be considered to be a first step or the base level of a quality system.
- It is defined as the quality assurance system in which quality components can be organizational structure, responsibilities, procedures, processes, and resources for implementing quality management. Quality assurance systems are created to help organizations ensure their products and services satisfy customer expectations by meeting their specifications.
- **ISO 9000 series of Standards**
  - ISO 9001:2015: Quality Management Systems - Requirements
  - ISO 9000:2015: Quality Management Systems - Fundamentals and Vocabulary (definitions)
  - ISO 9004:2018: Quality Management - Quality of an Organization - Guidance to Achieve Sustained Success (continuous improvement)
  - ISO 19011:2018: Guidelines for Auditing Management Systems

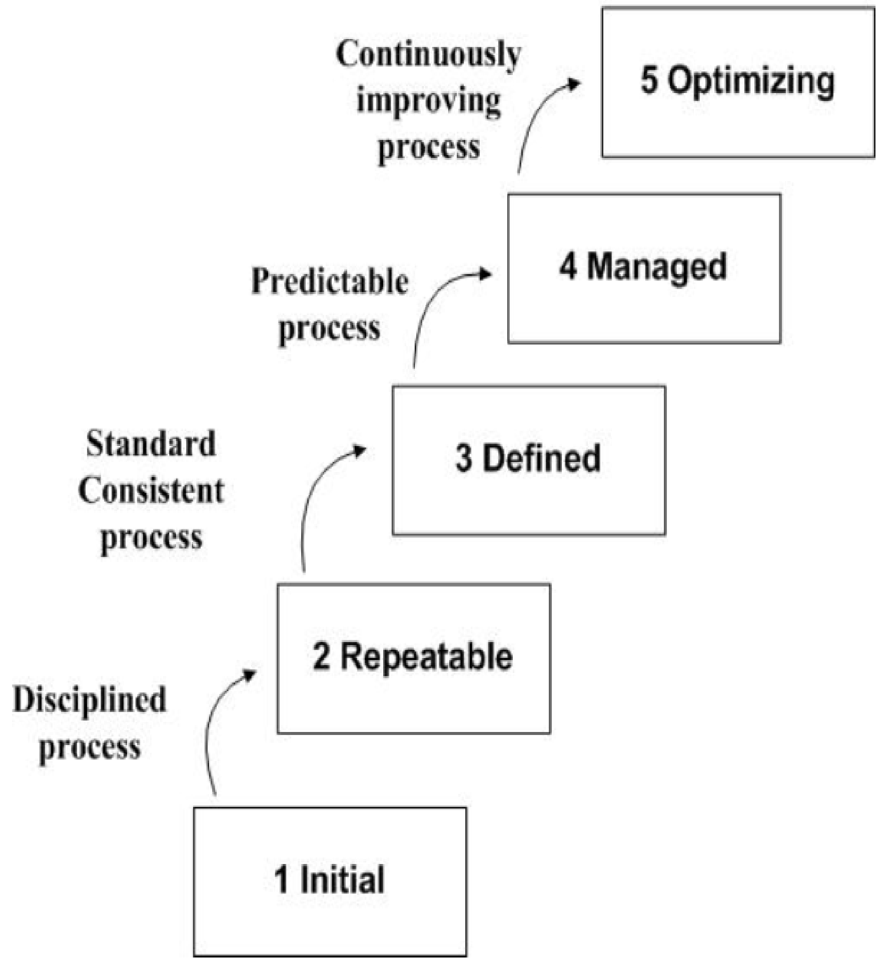
- **ISO 9000 series Quality Management Principles**
  - Principle 1 – Customer focus
    - Organizations depend on their customers and therefore should understand current and future customer needs, should meet customer requirements and strive to exceed customer expectations.
  - Principle 2 – Leadership
    - Leaders establish unity of purpose and direction of the organization. They should create and maintain the internal environment in which people can become fully involved in achieving the organization's objectives.
  - Principle 3 – Engagement of people
    - People at all levels are the essence of an organization and their full involvement enables their abilities to be used for the organization's benefit.
  - Principle 4 – Process approach
    - A desired result is achieved more efficiently when activities and related resources are managed as a process.
  - Principle 5 – Improvement
    - Improvement of the organization's overall performance should be a permanent objective of the organization.
  - Principle 6 – Evidence-based decision making
    - Effective decisions are based on the analysis of data and information.
  - Principle 7 – Relationship management
    - An organization and its external providers (suppliers, contractors, service providers) are interdependent and a mutually beneficial relationship enhances the ability of both to create value.
- **Benefits of ISO 9000:**
- Beyond meeting project owner or prime contractor requirements, certification can produce benefits that the contractor can enjoy. These include:
  - Higher perceived quality of customer service;
  - Improved customer satisfaction;
  - Competitive edge over non-certified competitors;
  - Increased market share;
  - Greater quality awareness;
  - Improved employee morale;
  - Better documentation.

Software Process Quality Standards

- **CMMI:** Capability Maturity Model Interface
- The Capability Maturity Model (CMM) is a procedure used to develop and refine an organization's software development process.
- The Capability Maturity Model Integration (CMMI) is a process and behavioral model that helps organizations streamline process improvement and encourage productive, efficient behaviors that decrease risks in software, product, and service development.
- CMM was developed and is promoted by the Software Engineering Institute (SEI), a research and development center promote by the U.S. Department of Defense (DOD).

- It is a process improvement approach that examines your current processes in place and identifies their weaknesses and strengths. Then appropriate process changes, improvements and modifications are made to change these weaknesses into strengths.
- CMMI model is a collection of set of very effective and reliable best practices that can help an organization improve quality, standards and efficiency. It consists of several process areas such as configuration management, project planning, etc.
- CMMI model is being widely adapted by organizations to prove their efficiency and reputation. Some of the advantages of adapting CMMI model are:
  - It provides good RoI (Return on Investment).
  - When it is adapted, business sees heighted success.
  - It is compatible with other quality related methodologies such as 6-Sigma, ISO Standards and ITIL.
  - CMMI model is always improving and evolving.

- **CMMI Levels:**
  - **Initial**
    - Essentially uncontrolled
    - The software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort and heroics.
  - **Repeatable**
    - Product management procedures defined and used
    - Basic project management processes are established to track cost, schedule, and functionality.
    - The necessary process discipline is in place to repeat earlier successes on projects with similar applications.
  - **Defined**
    - Process management procedures and strategies defined and used
    - The software process for both management and engineering activities is documented, standardized, and integrated into all processes for the organization.
    - All projects use an approved version of the organization's standard software process for developing and maintaining software.
  - **Managed**
    - Quality management strategies defined and used
    - Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.
  - **Optimising**
    - Process improvement strategies defined and used
    - Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.


Continuously improving process
5 Optimizing
Predictable process
4 Managed
Standard Consistent process
3 Defined
Disciplined process
2 Repeatable
1 Initial