SOFTWARE Modelling and Design

UNIT IV

Unit IV: Testing

Syllabus

Testing concepts, Principles of software testing, verification and validation, V-test model, defect management Testing strategies, unit, integration and system testing , acceptance, alpha, beta, performance, security testing ,white box and black box testing, basis path testing, equivalence testing, graph based testing, Test cases and test plan.

Testing Concepts

(Theoretical) Expected Behavior …… (Practically obtained) Actual Behavior

- **Error** : Refers to **difference** between Actual Output **and** Expected output.

- **Fault** : It is a condition that causes the software to **fail** to perform its required function.

- **Failure** : It is the inability of a system or component to perform required function according to its specification

- **Defects** can come from any SDLC phase

What is testing?

- Testing is the process of executing an application / program with the intent of finding defects.
- "Primary role of testing is not demonstration of correct performance, but the exposure of hidden defects." (**G. J. Myers**)
- Testing is process execution of application in controlled manner with the intent of finding the errors. It is nothing but “Detection”.
- Testing is the process of trying to discover every conceivable fault or weakness in a work product.
- A good test is one that has high probability of finding an as yet undiscovered defects/ error/bug.
- The objective of testing is to find all possible defects/error/bug in a work product.
- The goal of software tester is to find defects and find them as early as possible, and make sure they get fixed.
- Identifying the differences between expected and actual results

Cntd..

Why does software have bugs?

- Miscommunication or no communication
- Software complexity
- Programming errors
- Changing requirements
- Time pressures
- Poorly documented code
- software development tools

Principles of software testing

1. **Testing shows presence of defects**

- Testing principle states that - Testing talks about the presence of defects and don't talk about the absence of defects
- Sufficient testing reduces the presence of defects.
- In case testers are unable to find defects after repeated regression testing doesn't mean that the software is bug-free.

2. **Exhaustive testing is impossible:**

- Testing all the functionalities using all valid and invalid inputs and preconditions is known as Exhaustive testing.
- If we keep on testing all possible test conditions then the software execution time and costs will rise.
- So instead of doing exhaustive testing, risks and priorities will be taken into consideration while doing testing and estimating testing efforts.

Cntd..

3. Early testing

- Defects detected in early phases of SDLC are less expensive to fix. So conducting early testing reduces the cost of fixing defects.
- Assume two scenarios, first one is you have identified an incorrect requirement in the requirement gathering phase and the second one is you have identified a bug in the fully developed functionality.
- It is cheaper to change the incorrect requirement compared to fixing the fully developed functionality which is not working as intended.

4. Defect clustering

- Defect Clustering in software testing means that a small module or functionality contains most of the bugs or it has the most operational failures.
- As per the Pareto Principle (80-20 Rule), 80% of issues comes from 20% of modules and remaining 20% of issues from remaining 80% of modules. So we do emphasize testing on the 20% of modules where we face 80% of bugs.

Cntd..

5.Pesticide paradox

- Pesticide Paradox in software testing is the process of repeating the same test cases again and again, eventually, the same test cases will no longer find new bugs.

- So to overcome this Pesticide Paradox, it is necessary to review the test cases regularly and add or update them to find more defects.

6. Testing is context dependent

- Testing approach depends on the context of the software we develop.
- We do test the software differently in different contexts.
- example, online banking application requires a different approach of testing compared to an e-commerce site.

Cntd..

7. Absence of error – fallacy

- 99% of bug-free software may still be unusable, if wrong requirements were incorporated into the software and the software is not addressing the business needs.
- The software which we built not only be a 99% bug-free software but also it must fulfill the business needs otherwise it will become an unusable software.

Myth: "Principles are just for reference. I will not use them in practice ."

This is so very untrue. Test Principles will help you create an effective Test Strategy and draft error catching test cases.

Objectives of testing

- Executing a program with the intent of finding an *error*.
- To check if the system meets the requirements and be executed successfully in the Intended environment.
- To check if the system is “ Fit for purpose”.
- To check if the system does what it is expected to do.
- A good test case is one that has a probability of finding an as yet undiscovered error.
- A successful test is one that uncovers a yet undiscovered error.
- A good test is not redundant.
- A good test should be “best of breed”.
- A good test should neither be too simple nor too complex

verification and validation

VERIFICATION

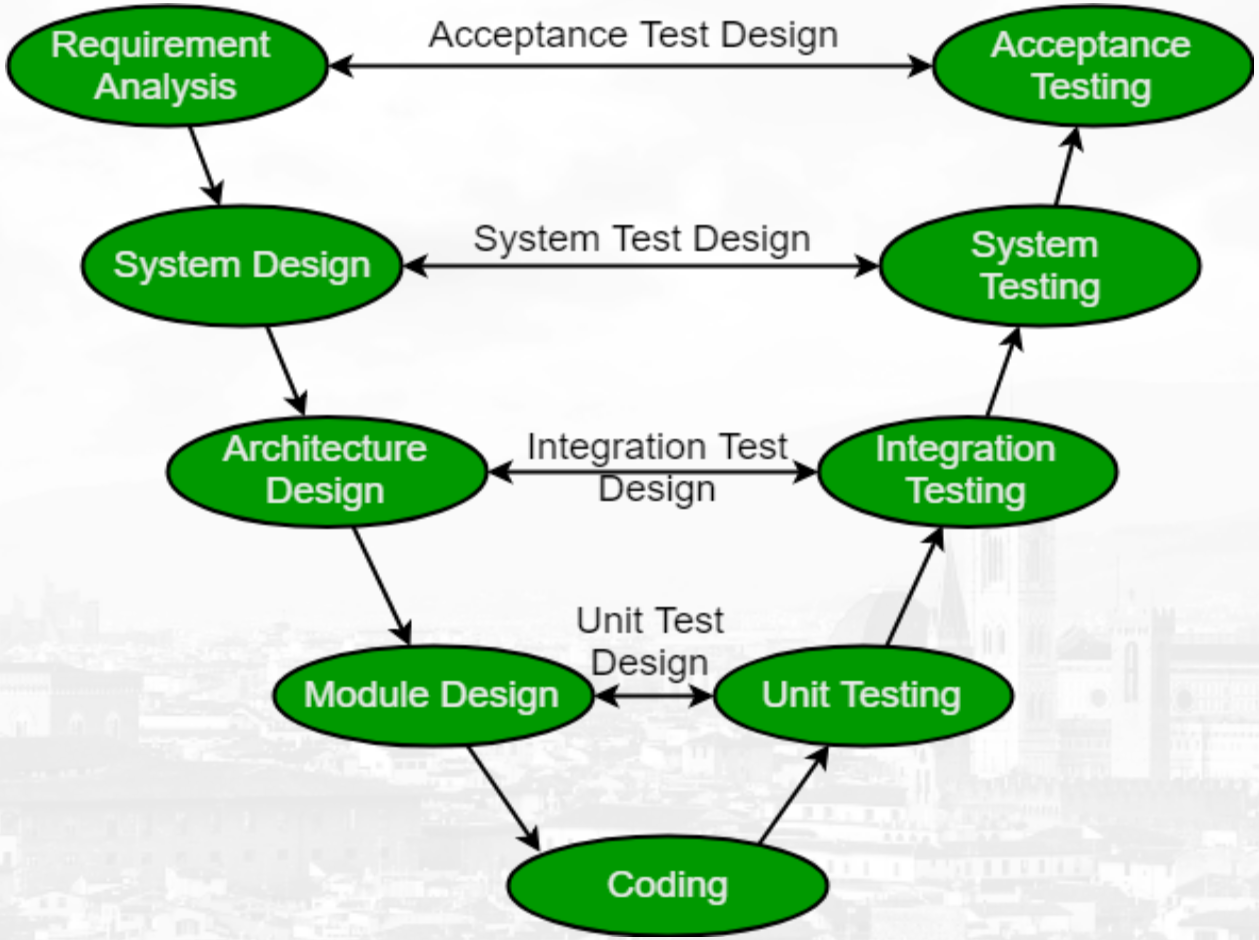
- Is the process of evaluating a system or component to determine whether the products of a given phase satisfy the conditions imposed at the start of that phase.

VALIDATION:

- Is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies the specified requirements

Sr. No.	VERIFICATION	VALIDATION
1	It is the process of confirming whether software meets its specification.	It is the process of confirming whether software meets user requirements.
2	It is the process of reviewing /inspecting deliverables throughout the life cycle.	It is the process of executing something to see how it behaves.
3	Inspections, Walkthrough and Reviews are the examples of Verification techniques.	Unit, Integration, System, & Acceptance testing are the examples of Validation Testing.
4	Verification is usually performed by Static Testing, Inspecting without executing on a computer.	Validation is usually performed by Dynamic Testing, Testing with execution on a computer.
5	It is the process of determining if phase completed correctly.	It is the process of determining, if product as a whole satisfies the requirement.
6	Verification is the process of examining the product to discover its defects.	Validation is the process of executing a product to expose its defects.
7	Are we building the product right?	Are we building the right product?
8	An effective document for verification is checklist.	Various Manual and Automated test tool are available. (MS Excel, Win Runner, QTP, etc.)
9	It finds the errors early in the requirement & design phase, Hence reduces the cost of fixing.	It finds the errors only after the code is ready, Hence the cost of fixing is higher than it is at verification.

V-Test Model



V-Test Model

Cntd..

The V-model is a type of SDLC model where process executes in a sequential manner in V-shape.

- It is also known as Verification and Validation model.

Verification: It involves static analysis technique (review) done without executing code.

- It is the process of evaluation of the product development phase to find whether specified requirements meet.

Validation: It involves dynamic analysis technique (functional, non-functional), testing done by executing code.

- Validation is the process to evaluate the software after the completion of the development phase to determine whether software meets the customer expectations and requirements.

So V-Model contains Verification phases on one side of the Validation phases on the other side.

Verification and Validation phases are joined by coding phase in V-shape. Thus it is called V-Model.

Defect Management Testing Strategies

- **What is Bug?**

A bug is the consequence/outcome of a coding fault.

What is Defect?

- A defect is a variation or deviation from the original business requirements

These two terms have very thin line of difference, In the Industry both are faults that need to be fixed and so interchangeably used by some of the Testing teams.

When a tester executes the test cases, he/she might come across the test result which is contradictory to expected result. This variation in the test result is referred as a **Software Defect**.

These defects or variation are referred by different names in a different organization like **issues, problem, bug or incidents**.

Cntd..

Defect Management Process

Cntd..

Discovery

- In the discovery phase, the project teams have to discover as **many** defects as **possible,** before the end customer can discover it.
- A defect is said to be discovered and change to status **accepted** when it is acknowledged and accepted by the developers

Cntd.

Let's have a look at the following scenario; your testing team discovered some issues in the Bank website.

They consider them as defects and reported to the development team, but there is a conflict -

Cntd..

In such case, as a Test Manager, what will you do?

A) Agree With the test team that its a defect

B) Test Manager takes the role of judge to decide whether the problem is defect or not

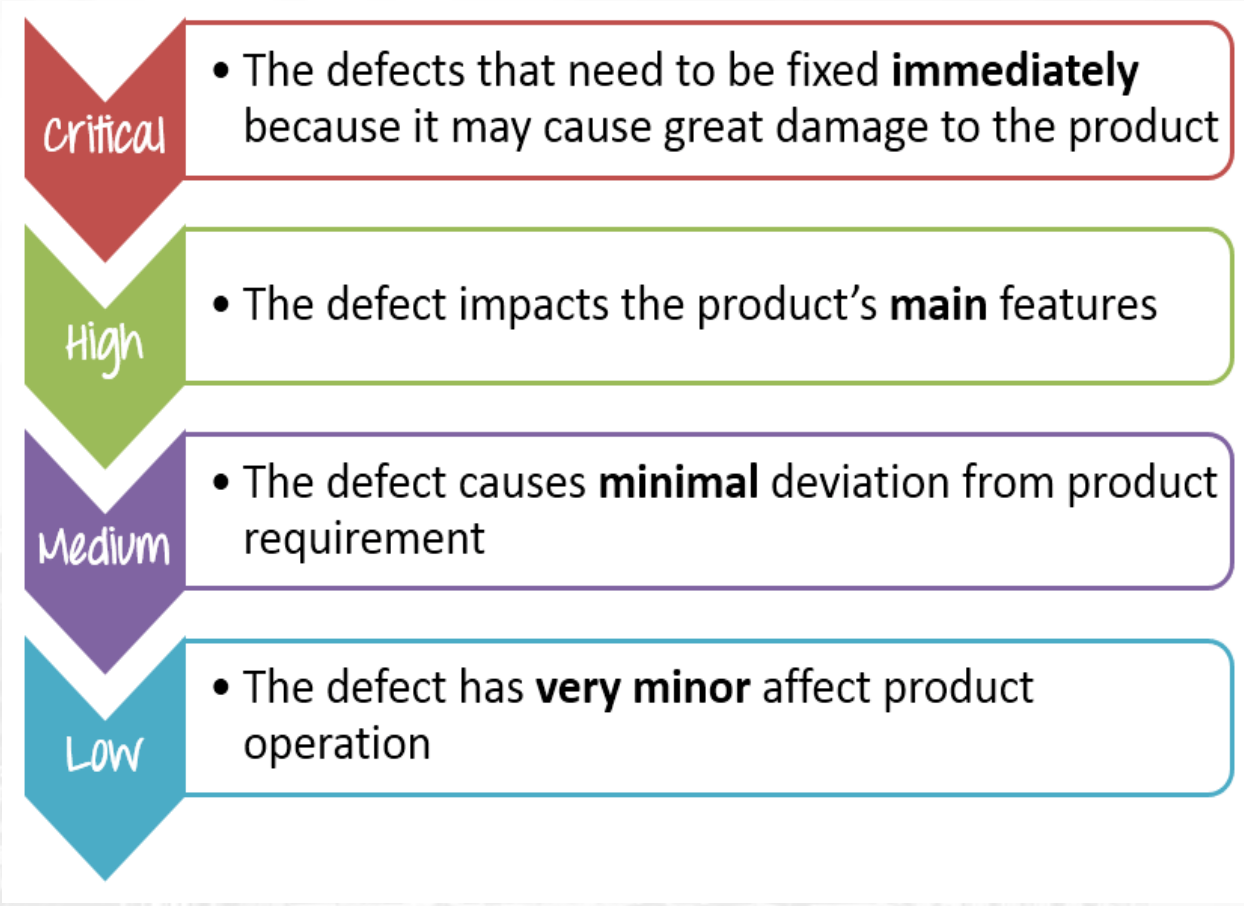
C) Agree with the development team that is not a defect

- In such case, a resolution process should be applied to solve the conflict, you take the role as a judge to decide whether the website problem is a defect or not.

Cntd.

Categorization: Defect categorization help the software developers to prioritize their tasks.

That means that this kind of priority helps the developers in fixing those defects first that are highly crucial



Cntd..

Defects are usually categorized by the Test Manager –

Let's do a small exercise as following **Drag & Drop the Defect Priority Below**

Low, Medium, High, Critical

1) The website performance is too slow
2) The login function of the website does not work properly
3) The GUI of the website does not display correctly on Mobile devices
4) The website could not remember the user login session
5) Some links doesn't work

Cntd.

No.	Description	Priority	Explanation
1	The website performance is too slow	High	The performance bug can cause huge inconvenience to user.
2	The login function of the website does not work properly	Critical	Login is one of the main function of the banking website if this feature does not work, it is serious bugs
3	The GUI of the website does not display correctly on mobile devices	Medium	The defect affects the user who use Smartphone to view the website.
4	The website could not remember the user login session	High	This is a serious issue since the user will be able to login but not be able to perform any further transactions
5	Some links doesn't work	Low	This is an easy fix for development guys and the user can still access the site without these links

Cntd..

Resolution: Once the defects are accepted and categorized, you can follow the following steps to fix the defect.

Cntd..

- **Assignment**: Assigned to a developer or other technician to fix, and changed the status to **Responding**.
- **Schedule fixing**: The developer side take charge in this phase. They will create a schedule to fix these defects, depend on the defect priority.
- **Fix the defect**: While the development team is fixing the defects, the Test Manager tracks the process of fixing defect compare to the above schedule.
- **Report the resolution**: Get a report of the resolution from developers when defects are fixed.

Cntd..

Verification

- After the development team **fixed** and **reported** the defect, the testing team **verifies** that the defects are actually resolved.
- For example, in the above scenario, e.g when the development team reported that they already fixed 61 defects, your team would test again to verify these defects were actually fixed or not.

Closure

- Once a defect has been resolved and verified, the defect is changed status as **closed**. If not, you have send a notice to the development to check the defect again.

Reporting

- The management board has right to know the defect status. They must understand the defect management process to support you in this project. Therefore, you must report them the current defect situation to get feedback from them.

Most bugs are not because of mistakes in the code …

- Specification (~= 55%)
- Design (~= 25%)
- Code (~= 15%)
- Other (~= 5%)

SOFTWARE TESTING LIFECYCLE - PHASES

- Requirements Analysis
- Test Planning
- Test Case Development
- Environment Set up
- Test Execution
- Test cycle closure

- **Requirement Analysis** is the first step involved in Software testing life cycle. In this step, Quality Assurance (QA) team understands the requirement in terms of what we will testing & figure out the testable requirements.

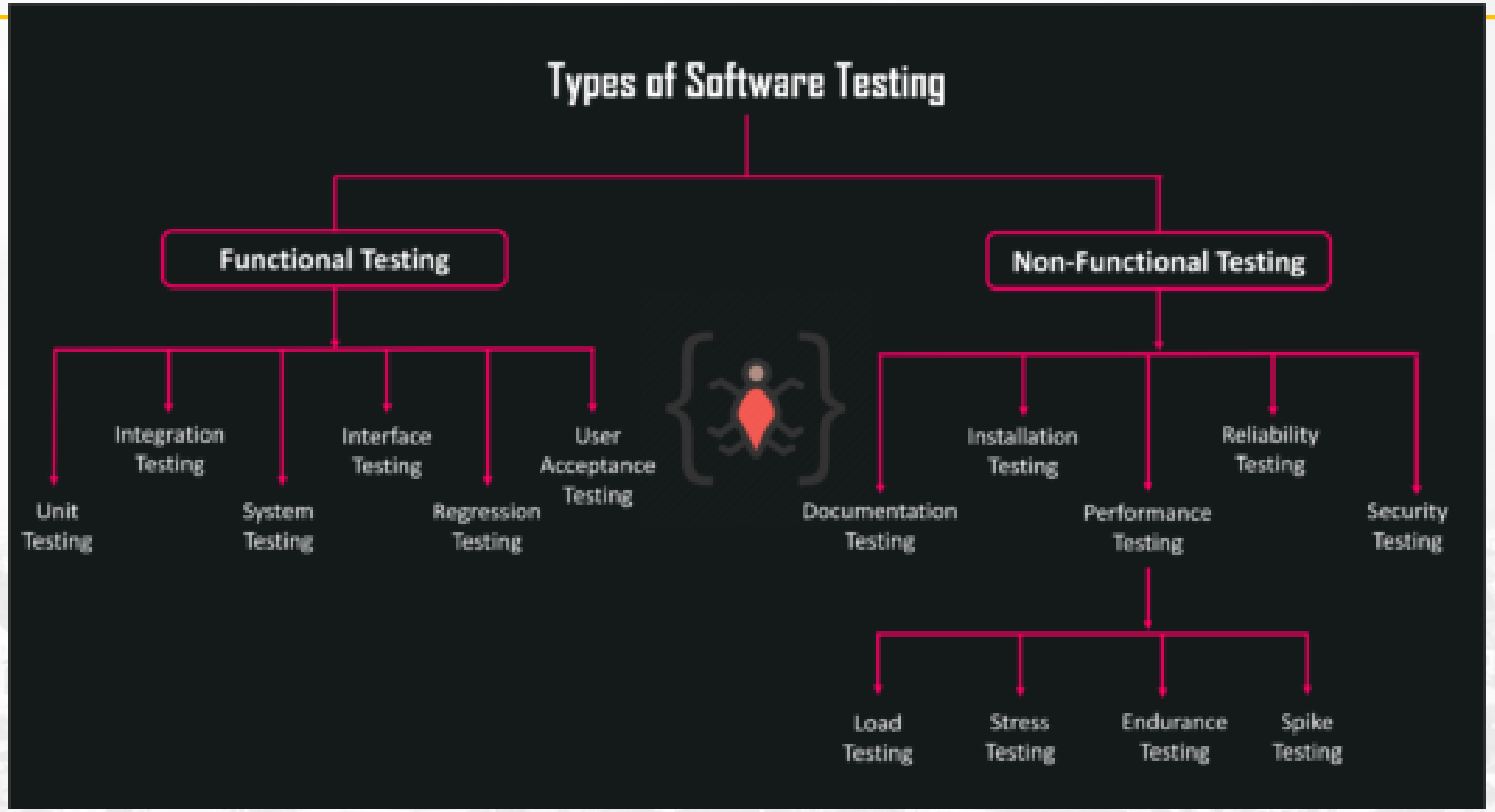
- **Test Planning** – Test Planning is most important phase of Software testing life cycle where all testing strategy is defined. This phase is also called as **Test Strategy** phase. In this phase, Test Manager is involved to determine the effort and cost estimates for entire project. It defines the objective & scope of the project.

- **Test Case Development** – The Test case development begins once the test planning phase is completed. This is the phase of STLC where testing team notes the detailed test cases. Along with test cases, testing team also prepares the test data for testing. Once the test cases are ready then these test cases are reviewed by peer members or QA lead.

- **Test Environment Setup** – Setting up the test environment is vital part of the Software Testing Life Cycle. A testing environment is a setup of software and hardware for the testing teams to execute test cases. It supports test execution with hardware, software and network configured.

- **Test Execution** – The next phase in Software Testing Life Cycle is Test Execution. Test execution is the process of executing the code and comparing the expected and actual results. When test execution begins, the test analysts start executing the test scripts based on test strategy allowed in the project.

- **Test Cycle Closure** – The final phase of the Software Testing Life Cycle is Test Cycle Closure. It involves calling out the testing team member meeting & evaluating cycle completion criteria based on Test coverage, Quality, Cost, Time, Critical Business Objectives, and Software.


Types of Software Testing
Functional Testing
Non-Functional Testing
Unit Testing
Integration Testing
System Testing
Interface Testing
Regression Testing
User Acceptance Testing
Documentation Testing
Installation Testing
Performance Testing
Reliability Testing
Security Testing
Load Testing
Stress Testing
Endurance Testing
Spike Testing

Non-Functional Testing

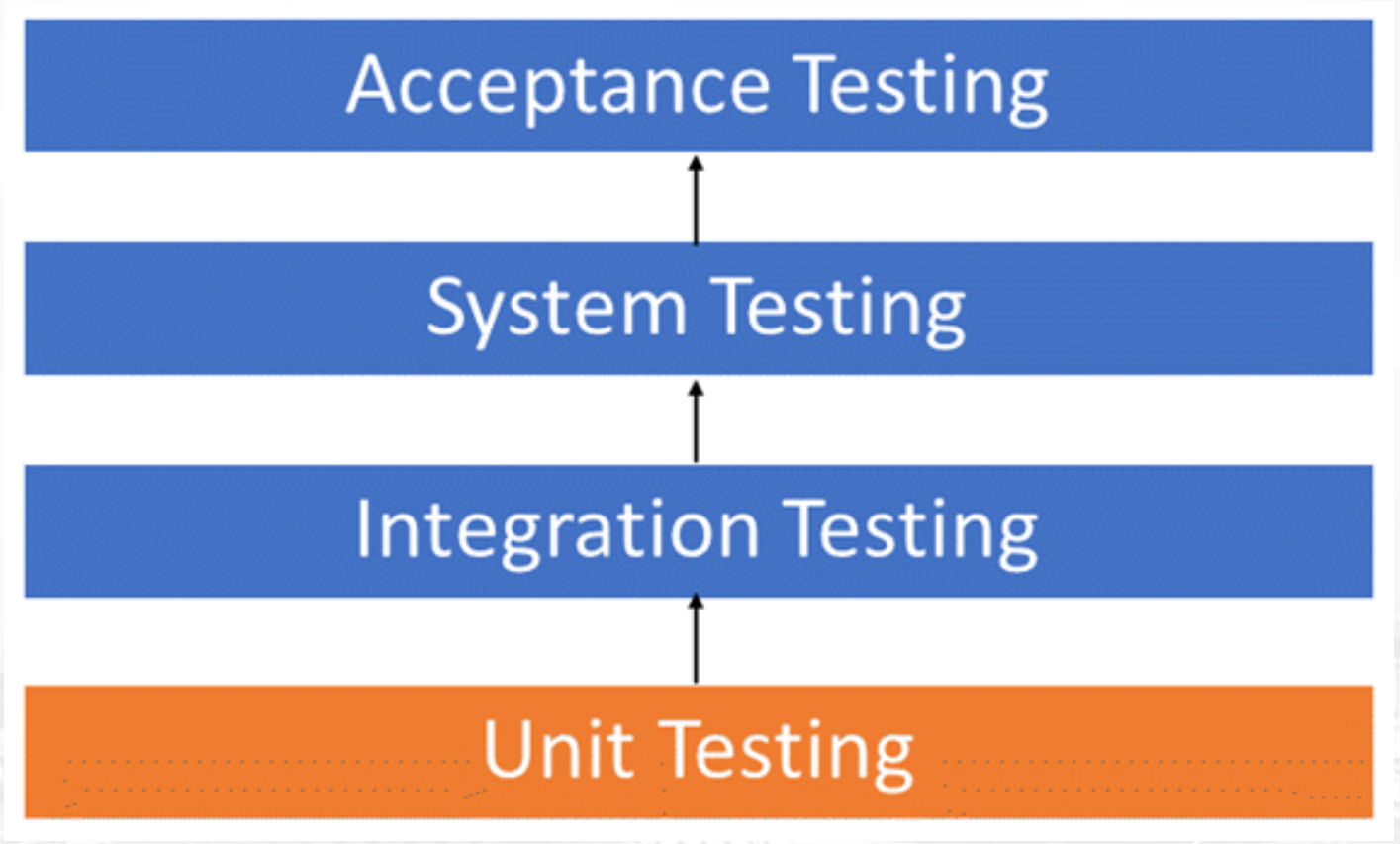
- There are a bunch of software testing types which differentiate the job work for the QA while testing the apps. It is a testing to determine the performance of the system to measure, validate or verify quality attribute of the system.
- The different **Types of Non-Functional Testing** are :
  - **Documentation Testing**
  - **Installation Testing**
  - **Performance Testing**
  - **Reliability Testing**
  - **Security Testing**

- **Functional Testing**

  Functional Testing is defined as a type of testing which verifies that each function of the software application operates in conformance with the requirement specification. The functional testing focuses on manual as well as automation testing.

- The Different Types of Functional Testing include :
  - Unit testing
  - Integration testing
  - System testing
  - Interface Testing
  - Acceptance testing
  - Regression Testing

Cntd..


Acceptance Testing
System Testing
Integration Testing
Unit Testing

Unit Testing

- UNIT Testing is defined as a type of software testing **where individual units/ components of a software are tested**.
- Unit Testing of software applications is done during the development (coding) of an application.
- The objective of Unit Testing is to isolate a section of code and verify its correctness.
- In procedural programming, a unit may be an individual function or procedure. Unit Testing is usually performed by the developer.
- In SDLC, STLC, V Model, Unit testing is first level of testing done before integration testing.
- Unit testing is a White Box testing technique that is usually performed by the developer.

Cntd..

Unit-test considerations:

1. The **module interface is tested** to ensure that information properly flows into and out of the program unit under test.
2. **Local data structures** are examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution.
3. **All independent paths** through the control structure are exercised to ensure that all statements in a module have been executed at least once.

Cntd..

4. **Boundary conditions are tested** to ensure that the module operates properly at boundaries established to limit or restrict processing.

5. And finally, **all error-handling paths are tested.**

6. **Data flow across a component interface** is tested before any other testing is initiated.

Cntd..

Why Unit Testing?

- Sometimes software developers attempt to save time by doing minimal unit testing.
- This is a myth because skipping on unit testing leads to higher Defect fixing costs during System Testing, Integration Testing and even Beta Testing after the application is completed.
- Proper unit testing done during the development stage saves both time and money in the end. Here, are key reasons to perform unit testing.

Cntd..

- Unit Tests fix bug early in development cycle and save costs.
- It helps understand the developers the code base and enable them to make changes quickly
- Good unit tests serve as project documentation
- Unit tests help with code re-use. Migrate both your code **and** your tests to your new project.
- Tweak the code till the tests run again.

Cntd..

How to do Unit Testing

Unit Testing is of two types

- Manual
- Automated

Unit testing is commonly automated but may still be performed manually.

Software Engineering does not favor one over the other but automation is preferred.

A manual approach to unit testing may employ a step-by-step instructional document.

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. Integration Testing focuses on checking data communication amongst these modules. Hence it is also termed as **'I & T'** (Integration and Testing)

Example of Integration Test Case:

- Integration Test Case differs from other test cases in the sense it **focuses mainly on the interfaces & flow of data/information between the modules**.
- Here priority is to be given for the **integrating links** rather than the unit functions which are already tested.
- Sample Integration Test Cases for the following scenario: Application has 3 modules say 'Login Page', 'Mailbox' and 'Delete emails' and each of them is integrated logically.
- Here do not concentrate much on the Login Page testing as it's already been done in Unit Testing. But check how it's linked to the Mail Box Page.
- Similarly Mail Box: Check its integration to the Delete Mails Module.

Cntd.

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

Cntd..

Approaches/Methodologies/Strategies of Integration Testing:

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

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

Cntd..

Big Bang Approach:

- Here all component are integrated together at **once** and then tested.

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.
- 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.
- 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.

Cntd..

Incremental Approach

- In this approach, testing is done by joining two or more modules that are ***logically related***. Then the other related modules are added and tested for the proper functioning. The process continues until all of the modules are joined and tested successfully.

Incremental Approach, in turn, is carried out by two different Methods:

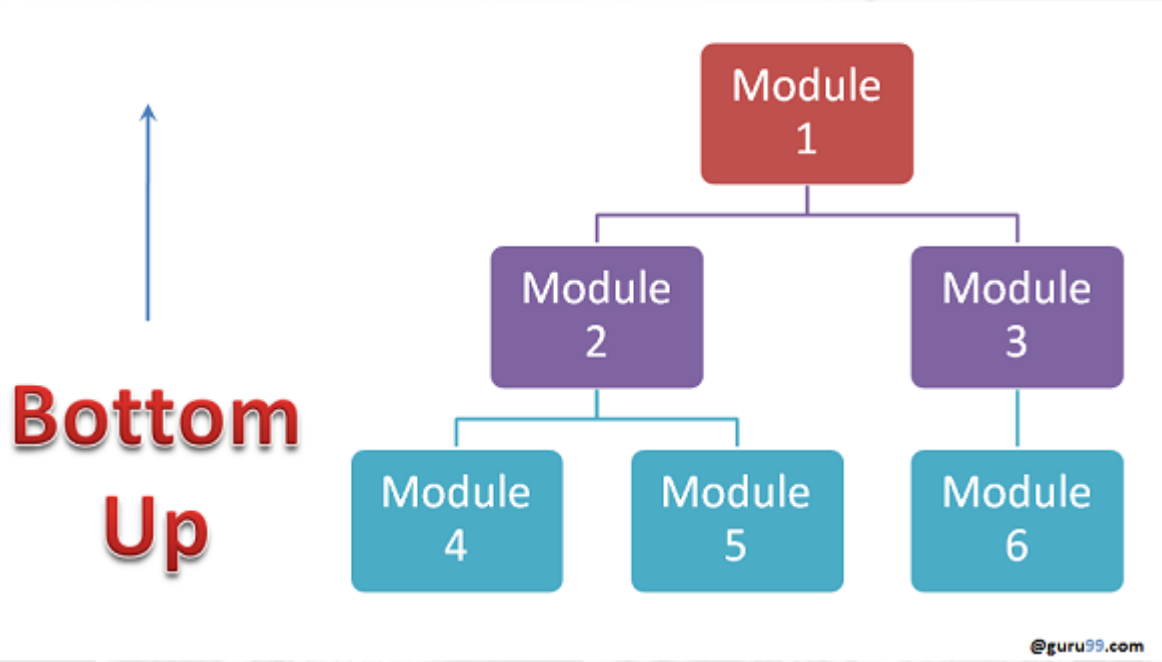
- Bottom Up
- Top Down

Bottom-up Integration

Stub and Driver

Incremental Approach is carried out by using dummy programs called **Stubs and Drivers**. Stubs and Drivers do not implement the entire programming logic of the software module but just simulate data communication with the calling module.

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



Cntd..

Advantages:

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

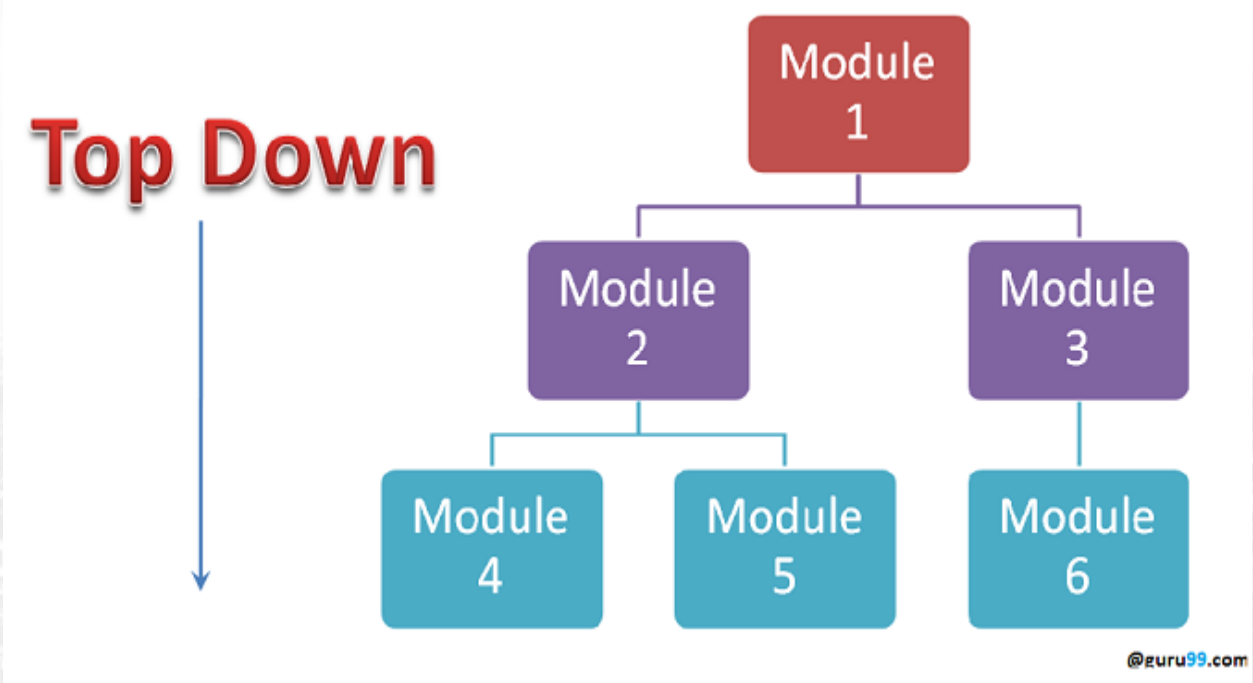
Disadvantages:

- 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

Cntd..

Top-down Integration:

- In Top to down approach, testing takes place from top to down following the control flow of the software system.
- Takes help of stubs for testing.



Cntd..

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.

System Testing

- System Testing is the testing of a complete and fully integrated software product.
- Usually, software is only one element of a larger computer-based system.
- Ultimately, 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.

Two Category of Software Testing: **Black Box Testing & White Box Testing**

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

White box testing is the testing of the internal workings or code of a software application. **In contrast**, black box or System Testing is the opposite.

System test involves the external workings of the software from the user's perspective.

Acceptance testing - beta testing of the product done by the actual end users.

Cntd..

Different Types of System Testing:

Usability Testing - Usability Testing mainly focuses on the user's ease to use the application, flexibility in handling controls and ability of the system to meet its objectives

Load Testing - Load Testing is necessary to know that a software solution will perform under real-life loads.

Regression Testing- Regression Testing involves testing done to make sure none of the changes made over the course of the development process have caused new bugs.
It also makes sure no old bugs appear from the addition of new software modules over time.

Recovery Testing - Recovery testing is done to demonstrate a software solution is reliable, trustworthy and can successfully recover from possible crashes.

Cntd..

Migration Testing - Migration testing is done to ensure that the software can be moved from older system infrastructures to current system infrastructures without any issues.

Hardware/Software Testing - IBM refers to Hardware/Software testing as "HW/SW Testing". This is when the tester focuses his/her attention on the interactions between the hardware and software during system testing.

Acceptance Testing

- User Acceptance is defined as a type of testing performed by the Client to certify the system with respect to the requirements that was agreed upon.
- This testing happens in the final phase of testing before moving the software application to the Market or Production environment.
- The main purpose of this testing is to validate the end to end business flow.
- It does NOT focus on cosmetic errors, Spelling mistakes or System testing.
- This testing is carried out in a separate testing environment with production like data setup. It is a kind of black box testing where two or more end users will be involved.

Cntd..

Who Performs UAT?

- Client
- End users

Need of User Acceptance Testing:

- Once software has undergone Unit, Integration, and System testing the need of Acceptance Testing may seem redundant. **But Acceptance Testing is required because**

1. Developers have included features on their "own" understanding
2. Requirements changes "not communicated" effectively to the developers

Cntd..

- **Prerequisites of User Acceptance Testing:**

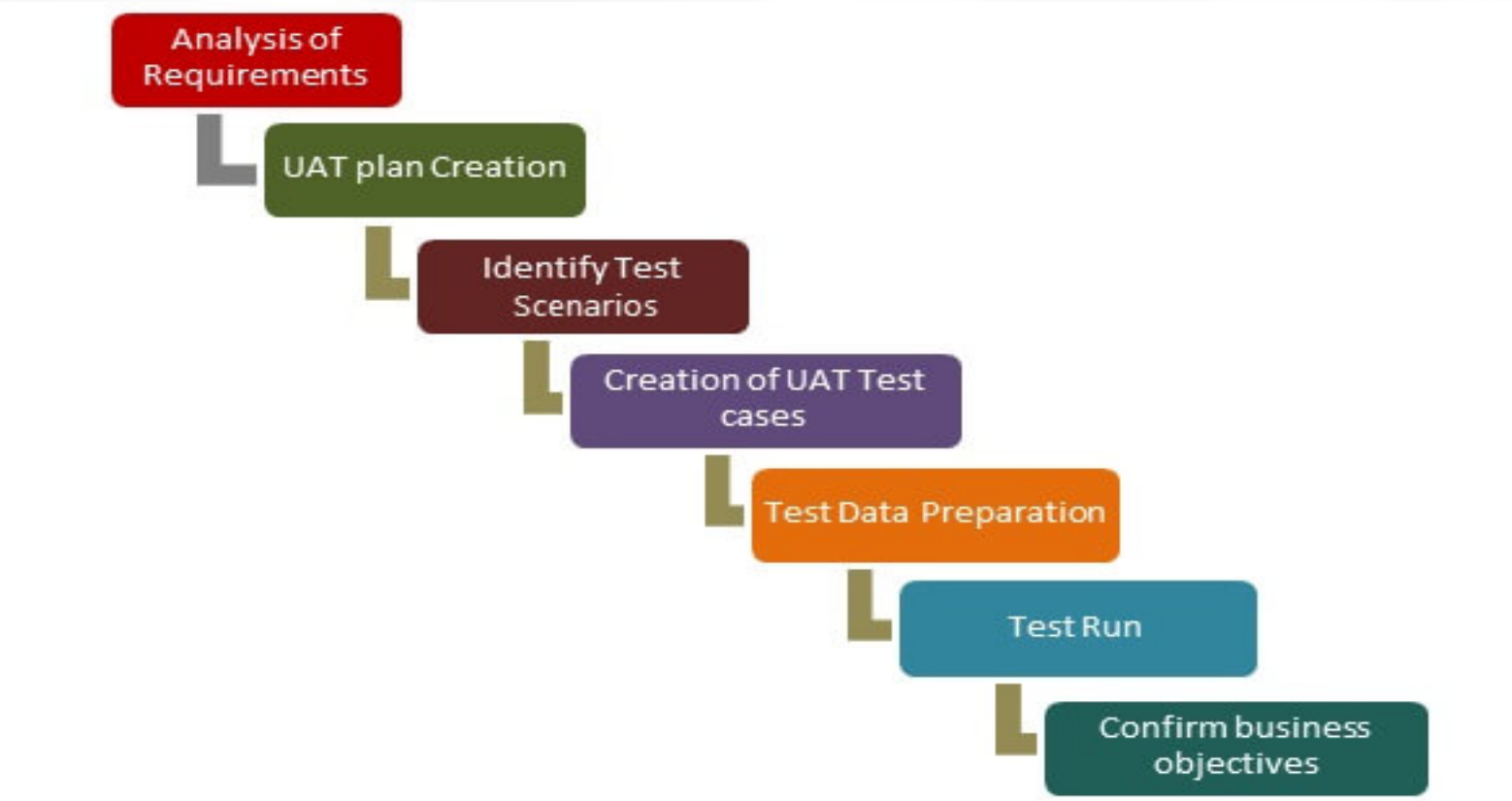
Following are the entry criteria for User Acceptance Testing:

- Business Requirements must be available.
- Application Code should be fully developed
- Unit Testing, Integration Testing & System Testing should be completed
- No Show-stoppers, High, Medium defects in System Integration Test Phase
- Only Cosmetic error is acceptable before UAT
- Regression Testing should be completed with no major defects
- All the reported defects should be fixed and tested before UAT
- Traceability matrix for all testing should be completed
- UAT Environment must be ready
- Sign off mail or communication from System Testing Team that the system is ready for UAT execution

Cntd..

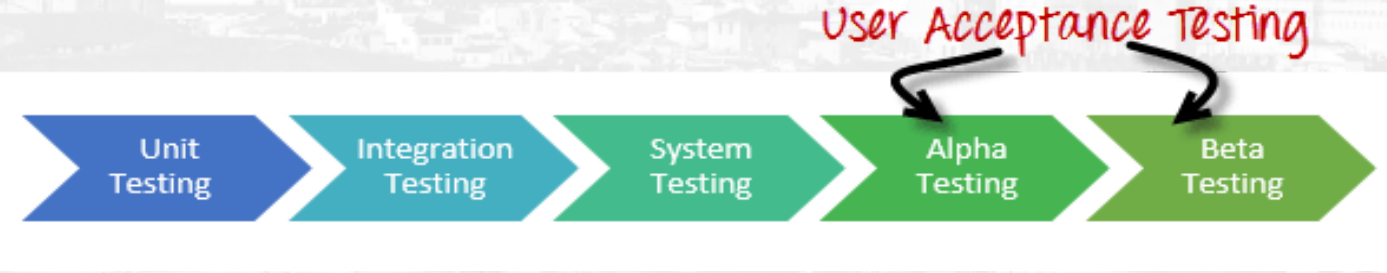
How to do UAT Testing

- UAT is done by the intended users of the system or software.
- This type of Software Testing usually happens at the client location which is known as Beta Testing.
- Once Entry criteria for UAT are satisfied, following are the tasks need to be performed by the testers:



Alpha Testing

- Alpha testing is a type of acceptance testing; performed to identify all possible issues/bugs before releasing the product to everyday users or the public.
- The focus of this testing is to simulate real users by using a black box & white box techniques.
- The aim is to carry out the tasks that a typical user might perform.
- Alpha testing is carried out in a lab environment and usually, the testers are internal employees of the organization.
- To put it as simple as possible, this kind of testing is called alpha only because it is done early on, near the end of the development of the software, and before beta testing.



Cntd.

Advantages of Alpha Testing:

- Provides better view about the reliability of the software at an early stage
- Helps simulate real time user behavior and environment.
- Detect many showstopper or serious errors
- Ability to provide early detection of errors with respect to design and functionality

Disadvantages of Alpha Testing:

- In depth, functionality cannot be tested as software is still under development stage Sometimes developers and testers are dissatisfied with the results of alpha testing

Beta Testing

- Beta Testing of a product is performed by "real users" of the software application in a "real environment" and can be considered as a form of external User Acceptance Testing.
- Beta version of the software is released to a limited number of end-users of the product to obtain feedback on the product quality.
- Beta testing reduces product failure risks and provides increased quality of the product through customer validation.
- It is the final test before shipping a product to the customers.
- Direct feedback from customers is a major advantage of Beta Testing.
- This testing helps to tests the product in customer's environment.

Cntd.

Advantages of Beta Testing

- Reduces product failure risk via customer validation.
- Beta Testing allows a company to test post-launch infrastructure.
- Improves product quality via customer feedback
- Cost effective compared to similar data gathering methods
- Creates goodwill with customers and increases customer satisfaction

Disadvantages of Beta Testing

- Test Management is an issue.
- As compared to other testing types which are usually executed inside a company in a controlled environment, beta testing is executed out in the real world where you rarely have control.
- Finding the right beta users and maintaining their participation could be a challenge

Alpha Testing Vs. Beta testing:

Alpha Testing	Beta Testing
1.Alpha testing performed by Testers who are usually internal employees of the organization	Beta testing is performed by Clients or End Users who are not employees of the organization
2.Alpha Testing performed at developer's site	Beta testing is performed at a client location or end user of the product
3.Reliability and Security Testing are not performed in-depth Alpha Testing	Reliability, Security, Robustness are checked during Beta Testing
4.Alpha testing involves both the white box and black box techniques	Beta Testing typically uses Black Box Testing
5.Alpha testing requires a lab environment or testing environment	Beta testing doesn't require any lab environment or testing environment. The software is made available to the public and is said to be real time environment
6.Long execution cycle may be required for Alpha testing	Only a few weeks of execution are required for Beta testing
7.Critical issues or fixes can be addressed by developers immediately in Alpha testing	Most of the issues or feedback is collected from Beta testing will be implemented in future versions of the product
8.Alpha testing is to ensure the quality of the product before moving to Beta testing	Beta testing also 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.

Performance Testing

- Performance Testing is defined as a type of software testing to ensure software applications will perform well under their expected workload.
- Features and Functionality supported by a software system is not the only concern.
- A software application's performance like its response time, reliability, resource usage and scalability do matter.
- The goal of Performance Testing is not to find bugs but to eliminate performance bottlenecks.

The focus of Performance Testing is checking a software program's

Speed - Determines whether the application responds quickly

Scalability - Determines maximum user load the software application can handle.

Stability - Determines if the application is stable under varying loads

Types of Performance Testing

Load testing - checks the application's ability to perform under anticipated user loads.

The objective is to identify performance bottlenecks before the software application goes live.

Stress testing - involves testing an application under extreme workloads to see how it handles high traffic or data processing.

The objective is to identify the breaking point of an application.

Endurance testing - is done to make sure the software can handle the expected load over a long period of time.

Spike testing - tests the software's reaction to sudden large spikes in the load generated by users.

Volume testing - Under Volume Testing large no. of. Data is populated in a database and the overall software system's behavior is monitored.

The objective is to check software application's performance under varying database volumes.

Scalability testing - The objective of scalability testing is to determine the software application's effectiveness in "scaling up" to support an increase in user load.

It helps plan capacity addition to your software system.

Cntd..

Example Performance Test Cases

- Verify response time is not more than 4 secs when 1000 users access the website simultaneously.
- Verify response time of the Application Under Load is within an acceptable range when the network connectivity is slow
- Check the maximum number of users that the application can handle before it crashes.
- Check database execution time when 500 records are read/written simultaneously.
- Check CPU and memory usage of the application and the database server under peak load conditions
- Verify response time of the application under low, normal, moderate and heavy load conditions.

Which Applications should we Performance Test?

Performance Testing is always done for client-server based systems only. This means, any application which is not a client-server based architecture, must not require Performance Testing.

Security Testing

- Security Testing is defined as a type of Software Testing that ensures software systems and applications are free from any vulnerabilities, threats, risks that may cause a big loss.
- Security testing of any system is about finding all possible loopholes and weaknesses of the system which might result into a loss of information, revenue, repute at the hands of the employees or outsiders of the Organization.
- The goal of security testing is to identify the threats in the system and measure its potential vulnerabilities, so the system does not stop functioning or is exploited.
- It also helps in detecting all possible security risks in the system and help developers in fixing these problems through coding.

Cntd..

Types of Security Testing:

- Vulnerability Scanning
- Security Scanning
- Penetration testing
- Risk Assessment
- Security Auditing
- Posture Assessment
- Ethical hacking

Cntd.

- **Vulnerability Scanning**: This is done through automated software to scan a system against known vulnerability signatures.
- **Security Scanning:** It involves identifying network and system weaknesses and later provides solutions for reducing these risks.

  This scanning can be performed for both Manual and Automated scanning.
- **Penetration testing**: This kind of testing simulates an attack from a malicious hacker.

  This testing involves analysis of a particular system to check for potential vulnerabilities to an external hacking attempt.
- **Risk Assessment:** This testing involves analysis of security risks observed in the organization.

  Risks are classified as Low, Medium and High. This testing recommends controls and measures to reduce the risk.
- **Security Auditing:** This is an internal inspection of Applications and Operating systems for security flaws. An audit can also be done via line by line inspection of code

Cntd.

- **Ethical hacking:** It's hacking an Organization Software systems. Unlike malicious hackers, who steal for their own gains, the intent is to expose security flaws in the system.
- **Posture Assessment:** This combines Security scanning, Ethical Hacking and Risk Assessments to show an overall security posture of an organization.

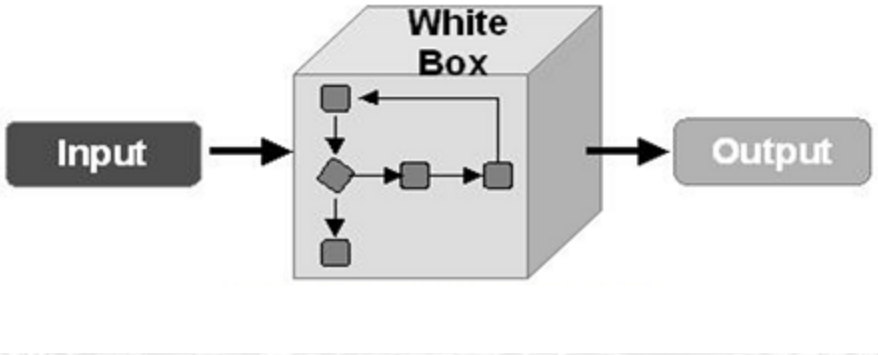
White Box Testing

- 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.

Cntd.

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



Cntd.

1. Static testing by humans:

Advantages

- Can find errors the computers cannot find.
- More humans working together = multiple perspectives, better than computer.
- Humans can compare code against the specification
- Saves computer resources (more human resources)
- Sooner is the defect find, lesser is the cost of fixing it.

Cntd.

A.DESK CHECKING OF THE CODE:

- Done manually by author of the code
- Method to verify portions of the code for correctness.
- Code is compared with the design or specification.
- No structured method / formalism
- No maintaining of logs

Disadvantages:

- Developer is not best person to test his own code.
- Developers prefer writing new code than testing the previous
- Is person dependent method

Cntd.

B. Code walkthrough

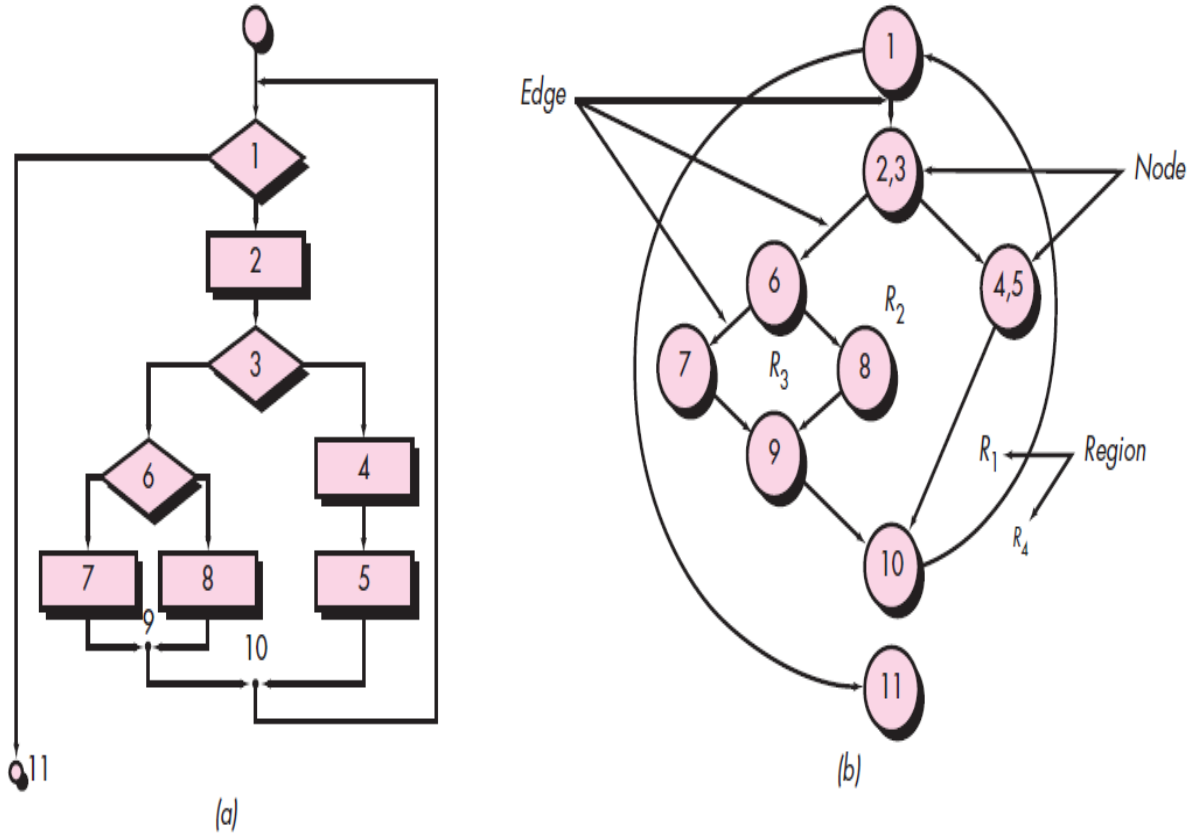
- Are group oriented methods.
- A set of people look at the code and raise questions for the author of code. Author explains logic of code and answers the questions.
- If author is unable to answer some questions, he takes those questions and finds answers.
- **COMPLETENESS**

Cntd.

C. Formal inspection

- code inspection
- High degree of formalism
- Focus is to detect all faults, violations, other side effects.

- Mapping of flowchart to flowgraph



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 + 2

  where E is the number of flow graph edges and N is the number of flow graph nodes.
- **3. Cyclomatic complexity V(G) for a flow graph G is also defined as**

  V(G) = P + 1

  where P is the number of predicate nodes contained in the flow graph G.

1. The flow graph has four regions.

2. V(G) = 11 edges - 9 nodes + 2 = 4

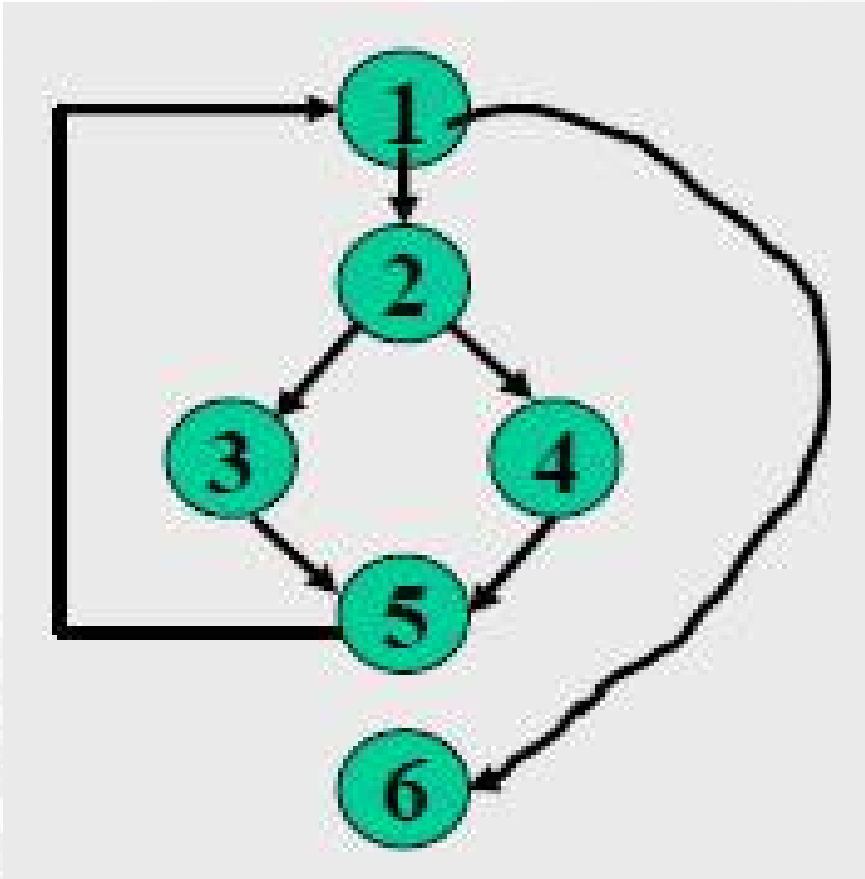
3. V(G) = 3 predicate nodes + 1 = 4

Therefore, the cyclomatic complexity of the flow graph is 4.

- The value for V(G) provides you
  - An **upper bound for the number of independent paths that form the basis set**
  - **An upper bound on the number of tests** that must be designed and executed to **guarantee coverage of all program statements**.

- **Example**

```
int f1(int x,int y)
{
 while (x != y)
{
 if (x>y) then
 x=x-y;
else y=y-x;
 }
return x;
}
```



Cyclomatic complexity = 7 – 6 + 2 = 3.

V(G) = Total number of bounded areas + 1

Example

From a visual examination of the CFG:

The number of bounded areas is 2.

cyclomatic complexity = 2+1=3.

- Number of independent paths: 3
- 1, 6 test case (x=1, y=1)
- 1, 2, 3, 5, 1, 6 test case(x=1, y=2)
- 1, 2, 4, 5, 1, 6 test case(x=2, y=1)

Cntd.

Challenges in white box testing

- Sound knowledge of code and programming language needed.
- developers must be involved.
- No code can be fully tested to map real world scenario.

Example of test cases:

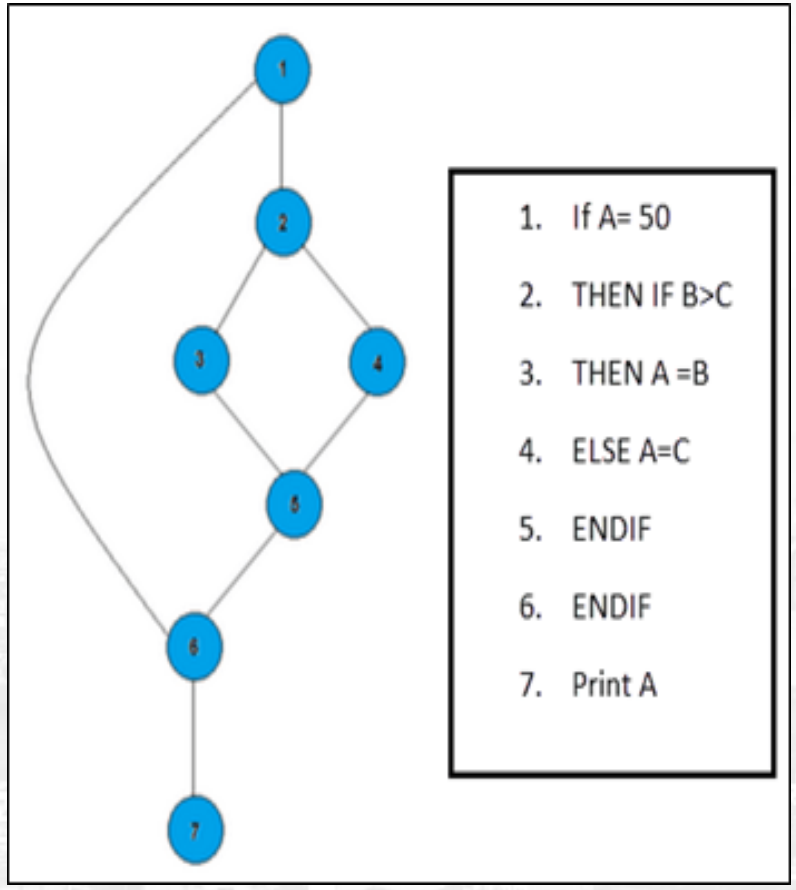
Login page

Basis Path Testing

- The basis path testing is based on a White Box Testing method, that defines test cases based on the flows or logical path that can be taken through the program.
- Basis path testing involves execution of all possible blocks in a program and achieves maximum path coverage with the least number of test cases.
- .The objective behind basis path in software testing is that it defines the number of independent paths, thus the number of test cases needed can be defined explicitly (maximizes the coverage of each test case).

Cntd.

simple example, to get a better idea what is basis path testing include



Cntd.

In the above example, we can see there are few conditional statements that is executed depending on what condition it 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

Steps for Basis Path testing

The basic steps involved in basis path testing include

- Draw a control graph (to determine different program paths)
- Calculate Cyclomatic complexity (metrics to determine the number of independent paths)
- Find a basis set of paths
- Generate test cases to exercise each path

Cntd.

Advantages of Basis Path Testing

- 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

Conclusion:

- Basis path testing helps to determine all faults lying within a piece of code.

Graph Based Testing

- It is one of black box testing method.
- Each and every application is build up of some objects.
- All such objects are identified and the graph is prepared.
- From this object graph, each object relationship is identified and test cases are written accordingly to discover the errors.

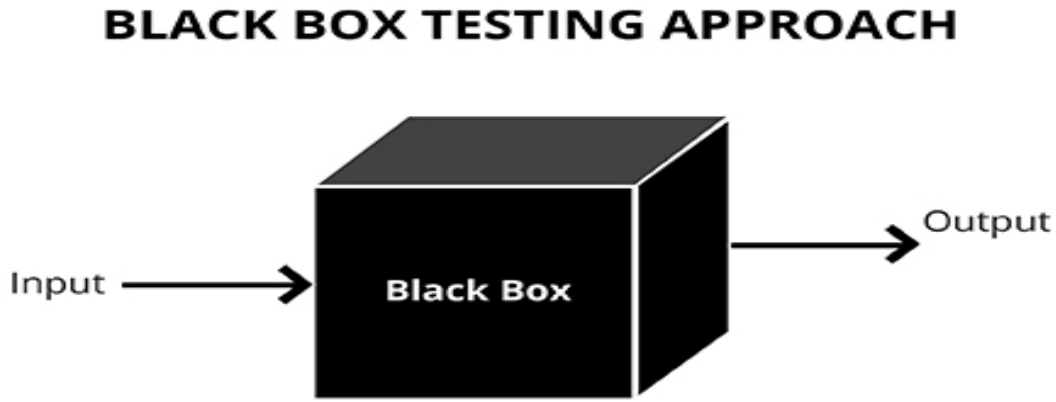
Test case:

- Test result: pass or fail

Test Id	Element Name	Test Case	EXPECTED OUTPUT (Theoretical Output)	ACTUAL OUTPUT (Practical Output)	Test Result

- **Black box testing**
  - No knowledge of internal design or code required.
  - Tests are based on requirements and functionality
- **White box testing**
- Knowledge of the internal program design and code required.
- Tests are based on coverage of code statements, branches,paths,conditions.

Black box testing



Black-box testing, also called **behavioral testing**, focuses on the functional requirements of the software.

That is, black-box testing techniques enable you to derive sets of input conditions that will fully exercise all functional requirements for a program.

Cntd.

- Tests are designed to answer the following questions:
- How is functional validity tested?
- How are system behavior and performance tested?
- What classes of input will make good test cases?
- Is the system particularly sensitive to certain input values?
- How are the boundaries of a data class isolated?

Cntd.

- What data rates and data volume can the system tolerate?
- What effect will specific combinations of data have on system operation
- Black-box testing is not an alternative to white-box techniques.
- Rather, it is a complementary approach that is likely to uncover a different class of errors than whitebox methods.

Cntd.

Black-box testing attempts to find errors in the following categories:

(1) incorrect or missing functions,

(2) interface errors

(3) errors in data structures or external database access,

(4) behavior or performance errors, and

(5) initialization and termination errors.

Cntd.

- Unlike white-box testing, which is performed early in the testing process, blackbox testing tends to be applied during later stages of testing

- Because black-box testing purposely disregards control structure, attention is focused on the information domain.

Cntd.

- Black box testing types

1. Requirements based testing

- Validating the requirements given in the SRS
- Requirements Review ensures:
- Consistent
- Correct
- Complete
- Testable
- Some Implicit requirements may be converted to Explicit Requirements.

Cntd.

2.Boundary value analysis

- Most defects occur at the boundaries
- Limits of the values of various variables.

Cntd.

3. Equivalence partitioning

- Identifies a small set of representative input values that produces many different output conditions as possible.

- Increases the coverage of testing and reduces the effort of testing.

Equivalence Class partitioning

- Divide the input space into equivalent classes
- If the software works for a test case from a class the it is likely to work for all
- Can reduce the set of test cases if such equivalent classes can be identified
- Getting ideal equivalent classes is impossible
- Approximate it by identifying classes for which different behavior is specified

Equivalence Class Examples

In a computer store, the computer item can have a quantity ?between -500 to +500. What are the equivalence classes

Answer: Valid class: -500 <= QTY <= +500
Invalid class: QTY > +500
Invalid class: QTY < -500

Equivalence Class Examples

Account code can be 500 to 1000 or 0 to 499 or 2000 (the field type is integer). What are the equivalence classes?

Answer:

Valid class: 0 <= account <= 499

Valid class: 500 <= account <= 1000

Valid class: 2000 <= account <= 2000

Invalid class: account < 0

Invalid class: 1000 < account < 2000

Invalid class: account > 2000

Equivalent class partitioning..

- Every condition specified as input is an equivalent class
- Define invalid equivalent classes also
- E.g. range 0< value<Max specified
  - one range is the valid class
  - input < 0 is an invalid class
  - input > max is an invalid class
- Whenever that entire range may not be treated uniformly - split into classes

Equivalence class…

- Once equivalence classes selected for each of the inputs, test cases have to be selected
  - Select each test case covering as many valid equivalence classes as possible
  - Or, have a test case that covers at most one valid class for each input
  - Plus a separate test case for each invalid class

Boundary value analysis

- Programs often fail on special values
- These values often lie on boundary of equivalence classes
- Test cases that have boundary values (BVs) have *high yield*
- These are also called *extreme cases*
- A BV test case is a set of input data that lies on the edge of an equivalence class of input/output

Boundary value analysis (cont)…

- For each equivalence class
  - choose values on the edges of the class
  - choose values just outside the edges
- E.g. if 0 <= x <= 1.0
  - 0.0 , 1.0 are edges inside
  - -0.1,1.1 are just outside
- E.g. a bounded list - have a null list , a maximum value list
- Consider outputs also and have test cases generate outputs on the boundary

Cntd.

4. Graph based / state based testing:

- when system can be represented as a data flow model
- Eg: employee hiring process
- Testing: tests each condition and state transition

Test cases

A test case is typically defined as a document that lays out the following:

- Test data
- Procedures/inputs
- Scenarios
- Descriptions
- Testing environment
- Expected results
- Actual results

Cntd.

Check Login Functionality there many

- Test Case 1: Check results on entering valid User Id & Password
- Test Case 2: Check results on entering Invalid User ID & Password
- Test Case 3: Check response when a User ID is Empty & Login Button is pressed, and many more possible test cases are:

Test Case #	Test Case Description	Test Data	Expected Result	Actual Result	Pass/Fail

Cntd.

Test Case ID #	Test Case Scenario	Test Step	Test Data	Expected Result	Actual Result	Pass/Fail
TU01	Check Customer Login with valid Data	Go to site Enter UserId Enter Password Click Submit	Userid = guru99 Password = pass99	User should Login into an application	As Expected	Pass
TU02	Check Customer Login with invalid Data	Go to site Enter UserId Enter Password Click Submit	Userid = guru99 Password = glass99	User should not Login into an application	As Expected	Pass

Test case format

Test Case ID		Test Case Description			
Created By		Reviewed By		Version	

QA Tester's Log

Tester's Name		Date Tested		Test Case (Pass/Fail/Not	

S #	Prerequisites:
1	
2	
3	
4	

S #	Test Data
1	
2	
3	
4	

Test Scenario Verify on entering valid userid and password, the customer can login

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended

Test Case format (example)

Test Case ID	BU_001	Test Case Description	Test the Login Functionality in Banking		
Created By	Mark	Reviewed By	Bill	Version	2.1
QA Tester's Log	Review comments from Bill incorprate in version 2.1				
Tester's Name	Mark	Date Tested	1-Jan-2017	Test Case (Pass/Fail/Not	Pass

S #	Prerequisites:
1	Access to Chrome Browser
2	
3	
4	

S #	Test Data
1	Userid = mg12345
2	Pass = df12@434c
3	
4	

Test Scenario: Verify on entering valid userid and password, the customer can login

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Navigate to http://demo.guru99.com	Site should open		Pass
2	Enter Userid & Password	Credential can be entered	As Expected	Pass
3	Click Submit	Cutomer is logged in	As Expected	Pass
4				

Test plan.

- A test plan is a detailed document that outlines the test strategy, Testing objectives, resources (manpower, software, hardware) required for testing, test schedule, Test Estimation and test deliverables.
- The test plan serves as a blueprint to conduct software testing activities as a defined process which is minutely monitored and controlled by the test manager.

Cntd.

Importance of Test Plan

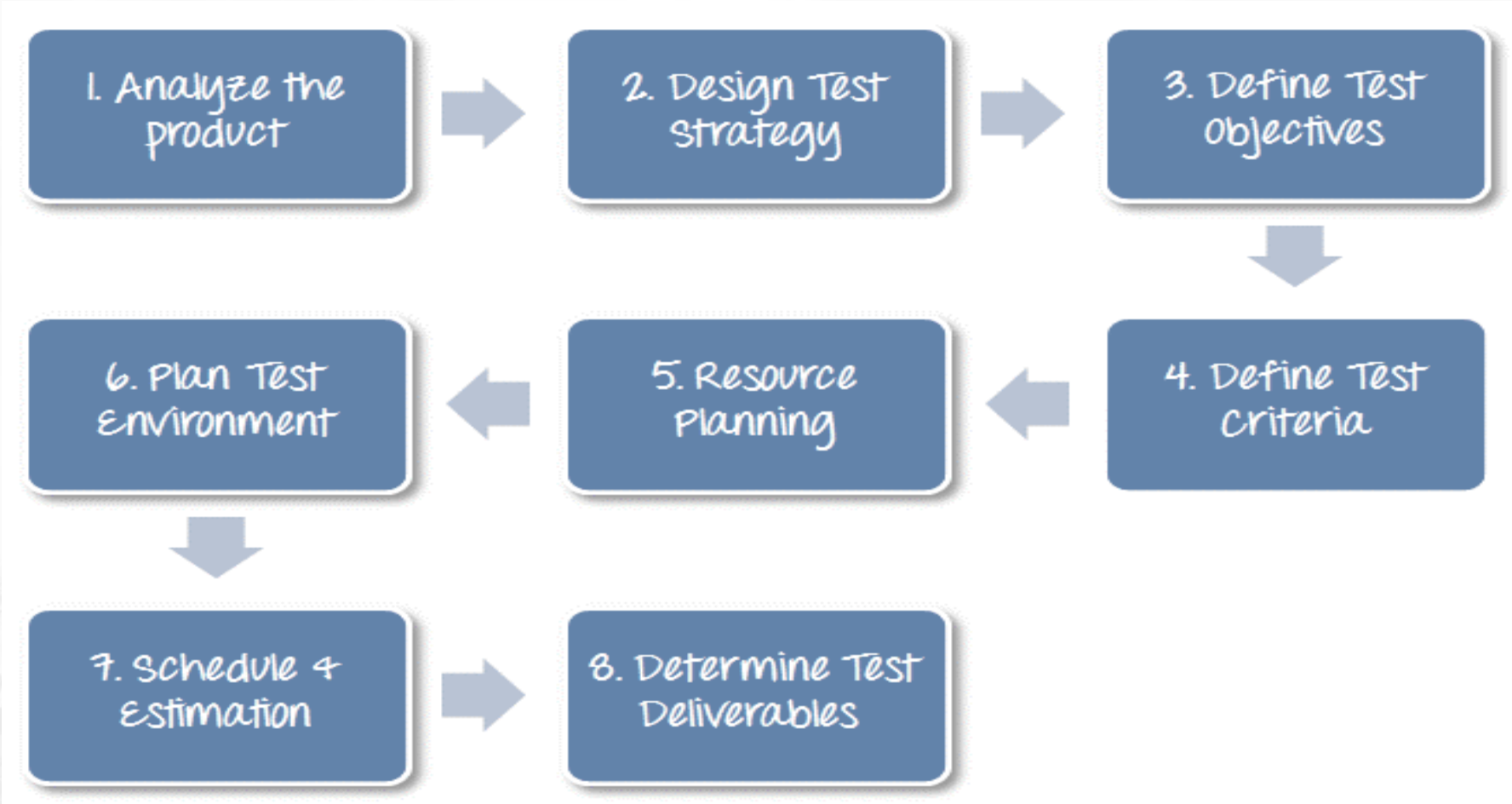
- Making Test Plan has multiple benefits
- Test Plan helps us determine the **effort** needed to validate the quality of the application under test
- Help people outside the test team such as developers, business managers, customers **understand** the details of testing.
- Test Plan **guides** our thinking.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

Cntd.

How to write a Test Plan

- You already know that making a **Test Plan** is the most important task of Test Management Process. Follow the seven steps below to create a test plan as per IEEE 829
- Analyze the product
- Design the Test Strategy
- Define the Test Objectives
- Define Test Criteria
- Resource Planning
- Plan Test Environment
- Schedule & Estimation
- Determine Test Deliverables

Cntd.


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