Software and its Quality – fundamentals

*Computer*

A computer is a programmable device that stores, retrieves, and processes data.

Computer Architecture:

is a set of rules and methods that describe the functionality, organization, and implementation of computer systems. ... In other definitions computer architecture involves instruction set architecture design, microarchitecture design, logic design, and implementation.

Types:

Computer Organization:

Software:

the programs and other operating information used by a computer.

Web application:

A web application is an application software that runs on a web server, unlike computer-based software programs that are stored locally on the Operating System of the device.

*Web applications are accessed by the user through a web browser with an active internet connection.*

(Web app) is an application program that is stored on a remote server and delivered over the Internet through a browser interface.

Server:

A Server is a centralized computer which offers A specific service via Internet / LAN.

Example: Web Server, Email Server, DB Server.

ECC (Error Correcting Code) RAM is used in processors of Servers.

Server needs an OS.

Proxy server:

Browser:

a computer program with a graphical user interface for displaying HTML files, used to navigate the World Wide Web.

*“Browser displays the webpage” –* URL is required to fetch the resource(GET method),

Browser

parses HTML file.

Fetches linked resources

Builds the document

Renders the document

Publish the DOM.

Runs Java script using the DOM.

Client Server architecture:

*Request by a client and response by a Server.*

Different Service by a Different Server having Different Software providing Different protocols and each protocol having Different port numbers.

Port number:

a port is a communication endpoint.

DNS - Domain Name System:

Computers will have numbers(ip address) instead of names which will be used to get identified over a network.

DNS – resolves names to numbers (domain names to ip address).

If browser/OS cannot find the ip address for the name (URI), query will be sent to “Resolver” which is nothing but the ISP(Internet Service Provider)

If ISP cannot find the query it will be forwarded to “Root Server”

Note: 13 sets of “Root Server” are placed in the world, operated by 12 different organizations. Each set has unique IP address.

“Root Server” will direct the resolver to “TLD Server” [Top Level Domain ]

[TLD Server – stores info of top-level domains like .com,.org,.net]

TLD Server will direct the resolver to “Authoritative Name Servers”.

Once ip address is obtained, Resolver stores it in its cache, so that it can be fed against any further request for the same URI from the browser.

VPN – Virtual Private Network

Is used to encrypt the data and add a layer of privacy to protect the identity.

*Server access the internet on behalf of the client.*

VPN is not an Internet connection; it is *a secure way to access the Internet.*

IP address:

An Internet Protocol address (IP address) is a numerical label assigned to each device connected to a computer network that uses the Internet Protocol for communication.

An IP address serves two main functions: host or network interface identification and location addressing.

Types: 1. IPV4 -32 bits, 4 binary numbers separated by .

2. IPV6 -128 bits 8 Hexadecimal numbers separated by :

Classification:

1. Dynamic – ISP provides from available range.

2. Static – never change, permanent IP address, DNS Server assigns this.

Includes- Continent, Country, Place and which ISP provides the Internet.

Less secure and easy to track.

Also known as dedicated ip.

Note:

A Static IP is assigned by a user manually.

DHCP: Dynamic Host Configuration Protocol

Note:

IP Conflict : Multiple computers(machines) having same ip address which makes those computers not have network access.

Dynamic IP is obtained by DHCP Server automatically.

DHCP assigns a Computer the following items:

IP address

Subnet mask

Default Gateway

DNS Server

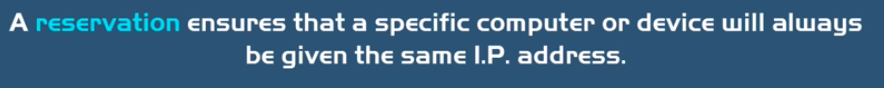
DHCP assigns an IP address to Computer in the network from its “SCOPE”

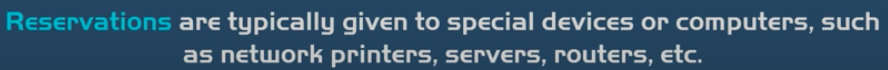
“SCOPE” – A range of IP addresses that a DHCP Server can handout.

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated





A close up of a sign

Description automatically generated

| URL | URI |
| --- | --- |
| URL is used to describe the identity of an item. | URI provides a technique for defining the identity of an item. |
| URL links a web page, a component of a web page or a program on a web page with the help of accessing methods like protocols. | URI is used to distinguish one resource from other regardless of the method used. |
| URL provides the details about what type of protocol is to be used. | URI doesn’t contains the protocol specification. |
| URL is a type of URI. | URI is the superset of URL. |

Network:

A network is a collection of computers, servers, mainframes, network devices, peripherals, or other devices connected to one another  
 to allow the sharing of data.

Common types of area networks are:  
Internet  
LAN - Local Area Network.  
WAN - Wide Area Network.  
WLAN - Wireless Local Area Network.  
MAN - Metropolitan Area Network.  
SAN - Storage Area Network, System Area Network, Server Area Network, or sometimes Small Area Network.

OSI Model:

The Open Systems Interconnection model (OSI model) is a conceptual model that characterises and standardises the communication functions  
 of a telecommunication or computing system without regard to its underlying internal structure and technology.  
 Its goal is the interoperability of diverse communication systems with standard communication protocols.   
The model partitions a communication system into abstraction layers.

TCP/IP:

a set of rules that governs the connection of computer systems to the internet.

TCP/IP is composed of layers, each layer is responsible for performing certain operations on the data it receives and passing it to the higher layers.

Each layer deals with certain issues related to the transmission of data and renders certain services to its higher layers.

When Transmission Control Protocol (TCP) couples with IP, you get the internet highway traffic controller.

TCP and IP work together to transmit data over the internet but at different levels.

Since IP does not guarantee reliable packet delivery over a network, TCP takes the charge of making the connection reliable.

HTTP:

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems.

HTTP is the foundation of data communication for the World Wide Web,

where hypertext documents include hyperlinks to other resources that the user can easily access.

HTTP status codes:

Status codes are issued by a server in response to a client's request made to the server.

It includes codes from IETF Request for Comments (RFCs), other specifications, and some additional codes used in some common applications of the HTTP.

The Internet Assigned Numbers Authority (IANA) maintains the official registry of HTTP status codes.

The first digit of the status code defines the class of response, while the last two digits do not have any classifying or categorization role.

There are five classes defined by the standard:

1xx informational response – the request was received, continuing process

2xx successful – the request was successfully received, understood, and accepted

3xx redirection – further action needs to be taken in order to complete the request

4xx client error – the request contains bad syntax or cannot be fulfilled

5xx server error – the server failed to fulfil an apparently valid request.

HTTP methods:

HTTP defines a set of request methods to indicate the desired action to be performed for a given resource.

The GET Method

GET is used to request data from a specified resource.

GET requests can be cached

GET requests remain in the browser history

GET requests can be bookmarked

GET requests should never be used when dealing with sensitive data

GET requests have length restrictions

GET requests are only used to request data (not modify).

The POST Method

POST is used to send data to a server to create/update a resource.

POST requests are never cached

POST requests do not remain in the browser history

POST requests cannot be bookmarked

POST requests have no restrictions on data length.

The PUT Method

PUT is used to send data to a server to create/update a resource.

The difference between POST and PUT is that PUT requests are idempotent.

That is, calling the same PUT request multiple times will always produce the same result. In contrast,

calling a POST request repeatedly have side effects of creating the same resource multiple times.

The HEAD Method

HEAD is almost identical to GET, but without the response body.

In other words, if GET /users returns a list of users, then HEAD /users will make the same request but will not return the list of users.

HEAD requests are useful for checking what a GET request will return before actually making a GET request - like before downloading a large file or response body.

The DELETE Method

The DELETE method deletes the specified resource.

The OPTIONS Method

The OPTIONS method describes the communication options for the target resource.

PUT v/s POST:

World Wide Web (WWW):

is an information system where documents and other web resources are identified by

Uniform Resource Locators may be interlinked by hypertext, and are accessible over the Internet.

Hypertext:

a software system that links topics on the screen to related information and graphics, which are typically accessed by a point-and-click method.

programming language:

A programming language is a formal language that specifies a set of instructions that can be used to produce various kinds of output.

Programming languages generally consist of instructions for a computer.

Programming languages can be used to create programs that implement specific algorithms.

[Algorithms - An algorithm is a procedure or formula for solving a problem,based on conducting a sequence of specified actions.]

A computer program can be viewed as an elaborate algorithm.

In mathematics and computer science, an algorithm usually means a small procedure that solves a recurrent problem.

The languages, machines,compilers and interpreters are only tools;

like brushes to painters.

The computer version of programming – coding and is mostly used for process automation/communication facilitation.

Classification Of Software Programming languages:

Imperative Declarative

instruct the computer how to do a task tell the computer what to do

1 . procedural - C

2. object-oriented languages– java 1. functional languages -a program is constructed by composing functions.

functional- Haskell;

2. logic programming languages - a program is constructed through a set of logical connections.

C, C++, and Java Logic programming - Prolog.

Statically-typed Dynamically-typed

which typing is checked

(and usually enforced)

prior to running the program

(typically during a compile phase); defer type checking to runtime.

C, C++, and Java Python, Ruby,

JavaScript, and Objective-C

weak typing strong typing

supports implicit type conversions supports implicit type conversions.

Note:

A language is statically typed if the type of a variable is known at compile time.

Scripting language:

A script or scripting language is a computer language with a series of commands within a file

that is capable of being executed without being compiled.

Good examples of server-side scripting languages include Perl, PHP, and Python.

The best example of a client side scripting language is JavaScript.

Basically, all scripting languages are programming languages.

The theoretical difference between the two is that scripting languages do not require the compilation step and are rather interpreted. ...

Generally, compiled programs run faster than interpreted programs because they are first converted to native machine code.

File archiver:

A file archiver is a computer program that combines a number of files together into one archive file, or a series of archive files, for easier transportation or storage.

File archivers employ lossless data compression in their archive formats to reduce the size of the archive.

Basic archivers just take a list of files and concatenate their contents sequentially into archives.

The archive files need to store metadata, at least the names and lengths of the original files, if proper reconstruction is possible.

More advanced archivers store additional metadata, such as the original timestamps, file attributes or access control lists.

Note:

Metadata: Metadata is "data [information] that provides information about other data".

Metadata types:

Descriptive metadata describes a resource for purposes such as discovery and identification

include elements such as title, abstract, author, and keywords.

Structural metadata containers of data and indicates how

compound objects are put together,

for example, how pages are ordered to form chapters. describes the types, versions, relationships and other characteristics of digital materials.

Administrative metadata information to help manage a resource,

such as when and how it was created,

file type and other technical information, and who can access it.

Reference metadata describes the contents and quality of statistical data

Statistical metadata also describe processes that collect, process, or produce statistical data.

also called process data.

Web Service:

A web service is a way for two applications or electronic devices to communicate over a network.

SOAP is a protocol. REST is an architectural style.

2) SOAP stands for Simple Object Access Protocol. REST stands for Representational State Transfer.

REST API has no official standard at all because it is an architectural style.

SOAP API, on the other hand, has an official standard because it is a protocol.

REST APIs uses multiple standards like HTTP, JSON, URL, and XML while SOAP APIs is largely based on HTTP and XML.

REST allows a greater variety of data formats, whereas SOAP only allows XML.

Coupled with JSON (which typically works better with data and offers faster parsing),

REST is generally considered easier to work with. ... REST is generally faster and uses less bandwidth.

Platform = OS+ Processor

Compiler: converts Source code to Assembly language

Assembly language:

instructions to Microprocessor

ex: Store data in some location

Assembler: Converts Assembly language to Machine code in bits

Microservice:

Traditional Application development

Modular Application development

Deploy apps to user's machine

Deploy web apps to Server

Monolithic architecture - dis advantages

1. difficult to deploy big applications.

2. scalability - unpredictable traffic spikes (elastic servers to overcome this).

Microservices are independent applications deployed independently on different servers and communicate

each other over REST APIs and collectively makes a single application for end user.

Each of these independent applications can be scaled independently in their respective servers.

Microservices - advantages

1. Deployment flexibility

2. Technology flexibility

3. Scaled separately

4. Separate Business logic functions.

5. Language independent

Microservices - dis advantages

1. Deployment / Architecture complexity.

2. Service discovery.

DevOps:

DevOps is Development and Operation's Collaboration, It's a Union of Process, People and Working Product that enable continuous integration and continuous delivery of value to our end users.

DevOps accelerate the process to deliver applications and software services at high speed and high velocity.

Container

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly

and reliably from one computing environment to another. ...

Secure: Applications are safer in containers and Docker provides the strongest default isolation capabilities in the industry.

Container orchestration

Container orchestration is the automatic process of managing or scheduling the work of individual containers for applications based on microservices within multiple clusters.

Kubernetes

- is an open source system for automating deployment, scaling and management of

containerized applications.

MVC architecture:

The Model-View-Controller (MVC) is an architectural pattern that separates an application into three main logical components:

the model, the view, and the controller. Each of these components are built to handle specific development aspects of an application.

Model

The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between

the View and Controller components or any other business logic-related data.

For example, a Customer object will retrieve the customer information from the database,

manipulate it and update it data back to the database or use it to render data.

View

The View component is used for all the UI logic of the application. For example, the Customer view will include

all the UI components such as text boxes, dropdowns, etc. that the final user interacts with.

Controller

Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the

Model component and interact with the Views to render the final output. For example, the Customer controller will handle all the interactions and inputs

from the Customer View and update the database using the Customer Model. The same controller will be used to view the Customer data.

mock API

A mock API server imitates a real API server by providing realistic responses to requests.

They can be on your local machine or the public Internet. Responses can be static or dynamic,

and simulate the data the real API would return, matching the schema with data types, objects, and arrays.

Software Environment

A software environment for a particular application could include the operating system, the database system, specific development tools or compiler.

Data Visualization

is the graphical representation of information and data.

Machine Learning

Machine learning is a method of data analysis that automates analytical model building.

It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention.

Business Analytics

Business analytics is the process of collating, sorting, processing, and studying business data, and using statistical models and iterative methodologies to transform data into business insights.

DSL and GPL

DSL: A domain-specific language (DSL) is a computer language specialized to a particular application domain.

This is in contrast to a general-purpose language (GPL), which is broadly applicable across domains. Simpler DSLs,

particularly ones used by a single application, are sometimes informally called mini- languages.

A programming language designed for a particular purpose.

For example, Tex is a language used for typesetting, SQL is used to query databases,

and Mathematica is used for computations. A domain-specific language (DSL) is more fine-tuned to the application environment

than a general-purpose programming language.

*Agile (*able to move quickly and easily*)*

A screenshot of a cell phone

Description automatically generated

|  |  |
| --- | --- |
| byte | 8 bits |
| bit | 0 or 1  0 = 0 volts,1 = 5 volts |
| nibble | 4 bits |
| giga | 1 billion |
| Decimal | based on 10 digits (0,1,2,3,4,5,6,7,8 and 9). |
| Binary | base-2 numeral system |
| Hexadecimal | The hexadecimal numeral system, often shortened to "hex",  is a numeral system made up of 16 symbols (base 16).  Each hexadecimal digit represents four binary digits, also known as a nibble,  which is half a byte. |
| Binary to Decimal |  |
| Hashcode | The value received from hashCode() is used as the bucket number for storing elements of the set/map.   How is hashCode calculated in Java? Simply put, hashCode() returns an integer value, generated by a hashing algorithm. Objects that are equal (according to their equals()) must return the same hash code.  It's not required for different objects to return different hash codes. |
| Unicode | Unicode is a universal character encoding standard. It defines the way individual characters are represented in text files, web pages, and other types of documents. ...  *While ASCII only uses one byte to represent each character, Unicode supports up to 4 bytes for each character.* |
| Code point | A code point is an integer value that uniquely identifies the given character.  Unicode characters can be encoded using different encodings, like UTF-8 or UTF-16.  These encodings specify how each character's Unicode code point is encoded, as one or more bytes. |
| UTF | Unicode Transformation Format.  The '8' means it uses 8-bit blocks to represent a character. |
|  | Unicode is a character set.  UTF-8 is encoding. Unicode is a list of characters with unique decimal numbers (code points). |
| Signed and Unsigned | Unsigned can hold larger positive values, but no negative values.  Unsigned uses the leading bit as a part of the value,  while the signed version uses the leftmost bit to identify whether the number is positive or negative. an unsigned 32-bit int can store up to 232-1, whereas its signed counterpart has a maximum positive value of 231-1. |

Software Testing:

Software testing is a process, to evaluate the functionality of a software application/product with an intent to find whether the developed software met the specified

requirements or not and to identify the defects to ensure that the product is defect free in order to produce the quality product.

Software Testing classification:

A screenshot of a cell phone

Description automatically generated

Functional Testing - Black Box

Black-box testing is a method of software testing that examines the functionality of

an application without peering into its internal structures or workings.

Black Box Testing - types

1. Smoke Testing - also known as “Build Verification Testing”, is a type of software testing that

comprises of a non-exhaustive set of tests that aim at ensuring that the most important functions work.

The result of this testing is used to decide if a build is stable enough to proceed with further testing.

a kind of Software Testing performed after software build to ascertain that the critical functionalities

of the program are working fine. It is executed "before" any detailed functional or regression

tests are executed on the software build.

Smoke testing is done at the build level to check the core functionalities of an application.

2. Sanity Testing - Sanity testing is the subset of regression testing

Sanity testing is performed to ensure that the code changes that are made are working as properly.

This is the first testing on the initial build.

Sanity Testing is done to check the new functionality/bugs have been fixed.

3. Regression Testing - to confirm that a recent program or code change has not adversely affected

existing features.

a full or partial selection of already executed test cases which are re-executed to ensure

existing functionalities work fine.

4. System Testing - the functionalities of the system are tested from an end-to-end perspective to evaluate the complete system's compliance against specified requirements.

5. Integration Testing - Majorly helps to build real-time use cases during the end to end testing.

individual units are combined and tested as a group.

determine if independently developed units of software work correctly when they are connected to each other.

6. UAT - user acceptance testing (UAT) —also called application testing, and end user testing—

is a phase of software development in which the software is tested in the "real world" by the intended audience.

also known as beta or end-user testing, is defined as testing the software by the user or client to determine whether it can be accepted or not. This is the final testing performed once the functional, system and regression testing are completed.

Test process:

Test process defines the complete set of Test activities in a SDLC. And it involves:

1. Planning and Control.

2. Analysis and Design.

3. Implementation and Execution.

4. Evaluating exit criteria and Reporting.

5. Test Closure activities.

Test Plan:

A document describing the scope, approach, resources and schedule of intended test activities.

It is the basis for formally testing any software/product in a project.

Test planning activities guides team to define Test coverage and testing scope.

Test strategy:

A test strategy is an outline that describes the testing approach of the software development cycle.

a high-level document describing the way testing will be carried out in an organization.

A testing strategy is a general approach to the testing process rather than a method of

devising particular system or component tests.

Bug and defect

A bug is the result of a coding error,

A defect is a deviation from the requirements

Verification and Validation

Validation is the process of checking whether the specification captures the customer's needs,

while verification is the process of checking that the software meets the specification.

SDET

SDET stands for Software Development Engineer in Test or Software Design Engineer in Test,

this kind of role is originated from Microsoft.

SDET is an IT professional who can work equally effectively in development and testing roles.

SDET takes part

in the complete software development process.

An SDET's professional's knowledge is entirely focused on

testability, robustness, and performance. They are also able to play a contributory or reviewer

role in the creation of designs for production software.

> Knows the entire system start to end

> SDET is involved in every step of the software development process like

Designing, development, and testing.

> Highly skilled professional with development as well as testing knowledge.

> SDET can participate in test automation tool development and may make it for generic use.

> SDETs need to perform duties like performance testing, automated generation of test data, etc.

> Know requirements and guidelines for the products

White box Testing:

testing of a software solution's internal structure, design, and coding.

It focuses primarily on verifying the flow of inputs and outputs through the application,

improving design and usability, strengthening security.

also known as Clear Box testing, Open Box testing, Structural testing, Transparent Box testing

,Code-Based testing, and Glass 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.

objective is to verify a working flow for an application.

two basic steps of Whitebox testing:

1. UNDERSTAND THE SOURCE CODE

2. CREATE TEST CASES AND EXECUTE

White Box Testing Techniques

Code Coverage analysis techniques:

1. Statement coverage

2. Branch coverage

3. Condition Coverage

4. Function Coverage

5. Path Coverage

Software testing summary:

Software testing is an activity to check whether the actual results match the expected results and to ensure that the software system is Defect free.

It involves execution of a software component or system component to evaluate one or more properties(functionality/behaviour) of interest.

Software testing also helps to identify errors, gaps or missing requirements in contrary to the actual requirements. It can be either done manually or using automation tools.

Testing is important because software bugs could be expensive or even dangerous.

Software bugs can potentially cause monetary and human loss.

7 Software Testing Principles

To determine right strategy for testing? need some basic testing principles.

Exhaustive testing is not possible - the optimal amount of testing based on the risk assessment of the application. defects are likely to be found in multi-tasking activity and need to be tested thoroughly.

Defect Clustering

Defect Clustering which states that a small number of modules contain most of the defects detected. This is the application of the Pareto Principle to software testing: approximately 80% of the problems are found in 20% of the modules.

By experience, we can identify such risky modules.

Pesticide Paradox

If the same set of repetitive tests are conducted, the method will be useless for discovering new defects.

To overcome this, the test cases need to be regularly reviewed & revised, adding new & different test cases to help find more defects.

Testers cannot simply depend on existing test techniques. He must look out continually to improve the existing methods to make testing more effective.

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. i.e. Software Testing reduces the probability of undiscovered defects remaining in the software but even if no defects are found, it is not a proof of correctness.

Absence of Error

It is possible that software which is 99% bug-free is still unusable. This can be the case if the system is tested thoroughly for the wrong requirement. Software testing is not mere finding defects, but also to check that software addresses the business needs. Absence of Error is a Fallacy i.e.. Finding and fixing defects does not help if the system build is unusable and does not full fill the user's needs & requirements.

To solve this problem, the next principle of testing states that Early Testing

Early Testing

Early Testing - Testing should start as early as possible in the Software Development Life Cycle. So that any defects in the requirements or design phase are captured in early stages. It is much cheaper to fix a Defect in early stages of testing. But how early one should start testing? It is recommended that you start finding the bug the moment the requirements are defined.

Testing is context dependent

All the developed software’s are not identical. You might use a different approach, methodologies, techniques and types of testing depending upon the application type.

Summary of the Seven Testing Principles

|  |  |
| --- | --- |
| Principle 1 | Testing shows presence of defects |
| Principle 2 | Exhaustive testing is impossible |
| Principle 3 | Early Testing |
| Principle 4 | Defect Clustering |
| Principle 5 | Pesticide Paradox |
| Principle 6 | Testing is context dependent |
| Principle 7 | Absence of errors - fallacy |

\*\* Test Principles will help to create an effective Test Strategy and draft error catching test cases.

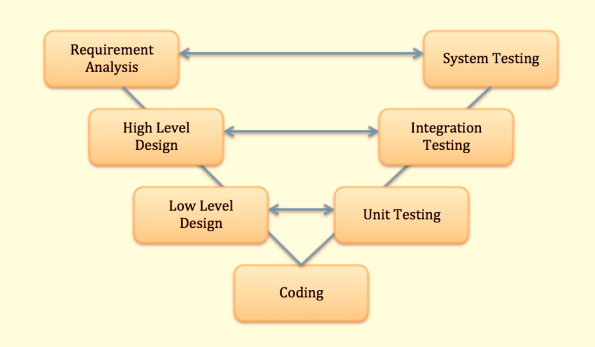
SDLC & STLC

|  |  |
| --- | --- |
| Different phases of Software Development Cycle | Activities performed in each stage |
| **Requirement Gathering stage** | Gather as much information as possible about the details & specifications of the desired software from the client. |
| Design Stage | Plan the programming language like Java, PHP, .net; database like Oracle, MySQL, etc. Which would be suited for the project, also some high-level functions & architecture. |
| **Build Stage** | actually, code the software |
| Test Stage | test the software to verify that it is built as per the specifications |
| **Deployment stage** | Deploy the application in the respective environment. |
| **Maintenance stage** | Once your system is ready to use, you may require to change the code later on as per customer request |

\*\*\* All these levels constitute the **waterfall method** of software development lifecycle. T**esting in the model starts only after implementation is done**.

\*\*\* **defects introduced during requirements & design make up close to half of the total number of defects. The earlier in life cycle a defect is detected, the cheaper it is to fix it.**As the say, "A stitch in time saves a nine."

\*\*\*\* the V model of testing was developed where for every phase, in the Development life cycle there is a corresponding Testing phase.



The left side of the model is Software Development Life Cycle - SDLC

The right side of the model is Software Test Life Cycle - STLC

The entire figure looks like a V, hence the name V – model

Apart from V model, there are iterative development models, where development is carried in phases, with each phase adding a functionality to the software.

Each phase comprises of its independent set of development and testing activities.

Good examples of Development lifecycles following iterative method are Rapid Application Development, Agile Development.

STLC - Software Testing Life Cycle

Software Testing Life Cycle (STLC) is defined as a sequence of activities conducted to perform Software Testing.

It consists of series of activities carried out methodologically to help certify your software product.



Each of these stages have a definite Entry and Exit criteria, Activities & Deliverables associated with it.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STLC Stage** | **Entry Criteria** | **Activity** | **Exit Criteria** | **Deliverables** |
| Requirement  Analysis | Requirements Document available (both functional and non functional)  Acceptance criteria defined. | Analyse business functionality to know the business modules and module specific functionalities.  Identify all transactions in the modules.  Identify all the user profiles.  Gather user interface/authentication, geographic spread requirements.  Identify types of tests to be performed.  Gather details about testing priorities and focus.  Prepare Requirement Traceability Matrix (RTM).  Identify test environment details where testing is supposed to be carried out.  Automation feasibility analysis (if required). | Signed off RTM  Test automation feasibility report signed off by the client | RTM document  Test automation feasibility report |
| Test Planning | Requirements Documents  Requirement Traceability matrix.  Test automation feasibility document. | Analyse various testing approaches available  Finalize on the best suited approach  Preparation of test plan/strategy document for various types of testing  Test tool selection  Test effort estimation  Resource planning and determining roles and responsibilities. | Approved test plan/strategy document.  Effort estimation document signed off. | Test plan/strategy document.  Effort estimation document. |
| Test case development | Requirements Documents  RTM and test plan  Automation analysis report | Create test cases, test design, automation scripts (where applicable)  Review and baseline test cases and scripts  Create test data | Reviewed and signed test Cases/scripts  Reviewed and signed test data | Test cases/scripts  Test data |
| Test Environment setup | System Design and architecture documents are available  Environment set-up plan is available | Understand the required architecture, environment set-up  Prepare hardware and software development requirement list  Finalize connectivity requirements  Prepare environment setup checklist  Setup test Environment and test data  Perform smoke test on the build  Accept/reject the build depending on smoke test result | Environment setup is working as per the plan and checklist  Test data setup is complete  Smoke test is successful | Environment ready with test data set up  Smoke Test Results. |
| Test Execution | Baselined RTM, Test Plan , Test case/scripts are available  Test environment is ready  Test data set up is done  Unit/Integration test report for the build to be tested is available | Execute tests as per plan  Document test results, and log defects for failed cases  Update test plans/test cases, if necessary  Map defects to test cases in RTM  Retest the defect fixes  Regression Testing of application  Track the defects to closure | All tests planned are executed  Defects logged and tracked to closure | Completed RTM with execution status  Test cases updated with results  Defect reports |
| Test Cycle closure | Testing has been completed  Test results are available  Defect logs are available | Evaluate cycle completion criteria based on - Time, Test coverage , Cost , Software Quality , Critical Business Objectives  Prepare test metrics based on the above parameters.  Document the learning out of the project  Prepare Test closure report  Qualitative and quantitative reporting of the work product to the customer.  Test result analysis to find out the defect distribution by type and severity. | Test Closure report signed off by client | Test Closure report  Test metrics |

*What is Manual testing?*

Manual Testing is a type of Software Testing where Testers manually execute test cases without using any automation tools.

Manual testing is the most primitive of all testing types and helps find bugs in the software system.

 Any new application must be manually tested before its testing can be automated. Manual testing requires more effort, but is necessary to check automation feasibility.

Manual Testing does not require knowledge of any testing tool.

One of the Software Testing Fundamental is "**100% Automation is not possible**".

This makes Manual Testing imperative.

Goal of Manual Testing

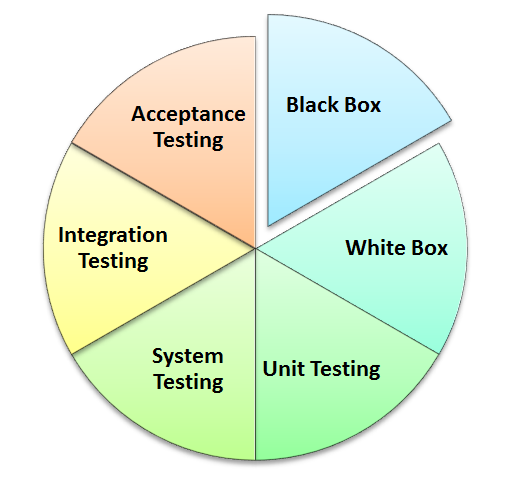
The key concept of Manual Testing is to ensure that the application is error free and it is working in conformance to the specified functional requirements.

Test Suites or cases, are designed during the testing phase and should have 100% test coverage.

It also makes sure that reported defects are fixed by developers and re-testing has been performed by testers on the fixed defects.

Basically, this testing checks the quality of the system and delivers bug-free product to the customer.

Types of Manual Testing:



Myths of Manual Testing

Myth: Anyone can do manual testing

**Fact**: Testing requires many skill sets

Myth: Testing ensures 100% Defect free product

**Fact**: Testing attempts to find as many defects as possible.

Identifying all possible defects is impossible.

Myth: Automated testing is more powerful than manual testing

**Fact**: 100% test automation cannot be done. Manual Testing is also essential.

Myth: Testing is easy

**Fact**: Testing can be extremely challenging.

Testing an application for possible use cases with minimum test cases requires high analytical skills.

Manual Testing vs Automation Testing

|  |  |
| --- | --- |
| Manual Testing | Test Automation |
| Manual testing requires human intervention for test execution. | Test Automation is use of tools to execute test cases |
| Manual testing will require skilled labour, long time & will imply high costs. | Test Automation saves time, cost and manpower. Once recorded, it's easier to run an Test Automation suite |
| Any type of application can be tested manually, certain testing types like ad-hoc and monkey testing are more suited for manual execution. | Test Automation is recommended only for stable systems and is mostly used for Regression Testing |
| Manual testing can be become repetitive and boring. | The boring part of executing same test cases time and again, is handled by automation software in Test Automation. |

*Test Automation:*

Test Automation means using an automation tool to execute your test case suite.

The automation software can also enter test data into the System Under Test ,  compare  expected and actual  results and generate detailed test  reports.

Test Automation demands considerable investments of money and resources.

Successive development cycles will require execution of same test suite repeatedly.

Using a test automation tool it's possible to record this test suite and re-play it  as required.

Once the test suite is automated, no human intervention is required.

This improves ROI of Test Automation.

Goal of Automation is to reduce number of test cases to be run manually and not eliminate manual testing all together.

Test Automation is important due to following reasons:

Manual Testing of all work flows, all fields, all negative scenarios is time and cost consuming

It is difficult to test for multi lingual sites manually

Test Automation does not require Human intervention.

we can run automated test unattended (overnight)

Automation increases speed of test execution

Automation helps increase Test Coverage

Manual Testing can become boring and hence error prone.

Which Test Cases to Automate?

High Risk - Business Critical test cases

Test cases that are executed repeatedly

Test Cases that are very tedious or difficult to perform manually

Test Cases which are time consuming.

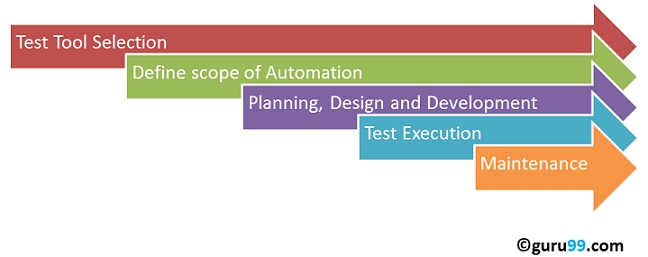
The following category of test cases are not suitable for automation:

Test Cases that are newly designed and not executed manually at least once

Test Cases for which the requirements are changing frequently

Test cases which are executed on ad-hoc basis.

Automated Testing Process:



Test tool selection

Test Tool selection largely depends on the technology the Application Under Test is built on.

**It's a good idea to conduct Proof of Concept of Tool on AUT.**

Define the scope of Automation

Scope of automation is the area of your Application Under Test which will be automated.

points determine scope:

Features of Business importance.

Scenarios which have **large amount of data**

**Common functionalities** across applications

Technical feasibility

Extent to which business components are reused

**Complexity** of test cases

Ability to use the same test cases for cross browser testing.

Planning, Design and Development

During this phase, create Automation strategy & plan, which contains following details-

Automation tools selected

Framework design and its features

In-Scope and Out-of-scope items of automation

Automation test bed preparation

Schedule and Timeline of scripting and execution

Deliverables of automation testing

Test Execution

Automation Scripts are executed during this phase. The scripts need input test data before there are set to run. Once executed they provide detailed test reports.

Execution can be performed using the automation tool directly or through the Test Management tool (CI/CD) which will invoke the automation tool.

Scripts can be executed in a single machine or a group of machines.

Maintenance

As new functionalities are added to the System Under Test with successive cycles, Automation Scripts need to be added, reviewed and maintained for each release cycle.

**Maintenance becomes necessary to *improve effectiveness of Automation Scripts.***

Framework in Automation

A framework is set of automation guidelines which help in

Maintaining consistency of Testing

Improves test structuring

Minimum usage of code

Less Maintenance of code

Improve re-usability

Non-Technical testers can be involved in code

Training period of using the tool can be reduced

Involves Data wherever appropriate

Five types of framework used in automation software testing:

Linear Automation Framework

Modular/Functional Automation Framework

Data Driven Automation Framework

Keyword Driven Automation Framework

Hybrid Automation Framework

Automation Tool Best Practices

Scope of Automation needs to be determined in detail before the start of the project. This sets expectations from Automation right.

Select the right automation tool: A tool must not be selected based on its popularity but it's fit to the automation requirements.

Choose appropriate framework.

Scripting Standards- Standards have to be followed while writing the scripts for Automation. Some of them are-

Create uniform scripts, comments and indentation of the code

Adequate Exception handling - How error is handled on system  failure or unexpected behaviour of the application.

User defined messages should be coded or standardized for Error Logging for testers to understand.

Measure metrics- Success of automation can be determined by capturing the following metrics.

Percent of defects found

Time required for automation testing for each and every release cycle

Minimal Time taken for release

Customer/End user satisfaction Index

Productivity improvement.

Benefits of Automation Testing



70% faster than the manual testing

Wider test coverage of application features

Reliable in results

Ensure Consistency

Saves Time and Cost

Improves accuracy

Human Intervention is not required while execution

Increases Efficiency

Better speed in executing tests

Re-usable test scripts

Test Frequently and thoroughly

More cycle of execution can be achieved through automation

Early time to market

Increased ROI

How to Choose an Automation Tool?

Environment Support

Ease of use

Testing of Database

Object identification

Image Testing

Error Recovery Testing

Object Mapping

Scripting Language Used

Support for various types of test - including functional, test management, mobile, etc...

Support for multiple testing frameworks

Easy to debug the automation software scripts

Ability to recognize objects in any environment

Extensive test reports and results

Minimize training cost of selected tools

\*\* Identify the requirements, explore various tools and its capabilities, set the expectation from the tool and go for a Proof of Concept.

Conclusion

Right selection of Test Automation tool, testing process and team, are important players for automation to be successful. Manual and automation methods go hand-in hand for successful testing.

Software Test types (basic):

UNIT Testing

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.

The goal of unit testing is to isolate each part of the program and show that the individual parts are correct.

Unit testing is usually performed by the developer.

How to Create Unit Test Cases?

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

The IEEE does not favour one over the other.

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

Under the automated approach-

A developer could write another section of code in the application just to test the function. They would later comment out and finally remove the test code when the application is done.

They could also isolate the function to test it more rigorously. This is a more thorough unit testing practice that involves copy and pasting the function from its own testing environment to other than its natural environment.

Isolating the code helps in revealing unnecessary dependencies between the code being tested and other units or data spaces in the product.

These dependencies can then be eliminated.

A coder may use a Unit Test Framework to develop automated test cases.

Using an automation framework, the developer codes criteria into the test to verify the correctness of the unit.

During execution of the test cases, the framework logs those fail any criteria.

Many frameworks will also automatically flag and report in a summary these failed test cases.

Depending upon the severity of a failure, the framework may halt subsequent testing.

Mock Objects

Unit testing relies on mock objects being created to test sections of code that are not yet part of a complete application.

*Mock objects fill in for the missing parts of the program.*

For example, you might have a function that needs variables or objects that are not created yet. In unit testing, those will be accounted for in the form of mock objects created solely for the purpose of the unit testing done on that section of code.

Unit Testing Benefits and Advantage:

Developers looking to learn what functionality is provided by a unit and how to use it can look at the unit tests to gain a *basic understanding of the unit API.*

Unit testing allows the programmer to refactor code at a later date, and make sure the module still works correctly.

The procedure is to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified and fixed.

Due to the modular nature of the unit testing, we can test parts of project without waiting for others to be completed.

Unit Testing Limitations

Unit testing can't be expected to catch every error in a program.

It is not possible to evaluate all execution paths even in the most trivial programs.

Unit testing by its very nature focuses on a unit of code. Hence it can't catch integration errors or broad system level errors.

Unit Testing Techniques

Structural Techniques

Functional Testing Techniques

Error Based Techniques

Unit Testing Best Practices:

Unit Test cases should be independent.

In case of any enhancements or change in requirements, unit test cases should not be affected.

Test only one code at a time.

Follow clear and consistent naming conventions for your unit tests.

In case of change in code in any module, ensure there is a corresponding unit Test Case for the module and the module passes the tests before changing the implementation

Bugs identified during unit testing must be fixed before proceeding to the next phase in SDLC.

Adopt a "test as you code" approach. The more code you write without testing the more paths you have to check for errors.

INTEGRATION Testing: Big Bang, Top Down & Bottom Up

In Integration Testing, individual 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), **'String Testing'** and sometimes 'Thread Testing'.

Why do Integration Testing is required?

Although each software module is unit tested, defects still exist for various reasons like A Module in general is designed by an individual software developer whose understanding and programming logic may differ from other programmers.

Integration testing becomes necessary to *verify the software modules work in unity.*

At the time of module development, there are wide chances of change in requirements by the clients. These new requirements may not be unit tested and hence system integration testing becomes necessary.

Interfaces of the software modules with the database could be erroneous

External Hardware interfaces, if any, could be erroneous

Inadequate exception handling could cause issues.

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.

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case ID | Test Case Objective | Test Case Description | Expected Result |

Approaches/Methodologies/Strategies of Integration Testing:

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 links to be tested could be missed easily.

Since the integration testing can commence only after "all" the modules are designed, 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.

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. Process continues until all of the modules are joined and tested successfully.

This process 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.

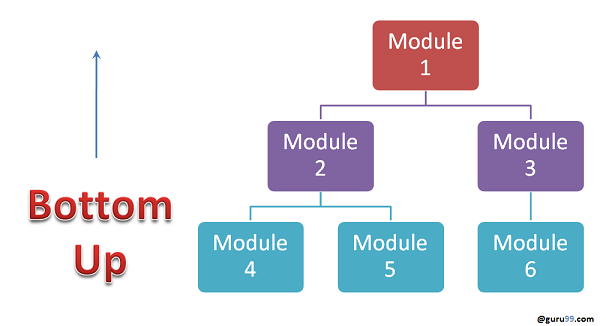
Incremental Approach in turn is carried out by two different Methods:

**Bottom Up**

**Top Down**

Bottom up Integration

In the bottom up strategy, each module at lower levels is tested with higher modules until all modules are tested. It takes help of Drivers for testing



Advantages:

Fault localization is easier.

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

Disadvantages:

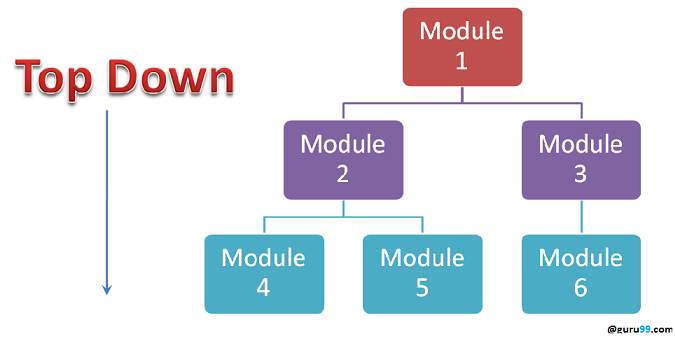
Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.

Early prototype is not possible

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.



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 lower level are tested inadequately.

Integration Testing Procedure

The integration test procedure irrespective of the test strategies:

Prepare the Integration Tests Plan

Design the Test Scenarios, Cases, and Scripts.

Executing the test Cases followed by reporting the defects.

Tracking & re-testing the defects.

Above 2 steps are repeated until the completion of Integration is successfully.

Brief Description of Integration Test Plans:

Methods/Approaches to test (as discussed above).

Scopes and Out of Scopes Items of Integration Testing.

Roles and Responsibilities.

Pre-requisites for Integration testing.

Testing environment.

Risk and Mitigation Plans.

Entry and Exit Criteria.

**Entry Criteria:**

Unit Tested Components/Modules

All High prioritized bugs fixed and closed

All Modules to be code completed and integrated successfully.

Integration tests Plan, test case, scenarios to be signed off and documented.

Required Test Environment to be set up for Integration testing

**Exit Criteria:**

Successful Testing of Integrated Application.

Executed Test Cases are documented

All High prioritized bugs fixed and closed

Technical documents to be submitted followed by release Notes.

Best Practices/ Guidelines for Integration Testing

First determine the Integration Test Strategy that could be adopted and later prepare the test cases and test data accordingly.

Study the Architecture design of the Application and identify the Critical Modules. These need to be tested on priority.

Obtain the interface designs from the Architectural team and create test cases to verify all of the interfaces in detail. Interface to database/external hardware/software application must be tested in detail.

After the test cases, it's the test data which plays the critical role.

Always have the mock data prepared, prior to executing. Do not select test data while executing the test cases.

**What is 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.

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

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

System testing involves testing the software code for following

**Testing the fully integrated applications** including external peripherals in order to check how components interact with one another and with the system as a whole. This is also called End to End testing.

Verify thorough testing of every input in the application to check for desired outputs.

Testing of the user's experience with the application.

That is a very basic description of what is involved in system testing. You need to build detailed test cases and test suites that test each aspect of the application as seen from the outside without looking at the actual source code.

**Software Testing Hierarchy**

The following is a list of software testing categories arranged in chronological order.

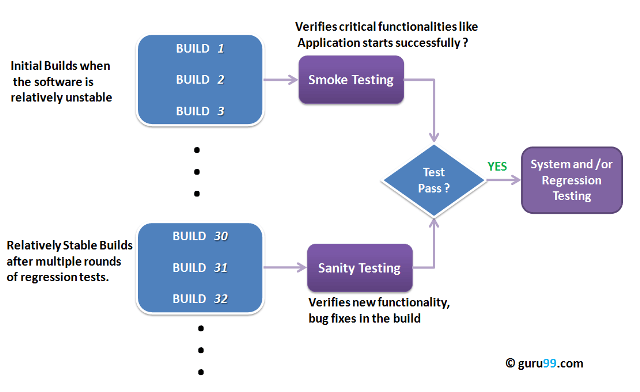
**Unit testing -** testing performed on each module or block of code during development. Unit Testing is normally done by the programmer who writes the code.

**Integration testing -** testing done before, during and after integration of a new module into the main software package. This involves testing of each individual code module. One piece of software can contain several modules which are often created by several different programmers. It is crucial to test each module's effect on the entire program model.

**System testing -** testing done by a professional testing agent on the completed software product before it is introduced to the market.

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

**Sanity Testing Vs Smoke Testing:**



**What is Smoke Testing?**

Smoke Testing is a kind of Software Testing performed after software build to ascertain that the critical functionalities of the program is working fine. It is executed "before" any detailed functional or regression tests are executed on the software build. The purpose is to reject a badly broken application, so that the QA team does not waste time installing and testing the software application.

In Smoke Testing, the test cases chosen cover the most important functionality or component of the system. The objective is not to perform exhaustive testing, but to verify that the critical functionalities of the system is working fine.  
For Example a typical smoke test would be - Verify that the application launches successfully, Check that the GUI is responsive ... etc.

**What is Sanity Testing?**

Sanity testing is a kind of Software Testing performed after receiving a software build, with minor changes in code, or functionality, to ascertain that the bugs have been fixed and no further issues are introduced due to these changes. The goal is to determine that the proposed functionality works roughly as expected. If sanity test fails, the build is rejected to save the time and costs involved in a more rigorous testing.

The objective is "not" to verify thoroughly the new functionality, but to determine that the developer has applied some rationality (sanity) while producing the software.

**Smoke Testing Vs Sanity Testing - Key Differences**

|  |  |
| --- | --- |
| **Smoke Testing** | **Sanity Testing** |
| Smoke Testing is performed to ascertain that the critical functionalities of the program is working fine | Sanity Testing is done to check the new functionality / bugs have been fixed |
| The objective of this testing is to verify the "stability" of the system in order to proceed with more rigorous testing | The objective of the testing is to verify the "rationality" of the system in order to proceed with more rigorous testing |
| This testing is performed by the developers or testers | Sanity testing is usually performed by testers |
| Smoke testing is usually documented or scripted | Sanity testing is usually not documented and is unscripted |
| Smoke testing is a subset of Regression Testing | Sanity testing is a subset of Acceptance testing |
| Smoke testing exercises the entire system from end to end | Sanity testing exercises only the particular component of the entire system |
| Smoke testing is like General Health Check Up | Sanity Testing is like specialized health check up |

**Points to note.**

Both sanity tests and smoke tests are ways to avoid wasting time and effort by quickly determining whether an application is too flawed to merit any rigorous testing.

Sanity Testing is also called tester acceptance testing.

Smoke testing performed on a particular build is also known as a build verification test.

One of the best industry practice is to conduct a Daily build and smoke test in software projects.

Both smoke and sanity tests can be executed manually or using an automation tool.  When automated tools are used, the tests are often initiated by the same process that generates the build itself.

As per the needs of testing, you may have to execute both Sanity and Smoke Tests on the software build. In such cases, you will first execute Smoke tests and then go ahead with Sanity Testing. In industry, test cases for Sanity Testing are commonly combined with that for smoke tests, to speed up test execution. Hence, it's a common that the terms are often confused and used interchangeably.

**Software Quality:**

Software quality is the degree of conformance to explicit or implicit requirements and expectations.

Explicit: clearly defined and documented

Implicit: not clearly defined and documented but indirectly suggested

Requirements: business/product/software requirements

Expectations: mainly end-user expectations

Definition by IEEE

The degree to which a system, component, or process meets specified requirements.

The degree to which a system, component, or process meets customer or user needs or expectations.

Definition by ISTQB

quality: The degree to which a component, system or process meets specified requirements and/or user/customer needs and expectations.

software quality: The totality of functionality and features of a software product that bear on its ability to satisfy stated or implied needs.

As with any definition, the definition of ‘software quality’ is also varied and debatable. Some even say that ‘quality’ cannot be defined and some say that it can be defined but only in a particular context. Some even state confidently that ‘quality is lack of bugs’. Whatever the definition, it is true that quality is something we all aspire to.

Software quality has many dimensions.

**Dimensions of Software Quality:**

Accessibility: The degree to which software can be used comfortably by a wide variety of people, including those who require assistive technologies like screen magnifiers or voice recognition.

Compatibility: The suitability of software for use in different environments like different Operating Systems, Browsers, etc.

Concurrency: The ability of software to service multiple requests to the same resources at the same time.

Efficiency: The ability of software to perform well or achieve a result without wasted energy, resources, effort, time or money.

Functionality: The ability of software to carry out the functions as specified or desired.

Install ability: The ability of software to be installed in a specified environment.

Localizability: The ability of software to be used in different languages, time zones etc.

Maintainability: The ease with which software can be modified (adding features, enhancing features, fixing bugs, etc)

Performance: The speed at which software performs under a particular load.

Portability: The ability of software to be transferred easily from one location to another.

Reliability: The ability of software to perform a required function under stated conditions for stated period of time without any errors.

Scalability: The measure of software’s ability to increase or decrease in performance in response to changes in software’s processing demands.

Security: The extent of protection of software against unauthorized access, invasion of privacy, theft, loss of data, etc.

Testability: The ability of software to be easily tested.

Usability: The degree of software’s ease of use.

**Software Quality Assurance**

Software Quality Assurance (SQA) is a set of activities for ensuring quality in software engineering processes (that ultimately result in quality in software products).

It includes the following activities:

Process definition and implementation

Auditing

Training

Processes could be:

Software Development Methodology

Project Management

Configuration Management

Requirements Development/Management

Estimation

Software Design

Testing

etc

Once the processes have been defined and implemented, Quality Assurance has the following responsibilities:

identify weaknesses in the processes

correct those weaknesses to continually improve the process

The quality management system under which the software system is created is normally based on one or more of the following models/standards:

* **CMMI**
* **Six Sigma**
* **ISO 9000**

Note: There are many other models/standards for quality management but the ones mentioned above are the most popular.

Software Quality Assurance encompasses the entire software development life cycle and the goal is to ensure that the development and/or maintenance processes are continuously improved to produce products that meet specifications/requirements.

The process of Software Quality Control (SQC) is also governed by Software Quality Assurance (SQA).

SQA is generally shortened to just QA.

**Software Quality Control**

SOFTWARE QUALITY CONTROL Fundamentals

Software Quality Control (SQC) is a set of activities for ensuring quality in software products.

It includes the following activities:

Reviews

Requirement Review

Design Review

Code Review

Deployment Plan Review

Test Plan Review

Test Cases Review

Testing

Unit Testing

Integration Testing

System Testing

Acceptance Testing

Software Quality Control is limited to the Review/Testing phases of the Software Development Life Cycle and the goal is to ensure that the products meet specifications/requirements.

The process of Software Quality Control (SQC) is governed by Software Quality Assurance (SQA). While SQA is oriented towards prevention, SQC is oriented towards detection.

***Requirements Traceability Matrix***

Table shows **Requirements/Use cases to Test Cases** relationship.

Business requirements > Software requirements >System Architecture >Design Specifications > Test Cases > Automated Test Scripts

Forward and Backward Traceability.

RTM – 100% Test Coverage.

Reference Docs – BRD, FRD, TDD / Use Case / Wireframes / Documented in a meeting.

***Test Scenarios – intermediaries between requirements and Test Cases.***

**< Wireframes** - A basic visual representation of a website (Web Services) /highlights the functional aspects and APIs of a web App. It won’t focus on UI aspects much / It establishes a hierarchy of info. Simplifies Communications & as a Visual Blueprint of a Web App. >

***Test Strategy***

\*Test Strategy defines the outline of an approach of **testing an Application**.

Test Strategy describes the level of Testing 🡪 Primarily 3 levels – Unit, Integration, System.

Test Strategy notifies Risks and Mitigations.

Test Strategy has to be approved by both Test Manager and Development Managers before testing begins.

Created based on Development Design Documents and System Design Document.

It defines **Entry and Exit criteria for the complete Project.**

***Test Plan***

Day to day activities.

Usually written by one person with consent from Team members.

Test Plan Doc. (by QA Team)

🡪 **Scope** – define the boundaries / Test Scenarios / Test Objectives are validated.

* **Out of Scope** – Enhanced clarity on what is not covered.
* **Assumptions** – Prerequisites needed by QA to go ahead for Testing
* **Schedules** – Test Scenario Preparation / Test Documentation – Test Case, Test Data Creation, Test Environment Set up. / Test Execution /Test Cycle

Start and End date for Testing.

Mile Stone Documentation.

* **Roles & Responsibilities** - Team Members Details. / who is to do what?
* **Deliverables** – What docs (Test Artefacts) are delivered at What time?
* **Environment** – who is in charge? What to do in case of issues?
* **Tools**
* **Defect Management** – Defect Tracking and Flow Chart. / Reporting Process.
* **Risk and Risk Management** – Mitigation Aspects of Risks.
* **Exit Criteria** – when to stop Testing.

\* Test Plan talks about – What, When, How to Test, in which Environment, Tools Used, What Data to Use, Resources and Risk Management.

***Software Test case creation / Generation***

\*\* A test case is a document with User Actions and response (expected result) to it.

Test case writing techniques: Black box techniques. (Normally) / Normally written in excel sheet.

To write Test Cases effectively 🡪

Rules: **Naming conventions for Test Case ID.**

**Simple and Comprehensive**

**Positive (Valid data) and Negative (Invalid data) Test cases.**

Write / Understandable / Run / change, maintain

\* Test Case Template is customizable.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case ID** | **Test Case Objective** | **Test Case Description** | **Expected Result** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Steps** | **Test Data** | **Expected Result** | **Actual Result** | **Status**  **<Not Executed /Blocked /Fail /**  **Pass** | **Comments** |
| **ProjectName**  **\_ModuleName**  **\_Functionality**  **\_TestScenario**  **\_SerialNumber** |  |  |  |  |  |  |

**\*Hierarchy of Testing Activities in a SDLC:**

Application 🡪 Project 🡪 Requirements <High Level> 🡪 Requirements <low Level> 🡪 **Testable** Requirements 🡪 Test Functionalities 🡪 Test Scenarios 🡪Test Cases

(Test Data), (Test Scripts)🡪 Health Check 🡪Exit Report.

***Test Design Techniques:***

To achieve maximum test coverage with minimum (optimal) Test cases.

**Equivalence Class Partitioning:** in Unit, System & Functional Testing Divide Test Data in to Parts.

Valid and Invalid Partitions.

**Boundary Value Analysis:** Test on either side of the boundary.

**Decision Table:** A Black Box Design Technique to determine the Test Scenarios for complex business logic. / also known as **Cause Effect Table.**

Input Data is chosen from Data Set / Combinations based on some logical conditions.

To document complex logic / Test all combinations of conditions / CONDITIONS & ACTIONS.

**Use Case Testing:**

Use Case – A list of steps to achieve a goal in the system / Steps define the interactions between the actor and the system. /Use case captures the functional requirements of the system / It defines the outcomes of the error during System use. / has 2 parts –Main scenarios and Optional Exceptional Scenarios.

< scenarios – Workflow / Sequence of events>

Use Case Testing - Review the Use Case / write Test Cases for both Normal and alternate Work flow.

***Test Case Management***

Use Test Management Tools for Test Case Management.

***Test Data Generation***

Input and Output Test Data

Created on the fly and designed before Test

Valid Test Data / No Test Data / Invalid Test Data

Whole Range of Test Data

System Configuration Data – ex: Connection Strings / Default Settings / Server access Settings

System Data – Menus / Products

Transaction Data – output from user actions.