CATEGORIES/TYPES OF SOFTWARE

Categories of software are

1. System Software
2. Application Software
3. Programming Software
4. Custom Software

SYSTEM SOFTWARE

It provides the basic functions for computer usage and helps operating and managing computer operations. System Software is responsible for managing a variety of independent hardware components.

Examples are Operating Systems, Servers, Drivers etc.

APPLICATION SOFTWARE

The softwares that are used to aid in any task that benefits from computation. Application Softwares are generally available and are ready to use.

Examples are Image editing softwares, MS Office, Games, Web browsers, databases etc.

PROGRAMMING SOFTWARE

The softwares which helps in developing other softwares are called programming softwares.

Examples are compilers, debuggers, text editors etc.

CUSTOM SOFTWARE

Softwares which are developed based on clients requirements are called Custom softwares.

Examples are Customer Relationship Management (CRM), Hospital Management System, Banking applications etc.

Note: Application, software or system is used interchangebaly through out this tutorial.

WHAT IS TESTING ?

Investigating the system to check if it meets the requirements that it was designed/developed/built for. And also to check whether its working as expected.

One can define testing as,

Verification and **Validation** of predefined set of rules, conditions, specifications or attributes.

VERIFICATION

Verification is the process of confirming the requirements in order to develop a quality system. It is used to evaluate whether system complies with specifications, or conditions imposed at the start of a development phase.

The documents used in developing a system are verified against one another in order to investigate if any requirements are missing.

Verification is the process of Quality Control. System (under test) is not used in verification process.

VALIDATION

The process of checking the system to certify that it will meet the business requirements it was intended for.

Validation is the process of Quality Assurance. System (under test) is actually used in validation process.

Note: Verification can be expressed by the query **"Are we building the software right?"** and Validation by **"Are we building the right software?"**.

ENVIRONMENTS

Set of conditions in which one operates a computer is called environment. It comprises of hardware devices, operating system, configurations etc.

In Software development process, we have

1. Development Environment
2. Test Environment
3. Production Environment

DEVELOPMENT ENVIRONMENT

The environment in which a software coding is done. It includes the technology used for front end and back end, computer configurations, memory, disk space etc.

It also includes the software development model (will be introduced later) followed, number of programmers, the organization policies, or any kind of support needed in coding a software.

TEST ENVIRONMENT

The environment is one in which testing of a software is done. It comprises of circumstances or conditions influencing and effecting testing of a software.

It also includes the organization policies, culture, test process used, methods to improve testing etc.

PRODUCTION ENVIRONMENT

The environment in which software is actually used by the client. It comprises of Operating system, computer configurations, Browser(s) used by client, number of software users etc.

QUALITY STANDARDS

Organizations (any kind) are working hard to meet customer needs and expectations. Many management tools are available to help organizations achieve this goal.

Some of these are in the form of International Quality Standards. This standards can be used by organizations to deliver quality services or products to their clients.

Here we will see

1. ISO
2. CMMI
3. Six Sigma

ISO

International Organization for Standardization famously known as ISO is an International-standard setting body. The purpose of ISO is to promote the development of standardization in the world.

The emphasis is greater on customer focus and satisfaction. ISO standards help acheive quality that is recognized and respected throughout the world.

Following ISO standards helps improve performance activities, resource management, quality control process etc.

CMMI

Capability Maturity Model Integration (CMMI) is a process improvement model which helps organizations improve their performance.

There are five different maturity levels defined by CMMI

1. Level 1 - Initial
2. Level 2 - Managed
3. Level 3 - Defined
4. Level 4 - Quantitatively Managed
5. Level 5 - Optimizing

CMMI Level-1 : Process followed by organizations is undocumented, chaotic, uncontrolled or poorly controlled and environment is unstable.

CMMI Level-2 : Process is managed to some extent, it is undocumented, little or no effort is made to control or assure quality. The process is generally reactive but not to the extent of customer satisfaction.

CMMI Level-3 : Process is managed, efforts are made to control and assure quality, risks are managed, incidents are resolved and prevented. Emphasis is on customer satisfaction.

CMMI Level-4 : Process is measured, controlled and is documented. Quality is controlled and assured, incidents are prevented. Emphasis is on customer satisfaction.

CMMI Level-5 : Apart from all the level-4 qualities, the focus is on improving process.

SIX SIGMA

It is a business management strategy widely used in many sectors of industry. It helps to improve quality of process by identifying and removing the causes of defects.

A six sigma is a process in which 99.99966% of the products developed are free from defects.

SDLC

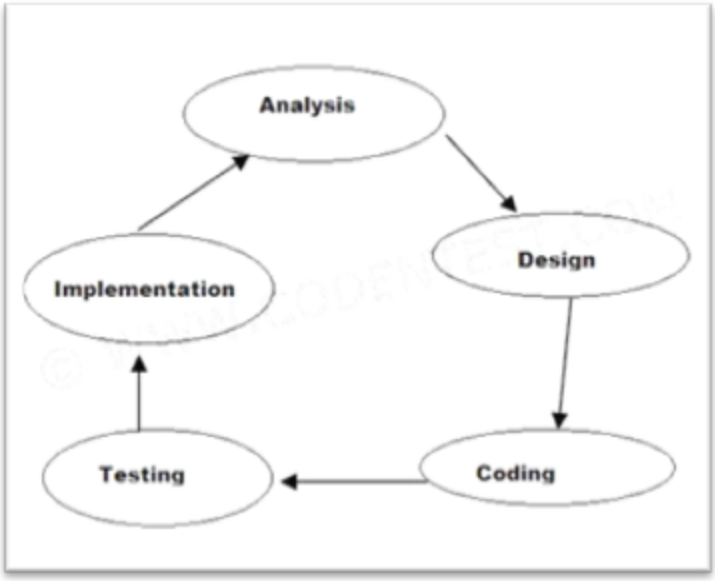
Software Development Life Cycle (SDLC) is a detailed plan to create, develop, test and eventually implement a software.

Software development is complex, and to manage this complexity a number of SDLC models or methodologies have been created.

Any SDLC methodology should result in a high quality software that meets or exceeds customer expectations, works effectively and efficiently, reaches completion within time and cost estimates.

SDLC methodologies help in project planning, software analysis, requirements, design, coding, testing, deployment and maintenance.

Below is a diagram of fundamental SDLC process.



Planning/Analysis : An important task in creating software is requirements analysis and establishing a high-level view of the project and determining its goals.

Design : Desired features and operations are designed in detail, including screen layouts, business rules, process diagrams, pseudocode.

Coding : Software code is written.

Testing : Checking for errors, bugs in the software.

Implemention : The software is put into production and runs actual business.

In the next sections we will learn about

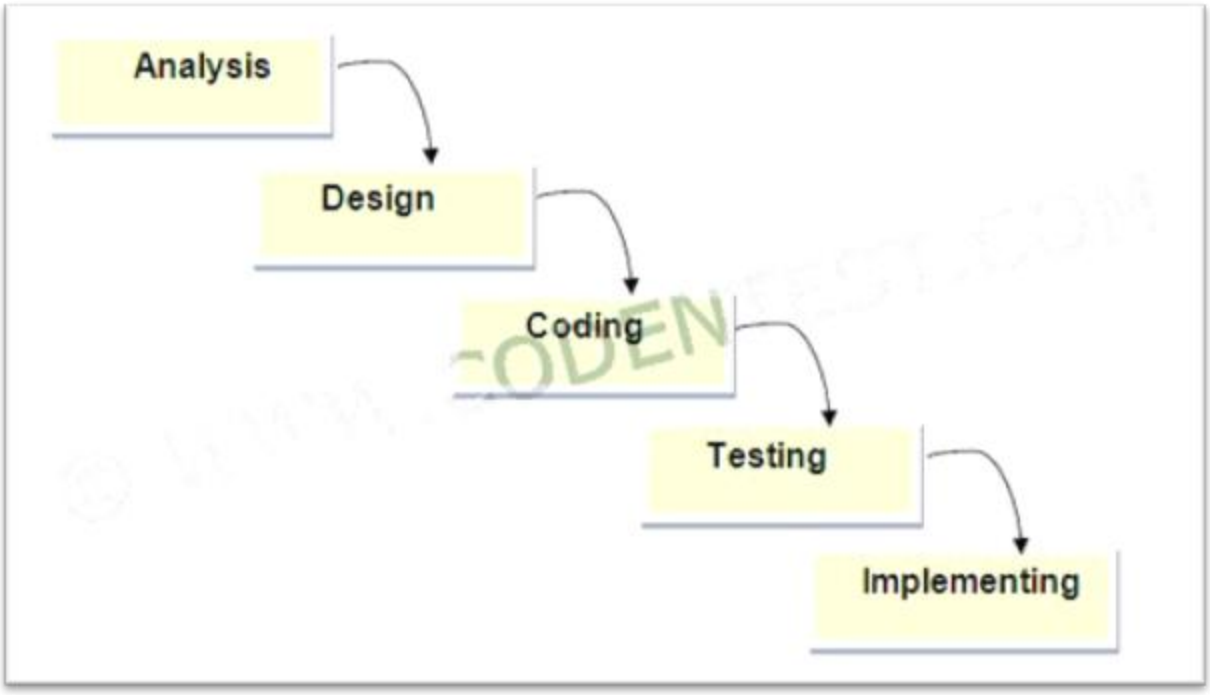
1. Waterfall Model
2. V Model
3. Scrum
4. Extreme Programming (XP)

WATERFALL MODEL

Waterfall Model is the idea of creating a software using sequential design process. This model maintains that one should move to a phase only when its previous phase is completed and perfected. There is no upward flow in pure waterfall model, just downwards, hence the name Waterfall.

The phases in a traditional waterfall model are

1. Analysis
2. Design
3. Coding
4. Testing
5. Implementation



Analysis : The phase is to analyse the system, gather requirements, and sometimes breaking down the system in different pieces to analyze the situation, analyzing project goals, breaking down what needs to be created and attempting to engage clients so that accurate requirements can be defined.

Normally there has to be a lot of communication between clients and software developing team to understand the requirements. Requirement gathering is the most crucial aspect in any kind of SDLC process. Many times communication gaps or misunderstanding in this phase leads to validation errors and bugs in the software.

When the system requirements are clear and perfect, a system requirement specification (SRS) document is prepared which has all the requirements agreed by client and the team. Only then one should move to the next phase in a pure waterfall model.

Design : After the requirement specifications are determined, problem solving and planning a software takes place.

Process is elaborated and software architecture (refers to the overall structure of the software) is divided into components called modules. Each module is again divided into several units (elements of a module). The User Interface screens are designed.

In this phase Functional Requirement Specifications (FRS) are defined. An FRS document includes all the functionalities of software components and the whole software system in detail. A functionality is described as a set of inputs, the behavior, and outputs.

Back end i.e. data structure is designed to store and organize data. Structuring data helps in managing huge amounts of data efficiently.

Coding : Some call this phase as Implementation, because the software designed is implemented in this phase. The actual programming code is written. The purpose of programming is to create a program that performs specific operations. Tables in database are created.

One should go to next phase, only when the coding for the entire software is completed.

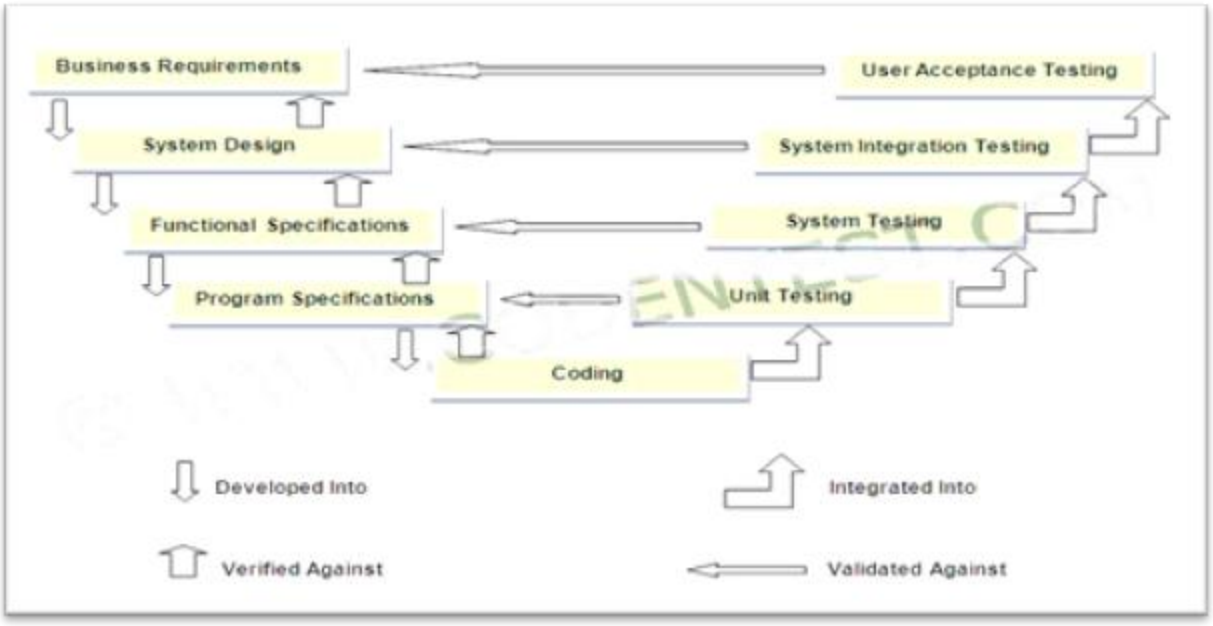
Testing : The software is tested at various levels in this phase to check for errors and also to check whether the software is meeting the requirements it was developed for.

Implemention : The software is deployed at the clients end. Software is maintained which is an important aspect of SDLC. In future, if there are any changes or improvements in a software it will be implemented by updating the software.

V MODEL

In this model, the process flow is bent downwards, upwards and sideways.

See the diagram given below, the left side phases (Business Requirements, System Design, Functional Specifications and Program Specifications) are called Verification phases. Where as the right side phases (Unit testing, Integration Testing, System Testing and User Acceptance Testing) are called Validation phases.



VERIFICATION PHASES

Business Requirements : In this phase, the business requirements or simply requirements of the software are collected by analyzing the clients need. This phase is concerned with establishing what the ideal system has to perform.

Clients are interviewed and a Business Requirements Specification (BRS) document is generated. This document will describe the system's functional, interface, performance, data, security, etc.

System Design : In this phase system engineers analyze and understand the business of the software by studying the requirements document created in business requirements phase.

System/Software Requirement Specifications (SRS) is prepared, which is a complete description of the behavior of a software to be developed. SRS is verified against BRS to check if it covers all the requirements mentioned in BRS.

Functional Specifications : In this phase the functionalities of the elements are specified, like what should software do when "Submit" button is clicked.

Functional Requirement Specification (FRS) is created which gives a precise idea of the element functionalities.

FRS is verified against SRS to check if it covers all the functionalities mentioned in the SRS. FRS helps coders in developing the software and testers in validating the software.

Program Specifications : In this phase the specifications are specified in detail, like what elements should be mandatory, what should be the inputs and outputs, database tables, with all elements, including their type and size etc.

This specifications are verified against FRS by the developers and after that actual coding starts.

CODING

The process of writing source code of the software. This source code is written in one or more programming languages. The purpose of programming is to create a program that performs specific operations or exhibits a certain desired behavior.

Coding is verified against the specified program specifications.

VALIDATION PHASES

Unit Testing : Unit testing is a method by which individual units of source code are tested to determine if they are fit for use. A unit is the smallest testable part of an application. In programming a unit may be an individual function.

Unit testing is usually done by developers or white box testers. Testing in this phase is validated against the specified program specifications (see the diagram above).

Note: Some organizations have Unit Integration Testing, after the Unit Testing phase and before System Testing phase. In this all the units tested in Unit Testing are integrated into a component and tested. Some call it component testing.

System Testing : This is the actual testing where all the functionalities specified in functional requirements are tested. System testing is performed on the entire system in the context of FRS and/or a SRS.

Both functional (like smoke, sanity, regression etc) and non-functional (like Load, Stress, Recovery, Performance etc) testing is done in this phase.

System Integration Testing : It is the process of checking the synchronization between two or more software systems.

As an example if we are providing a solution for a software consumer as enhancement to their existing solution, then we should integrate our application layer and our DB layer with consumer's existing application and existing DB layers. After the integration process completed both software systems should be synchronized.

User Acceptance Testing : acceptance testing is a test conducted to determine if the specified business requirements are met or not.

The objective is to provide confidence that the delivered system meets the business requirements of both sponsors and users. The acceptance phase may also act as the final quality gateway.

Note: The validation phases will be discussed in detail in the later sections.

SCRUM METHODOLOGY

Scrum model is an agile software development methodology. In this, a part of workable software is delivered to the customer on a regular basis.

A key principle of Scrum is its recognition that during a project the customers can change their minds about what they want and need, often called requirements churn.

Scrum enables the creation of self-organizing teams by encouraging co-location of all team members, and verbal communication between all team members and disciplines in the project.

In scrum model, we have

- Roles
- Sprint
- Meetings
- Artifacts

ROLES

The main roles in Scrum are

- Product Owner
- Scrum Master
- Team

Product Owner represents the voice of the customer and is accountable for ensuring that the Team delivers value to the business. Product owner represents the stakeholders and the business.

The Product Owner writes what the user wants to achieve, prioritizes them, and adds them to the product backlog.

Scrum Master is one who heads the scrum. He is accountable for removing impediments to the ability of the team to deliver the sprint goal/deliverables. The Scrum Master is not the team leader but acts as a buffer between the team and any distracting influences. The ScrumMaster ensures that the Scrum process is used as intended.

A key part of the ScrumMaster's role is to protect the team and keep them focused on the tasks at hand.

Team is responsible for delivering the software. It consists of people with cross-functional skills who do the actual work like analyse, design, develop, test, technical communication, document, etc.

It is recommended that the Team be self-organizing and self-led, but often work with some form of project or team management.

SPRINT

A sprint is the basic unit of development in Scrum. Sprints tend to last between one week and one month, and are restricted to a specific duration.

Each sprint is preceded by a planning meeting, where the tasks for the sprint are identified and an estimated commitment for the sprint goal is made, and followed by a review meeting where the progress is reviewed and activities for the next sprint are identified.

During each sprint, the team creates a potentially deliverable product increment. The set of features that go into a sprint come from the product "backlog", which is a prioritized set of high level requirements of work to be done. Which backlog items go into the sprint is determined during the sprint planning meeting.

The team then determines how much of this they can commit to complete during the next sprint, and records this in the sprint backlog. During a sprint, no one is allowed to change the sprint backlog, which means that the requirements are frozen for that sprint.

MEETINGS

Sprint Planning Meeting is held at the beginning of the sprint cycle to

- Select what work is to be done
- Prepare the Sprint Backlog
- Identify and communicate how much of the work is likely to be done during the current sprint.

At the end of a sprint cycle, two meetings are held: the "Sprint Review Meeting" and the "Sprint Retrospective".

Sprint Review Meeting , is held to

- Review the work that was completed and not completed
- Present the completed work to the stakeholders
- Incomplete work cannot be demonstrated

Sprint Retrospective , is for

- All team members reflect on the past sprint
- Make continuous process improvements
- Two main questions are asked in the sprint retrospective: What went well during the sprint? What could be improved in the next sprint?

ARTIFACTS

Product backlog is a high-level list that is maintained throughout the entire project. It aggregates backlog items: broad descriptions of all potential features, prioritized as an absolute ordering by business value. It is therefore the "What" that will be built, sorted by importance.

Sprint backlog is the list of work the team must address during the next sprint. Features are broken down into tasks. With this level of detail the whole team understands exactly what to do.

The sprint backlog is the property of the team, and all included estimates are provided by the Team.

Burn down is a chart which shows remaining work in the sprint backlog. It is updated every day, it gives a simple view of the sprint progress. It also provides quick visualizations for reference. There are also other types of burndown, for example the release burndown chart that shows the amount of work left to complete.

EXTREME PROGRAMMING

Extreme programming (XP) is an agile software development method which is intended to improve software quality and responsiveness to changing customer requirements.

Like any other agile methodology, XP advocates frequent releases in short development cycles, which is intended to improve productivity and introduce checkpoints where new customer requirements can be adopted.

The concept of this methodology is to focus on

- Goals
- Activities
- Values
- Principles

GOALS

The goal of extreme programming is to produce higher quality software more productively. XP attempts to reduce the cost of changes in requirements by having multiple short development cycles, rather than one long one.

Extreme Programming empowers developers to confidently respond to changing customer requirements, even late in the life cycle.

Extreme programming also introduces a number of basic values, principles and practices on top of the agile programming process.

ACTIVITIES

XP describes four basic activities that are performed within the software development process: coding, testing, listening, and designing.

Coding : The advocates of XP argue that coding is the meat of XP methodology. Coding is used to figure out the most suitable solution, to communicate thoughts about programming problems.

Testing : Extreme programming approach is that if a little testing can eliminate a few flaws, a lot of testing can eliminate many more flaws.

Unit tests determine whether a given feature works as intended. Testing is done to break the code, if all tests run successfully, then the coding is complete. Every piece of code that is written is tested before moving on to the next feature.

Acceptance tests verify that the requirements as understood by the programmers satisfy the customer's actual requirements.

Listening : Programmers must listen to what the customers need the system to do, what "business logic" is needed. They must understand these needs well enough to give the customer feedback about the technical aspects of how the problem might be solved, or cannot be solved.

Designing : Normally software development becomes too complex and the dependencies within the system cease to be clear. One can avoid this by creating a design structure that organizes the logic in the system. Good design will avoid lots of dependencies within a system; this means that changing one part of the system will not affect other parts of the system.

VALUES

Values in extreme programming are Communication, Simplicity, Feedback, Courage and Respect.

Communication : Building software systems requires communicating system requirements to the developers of the system. The goal is to give all developers a shared view of the system which matches the view held by the users of the system.

Extreme programming favors simple designs, collaboration of users and programmers, frequent verbal communication, and feedback.

Simplicity : Extreme programming encourages starting with the simplest solution. Extra functionality can then be added later. The difference between this approach and more conventional system development methods is the focus on designing and coding for the needs of today instead of those of tomorrow.

Simplicity in design and coding should improve the quality of communication. A simple design with very simple code could be easily understood by most programmers in the team.

Feedback : Within extreme programming, feedback relates to different dimensions of the system development

- Feedback from the customer
- Feedback from the team
- Feedback from the system

Feedback is closely related to communication and simplicity.

Courage : This is an effort to avoid getting bogged down in design and requiring a lot of effort to implement anything else. Courage enables developers to feel comfortable with refactoring their code when necessary. This means reviewing the existing system and modifying it so that future changes can be implemented more easily.

Another example of courage is knowing when to throw code away, to remove source code that is obsolete, no matter how much effort was used to create that source code.

Respect : The respect value includes respect for others as well as self-respect. Members respect their own work by always striving for high quality and seeking for the best design for the solution at hand.

PRINCIPLES

The principles are intended to be more concrete than the values. The principles in XP are Feedback, Assuming Simplicity and Embracing Change.

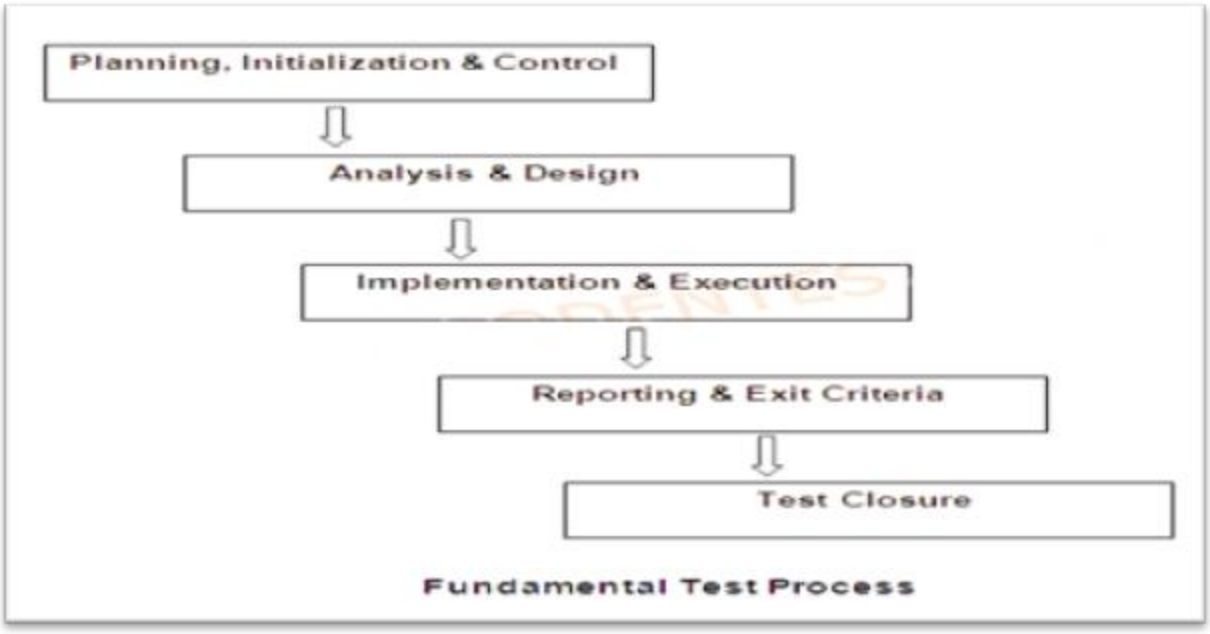
Feedback : Extreme programming sees feedback as most useful if it is done rapidly and expresses that the time between an action and its feedback is critical to learning and making changes.

Assuming simplicity : This is about treating every problem as if its solution were extremely simple. Traditional system development methods say to plan for the future and to code for reusability. Extreme programming rejects these ideas. Instead applies incremental changes thus making things simple.

Embracing change : The principle of embracing change is about not working against changes but embracing them. For instance, if at one of the iterative meetings it appears that the customer's requirements have changed dramatically, programmers are to embrace this and plan the new requirements for the next iteration.

TEST PROCESS

The process which defines procedures and activities required to successfully complete a testing project is Test Process.



PLANNING, INITIALIZATION & CONTROL

Initialization in testing is the process of starting the testing project. A high level Project Plan or Project Strategy is created. Since many activities will be carried out during testing, plans and strategies are needed.

Test planning is a detailed plan to test a system. It is the activity of verifying the mission, defining the objectives of the testing in order to successfully complete a testing project.

Test control is an ongoing activity in which actual progress is compared against the plan. It involves taking actions necessary to meet the mission and objectives of testing project.

ANALYSIS & DESIGN

Test Analysis is identifying and prioritization of Test Scenarios, Test Cases and Test Scripts.

Test Environment is setup with the required infrastructure. Data requirements are also identified in this phase.

Designing is preparing Test Scenarios, Test Cases and Test Scripts.

IMPLEMENTATION & EXECUTION

Implementation & Execution is a phase in which all the designed and analyzed Test Scenarios, Cases and Scripts are implemented and executed as per schedule.

Test execution can be automatic or manual. The outcomes of the tests are analysed by comparing the actual result and expected result.

REPORTING & EXIT CRITERIA

Defects detected are tracked till they are closed. Some organizations report defects in the execution phase, which is ok.

Preparing Test Summary report for the stakeholders.

Exit Criteria is the activity where test execution is assessed against the defined objectives.

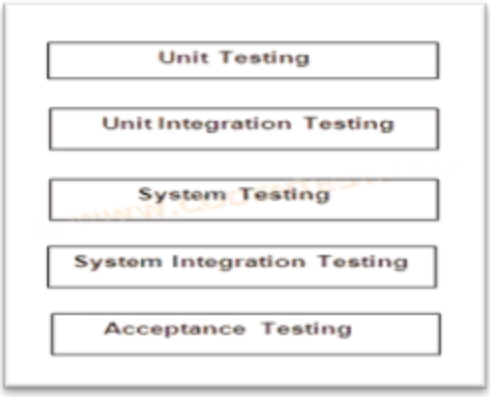
TEST CLOSURE

In **Test Closure** phase, information is collected from the completed test activities. For example, when testing was started, when completed, when a software was released.

Apart from this, Test Closure also specifies what milestones were achieved, problems faced and how the problems were tackled etc to implement the test process better in future projects.

TEST PHASES

Different types of tests a software undergoes during testing are called Test Phases. It can also be defined as different stages of software testing.



The above diagram gives an overview of the testing phases, which are again divided into different testing types which we will be learning throughout this tutorials.

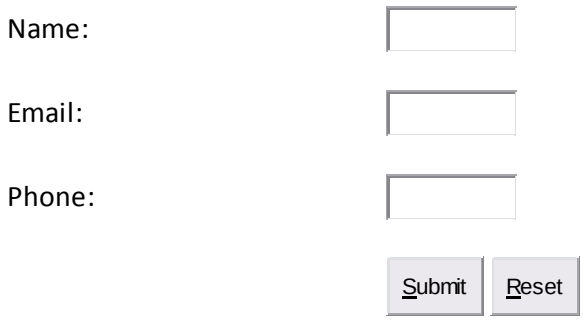
UNIT TESTING

Unit is smallest testable part of software. In Unit testing the source code of a unit is tested to check its behavior. It finds defects early in the testing stage. Normally large percentage of defects are identified in unit testing stage.

Unit testing can performed manually or automatically. It also verifies that the code is efficient as per the adopted coding standards.

In the below given example form, each unit is tested seperately. Name, Email, Phone, Submit button and Reset button are tested in unit testing.

EXAMPLE



COMPONENT TESTING

After unit testing two or more units that have already been tested are combined into a component and tested.

In component testing, programs or subprograms are tested prior to integrating them into larger components.

The above example form can be termed as a component.

After each component is tested, two or more components are integrated to form a larger component and the interface is tested.

As stated before, two or more components are combined to form a larger component and tested. What if all the components are not developed, but testing has to be done anyway.

In such situation **Stubs** and **Drivers** are introduced.

STUBS AND DRIVERS

Stub is the pseudo code written for a called component.

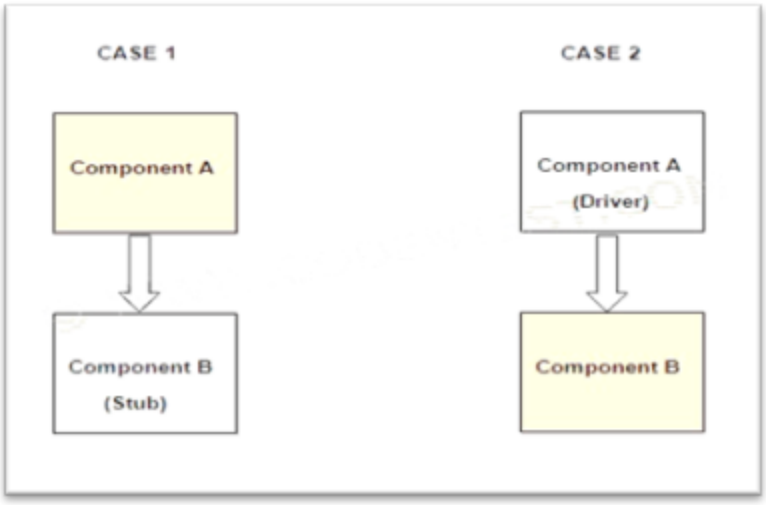
Let us assume that there are two components, component A and component B. There are few functionalities in A that depend on B, in order to work properly. A is developed and B is not but testing has to be carried out anyway.

In order to support component A, pseudo code is written for component B which is called **Stud**.

Driver is the pseudo code written for a calling component.

Now lets assume that component B is developed but component A is not, and testing has to be done. In this situation a pseudo code for coponent A is written in order to support component B, which is called **Driver**.

The diagram given below will give you clear idea about stubs and drivers.



SYSTEM TESTING

All the integrated components which have passed testing are put together and tested, which is called System Testing.

It is conducted on a complete, integrated system to to check whether the system is meeting the specified requirements. Inner code knowledge is not required in System testing.

The entire system is validated against Functional Requirement Specifications (FRS) and System Requirement Specifications (SRS).

System testing includes (but not limited to) the following types of testing.

- Smoke Testing
- Sanity Testing
- Retesting Testing
- Regression Testing
- Compatibility Testing
- Performance Testing
- Load Testing

- Stress Testing
- Volume Testing
- Usability Testing
- Security Testing

In this section, we will discuss about Smoke testing, Sanity testing, Retesting and Regression testing. Others are discussed in Non-functional testing section.

SMOKE TESTING

Smoke testing is done when a software is given for testing, to check whether thorough testing can be carried out or not. It is preliminary to further testing.

Most important functionalities are tested, not bothering about the less crucial functionalities or finer details.

If a software fails smoke testing, it is sent back for fixing.

SANITY TESTING

Sanity testing is a very brief run-through of the functionality of a software, to assure that part of the software works as expected. This is often prior to a more exhaustive round of testing.

Sanity testing may be a tool used while manually debugging software. When the overall software is not working as expected, a sanity test can be used to check the working parts of software, until the problem with non-working parts software is solved.

RETESTING

When a functionality fails to work as expected, we say defect has occured, and it is sent for fixing. The fixed functionality is tested again to ensure that the defect is fixed, this is called **Retesting**.

REGRESSION TESTING

Regression testing is the process in which previously tested and passed functionalities are tested again to ensure that a change, such as a defect fixing, did not introduce new defect.

There is a high possibility that when some functionality is fixed, the others might not work as expected. So a thorough regression testing is done after fixing the defects.

FUNCTIONAL TESTING

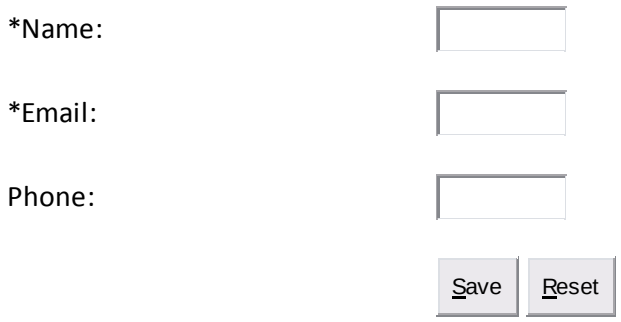
Functions are tested by giving different inputs in different conditions and the output is examined, which is compared with the expected result.

Most people think Functional testing and System testing is same. But they differ slightly in that functional testing verifies a software by checking it against designed specification documents while system testing validates a software by checking it against the user requirements.

Functional testing involves

1. Identifying how a functionality is expected to behave
2. Creating input data to check the functionality based on the functional requirements.
3. The execution of the test cases prepared for that function.
4. The comparison of actual and expected outputs

EXAMPLE



Lets assume we are testing the functionality of Save button.

Functionality behavior : When valid data is entered in all the mandatory fields and Save button is clicked, the data entered should be saved in the database table (otherwise not).

Creating input data : Both valid and invalid data is created.

Valid data for

- Name is alphabets with spaces (ex: Sam Washington)
- Email is valid email format (ex: washington@something.com)
- Phone is digits (ex: 23456789)

Data apart from the mentioned one is invalid.

Example, for

- Name digits or special characters (ex: Sam123_4u)
- for Email data other than email format (ex: sam.com)

- for Phone data other than digits (ex: 65^&%34tr)

Execution of test cases : Test cases prepared to test Save button are executed. Below are four test cases, two positive and two negative.

Test save button by

1. Entering valid data in mandatory fields only.
2. Entering valid data in mandatory fields and invalid in non-mandatory fields.
3. Entering invalid data in mandatory fields.
4. Not entering any data in the fields.

Comparision of actual and Expected results : Expected result for first two test cases is that the data should be saved in the database table. Expected result for the last two test cases is that the data should not be saved.

When actual result does not match expected result, a defect is raised.

NON-FUNCTIONAL TESTING

In functionality testing we tested the functionalities in a software. In non-functional testing, we test the non-functionalities of a software, like the security, compatibility, reliability of software etc.

Types of Non-functional testing are

- Performance Testing
- Load Testing
- Stress Testing
- Volume Testing
- Compatibility Testing
- Security Testing
- Usability Testing
- Sociability Testing
- Installation Testing
- Recovery Testing
- Compliance Testing

PERFORMANCE TESTING

Performance testing is the testing performed to determine how fast some aspects of a software perform under a particular workload.

Performance testing demonstrates that the software meets performance criteria. It can compare two softwares of same type, to find which performs better. It also measures what parts of software are performing badly under certain workload.

Performance testing is usually automated to create a production like environment. Load and Stress testing are a part of Performance testing.

LOAD TESTING

A load test is done to understand the behavior of the software under some load conditions. This load can be the number of users on the software performing a specific number of transactions within the set duration.

Example, how will a software perform if fifty users are using it simultenously.

This test will give out the response times of all the important business transactions. Load testing is also done to find the capacity of the server, on which software resides.

STRESS TESTING

Stress testing is done to determine that software performs effectively when the load is beyond expected load.

Robustness of a software is determined in terms of extreme load which helps system administrators to determine if the software will perform effectively if the current load goes well above the expected load.

Lets assume that a software is developed for hundred users of an organization. Testing the software with upto 100 users is load testing, testing it with well over 100 users is stress testing.

Software should maintain certain level of effectiveness under unfavorable conditions.

VOLUME TESTING

Volume testing is testing a software with a certain amount of data. This amount can, be the database size or it could be the size of the file which interacts with software.

For example, if you want your software database to be of specific size, you will expand your database to that size and then test the software performance on it.

Lets assume a software has an option of uploading a file of not more then 10 MB size, format of the file could be anything. We will create a sample file of the size we want and then upload it to test.

COMPATIBILITY TESTING

Compatibility testing is done to evaluate the software compatibility with the environment, like hardware configurations, bandwidth, operating systems, databases, browsers etc.

Web based applications are tested on different web browsers, to ensure that the users have same visual experience irrespective of the browsers. In terms of functionality, the application must behave and respond the same way across different browsers.

SECURITY TESTING

Security testing is a process to determine that an information system protects data and maintains functionality against internal/external willful damage.

The things we should consider in security testing are confidentiality, integrity, authentication, availability, authorization and non-repudiation.

Confidentiality is a measure which protects against the disclosure of information to parties other than the intended recipient.

Integrity is to provide the receiver correct information.

Authentication is confirming the identity of a user. Example, only the valid users can use the software.

Authorization The process of determining that a requester is allowed to receive a service or perform an operation. Example, only the authorized screens should be displayed to the user.

Availability is insuring that the information is available to authentic or authorized users when they need it.

Non-repudiation is to ensure that a message has been sent and received by the parties claiming to have sent and received the message. Nonrepudiation is a way to guarantee that the sender of a message cannot later deny having sent the message and that the recipient cannot deny having received the message.

USABILITY TESTING

Usability testing in software is how a user users a software, like how many steps are involved to make certain transaction, what could be the user experience.

Usability testing measures the easiness or complexity of a software, i.e. how easy or how complex a software is for the user. In short it determines the user friendliness of a software.

SOCIABILITY TESTING

Sociablility testing is done to ensure that the software does not conflict with other softwares on the system. Software should not

- corrupt other software files
- consume other software resources
- hang-up or crash the other softwares etc

INSTALLATION TESTING

Installation testing is to test the installation of full, part of a sofware. It also tests the upgrading or uninstalling of a software.

RECOVERY TESTING

Recovery testing is the activity of testing how well an application is able to recover from crashes, hardware failures and other similar problems.

COMPLIANCE TESTING

Testing a software whether is meets the internationally recognized standards of a software. Products tested in such a manner are then advertised as being certified by that external organization as complying with the standards.

ACCEPTANCE TESTING

Acceptance testing is conducted to determine if the business requirements that are agreed upon are met. It is done prior to delivering software.

Acceptance testing is usually performed by subject experts, managers, end users etc.

It is performed on a completed software. The test environment should be identical, or as close as possible, to the user environment in which software will be used.

Acceptance testing done by user is termed as User Acceptance Testing (UAT). It is one of the final stages of a project and often occurs before a client or customer accepts the software.

Acceptance testing is divided into

- Alpha Testing
- Beta Testing

ALPHA TESTING

Acceptance testing done in the organization before the release of the software is called Alpha Testing. It is usually done by subject experts, stake holders, managers etc.

Some changes may still be made in a software as a result of Alpha testing.

BETA TESTING

Beta testing takes place at customers place, and involves testing by a group of customers who use the software.

Feedback is given after customers test the software. If the feedback is positive, the software is released. Otherwise changes are made before it is released.

DATA TYPES

Data is information that had been translated into a form that is understandable. In software testing, data is used to test the functionalities of a software.

Data in testing is of two types

- Test Data
- Production Data

TEST DATA

The data used to test a software is test data. Typically test data is used in Unit testing, Component testing, Functional and System testing.

PRODUCTION DATA

Production data is the data used by the customer when the software is in actual use.

EXAMPLE

Name:

Submit Reset

In the above example form, lets assume that Name must accept only alphabets and spaces.

Positive test data used to check Name can be

- indiana jones
- james bond
- amtyu chre etc

Negative test data used to check Name can be

- indiana743jones
- james@#$ bond
- 71234$ etc

Lets say the above example form is delivered to customer, who employs Sam Washington, Alice Smart and Jack black.

So the production data is

Sam Washington

Alice Smart

Jack Black

Note: Combination of production and test data is used in Acceptance testing. Normally in Alpha testing combination of test and production data is used, where as in Beta testing production data is used.

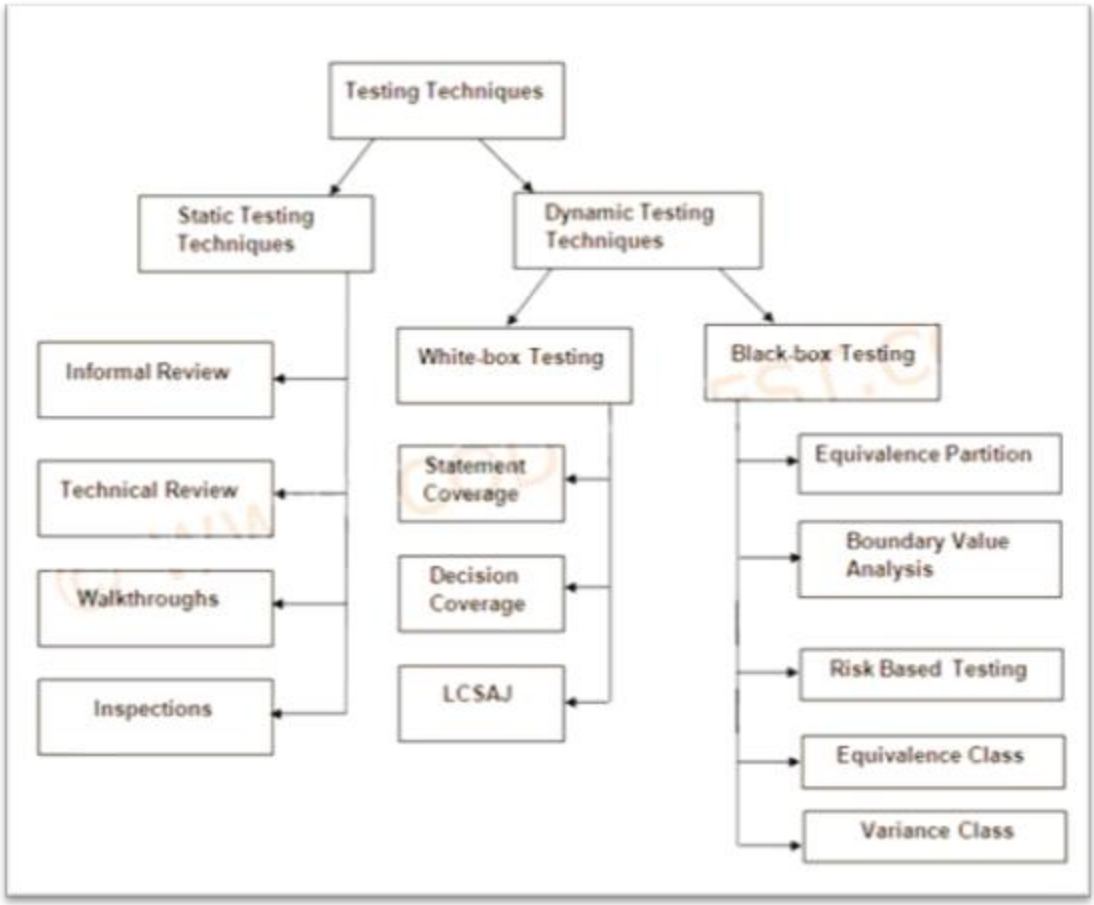
TESTING TECHNIQUES

The techniques used to easily find defects and successfully complete testing are called Testing techniques.

Testing techniques helps us in

- Measuring test efforts
- Testing effectively to find more defects
- Testing efficiently to find defects easily
- Deriving good test cases

Testing techniques are divided into **Static Testing Techniques** and**Dynamic Testing Techniques**, which are further divided. The diagram given below shows different testing techniques, that are normally used.



STATIC TESTING TECHNIQUES

Verification portion, of Verification and Validation, falls in Static testing techniques. In this software is not actually used. This technique is used to check the code, algorithms and documentations.

Defects discovered at this stage are less expensive to fix than later in the test cycle.

Static testing techniques are divided into

- Informal Reviews
- Technical Reviews
- Walkthroughs
- Inspections

INFORMAL REVIEW

No formal process is followed. **Pair Programming** is done in which two developers write code together. Technical lead will review the code designs.

Informal reviews sometimes are documented. Main purpose of informal review is to find errors in coding, improving both quality of software and the developers skills.

TECHNICAL REVIEW

Reviewing of code is done with managements praticipation. Results are documented, peer reviewing is done. Meetings are normally led by trained moderators.

WALKTHROUGHS

In walkthroughs the lead members of the team walks you through a software, and the participants, usually developers and testers, ask questions and make comments about possible errors.

Some organizations document walkthroughs, which is a good practice, and some don't.

INSPECTIONS

Inspections are auditing sessions which are led by trained individuals who look for defects using a well defined process.

An inspection is one of the most common sorts of review practices found in software projects. Normally Software Specifications and Project Plans are inspected. The goal of the inspection is to identify defects.

Note: Static Techniques are part of Quality Control activities in which the actual progress is compared against the plan and status is reported, including deviations from the plan.

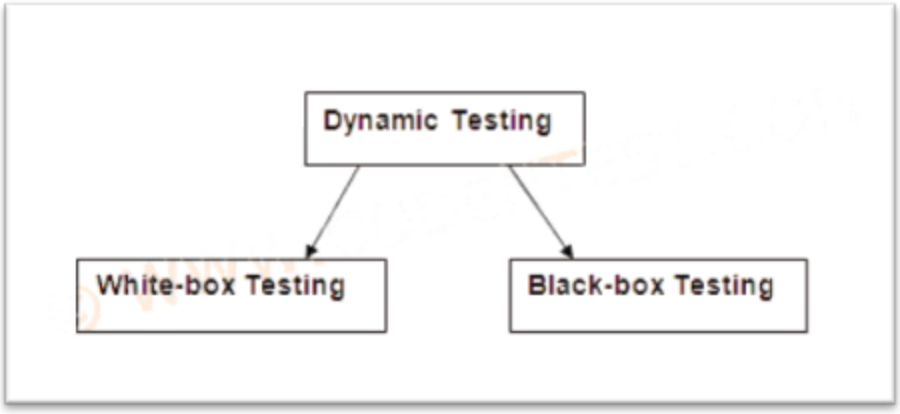
DYNAMIC TESTING TECHNIQUES

In Dynamic testing software is used, inputs are given and outputs are checked with expected results. The Validation portion, of Varification and Validation, falls in this category of testing.

Dynamic testing involves testing software based on prepared test cases. Unit Testing, Integration Testing, System Testing and Acceptance Testing are part of Dynamic testing techniques.

Dynamic testing is divided into

- White-box testing
- Black-box testing



WHITE-BOX TESTING

White box testing is a method of testing software that tests internal structures of an application. It is usually done at the unit level.

White box testingis also called as **Structural** or **Glass-box testing**.

White-box testing techniques are

- Statement Testing and Coverage
- Decision Testing and Coverage
- Linear Code Sequence And Jump (LCSAJ)

STATEMENT TESTING & COVERAGE

Statement Coverage is defined as the percentage of executable statements that has been exercised by a test suite.

The objective if the statement testing is to show that the executable statements within a program have been executed at least once. When all the statements have been executed then its 100% statement coverage. But that if we have decision statements? is it possible to cover all the statements?

Consider the following example code

```
1. x=input;
2. y=20;
3. if(x > y) {
4. print "x is greater then y"; }
5. else {
6. print "x is less then y"; }
7. print "end";
```

If we test the above code with **x=25**, the control flow will be

- line: 3, 4 and 7

If we test the above code with **x=15**, the control flow will be

- line: 3, 6 and 7

The flow does not cover all the statements.

DECISION TESTING & COVERAGE

The percentage of the decision outcomes that have been exercised by a test suite is called decision coverage.

In decision statement we cover all the possible conditions. In the above example, with two test cases we can achieve two possible conditions and hence achieve 100% coverage.

Note: 100% decision coverage guarantees 100% statement coverage, but not vice versa.

LINEAR CODE SEQUENCE & JUMP

Linear code sequence and jump (LCSAJ) is a software analysis method used to identify structural units in code under test. Dynamic software analysis is used to measure the quality and efficacy of software test data, where the quantification is performed in terms of structural units of the code under test.

When used to quantify the structural units exercised by a given set of test data, dynamic analysis is also referred to as coverage analysis.

An LCSAJ is a software code path fragment consisting of a sequence of code (a linear code sequence, or basic block) followed by a control flow Jump, and consists of the following three items:

- the start of the linear sequence (basic block) of executable statements
- the end of the linear sequence
- the target line to which control flow is transferred at the end of the linear sequence.

BLACK-BOX TESTING

Testing the functionality of software without the knowledge of internal code structures. In this testing is done depending on the specifications and requirements of a software, like what a software is supposed to do.

Both valid and invalid data is used to test the functionalities. Mostly black box testing is done at higher levels, like component testing, system testing, UAT etc but can also be done at unit levels.

Black box testing techniques are used to prepare test cases which can easily find defects in the functionalities of a software. It is also called as **Functional Technique**.

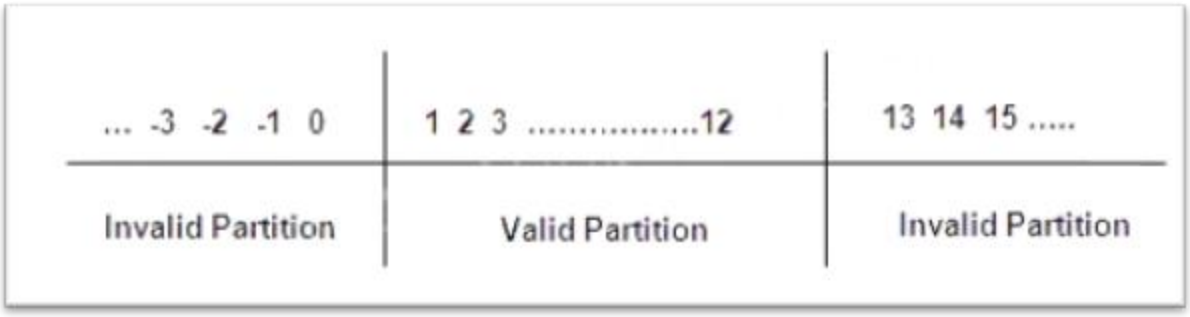
Following are the Black-box testing techniques

- Equivalence Partition
- Boundary Value Analysis (BVA)
- Risk based Testing
- Equivalence Class
- Variance Class

EQUIVALENCE PARTITION

Equivalence partition is a technique to divide the data into valid and invalid partitions.

Lets assume that there is a parameter "Month" in a software. The valid values for the Month are from 1 to 12, which is called valid partition. The invalid partitions are values which are <=0 and >12.



Depending on the valid and invalid partitions the following test cases for "Month" can be prepared.

1. Test Month with 5
2. Test Month with -3
3. Test Month with 15

The first one is a positive case and the next two are negative cases. The testing theory related to equivalence partitioning says that only one test case of each partition is needed.

BOUNDARY VALUE ANALYSIS (BVA)

In this technique, boundary values are entered as inputs. Values on the minimum and maximum edges of an equivalence partition are tested, since boundaries are common locations for errors.

Test cases should be created to generate inputs that will fall on either side of each boundary and on boundary, which results in three cases per boundary.

In the example mentioned above, the boundary values for "Month" are 0,1,2 and 11,12,13. So, the following test cases for "Month" can be prepared.

1. Test Month with 0
2. Test Month with 1
3. Test Month with 2
4. Test Month with 11
5. Test Month with 12
6. Test Month with 13

Another example lets assume there is a parameter called "Username" and number of characters it should accept are between 6 to 18. Boundary values are 5,6,7 and 17,18,19.

Test "Username" with

1. 5 characters
2. 6 characters
3. 7 characters
4. 17 characters
5. 18 characters
6. 19 characters

RISK BASED TESTING

Testing the functionalities which are more likely to be problematic is risk based testing.

For example, lets assume that user is booking a flight ticket, and has to enter date of his journey manually. The application should not allow the user to enter past dates.

EQUIVALENCE CLASS

In Equivalence Class testing, parameters with common requirements are grouped and test cases for them are prepared.

Lets assume we have two screens and "Email" parameter is present in both of them. Now we need not write test cases for "Email" both the times, test cases written for "Email" in one form can be used for "Email" in another form.

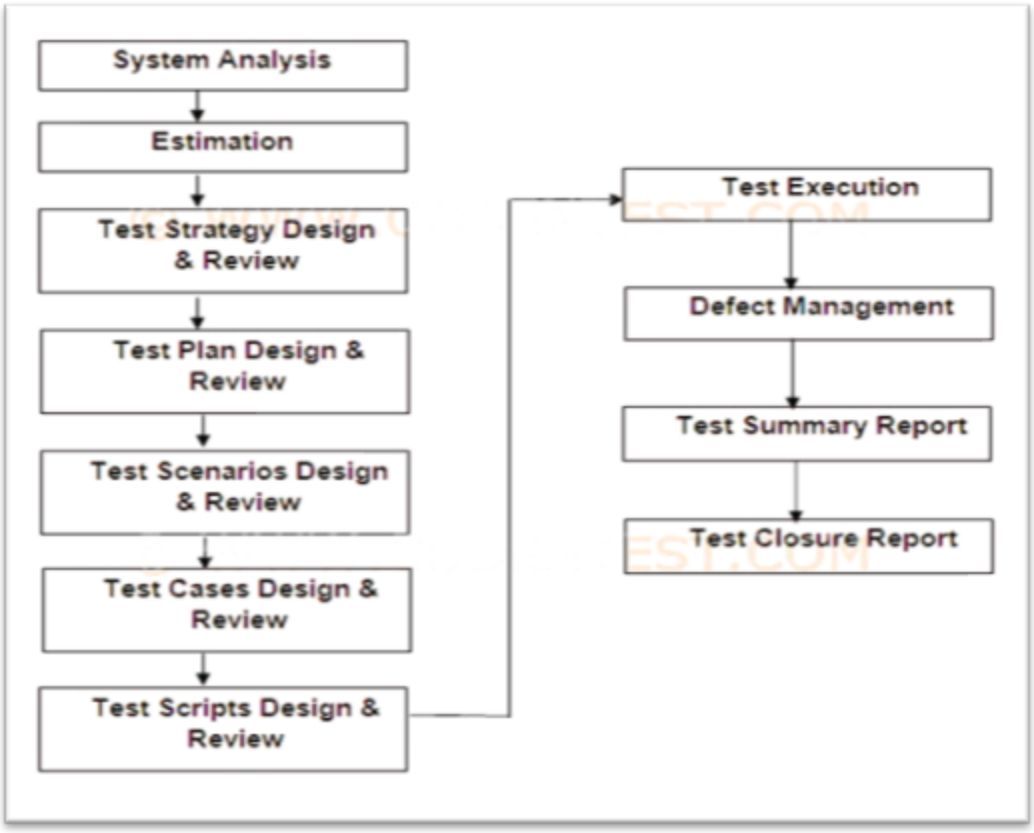
VARIANCE CLASS

Variance Class testing is opposite of Equivalence Class testing. In this, parameters with different requirements are grouped. We cannot use test cases of one parameter for another, seperate test cases must be prepared.

SOFTWARE TEST LIFE CYCLE

Software test life cycle (STLC) is a planning and executing activities to successfully complete a testing project. STLC is a part of Software Development Life Cycle (SDLC).

Test activities differ from organization to organization. The diagram given below shows the activities that are generic.



In this section, we will discuss about System Analysis and estimation.

SYSTEM ANALYSIS

The phase is to analyse the system, gather requirements, and sometimes breaking down the system in different pieces to analyze the situation, engaging clients so that accurate requirements can be defined.

This activity is same as analysis in SDLC.

TEST ESTIMATION

Test Estimation is estimating the testing size, testing effort, testing cost and testing schedule for a software testing project.

Testing Size is the amount (quantity) of testing that needs to be carried out.

Testing Effort is the amount of effort in days/hours necessary for conducting the tests.

Testing cost is the expenses for completing tests. It could be cost of the tools being used, hiring or training resources etc.

Testing Schedule is allocating time periods for different stages in testing. Like at what stage or time, what testing should be carried out.

Note: Each software is unique in its own domain, and it is a tough ask to estimate software accurately at first attemp. Estimations can go wrong, so revise the estimations frequently.

TEST STRATEGY

A test strategy is a high level plan that describes the testing part in software development process.

It is a high level document genereally prepared by managers, and is usually derived from the Business Requirements Specifications (BRS). It describes the standards and activities that are to be followed during testing.

Every organization has its own strategy. Some organizations break the test process in parts and seperate test strategy is prepared for each one. Some organizations include test strategy in test plan, they do not have seperate test strategy, specially for small projects.

Generally Test Strategy will have the following components

- Objectives of Strategy
- Risks and Mitigation
- Test Priorities
- Environmental needs
- Roles and Responsibilities
- Reporting Structure
- Test Deliverables
- Test Approach
- Test Metrices
- Test Summary Report

OBJECTIVES OF STRATEGY

In this the objectives of test strategy are defined. It typically includes, executing planned activities, understanding the risks and mitigations involved, clarify the responsibilities.

RISKS AND MITIGATION

Any risks that will effect the testing process must be listed along with the mitigation. By documenting a risk, its occurrence can be anticipated well ahead of time. Proactive action may be taken to prevent it from occurring.

TEST PRIORITIES

In this the requirements are prioritize, depending on the importance. The most important ones are critical and should be high in priority. Depending on the priorities test scenarios and test cases, we need to establish priorities. While testing software projects, certain test cases will be treated as the most important ones

ENVIRONMENTAL NEEDS

In this the test environment requirements are specified like hardware, operating systems, servers, automation tools, technical skills needed for a tester etc.

ROLES AND RESPONSIBILITIES

The roles and responsibilities of test leader, individual testers, project manager are to be clearly defined.

Both the testing managers and the development managers should approve the test strategy before testing can begin.

REPORTING STRUCTURE

In this, who interacts with the customers, whom do team members and team leaders report to etc is stated in this section.

Apart from that, when test cases are executed should be reported to the test leader and the project manager, to measure where exactly the project stands in terms of testing activities.

TEST DELIVERABLES

Test deliverabes are documents given to managers and customers during testing stages. The different types of Test deliverables are

- Test Strategy
- Test Schedule
- Test Plan
- Test Scenarios
- Test Cases
- Defect Reports
- Matrices
- Test Results
- Test Summary Reports
- User Manual

TEST APPROACH

In the approach is defined, how a software or a piece of it will be tested, will it be manual or automation or both. If manual, what types of testing must be carried out. If automation what tools must be used. If both what parts of software mst be tested manually and what automatically etc.

Apart from the above mentioned things, it also defines the defect management process that has to be followed.

TEST MATRICES

Test Matrices are tables which are used to keep track of test progress.

Normally, requirements are written in columns and functionalities in rows. Having said that every organization will have its own template of matrices, but one thing in common will be, matrices will keep track of the progress and helps us to know where we are at a given stage.

TEST SUMMARY REPORT

The senior management may like to have test summary on a weekly or monthly basis. If the project is very critical, they may need it even on daily basis. The test strategy must give a clear vision of what the testing team will do for the whole project for the entire duration.

TEST PLAN

Test plan is an approach to test a system. A test plan documents the strategy that will be used to verify and ensure that a product or system meets its design specifications and other requirements. A test plan is usually prepared by Leads with significant input from Test Engineers.

Test plan document formats vary as the products and organizations to which they apply. But every test plan should have one thing in common, it should answer "who tests what, when and how?"

Normally, contents in a test plan are

- Test plan identifier
- References
- Introduction
- Test items
- Features to be tested
- Features not to be tested
- Approach
- Item pass/fail criteria
- Entry and exit criteria
- Suspension and Resumption requirements
- Test deliverables
- Remaining testing tasks
- Environmental needs
- Responsibilities
- Staffing and training needs
- Schedule
- Risks and Mitigation
- Approvals
- Glossary

TEST PLAN IDENTIFIER

It is the version number given for test plan. This number is updated as test plan is revised. Sometimes it is the version of the software the test plan is prepared for.

REFERENCES

References are the documents that directly or indirectly support the test plan like Test Strategry, Schedule, Business Requirements, Functional Specifications etc.

INTRODUCTION

Introduction gives purpose of test plan. Its not the introduction of software we are going to test, rather it is introduction of test plan.

The scope and objectives of the plan are identified here.

TEST ITEMS

Items are are to be tested within the scope of this test plan. Remember, what you are testing is what you intend to deliver to the Client.

Specially in testing a complex software, we have different test plans for different stages. In such situation, test plan contains items within the scope of that plan.

FEATURES TO BE TESTED

This section contains features that are to be tested in a software. This is from the users prospective. Customers do not understand technical software terminology, they understand functions in software as they are related to their jobs.

To test features use simple ratings like High, Medium and Low. This is easily understood by customers.

FEATURES NOT TO BE TESTED

This section contains features that are not to be tested. This is from both the Users viewpoint of what the system does and management control view.

Identify why a feature is not to be tested, it could be

- The feature is not to be included in release of the Software
- Low risk, has been used before and is considered stable.

APPROACH

This is your overall test strategy for this test plan; it should be appropriate to the level of the plan and should be in agreement with all higher and lower levels of plans.

If this is a master test plan the overall project testing approach and coverage requirements must also be identified.

ITEM PASS/FAIL CRITERIA

This section mentions the completion criteria for plan. Keep in mind the following things before stating that a plan has been passed or failed.

- All test cases completed
- A specified percentage of cases completed with a percentage containing some number of minor defects
- All lower level plans completed.

ENTRY AND EXIT CRITERIA

This section specifies when to start and end testing. The following things must be considered to stop testing

- All the critical defects must be fixed
- Designed test cases are passed

- Bug rate is down to certain acceptable level
- Amount of confidence on quality of software
- Project deadlines
- Budget allocated for testing

SUSPENSION & RESUMPTION REQUIREMENTS

Know when to pause during testing. It is done when the density of defects is high and where the follow on testing has no value.

Similarly know when to resume testing. Typically when density of defects in a particular module are at an acceptable level to follow on testing.

TEST DELIVERABLES

The documents that are to be delivered to managers and customers as a part of the plan. The different types of Test deliverables are

- Test Strategy
- Test Schedule
- Test Plan
- Test Scenarios
- Test Cases
- Defect Reports
- Matrices
- Test Results
- Test Summary Reports
- User Manual

REMAINING TESTING TASKS

If a software is to be released in increments, there may be parts of the application that this plan does not address. Such plans may only cover a portion of the total functions. This status needs to be identified so that those other areas have plans developed for them and to avoid wasting resources tracking defects that do not relate to this plan.

ENVIRONMENTAL NEEDS

The enviromental needs that are needed for testing are defined here. Like softwares, testing tools, hardware, bandwidth, server capacity, database etc.

RESPONSIBILITIES

Define the roles and responsibilities of testers, team leaders and managers. Who is incharge of what is specified in this section of test plan.

Who is responsible for setting risks, deciding priorities, selecting features to be tested and features not to be tested etc.

STAFFING AND TRAINING NEEDS

Train the resources so that they understand the software they are dealing with. Train them for any automation tool that is to be used in later stages.

If you dont have enough or right man power to test the software staffing must be considered.

SCHEDULE

It is the project schedule, and should be based on realistic and valid terms.

Consider buffer time, while planning schedule. This will be very helpful in later stages when you run out of time and there are some features still to be tested.

RISKS AND MITIGATION

It includes, the critical areas where problems are likely to occur.

Some of the issues could be

- Poorly documented modules or change requests
- Modifications to components with a past history of failure
- misunderstanding of requirement documents

APPROVALS

This section contains the approval process, i.e when a stage of testing is complete who will approve the project to proceed to the next level

GLOSSARY

Used to define terms and acronyms used in the document, and testing in general, to eliminate confusion and promote consistent communications.

TEST SCENARIOS

Test Scenarios are the business processes. Scenario is an interaction between the user and software. Scenarios are written in plain language, with minimal technical details, so that designers, programmers, engineers, managers, marketing specialists etc can have a common understanding.

Scenarios are used directly to define the wanted behavior of software and are derived from Business Requirements Specifications.

Test scenarios are best explained with an example. Consider the form given below

EXAMPLE

*Name:

*Email:

Phone:

Save Reset

Below diagram shows the requirements for the example form.

Parameter	Type	Valid values	Min Length	Max Length
Name	Textbox	Alphabets & Spaces	6	18
Email	Textbox	Valid email format	N/A	N/A
Phone	Textbox	Digits	6	15
Save	Button	N/A	N/A	N/A
Reset	Button	N/A	N/A	N/A

Requirements for example form

A scenario should have a unique Id, description, pre and post conditions. Below are sample scenarios for the example form.

Scenario Id	Description	Pre-condition	Post-condition	Priority
1	Verify the functionality of "Save" button	Enter values in mandatory textboxes	Information should be saved	High
2	Verify the functionality of "Reset" button	Enter values in textbox(es)	Should refresh the form	Medium
3	Verify the functionality of "Name" textbox	N/A	N/A	Medium
4	Verify the functionality of "Email" textbox	N/A	N/A	Medium
5	Verify the functionality of "Phone" textbox	N/A	N/A	Low

Scenarios for example form

Observe the template, we have Scenario Id, Description, Pre and Post conditions and Priority.

Note: In scenarios, priority is importance of a test scenario.

EXPLANATION

The **first scenario** is, checking the functionality of "Save" button and to do that,

- We need to enter values in textboxes, which is the pre-condition.
- After entering values and clicking save button, the system must save the information, this is post-condition.
- Priority is high as save button functionality is important

Note: For the first scenario, the description can be **"Verify the process of saving information".**

The **second scenario**, is checking the functionality of "Reset" button.

- We need to enter values in atleast one textbox, which is the pre-condition.
- After entering values and clicking Reset button, the form must be refreshed, this is post-condition.
- Priority is medium as Reset button functionality is less important to some extent

The first two scenarios have multiple steps. Scenarios can even be prepared for individual parameters like we did in third, fourth and fifth scenarios. Such scenarios are normally called as unit level scenarios.

TEST CASES

Test case is a set of conditions under which a tester will determine whether a functionality of software is working correctly or not.

Test case can also be defined as idea of a tester to test a functionality by giving different inputs.

Test case should have a unique Id, test description, test steps, expected result, actual result, status, author name.

EXAMPLE

*Name:

*Email:

Phone:

Save Reset

This is the same example form from the test scenario chapter. Lets write test cases for scenario number one. (Please check the scenario diagram in the previous chapter)

Author	Test Engineer 1				
Test Id	Description	Test Steps	Expected Result	Actual Result	Status
1	Test the "Save" button with valid values in mandatory fields	Step:1 Enter valid values in "Name" Step:2 Enter valid values in "Email" Step:3 Click "Save"	The information should be saved		
2	Test the "Save" button with invalid values in mandatory fields	Step:1 Enter invalid values in "Name" Step:2 Enter invalid values in "Email" Step:3 Click "Save"	The information should not be saved. System should request the user to enter valid values.		
3	Test the "Save" button by not entering any data in fields	Step:1 Click "Save"	The information should not be saved. System should request the user to enter values.		

EXPLANATION

In the **first test case**, we are testing functionality of "Save" button with valid values in mandatory fields (Name and Email) and to do that, we have steps to follow

1. Enter valid values in Name
2. Enter valid values in Email
3. Click Save button

The test steps and the actual values entered in the fields are called **Test Scripts**.

And the expected result is, information should be saved. Actual result and status is entered when test case is executed.

Because of space constraint, only three test cases were shown in the diagram. Apart from the three test cases, you can go ahead and add the following test cases.

Test Save button with

1. valid values in Name and invalid in values in Email
2. invalid values in Name and valid values in Email
3. valid values in mandatory and non-mandatory fields
4. invalid values in mandatory and non-mandatory fields
5. valid values in mandatory and invalid in non-mandatory
6. invalid values in mandatory and valid in non-mandatory fields

Expected result for Test cases 1, 2, 4 and 6 is system should not save the information. And expected result for test cases 3 and 5 is information should be stored.

The above test cases are scenario based test cases, i.e. to execute one test case, we need to follow some steps. In our case, we enter valid or invalid values in Name, valid or invalid values in Email and click Submit.

Test cases can be written for individual elements or units. This type of test cases are usually called as Unit level test cases.

Test cases for Name: Please check the requirements for Name field in the previous chapter.

Some of the test cases for Name are

Test Name textbox with

1. Alphabets with no. of characters >6 but <18
2. Alphabets with no. of characters <6
3. Alphabets with no. of characters >18
4. Alphabets with 5 characters
5. Alphabets with 6 characters
6. Alphabets with 7 characters
7. Alphabets with 17 characters
8. Alphabets with 18 characters
9. Alphabets with 19 characters
10. Special Characters only
11. Digits only
12. Alphanumerics only

DEFECT REPORT

When actual result does not match the expected result we say defect has been raised.

Author	Test Engineer 1				
Test Id	Description	Test Steps	Expected Result	Actual Result	Status
1	Test the "Save" button with valid values in mandatory fields	Step:1 Enter valid values in "Name" Step:2 Enter valid values in "Email" Step:3 Click "Save"	The information should be saved		
2	Test the "Save" button with invalid values in mandatory fields	Step:1 Enter invalid values in "Name" Step:2 Enter invalid values in "Email" Step:3 Click "Save"	The information should not be saved. System should request the user to enter valid values.		
3	Test the "Save" button by not entering any data in fields	Step:1 Click "Save"	The information should not be saved. System should request the user to enter values.		

Lets assume we have executed the test cases given in the above diagram. Say, first and third test case has passed and second test case failed.

The diagram given below shows test cases after execution.

Author	Test Engineer 1				
Test Id	Description	Test Steps	Expected Result	Actual Result	Status
1	Test the "Save" button with valid values in mandatory fields	Step:1 Enter valid values in "Name" Step:2 Enter valid values in "Email" Step:3 Click "Save"	The information should be saved	The information is saved	Pass
2	Test the "Save" button with invalid values in mandatory fields	Step:1 Enter invalid values in "Name" Step:2 Enter invalid values in "Email" Step:3 Click "Save"	The information should not be saved. System should request the user to enter valid values.	The information is saved	Fail
3	Test the "Save" button by not entering any data in fields	Step:1 Click "Save"	The information should not be saved. System should request the user to enter values.	System is requesting the user to enter values in Name & Email	Pass

In the second test case, the actual result is not matching the expected result. Hence defect was raised. Given below is the defect description in a sample defect template.

S.No	Defect Description	Severity	Steps to reproduce the defects	Defect State	
1	System is accepting invalid values for "Name" and "Email"	Low		Open	

The above Defect report has Defect description, Severity, Steps to reproduce the defect (in case of complex defects), Defect State. Defect report template differs from organization to organization.

SEVERITY & PRIORITY

Severity is the impact of the defect on the software, i.e. how badly the defect is effecting the software and test process. Where as **Priority** is how fast the defect has to be fixed.

Generally critical functionalities which fail are given high severity and high priority.

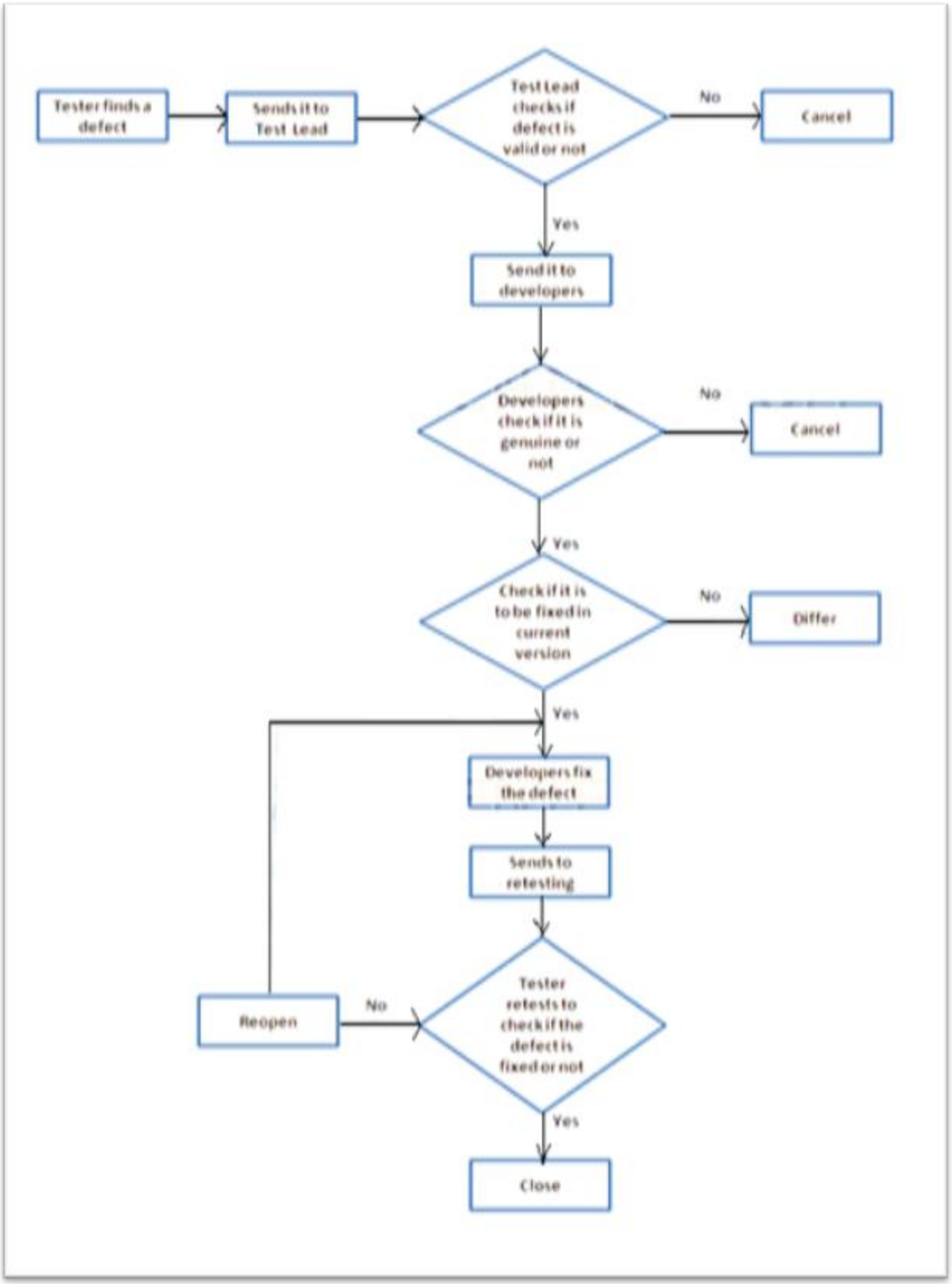
Note: Defect states are discussed in the next chapter.

DEFECT MANAGEMENT PROCESS

When actual output does not match the expected output, we say defect has been raised.

Defect Management is a process of tracing, tracking, fixing and eventually closing the defect. In order to effectively manage defects, an organization has to establish a process.

Below is a diagram which shows the defect management process in general.


Tester finds a defect
Sends it to Test Lead
Test Lead checks if defect is valid or not
No
Cancel
Yes
Send it to developers
Developers check if it is genuine or not
No
Cancel
Yes
Check if it is to be fixed in current version
No
Differ
Yes
Developers fix the defect
Sends to retesting
Tester retests to check if the defect is fixed or not
No
Reopen
Yes
Close

The steps in the above diagram are

1. When tester finds a defect, its state is **new** and is reported to Team Lead. (Normally defects are reported on daily basis)
2. Team Lead checks if the defect is valid or not. If yes, defect state is **open** and is sent to developers. If not, defect is **cancelled**.
3. Developers check if the defect is genuine or not. If yes, it is sent for fixing. If not, defect is **cancelled** and the reason is explained to testers and/or stake-holders.
4. Sometimes a defect raised is supposed to be fixed in later versions of a software, in such cases, the defect is **differed**.
5. When developers fix the defect, its state is **fixed** and it is sent for retesting.
6. Testers check if the defect is fixed or not. If fixed, they **close** it, if not the defect is **re-opened** and is sent back to the developers.
7. Steps 5 and 6 are followed until the defect is closed.

TEST SUMMARY REPORT

Test summary is a report providing information uncovered by the tests accomplished, and including assessments of the quality of the testing effort, the quality of the software system under test, and statistics derived from Incident Reports.

A detailed Test Report will have number of test scenarios and cases prepared, defects raised, fixed, differed etc. Sometimes a report will have feedback given by testers.

The report will also have what testing was done and how long it took, in order to improve any future test planning. This final document is used to indicate whether the software system under test is fit for purpose according to whether or not it has met acceptance criteria defined by project stakeholders.

TESTING PRINCIPLES

Every Test Engineer has to maintain some principles and follow some rules. These principles give us an overview of some key practices that help make software testing successful.

Some of the widely accepted testing principles are

- Testing shows presence of defects
- Exhaustive testing is highly not possible
- Timely and early testing
- Defect clustering
- Balanced testing
- Pesticide paradox
- Testing is content dependent
- Provable
- Sensible
- Absence of errors fallacy

TESTING SHOWS PRESENCE OF DEFECTS

Testing increases the probability of discovering defects in the software. At the same time, if no defects are found, it does not mean that their are no defects in the software.

EXHAUSTIVE TESTING IS HIGHLY NOT POSSIBLE

Testing everything, all combinations of inputs (positive and negative) and preconditions is not possible.

There is no way of validating that your software will efficetively on every variation of system in the world. And testing every possible condition is not possible.

TIMELY AND EARLY TESTING

Earlier the defects are identified in testing life cycle, lesser is the cost associated with fixing it.

Test depending on priority factors. Functionalities which are more important and crucial must be tested first, rather than testing less important functionalities.

DEFECT CLUSTERING

Identifying the density of bugs in a module is called defect clustering. If the density is high in particular module, it is reported that defect clustering has occured in that module.

Once defect clustering has occured, there is no use of testing that module. Instead it is sent back for fixing.

BALANCED TESTING

Testing should be balanced with time, budget and resources available. Significant efforts are needed to test functions that are necessary rather than alignments, colors etc.

Clarity of roles and responsibilities reduces replications and ensures maximum coverage.

PESTICIDE PARADOX

If you keep applying the same pesticide to insects, they eventually build a resistance and become immune, and the pesticide no longer works.

Similarly in software testing, if same tests are repeated over and over again, the tests will no longer find any new defects. To overcome this, the test cases should be regularly revised.

TESTING IS CONTENT DEPENDENT

Testing is done differently in different contexts. For example, a Hospital Management Application is different than a Banking Application, and thus should be tested differently.

PROVABLE

Defects raised by testers are again reproduced by developers to understand the problem and fix them. The defects must be clearly described in defect reporting.

Sometimes a defect can be observed in test environment but not in development environment. Such defects are termed as Environmental issues and at times need to be recorded or screen printed to prove.

SENSIBLE

Many softwares are so complex and run on interdependent environment that it is not possible to test everything. It is necessary to test sensibly by identify the most critical combinations and preconditions.

Another factor is when to decide that enough testing has been done. The following things must be considered to stop testing

- All the critical defects must be fixed
- Designed test cases are passed
- Bug rate is down to certain acceptable level
- Amount of confidence on quality of software
- Project deadlines
- Budget allocated for testing

ABSENCE OF ERRORS FALLACY

A test team can say that, they did not find any defects in the software but they cannot they that a software is defect-free. Testers should assume that all software contains some defects, if not critical, atleast environmental.

SOFT SKILLS NEEDED FOR A TESTER

While the technical skills are important for a tester there are certain soft skills that should complement the technical abilities. Both technical and soft skills compliment each other and the balance between them is what makes a tester a true professional.

ATTITUDE

Before testing, ensure that the requirements are clearly understood. Do not make assumotions regarding the requirements that are not clear.

Most of the times it is difficult to understand the software requirements at once. Begin with simple things and expand to include the complex ones.

Documents will not be available all the times, request assistance from the individuals who understand the software and requirements. Its not a bad idea to build your owm requirement specifications for your reference.

DISCIPLINE

Testing can be extremely repetitive and tiresome. Most of the times testers have to do the same work again and again, and obvious reaction to this is getting tired of of the repetitive work.

Find ways to make your work interesting, concentrate on tasks at hand. Usually its the small things followed by a discipline acheive higher levels of quality.

COMMUNICATION

Communication skills are necessary to succeed in any profession. For a tester, both written and verbal communication are crucial.

From explaning the requirements to someone in the team to reporting defects to developers, or explaning the project status to the stakeholders and customers, communication skills of a tester plays a crucial role.

READING/UNDERSTANDING SKILLS

Testers are supposed to deal with large amount of information everyday, like reading/listening to customers or business analysts to understand the business process and requirements.

This skill makes a big difference when you are supposed to read and understand hundreds of requirement page.

TIME MANAGEMENT

Quality testing is imperative not quantity testing. In testing phase, or any phase of software development, deadlines are set which means that smarter testing has to be done rather than more testing.

Test areas which have been identifies as problematic or complex and where test where bugs have been found before. Testing should be driven by risk factors.

NEGATIVE THINKING

There are some situations in which tester should think neagtively. This will help in preparing negative scenarios, negative test cases etc.

Good testers will never compromise on quality, People sometimes feel they have to sacrifice quality just to meet deadlines. Never do this, no one gets benifit, not your customer, not your organization not you.

Any hardcore tester will have a "test-to-break" attitude. Give such tester a software an organization boasts on quality, he will find defects.

LEARNING ABILITY

New concepts and techniques are introduced every now and then, to make things easier. Learn what is already there, and learn what is new.