QA Booklet

A brief collection of basic IT concepts for starting QA / testing job

Table of Contents

1 Testing Types and Techniques

1.1 Unit Testing

UNIT testing is defined as a type of software testing where individual units/ components of software are tested.

Unit testing of software applications is done during the development (coding) of an application. The objective of Unit Testing is to isolate a section of code and verify its correctness. Unit testing is usually performed by the developer.

In SDLC, STLC, V Model, Unit testing is first level of testing done before integration testing. Unit testing is a White Box testing technique that is usually performed by the developer. Though, in a practical world due to time crunch or reluctance of developers to tests, QA engineers also do unit testing.

Unit Tests fix bug early in development cycle and save costs. It helps understand the developers the code base and enable them to make changes quickly.

1.2 Smoke Testing and Sanity testing

Smoke Testing

Smoke Testing is a special type of testing performed on Software build to check the critical functionalities of the program. It is done at the early stage before regression testing or any other detailed testing is performed on the software build. The purpose of smoke testing is to reject badly performing application so that the QA team does not have to waste time in the installation or testing of a software application.

In smoke testing, the test cases are chosen to define the most critical functions of the application or component of the system. The objective is clear here to verify the most critical functionalities of the system either they are working fine or not.

For example, a typical smoke test involves:

- Verification of the application either it is launched successfully or not,

- Verify either GUI of the application is responsive or not.
- Verify either financial transaction is completed well and more….

In brief, Smoke Testing makes sure that build is testable or not received from the development team. It is also named as the “Day 0” check and performed on the build level. It saves time as you don’t have to test the whole application if the core functionalities of the program are not working. So, the primary focus will be the core application workflow in case of the Smoke testing.

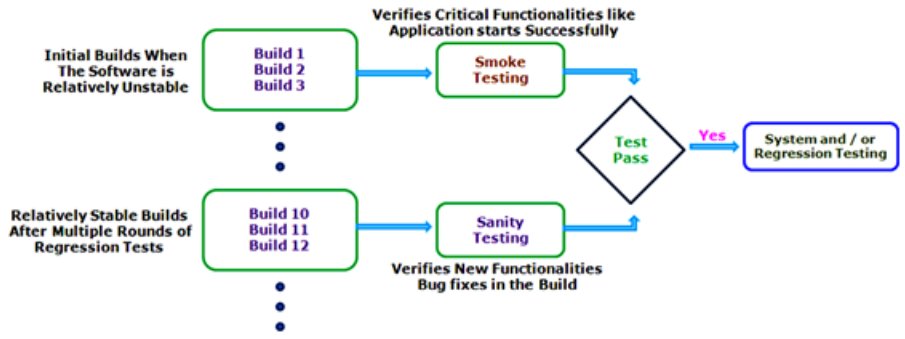
Sanity testing

Sanity Testing is a special type of software testing performed after receiving a software build with little changes in code or functionality to ascertain that certain bugs have been fixed in advance to resolve functionality issues. The goal of sanity testing is to determine that the proposed functionalities are working roughly as expected. If sanity testing fails then the build is rejected directly to save time and costs that are involved in more rigorous testing.

The objective of sanity testing is not to verify the core functionalities thoroughly but to determine that the developer has applied some rationality while building a software program. For example, if your scientific calculator fives the result of 2+2=5! For the instance, then there is no need to check the advanced functionalities like trigonometry calculations or more.

Sanity testing is performed during the release phase to check the main functionalities of an application without going into depth. It is named as the subset of regression testing. There are certain cases when regression testing is not done to the build due to time constraints and sanity testing is considered more suitable to check the main functionalities.

The major difference between both types of testing can be quickly understood by the diagram given below.



Smoke Testing Vs Sanity Testing

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 Acceptance testing	Sanity testing is a subset of Regression 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

1.3 System testing / End to End Testing

System Testing is the testing of a complete and fully integrated software product. Usually, software is only one element of a larger computer-based system. Ultimately, software is interfaced with other software/hardware systems. System Testing is actually a series of different tests whose sole purpose is to exercise the full computer-based system.

Two Category of Software Testing

- Black Box Testing
- White Box Testing

System Testing involves testing the software code for following

- Testing the fully integrated applications including external peripherals in order to check how components interact with one another and with the system as a whole. This is also called End to End testing scenario.
- Verify thorough testing of every input in the application to check for desired outputs.
- Testing of the user's experience with the application.

1.4 What is GUI Testing?

Graphical User Interface Testing is to test the interface between the application and the end user.

1.5 Retesting and Regression testing

Re-testing

Re-testing is a type of testing performed to check the test cases that were unsuccessful in the previous execution are now successfully passed after the defects are fixed.

Regression Testing

Regression Testing is a type of software testing executed to check whether a code change has not unfavorably disturbed current features & functions of an Application

Retesting Vs Regression Testing

Regression Testing	Re-testing
Regression Testing is carried out to confirm whether a recent program or code change has not adversely affected existing features	Re-testing is carried out to confirm the test cases that failed in the final execution are passing after the defects are fixed
The purpose of Regression Testing is that new code changes should not have any side effects to existing functionalities	Re-testing is done on the basis of the Defect fixes
Defect verification is not the part of Regression Testing	Defect verification is the part of re-testing
Based on the project and availability of resources, Regression Testing can be carried out parallel with Re-testing	Priority of re-testing is higher than regression testing, so it is carried out before regression testing
You can do automation for regression testing, Manual Testing could be expensive and time-consuming	You cannot automate the test cases for Retesting.
Regression testing is done for passed test cases	Retesting is done only for failed test cases
Regression testing checks for unexpected side-effects	Re-testing makes sure that the original fault has been corrected

1.6 Integration testing

Testing of all integrated modules to verify the combined functionality after integration is termed as Integration Testing. Modules are typically code modules, individual applications, client and server applications on a network, etc. This type of testing is especially relevant to client/server and distributed systems. Integration testing is the process of testing the interface between two software units or module. It's focus on determining the correctness of the interface. The purpose of the integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit tested, integration testing is performed.

What is Top-Down Approach?

Testing takes place from top to bottom. High-level modules are tested first and then low-level modules and finally integrating the low-level modules to a high level to ensure the system is working as intended. Stubs are used as a temporary module if a module is not ready for integration testing.

What is Bottom-Up Approach?

It is a reciprocate of the Top-Down Approach. Testing takes place from bottom to up. Lowest level modules are tested first and then high-level modules and finally integrating the high-level modules to a low level to ensure the system is working as intended. Drivers are used as a temporary module for integration testing.

Advantages of integration testing:

- Integration testing provides a systematic technique for assembling a software system while conducting tests to uncover errors associated with interfacing.
- The application is tested in order to verify that it meets the standards set by the client as well as reassuring the development team that assumptions which were made during unit testing are correct.
- Integration testing need not wait until all the modules of a system are coded and unit tested. Instead, it can begin as soon as the relevant modules are available.
- Integration testing or incremental testing is necessary to verify whether the software modules work in unity.
- System Integration testing includes a number of techniques like Incremental, Top- down, Bottom –Up, Sandwich and Big Bang Integration techniques.

1.7 Interface testing

Interface Testing is defined as a software testing type which verifies whether the communication between two different software systems is done correctly. A connection that integrates two components is called interface. This interface in a computer world could be anything like API's, web services, etc. Testing of these connecting services or interface is referred to as Interface Testing. An interface is actually software that consists of sets of commands, messages, and other attributes that enable communication between components. This testing ensures that end-users or customer should not encounter any problem when using a particular software product.

1.8 Functional vs. Non-Functional Testing

Functional Testing

Functional testing is a type of testing which verifies that each function of the software application operates in conformance with the requirement specification. This testing mainly involves black box testing, and it is not concerned about the source code of the application. Every functionality of the system is tested by providing appropriate input, verifying the output and comparing the actual results with the expected results. This testing involves checking of User Interface, APIs, Database, security, client/ server applications and functionality of the Application under Test. The testing can be done either manually or using automation.

Non-Functional Testing

Non-functional testing is a type of testing to check non-functional aspects (performance, usability, reliability, etc.) of a software application. It is explicitly designed to test the readiness of a system as per nonfunctional parameters which are never addressed by functional testing.

A good example of non-functional test would be to check how many people can simultaneously login into a software.

Non-functional testing is equally important as functional testing and affects client satisfaction.

Functional Vs. Non-Functional Testing

Parameters	Functional testing	Non-functional testing
Execution	It is performed before non-functional testing.	It is performed after the functional testing.
Focus area	It is based on customer's requirements.	It focuses on customer's expectation.
Requirement	It is easy to define functional requirements.	It is difficult to define the requirements for non-functional testing.
Usage	Helps to validate the behavior of the application.	Helps to validate the performance of the application.
Objective	Carried out to validate software actions.	It is done to validate the performance of the software.
Requirements	Functional testing is carried out using the functional specification.	This kind of testing is carried out by performance specifications
Manual testing	Functional testing is easy to execute by manual testing.	It's very hard to perform non-functional testing manually.
Functionality	It describes what the product does.	It describes how the product works.
Example Test Case	Check login functionality.	The dashboard should load in 2 seconds.

1.9 Performance testing

Performance Testing is defined as a type of software testing to ensure software applications will perform well under their expected workload.

Features and Functionality supported by a software system is not the only concern. A software application's performance like its response time, reliability, resource usage and scalability do matter. The goal of Performance Testing is not to find bugs but to eliminate performance bottlenecks.

The focus of Performance Testing is checking a software program

- Speed - Determines whether the application responds quickly
- Scalability - Determines maximum user load the software application can handle.
- Stability - Determines if the application is stable under varying loads

1.10 Load testing

Load testing is a kind of Performance Testing which determines a system's performance under real-life load conditions. This testing helps determine how the application behaves when multiple users access it simultaneously.

This testing usually identifies -

- The maximum operating capacity of an application
- Determine whether the current infrastructure is sufficient to run the application
- Sustainability of application with respect to peak user load
- Number of concurrent users that an application can support, and scalability to allow more users to access it.

It is a type of non-functional testing. In Software Engineering, Load testing is commonly used for the Client/Server, Web-based applications - both Intranet and Internet.

1.11 Stress testing

Stress Testing is defined as a type of Software Testing that verified the stability & reliability of the system. This test mainly determines the system on its robustness and error handling under extremely heavy load conditions.

It even tests beyond the normal operating point and evaluates how the system works under those extreme conditions. Stress Testing is done to make sure that the system would not crash under crunch situations.

Difference between Load and Stress testing

Load Testing	Stress Testing
Load Testing is to test the system behavior under normal workload conditions, and it is just testing or simulating with the actual workload	Stress testing is to test the system behavior under extreme conditions and is carried out till the system failure.
Load testing does not break the system	Stress testing tries to break the system by testing with overwhelming data or resources.

1.12 Recovery testing

Recovery testing is a type of non-functional testing technique performed in order to determine how quickly the system can recover after it has gone through system crash or hardware failure. Recovery testing is the forced failure of the software to verify if the recovery is successful.

The life cycle of the recovery process can be classified into the following five steps:

1. Normal operation
2. Disaster occurrence
3. Disruption and failure of the operation
4. Disaster clearance through the recovery process
5. Reconstruction of all processes and information to bring the whole system to move to normal operation

1.13 Penetration testing

Penetration testing is also called pen testing or ethical hacking, is the practice of testing a computer system, network or web application to find security vulnerabilities that an attacker could exploit. Penetration testing can be automated with software applications or performed manually. Either way, the process involves gathering information about the target before the test, identifying

possible entry points, attempting to break in -- either virtually or for real -- and reporting back the findings.

The main objective of penetration testing is to identify security weaknesses. Penetration testing can also be used to test an organization's security policy, its adherence to compliance requirements, its employees' security awareness and the organization's ability to identify and respond to security incidents.

1.14 Security testing

Security Testing is a type of Software Testing that uncovers vulnerabilities of the system and determines that the data and resources of the system are protected from possible intruders. It ensures that the software system and application are free from any threats or risks that can cause a loss. Security testing of any system is focuses on finding all possible loopholes and weaknesses of the system which might result into the loss of information or repute of the organization.

The goal of security testing is to:

- To identify the threats in the system.
- To measure the potential vulnerabilities of the system.
- To help in detecting every possible security risks in the system.
- To help developers in fixing the security problems through coding.

Major Focus Areas in Security Testing:

- Network Security
- System Software Security
- Client-side Application Security
- Server-side Application Security

1.15 Compatibility testing

Compatibility Testing is a type of Software testing to check whether your software is capable of running on different hardware, operating systems, applications, network environments or Mobile devices. Compatibility Testing is a type of Non-functional testing.

For example, to test that the software runs successfully on operating systems like Windows, Unix, Mac OS and in browsers like Firefox, Chrome, Internet Explorer etc.

1.16 Exploratory testing

Exploratory testing is all about discovery, investigation, and learning. It emphasizes personal freedom and responsibility of the individual tester. It is defined as a type of testing where Test cases are not created in advance but testers check system on the fly. They may note down ideas about what to test before test execution. The focus of exploratory testing is more on testing as a "thinking" activity.

1.17 Monkey testing

Monkey testing is a technique in software testing where the user tests the application by providing random inputs and checking the behavior (or try to crash the application). Mostly this technique is done automatically where the user enters any random invalid inputs and checks the behavior. As said earlier, there are no rules; this technique does not follows any predefined test cases or strategy and thus works on tester’s mood and gut feeling.

Advantages of Monkey Testing:

- Can identify some out of the box errors.
- Easy to set up and execute
- Can be done by “not so skilled” resources.
- A good technique to test the reliability of the software
- Can identify bugs which may have higher impact.
- Not costly

1.18 Ad hoc testing

Ad hoc testing is defined as an informal testing type with an aim to break the system. This Software Testing type is usually an unplanned activity. It does not follow any test design techniques to create test cases. In fact it does not create test cases altogether!

Ad hoc Testing does not follow any structured way of testing and it is randomly done on any part of application. Main aim of this testing is to find defects by random checking. Ad hoc testing can be achieved with the Software testing technique called Error Guessing. Error guessing can be done by the people having enough experience on the system to "guess" the most likely source of errors.

In Software Engineering, Ad-hoc Testing saves lot of time as it doesn't require elaborate test planning, documentation and Test Case design.

1.19 Accessibility testing

Accessibility Testing is defined as a type of Software Testing performed to ensure that the application being tested is usable by people with disabilities like hearing, color blindness, old age and other disadvantaged groups. It is a subset of Usability Testing.

People with disabilities use assistive technology which helps them in operating a software product. Examples of such software are:

- Speech Recognition Software - It will convert the spoken word to text, which serves as input to the computer.
- Screen reader software - Used to read out the text that is displayed on the screen
- Screen Magnification Software- Used to enlarge the monitor and make reading easy for vision-impaired users.
- Special keyboard made for the users for easy typing who have motor control difficulties

1.20 Usability testing

Usability Testing is defined as a type of software testing where a small set of target end-users, of a software system, "use" it to expose usability defects. This testing mainly focuses on the user's ease to use the application, flexibility in handling controls and the ability of the system to meet its objectives. It is also called User Experience (UX) Testing.

1.21 Acceptance testing

Acceptance Testing is a level of software testing where a system is tested for acceptability. The purpose of this test is to evaluate the system’s compliance with the business requirements and assess whether it is acceptable for delivery.

It is formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system.

1.22 Alpha and Beta testing

Alpha Testing

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

Beta Testing

Beta Testing of a product is performed by "real users" of the software application in a "real environment" and can be considered as a form of external User Acceptance Testing.

Beta version of the software is released to a limited number of end-users of the product to obtain feedback on the product quality. Beta testing reduces product failure risks and provides increased quality of the product through customer validation.

It is the final test before shipping a product to the customers. Direct feedback from customers is a major advantage of Beta Testing. This testing helps to tests the product in customer's environment.

Alpha Testing	Beta Testing

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

1.23 Positive and Negative testing

Positive testing is the type of testing that can be performed on the system by providing the valid data as input. It checks whether an application behaves as expected with positive inputs. This test is done to check the application that does what it is supposed to do.

Negative Testing is a variant of testing that can be performed on the system by providing invalid data as input. It checks whether an application behaves as expected with the negative inputs. This is to test the application does not do anything that it is not supposed to do so.

Following techniques are used for Positive and negative validation of testing is:

Boundary Value Analysis:

This is one of the software testing technique in which the test cases are designed to include values at the boundary. If the input data is used within the boundary value limits, then it is said to be Positive Testing. If the input data is picked outside the boundary value limits, then it is said to be Negative Testing.

Equivalence Partitioning:

This is a software testing technique which divides the input data into many partitions. Values from each partition must be tested at least once. Partitions with valid values are used for Positive Testing. While partitions with invalid values are used for negative testing.

Testing helps deliver quality software application and ensures the software is bug-free before the software is launched. For effective testing, use both - Positive and Negative testing which give enough confidence in the quality of the software.

1.24 Dynamic and Static Testing

Under Static Testing, code is not executed. Rather it manually checks the code, requirement documents, and design documents to find errors. Hence, the name "static".

The main objective of this testing is to improve the quality of software products by finding errors in the early stages of the development cycle. This testing is also called a Non-execution technique or verification testing.

Static testing involves manual or automated reviews of the documents. This review is done during an initial phase of testing to catch Defect early in STLC. It examines work documents and provides review comments.

Under Dynamic Testing, the code is executed. It checks for functional behavior of software system, memory/CPU usage and overall performance of the system. Hence the name "Dynamic"

The main objective of this testing is to confirm that the software product works in conformance with the business requirements. This testing is also called an Execution technique or validation testing.

Static Testing	Dynamic Testing
Testing was done without executing the program	Testing is done by executing the program
This testing does the verification process	Dynamic testing does the validation process
Static testing is about prevention of defects	Dynamic testing is about finding and fixing the defects
Static testing gives an assessment of code and documentation	Dynamic testing gives bugs/bottlenecks in the software system.
Static testing involves a checklist and process to be followed	Dynamic testing involves test cases for execution
This testing can be performed before compilation	Dynamic testing is performed after compilation
Static testing covers the structural and statement coverage testing	Dynamic testing covers the executable file of the code
Cost of finding defects and fixing is less	Cost of finding and fixing defects is high
Return on investment will be high as this process involved at an early stage	Return on investment will be low as this process involves after the development phase
More reviews comments are highly recommended for good quality	More defects are highly recommended for good quality.
Requires loads of meetings	Comparatively requires lesser meetings

1.25 Black box, White box and Gray box testing

Black box testing

Black box testing is defined as a testing technique in which functionality of the Application Under Test (AUT) is tested without looking at the internal code structure, implementation details and knowledge of internal paths of the software. In Black Box Testing we just focus on inputs and output of the software system without bothering about internal knowledge of the software program.

White box testing

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

Black Box Testing	White Box Testing
The main focus of black box testing is on the validation of your functional requirements.	White Box Testing (Unit Testing) validates internal structure and working of your software code
Black box testing gives abstraction from code and focuses on testing effort on the software system behavior.	To conduct White Box Testing, knowledge of underlying programming language is essential. Current day software systems use a variety of programming languages and technologies and it’s not possible to know all of them.

Black box testing facilitates testing communication amongst modules	White box testing does not facilitate testing communication amongst modules

Grey box testing

Gray Box Testing is a software testing technique which is a combination of Black Box Testing technique and White Box Testing technique. In Black Box Testing technique, tester is unknown to the internal structure of the item being tested and in White Box Testing the internal structure is known to tester. The internal structure is partially known in Gray Box Testing. This includes access to internal data structures and algorithms for purpose of designing the test cases.

Gray Box Testing is named so because the software program is like a semitransparent or grey box inside which tester can partially see. It commonly focuses on context-specific errors related to web systems.

1.26 Equivalence Partitioning and Boundary Value Analysis

Boundary value analysis and equivalence partitioning both are test case design strategies in black box testing.

Equivalence Partitioning:

In this method, the input domain data is divided into different equivalence data classes. This method is typically used to reduce the total number of test cases to a finite set of testable test cases, still covering maximum requirements.

In short, it is the process of taking all possible test cases and placing them into classes. One test value is picked from each class while testing.

E.g.: If you are testing for an input box accepting numbers from 1 to 1000 then there is no use in writing thousand test cases for all 1000 valid input numbers plus other test cases for invalid data.

Using equivalence partitioning method above test cases can be divided into three sets of input data called as classes. Each test case is a representative of a respective class.

So in above example, we can divide our test cases into three equivalence classes of some valid and invalid inputs.

Test cases for input box accepting numbers between 1 and 1000 using Equivalence Partitioning:
1) One input data class with all valid inputs. Pick a single value from range 1 to 1000 as a valid test case. If you select other values between 1 and 1000 the result is going to be same. So one test case for valid input data should be sufficient.

2) Input data class with all values below the lower limit. I.e. any value below 1, as an invalid input data test case.

3) Input data with any value greater than 1000 to represent third invalid input class.

So using equivalence partitioning you have categorized all possible test cases into three classes. Test cases with other values from any class should give you the same result.

We have selected one representative from every input class to design our test cases. Test case values are selected in such a way that largest number of attributes of equivalence class can be exercised.

Equivalence partitioning uses fewest test cases to cover maximum requirements.

Boundary value analysis:

It's widely recognized that input values at the extreme ends of input domain cause more errors in the system. More application errors occur at the boundaries of input domain. ‘Boundary value analysis' testing technique is used to identify errors at boundaries rather than finding those exist in center of input domain.

Boundary value analysis is the next part of Equivalence partitioning for designing test cases where test cases are selected at the edges of the equivalence classes.

Test cases for input box accepting numbers between 1 and 1000 using Boundary value analysis:
1) Test cases with test data exactly as the input boundaries of input domain i.e. values 1 and 1000 in our case.

2) Test data with values just below the extreme edges of input domains i.e. values 0 and 999.

3) Test data with values just above the extreme edges of input domain i.e. values 2 and 1001.

Boundary value analysis is often called as a part of stress and negative testing.

1.27 Database testing

Basically, database testing is a layered process. Database systems usually consist of four layers: the user interface (UI) layer, the business layer, the data access layer and the database itself. Testing at these different layers is important for a consistent database system.

Database testing is one of the major testing which requires tester to expertise in checking tables, writing queries and procedures. Testing can be performed in web application or desktop and database can be used in the application like SQL or Oracle. There are many projects like banking, finance, health insurance which requires extensive database testing.

Database testing basically include the following:

1. Data validity testing.
2. Data Integrity testing
3. Performance related to data base.
4. Testing of Procedure, triggers and functions.

- For doing data validity testing one should be good in SQL queries.
- For data integrity testing one should know about referential integrity and different constraint.
- For performance related things one should have idea about the table structure and design.
- For testing Procedure triggers and functions one should be able to understand the same.

1.28 Experience based testing

The experience based testing technique is based on the skill and experience of the testers, experts, users etc. It is conducted in an Ad-hoc manner because proper specifications are not available to test the applications. Here the tester depends on the past experiences with same technologies. The tester as per his experience focuses on the important areas of the software like- the areas mostly used by the customer or the areas most likely to be failed. This kind of testing is done even when there is no specification or have inadequate specification list.

The different types of experience based techniques are:

- Error guessing techniques
- Exploratory Techniques
- Checklist based testing

Some of the expected situations to use experience based techniques are

- Non –availability of requirements and specifications.
- Limited knowledge of software product
- Inadequate specification
- Restricted amount of time

All the members in the IT team can involve in experience based testing

1.29 Web Vs Desktop Testing

Desktop testing - Desktop application runs on work stations and personal computers, so in case you test the desktop application you are concentrating on a particular environment. You will test entire application widely in such categories as GUI, functionality, backend i.e. DB and Load.

Client-server testing - You have 2 various components to test in client server application. An Application is loaded on server machine meanwhile the application exe on each client machine. You will broadly test in such categories as GUI on both sides, Load, functionality, backend, client-server interaction. This environment for the most part is used in Intranet networks. You are informed about number of servers and locations and clients in the test scenario.

Web application testing - Application is loaded on the server and there is no exe installed on the client machine, you are required to test it on various web browsers. It is probable that web applications are to be tested on different OS platforms and browsers so, broadly speaking; web application is tested chiefly for system compatibility and browser compatibility, static pages, load testing, error handling and backend testing.

2 Quality assurance and Quality Control

2.1 Quality Assurance

Quality Assurance is popularly known as QA. QA focuses on improving the processes to deliver Quality Products to the customer. An organization has to ensure, that processes are efficient and effective as per the quality standards defined for software products.

2.2 Quality Control

Quality control popularly abbreviated as QC. It is a Software Engineering process used to ensure quality in a product or a service. It does not deal with the processes used to create a product; rather it examines the quality of the "end products" and the final outcome.

Difference between quality assurance and quality control

Quality Assurance (QA)	Quality Control (QC)
It is a procedure that focuses on providing assurance that quality requested will be achieved	It is a procedure that focuses on fulfilling the quality requested.
QA aims to prevent the defect	QC aims to identify and fix defects
It is a method to manage the quality-Verification	It is a method to verify the quality-Validation
It does not involve executing the program	It always involves executing a program
It's a Preventive technique	It's a Corrective technique
It's a Proactive measure	It's a Reactive measure
It is the procedure to create the deliverables	It is the procedure to verify that deliverables
QA involves in full software development life cycle	QC involves in full software testing life cycle
In order to meet the customer requirements, QA defines standards and methodologies	QC confirms that the standards are followed while working on the product
It is performed before Quality Control	It is performed only after QA activity is done
It is a Low-Level Activity, it can identify an error and mistakes which QC cannot	It is a High-Level Activity, it can identify an error that QA cannot
Its main motive is to prevent defects in the system. It is a less time-consuming activity	Its main motive is to identify defects or bugs in the system. It is a more time-consuming activity
QA ensures that everything is executed in the right way, and that is why it falls under verification activity	QC ensures that whatever we have done is as per the requirement, and that is why it falls under validation activity

It requires the involvement of the whole team	It requires the involvement of the Testing team

2.3 Software Quality

Quality is simply stated: "Fit for use or purpose." It is all about meeting the needs and expectations of customers with respect to functionality, design, reliability, durability, & price of the product.

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

Software quality is the characteristic of the software that defines how well the software meets the customer requirements, business requirements, coding standards etc. It can be divided into two categories:

Software Functional Quality: characteristics that define how well the software meets functional requirements, and how well it satisfies the end-users.

Software Non-Functional Quality: characteristics that define how well structural requirements are met that support the delivery of functional requirements. It is usually related to software code and internal structure.

2.4 Software Quality Parameters

Following are the different attributes (parameters) that are used to measure the software quality:

Testability – How easy it is to test the software and to what extent it can be tested.

Usability – It defines how user friendly the software is.

Maintainability – How easily the software can be maintained in terms of enhancements/new features/bug fixes.

Reliability – How reliable the software is in performing required functions under different scenarios/conditions.

Portability – How easily the software can be transported and run in different environments e.g. platforms like operating systems (Linux, Mac, Windows), machines it can run on.

Security – How secured the software is in terms of access authorization and personal data like passwords.

Robustness – How robust the software is under unexpected events like software crash, power-off etc. and saves its data.

Consistency – How far the software is consistent / uniform in terms of GUI, terminologies, conventions.

Efficiency – It defines the amount of work that the software can do in defined timeframe with minimum resources used.

Effectiveness – The extent to which the software satisfies the user and meets user requirements

Accuracy – How accurately the software works with gives the correct results.

2.5 Software Testing

Software testing is a process, to evaluate the functionality of a software application 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 also helps to identify errors, gaps or missing requirements in contrary to the actual requirements. It can be either done manually or using automated tools.

In simple terms, Software Testing means Verification of Application under Test (AUT).

Why is Software Testing Important?

Testing is important because software bugs could be expensive or even dangerous. Software bugs can potentially cause monetary and human loss, and history is full of such examples.

- Some of the Amazon's third party retailers saw their product price is reduced to 1p due to a software glitch. They were left with heavy losses.
- China Airlines Airbus A300 crashed due to a software bug on April 26, 1994, killing 264 innocent live

- In 1985, Canada's Therac-25 radiation therapy machine malfunctioned due to software bug and delivered lethal radiation doses to patients, leaving 3 people dead and critically injuring 3 others.

2.6 Verification and Validation

Criteria	Verification	Validation
Definition	The process of evaluating work-products (not the actual final product) of a development phase to determine whether they meet the specified requirements for that phase.	The process of evaluating software during or at the end of the development process to determine whether it satisfies specified business requirements.
Objective	To ensure that the product is being built according to the requirements and design specifications. In other words, to ensure that work products meet their specified requirements.	To ensure that the product actually meets the user's needs and that the specifications were correct in the first place. In other words, to demonstrate that the product fulfills its intended use when placed in its intended environment.
Question	Are we building the product right?	Are we building the right product?
Evaluation Items	Plans, Requirement Specs, Design Specs, Code, Test Cases	The actual product/software.
Activities	• Reviews • Walkthroughs • Inspections	• Testing

Verification Vs Validation: Key Difference

Verification	Validation
The verifying process includes checking documents, design, code, and program	It is a dynamic mechanism of testing and validating the actual product
It does not involve executing the code	It always involves executing the code
Verification uses methods like reviews, walkthroughs, inspections, and desk- checking etc.	It uses methods like Black Box Testing, White Box Testing, and non-functional testing
Whether the software conforms to specification is checked	It checks whether the software meets the requirements and expectations of a customer
It finds bugs early in the development cycle	It can find bugs that the verification process cannot catch
Target is application and software architecture, specification, complete design, high level, and database design etc.	Target is an actual product
QA team does verification and make sure that the software is as per the requirement in the SRS document.	With the involvement of testing team validation is executed on software code.

It comes before validation	It comes after verification

2.7 Requirement Traceability Matrix

Requirement Traceability Matrix or RTM captures all requirements proposed by the client or software development team and their traceability in a single document delivered at the conclusion of the life-cycle.

In other words, it is a document that maps and traces user requirement with test cases. The main purpose of Requirement Traceability Matrix is to see that all test cases are covered so that no functionality should miss while doing Software testing.

Why RTM is Important?

The main agenda of every tester should be to understand the client's requirement and make sure that the output product should be defect-free. To achieve this goal, every QA should understand the requirement thoroughly and create positive and negative test cases.

This would mean that the software requirements provided by the client have to be further split into different scenarios and further to test cases. Each of this case has to be executed individually.

A question arises here on how to make sure that the requirement is tested considering all possible scenarios/cases? How to ensure that any requirement is not left out of the testing cycle?

A simple way is to trace the requirement with its corresponding test scenarios and test cases. This is achieved through 'Requirement Traceability Matrix.'

The traceability matrix is typically a worksheet that contains the requirements with its all possible test scenarios and cases and their current state, i.e. if they have been passed or failed. This would help the testing team to understand the level of testing activities done for the specific product.

Parameters to include in Requirement Traceability Matrix

- Requirement ID
- Requirement Type and Description
- Test Cases with Status

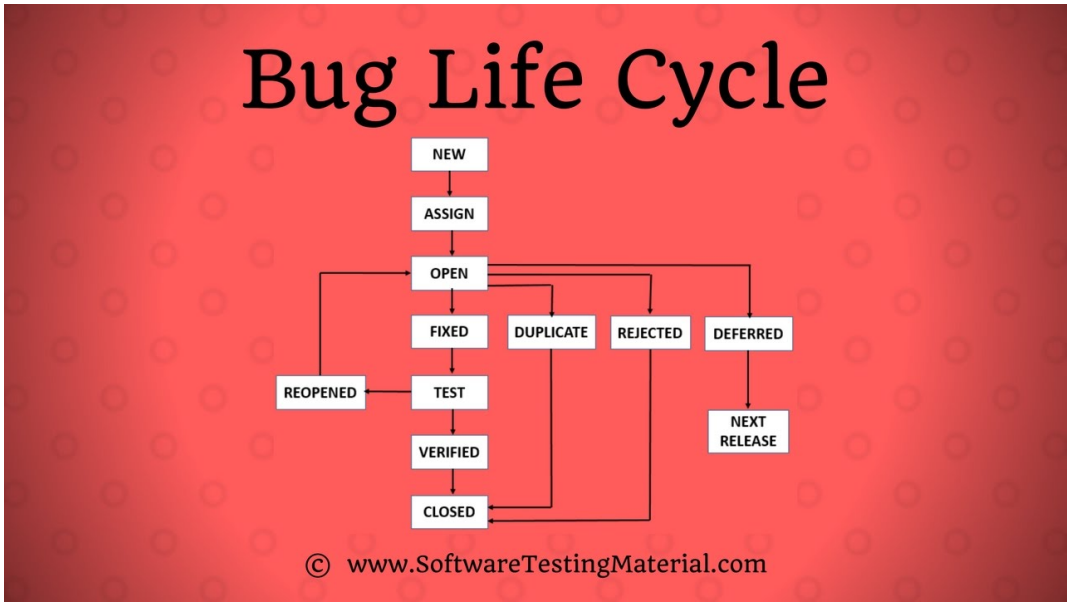
2.8 Defect lifecycle

Defect Life Cycle or Bug Life Cycle is the specific set of states that a Bug goes through from discovery to defect fixation.

Bug Life Cycle Status

The number of states that a defect goes through varies from project to project. Below lifecycle diagram, covers all possible states

- New: When a new defect is logged and posted for the first time. It is assigned a status as NEW.
- **Assigned:** Once the bug is posted by the tester, the lead of the tester approves the bug and assigns the bug to the developer team
- **Open:** The developer starts analyzing and works on the defect fix
- **Fixed:** When a developer makes a necessary code change and verifies the change, he or she can make bug status as "Fixed."
- **Retest:** Tester does the retesting of the code at this stage to check whether the defect is fixed by the developer or not and changes the status to "Re-test."
-



- **Verified:** The tester re-tests the bug after it got fixed by the developer. If there is no bug detected in the software, then the bug is fixed and the status assigned is "verified."
- **Reopen:** If the bug persists even after the developer has fixed the bug, the tester changes the status to "reopened". Once again the bug goes through the life cycle.
- **Closed:** If the bug is no longer exists then tester assigns the status "Closed."
- **Duplicate:** If the defect is repeated twice or the defect corresponds to the same concept of the bug, the status is changed to "duplicate."
- **Rejected:** If the developer feels the defect is not a genuine defect then it changes the defect to "rejected."
- **Deferred:** If the present bug is not of a prime priority and if it is expected to get fixed in the next release, then status "Deferred" is assigned to such bugs
- **Not a bug:** If it does not affect the functionality of the application then the status assigned to a bug is "Not a bug".

2.9 Difference between severity and priority of a defect

Severity is defined as the degree of impact a Defect has on the development or operation of a component application being tested.

Higher effect on the system functionality will lead to the assignment of higher severity to the bug. Quality Assurance engineer usually determines the severity level of defect.

Priority is defined as the order in which a defect should be fixed. Higher the priority the sooner the defect should be resolved.

Generally, defects that leave the software system unusable are given higher priority over defects that cause a small functionality of the software to fail.

Defect Severity and Priority Types

In Software Testing, Defect severity can be categorized into four classes

- **Critical:** This defect indicates complete shut-down of the process, nothing can proceed further
- **Major:** It is a highly severe defect and collapses the system. However, certain parts of the system remain functional
- **Medium:** It causes some undesirable behavior, but the system is still functional
- **Low:** It won't cause any major break-down of the system

Defect priority can be categorized into three classes

- **Low:** The Defect is an irritant but repair can be done once the more serious Defect has been fixed
- **Medium:** During the normal course of the development activities defect should be resolved.

It can wait until a new version is created

- **High:** The defect must be resolved as soon as possible as it affects the system severely and cannot be used until it is fixed

Priority Vs Severity: Key Difference

Priority	Severity
• Defect Priority has defined the order in which the developer should resolve a defect	• Defect Severity is defined as the degree of impact that a defect has on the operation of the product
• Priority is categorized into three types o Low o Medium o High	• Severity is categorized into five types o Critical o Major o Moderate o Minor o Cosmetic
• Priority is associated with scheduling	• Severity is associated with functionality or standards
• Priority indicates how soon the bug should be fixed	• Severity indicates the seriousness of the defect on the product functionality
• Priority of defects is decided in consultation with the manager/client	• QA engineer determines the severity level of the defect
• Priority is driven by business value	• Severity is driven by functionality
• Its value is subjective and can change over a period of time depending on the change in the project situation	• Its value is objective and less likely to change
• High priority and low severity status indicates, defect have to be fixed on immediate bases but does not affect the application	• High severity and low priority status indicates defect have to be fixed but not on immediate bases
• Priority status is based on customer requirements	• Severity status is based on the technical aspect of the product
• During UAT the development team fix defects based on priority	• During SIT, the development team will fix defects based on the severity and then priority

2.10 Difference between Test scenario, Test case and a Test script

Test Case VS Test Scenario

A test case is a set of conditions for evaluation a particular feature of a software product to determine its compliance with the business requirements.

A Test Case is a set of actions executed to verify a particular feature or functionality of your software application. The Test Case has a set test data, precondition, certain expected and actual results developed for specific test scenario to verify any requirement.

A test case includes specific variables or conditions, using which a test engineer can determine as to whether a software product is functioning as per the requirements of the client or the customer.

Whereas, a test scenario is generally a one line statement describing a feature of application to be tested. It is used for end to end testing of a feature and is generally derived from the use cases.

Let's now see the difference between the two -

#	Test Case	Test Scenario
1.	A test case contains clearly defined test steps for testing a feature of an application.	A test scenario contains a high level documentation, describing an end to end functionality to be tested.
2.	Test cases focus on "what to test" and "how to test".	Test scenarios just focus on "what to test".

3.	Test cases have clearly defined step, pre-requisites, expected results etc. Hence, there is no ambiguity.	Test scenarios are generally one-liner. Hence, there is always possibility of ambiguity during testing.
4.	Test cases can be derived from test scenarios and have many to one relationship with them.	Test scenarios are derived from use cases.
5.	Test cases are efficient in exhaustive testing of application.	Test scenarios are beneficial in quick testing of end to end functionality of the application.
6.	More resources are required for documentation and execution of test cases.	Relatively less time and resources are required for creating and testing using scenarios.

Description of the test case

Test case ID:	The ID of the test case
Test case description:	The objective or summary of the test case
Prerequisites:	Any preconditions which need to be executed before starting the test
Test steps:	Procedure to execute the test
Test data:	Data required while executing the test
Expected Result:	The expected output of the test
Actual Result:	The actual output of the test
Status:	Pass, Fail, 'Not executed' when test case is not executed and 'Blocked' when high severity bug is found
Created By:	Name of the person who wrote the test case
Date of creation:	The date on which the test case was authored
Executed By:	Name of the person who ran or executed the test case
Date of execution:	The date on which the test case was executed

Test Script

A TEST SCRIPT is a set of instructions (written using a scripting/programming language) that is performed on a system under test to verify that the system performs as expected. Test scripts are used in automated testing.

Sometimes, a set of instructions (written in a human language), used in manual testing, is also called a Test Script but a better term for that would be a Test Case.

Some scripting languages used in automated testing are:

- JavaScript
- Perl
- Python
- Ruby
- VBScript

2.11 What is Test Suite?

Test Suite is a collection of test cases. The test cases which are intended to test an application.

2.12 What is test coverage?

Test coverage helps in measuring the amount of testing performed by a set of tests.
Test coverage can be done on both functional and non-functional activities. It assists testers to create tests that cover areas which are missing.

2.13 What is Test Bed?

An environment configured for testing. Test bed consists of hardware, software, network configuration, an application under test and other related software.

2.14 Difference between Use case and Test case

COMPARISON PARAMETER	USE CASE	TEST CASE
	A sequential actions which	A Group of test inputs,

Definition	is use to describe the interaction among the role and system to maintain a specified objective,	conditions and variables by which the characteristics of the software is defined.
Goal	To reach the last operation follow all sequential operation	Validating the software as it is working fine or not.
Iteration	it follows different paths	it follows single test case is tested at a time
Dependency	it is dependent on the requirements	it is dependent over the use case
Requirement	Documents and research is required	Test inputs scripts and each test scripts complete one step
Completion	complete all step once	The testing is done again and again then finish.
Interaction	User	Results
Working	it is working as following the step by step function ability of the software.	it is working with the help of testers to validate the software

2.15 Difference between build and release

The main difference between Build and Release in Software Testing is that Build is a version of a software the development team hands over to the testing team for testing purposes while Release is a software the testing team hands over to the customer.

Build

After developing the software module, developers convert the source codes into a standalone form or an executable code. Then the development team hands over the build to the testing team to perform testing. Build is in the testing phase; it may have already undergone testing or not. Software testing team checks this build. If it consists of multiple bugs and if it does not meet the requirements, then the software testing team rejects that build. Builds occur prior to the release, and they are generated more frequently.

Release

The release is the final application after completing development and testing. After testing the build, the testing team certifies that software and delivers it to the customer. It is possible for one release to have several builds. Therefore, it is the software delivered to the customer after completing the development and testing phases. Moreover, the release is based on builds, and it can have several builds.

2.16 Seven Principles of Software Testing

1. Testing Shows Presence of Defects:

Testing shows the presence of defects in the software. The goal of testing is to make the software fail. Sufficient testing reduces the presence of defects. In case testers are unable to find defects after repeated regression testing doesn't mean that the software is bug-free.

Testing talks about the presence of defects and don't talk about the absence of defects.

2. Exhaustive Testing is Impossible:

What is Exhaustive Testing?

Testing all the functionalities using all valid and invalid inputs and preconditions is known as Exhaustive testing.

Why it's impossible to achieve Exhaustive Testing?

Assume we have to test an input field which accepts age between 18 to 20 so we do test the field using 18, 19 and 20. In case the same input field accepts the range between 18 to 100 then we have to test using inputs such as 18, 19, 20, 21, …., 99, 100. It's a basic example, you may think that you could achieve it using automation tool. Imagine the same field accepts some billion values. It's impossible to test all possible values due to release time constraints.

If we keep on testing all possible test conditions then the software execution time and costs will

rise. So instead of doing exhaustive testing, risks and priorities will be taken into consideration whilst doing testing and estimating testing efforts.

3. Early Testing:

Defects detected in early phases of SDLC are less expensive to fix. So conducting early testing reduces the cost of fixing defects.

Assume two scenarios, first one is you have identified an incorrect requirement in the requirement gathering phase and the second one is you have identified a bug in the fully developed functionality. It is cheaper to change the incorrect requirement compared to fixing the fully developed functionality which is not working as intended.

4. Defect Clustering:

Defect Clustering in software testing means that a small module or functionality contains most of the bugs or it has the most operational failures.

As per the Pareto Principle (80-20 Rule), 80% of issues comes from 20% of modules and remaining 20% of issues from remaining 80% of modules. So we do emphasize testing on the 20% of modules where we face 80% of bugs.

5. Pesticide Paradox:

Pesticide Paradox in software testing is the process of repeating the same test cases again and again, eventually, the same test cases will no longer find new bugs. So to overcome this Pesticide Paradox, it is necessary to review the test cases regularly and add or update them to find more defects.

6. Testing is Context Dependent:

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

7. Absence of Error – Fallacy:

99% of bug-free software may still be unusable, if wrong requirements were incorporated into the software and the software is not addressing the business needs.

The software which we built not only be a 99% bug-free software but also it must fulfill the business needs otherwise it will become an unusable software.

3 Software Development Models

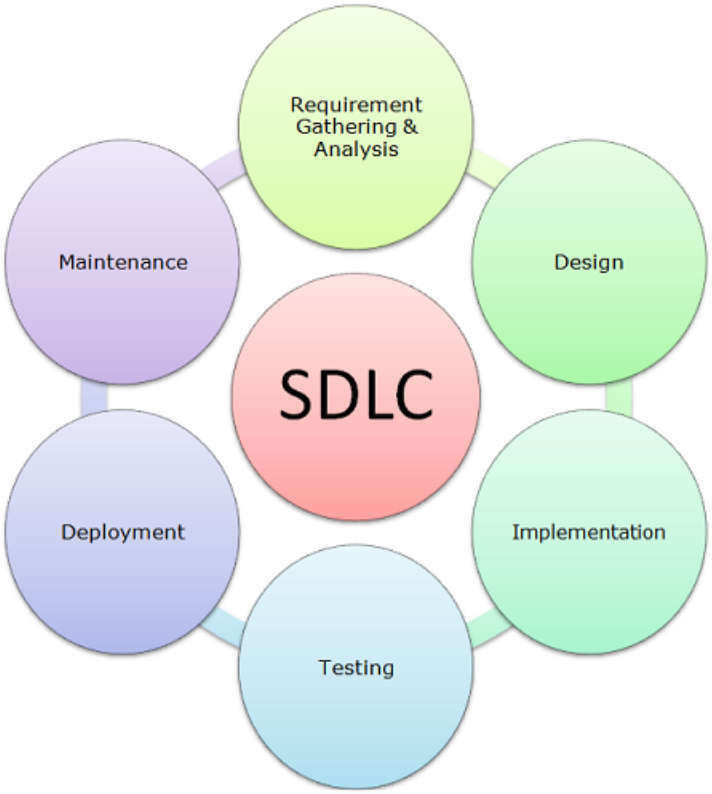
3.1 What is SDLC

The Software Development Lifecycle is a systematic process for building software that ensures and correctness of the software built. SDLC process aims to produce high-quality software which meets customer expectations. The software development should be complete in the pre-defined time frame and cost the quality

SDLC consists of a detailed plan which explains how to plan, build, and maintain specific software. Every phase of the SDLC lifecycle has its own process and deliverables that feed into the next phase.

3.1.1 SDLC Phases

The entire SDLC process divided into the following stages:



- Phase 1: Requirement collection and analysis
- Phase 2: Design:
- Phase 3: Coding/implementation
- Phase 4: Testing:
- Phase 5: Installation/Deployment:
- Phase 6: Maintenance

Requirement collection and analysis:

During this phase, all the relevant information is collected from the customer to develop a product as per their expectation. Any ambiguities must be resolved in this phase only.

Business analyst and Project Manager set up a meeting with the customer to gather all the information like what the customer wants to build, who will be the end-user, what is the purpose of the product. Before building a product a core understanding or knowledge of the product is very important.

For Example, A customer wants to have an application which involves money transactions. In this case, the requirement has to be clear like what kind of transactions will be done, how it will be done, in which currency it will be done, etc.

Once the requirement gathering is done, an analysis is done to check the feasibility of the development of a product. In case of any ambiguity, a call is set up for further discussion.

Once the requirement is clearly understood, the SRS (Software Requirement Specification) document is created. This document should be thoroughly understood by the developers and also should be reviewed by the customer for future reference.

Design:

In this phase, the requirement gathered in the SRS document is used as an input and software architecture that is used for implementing system development is derived.

Coding:

Implementation/Coding starts once the developer gets the Design document. The Software design is translated into source code. All the components of the software are implemented in this phase.

Testing:

Testing starts once the coding is complete and the modules are released for testing. In this phase, the developed software is tested thoroughly and any defects found are assigned to developers to

get them fixed.

Retesting, regression testing is done until the point at which the software is as per the customer's expectation. Testers refer SRS document to make sure that the software is as per the customer's standard.

Installation/Deployment:

Once the product is tested, it is deployed in the production environment or first UAT (User Acceptance testing) is done depending on the customer expectation.

In the case of UAT, a replica of the production environment is created and the customer along with the developers does the testing. If the customer finds the application as expected, then sign off is provided by the customer to go live.

Maintenance:

After the deployment of a product on the production environment, maintenance of the product i.e. if any issue comes up and needs to be fixed or any enhancement is to be done is taken care by the developers.

3.2 SDLC Models

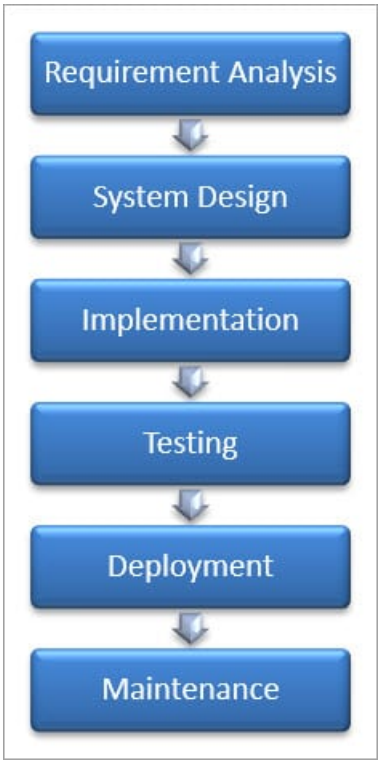
A software life cycle model is a descriptive representation of the software development cycle. SDLC models might have a different approach but the basic phases and activity remain the same for all the models.

Waterfall Model

Waterfall model is the very first model that is used in SDLC. It is also known as the linear sequential model.

In this model, the outcome of one phase is the input for the next phase. Development of the next phase starts only when the previous phase is complete.

- First, Requirement gathering and analysis is done. Once the requirement is freeze then only the System Design can start. Herein, the SRS document created is the output for the Requirement phase and it acts as an input for the System Design.
- In System Design Software architecture and Design, documents which act as an input for the next phase are created i.e. Implementation and coding.
- In the Implementation phase, coding is done and the software developed is the input for the next phase i.e. testing.
- In the testing phase, the developed code is tested thoroughly to detect the defects in the software. Defects are logged into the defect tracking tool and are retested once fixed. Bug logging, Retest, Regression testing goes on until the time the software is in go-live state.
- In the Deployment phase, the developed code is moved into production after the sign off is given by the customer.
- Any issues in the production environment are resolved by the developers which come under maintenance.



Advantages of the Waterfall Model:

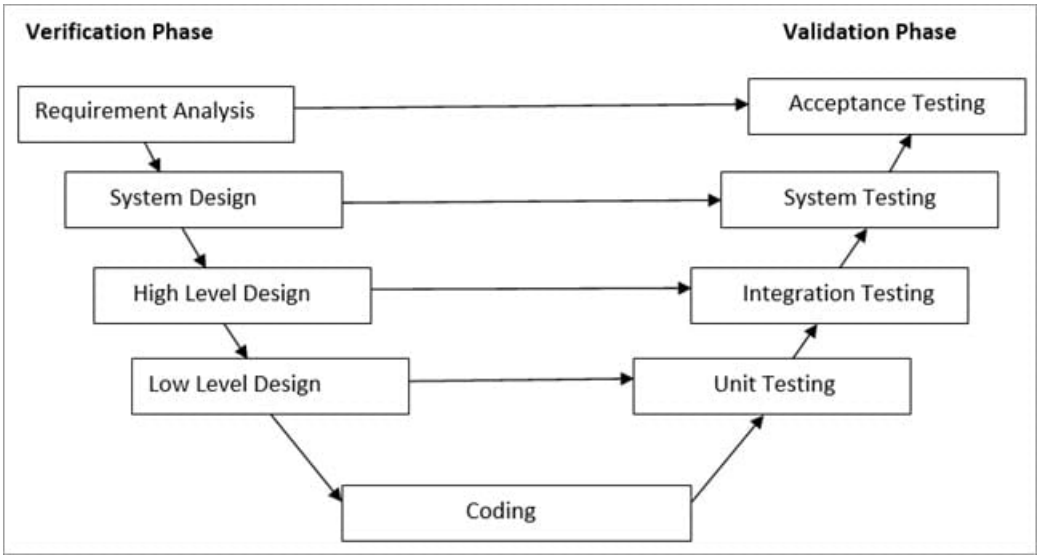
- Waterfall model is the simple model which can be easily understood and is the one in which all the phases are done step by step.
- Deliverables of each phase are well defined, and this leads to no complexity and makes the project easily manageable.

Disadvantages of Waterfall model:

- Waterfall model is time-consuming & cannot be used in the short duration projects as in this model a new phase cannot be started until the ongoing phase is completed.
- Waterfall model cannot be used for the projects which have uncertain requirement or wherein the requirement keeps on changing as this model expects the requirement to be clear in the requirement gathering and analysis phase itself and any change in the later stages would lead to cost higher as the changes would be required in all the phases.

V-Shaped Model

V- Model is also known as Verification and Validation Model. In this model Verification & Validation goes hand in hand i.e. development and testing goes parallel. V model and waterfall model are the same except that the test planning and testing start at an early stage in V-Model.



a) Verification Phase:

(i) Requirement Analysis:

In this phase, all the required information is gathered & analyzed. Verification activities include reviewing the requirements.

(ii) System Design:

Once the requirement is clear, a system is designed i.e. architecture, components of the product are created and documented in a design document.

(iii) High-Level Design:

High-level design defines the architecture/design of modules. It defines the functionality between the two modules.

(iv) Low-Level Design:

Low-level Design defines the architecture/design of individual components.

(v) Coding:

Code development is done in this phase.

b) Validation Phase:

(i) Unit Testing:

Unit testing is performed using the unit test cases that are designed and is done in the Low-level design phase. Unit testing is performed by the developer itself. It is performed on individual components which lead to early defect detection.

(ii) Integration Testing:

Integration testing is performed using integration test cases in High-level Design phase. Integration testing is the testing that is done on integrated modules. It is performed by testers.

(iii) System Testing:

System testing is performed in the System Design phase. In this phase, the complete system is tested i.e. the entire system functionality is tested.

(iv) Acceptance Testing:

Acceptance testing is associated with the Requirement Analysis phase and is done in the customer's environment.

Advantages of V – Model:

- It is a simple and easily understandable model.
- V –model approach is good for smaller projects wherein the requirement is defined and it freezes in the early stage.
- It is a systematic and disciplined model which results in a high-quality product.

Disadvantages of V-Model:

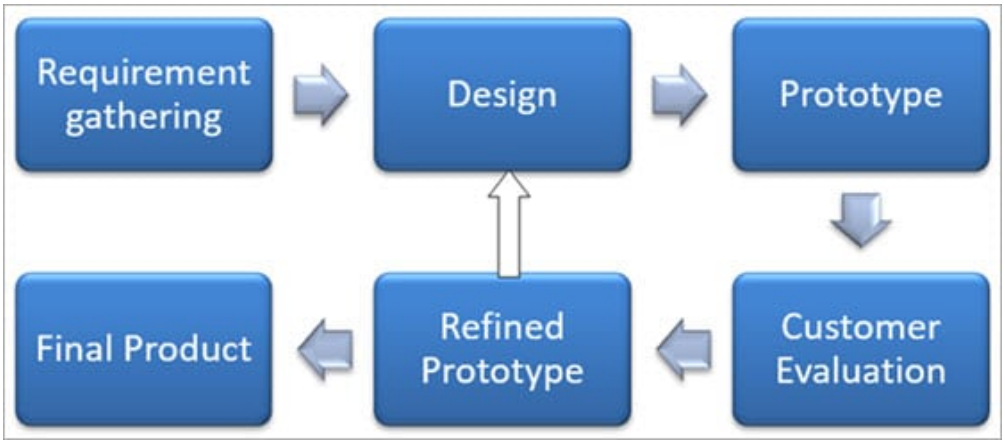
- V-shaped model is not good for ongoing projects.
- Requirement change at the later stage would cost too high.

Prototype Model

The prototype model is a model in which the prototype is developed prior to the actual software.

Prototype models have limited functional capabilities and inefficient performance when compared to the actual software. Dummy functions are used to create prototypes. This is a valuable mechanism for understanding the customers' needs.

Software



prototypes are built prior to the actual software to get valuable feedback from the customer. Feedbacks are implemented and the prototype is again reviewed by the customer for any change. This process goes on until the model is accepted by the customer.

Once the requirement gathering is done, the quick design is created and the prototype which is presented to the customer for evaluation is built.

Customer feedback and the refined requirement is used to modify the prototype and is again presented to the customer for evaluation. Once the customer approves the prototype, it is used as a requirement for building the actual software. The actual software is build using the Waterfall model approach.

Advantages of Prototype Model:

- Prototype model reduces the cost and time of development as the defects are found much earlier.
- Missing feature or functionality or a change in requirement can be identified in the evaluation phase and can be implemented in the refined prototype.
- Involvement of a customer from the initial stage reduces any confusion in the requirement or understanding of any functionality.

Disadvantages of Prototype Model:

- Since the customer is involved in every phase, the customer can change the requirement of the end product which increases the complexity of the scope and may increase the delivery time of the product.

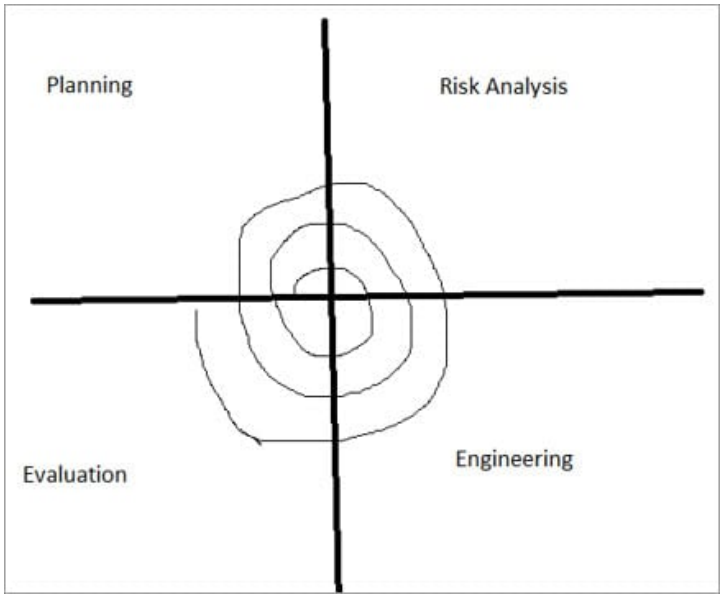
Spiral Model

The Spiral Model includes iterative and prototype approach.

Spiral model phases are followed in the iterations. The loops in the model represent the phase of the SDLC process i.e. the innermost loop is of requirement gathering & analysis which follows the Planning, Risk analysis, development, and evaluation. Next loop is designing followed by Implementation & then testing.

Spiral Model has four phases:

- Planning
- Risk Analysis
- Engineering
- Evaluation



(i) Planning:

The planning phase includes requirement gathering wherein all the required information is gathered from the customer and is documented. Software requirement specification document is created for the next phase.

(ii) Risk Analysis:

In this phase, the best solution is selected for the risks involved and analysis is done by building the prototype.

For Example, the risk involved in accessing the data from a remote database can be that the data access rate might be too slow. The risk can be resolved by building a prototype of the data access subsystem.

(iii) Engineering:

Once the risk analysis is done, coding and testing are done.

(iv) Evaluation:

Customer evaluates the developed system and plans for the next iteration.

Advantages of Spiral Model:

- Risk Analysis is done extensively using the prototype models.
- Any enhancement or change in the functionality can be done in the next iteration.

Disadvantages of Spiral Model:

- The spiral model is best suited for large projects only.
- The cost can be high as it might take a large number of iterations which can lead to high time to reach the final product.

Iterative Incremental Model

The iterative incremental model divides the product into small chunks.

For Example, Feature to be developed in the iteration is decided and implemented. Each iteration goes through the phases namely Requirement Analysis, Designing, Coding, and Testing. Detailed planning is not required in iterations.

Once the iteration is completed, a product is verified and is delivered to the customer for their evaluation and feedback. Customer's feedback is implemented in the next iteration along with the newly added feature.

Hence, the product increments in terms of features and once the iterations are completed the final build holds all the features of the product.

Phases of Iterative & Incremental Development Model:

- Inception phase
- Elaboration Phase
- Construction Phase
- Transition Phase

(i) Inception Phase:

Inception phase includes the requirement and scope of the Project.

(ii) Elaboration Phase:

In the elaboration phase, the working architecture of a product is delivered which covers the risk identified in the inception phase and also fulfills the non-functional requirements.

(iii) Construction Phase:

In the Construction phase, the architecture is filled in with the code which is ready to be deployed and is created through analysis, designing, implementation, and testing of the functional requirement.

(iv) Transition Phase:

In the Transition Phase, the product is deployed in the Production environment.

Advantages of Iterative & Incremental Model:

- Any change in the requirement can be easily done and would not cost as there is a scope of incorporating the new requirement in the next iteration.
- Risk is analyzed & identified in the iterations.
- Defects are detected at an early stage.
- As the product is divided into smaller chunks it is easy to manage the product.

Disadvantages of Iterative & Incremental Model:

- Complete requirement and understanding of a product are required to break down and build incrementally.

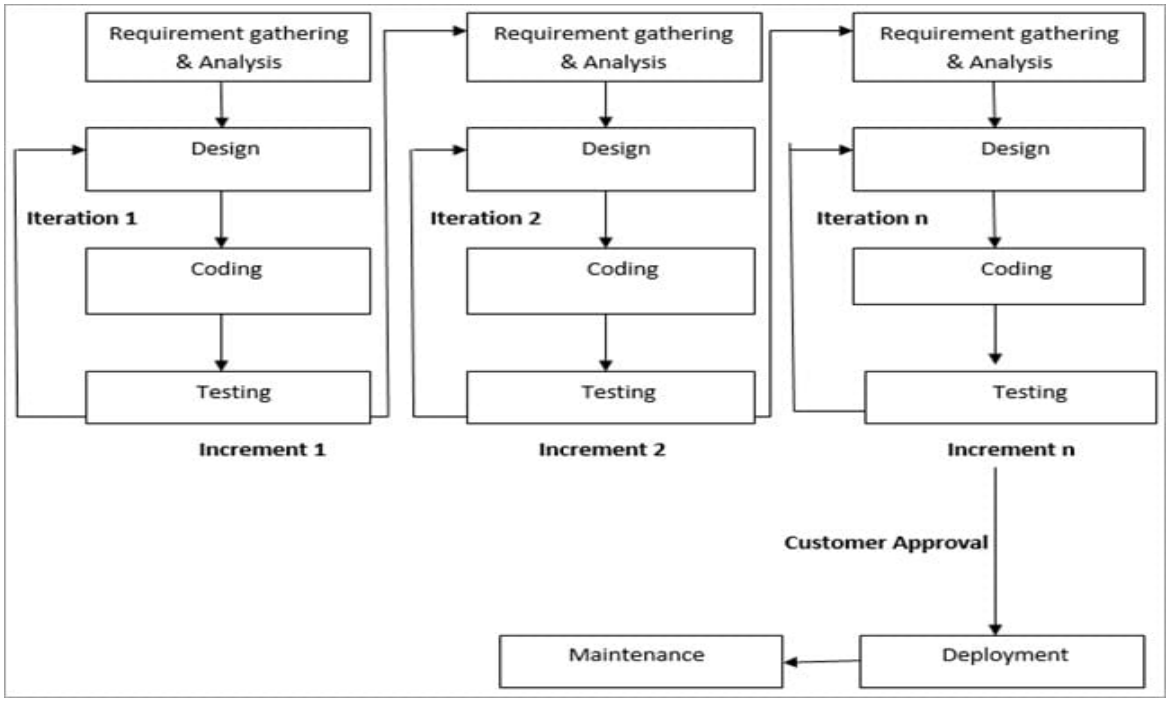
Agile Model

Agile Model is a combination of the Iterative and incremental model. This model focuses more on flexibility while developing a product rather than on the requirement.

In Agile, a product is broken into small incremental builds. It is not developed as a complete product in one go. Each build increments in terms of features. The next build is built on previous functionality.

In agile iterations are termed as sprints. Each sprint lasts for2-4 weeks. At the end of each sprint, the product owner verifies the product and after his approval, it is delivered to the customer.

Customer feedback is taken for improvement and his suggestions and enhancement are worked on in the next sprint. Testing is done in each sprint to minimize the risk of any failures.



Advantages of Agile Model:

- It allows more flexibility to adapt to the changes.
- The new feature can be added easily.
- Customer satisfaction as the feedback and suggestions are taken at every stage.

Disadvantages:

- Lack of documentation.
- Agile needs experienced and highly skilled resources.
- If a customer is not clear about how exactly they want the product to be, then the project would fail.

Conclusion

Adherence to a suitable life cycle is very important, for the successful completion of the Project. This, in turn, makes the management easier.

Different Software Development Life Cycle models have their own Pros and Cons. The best model for any Project can be determined by the factors like Requirement (whether it is clear or unclear), System Complexity, Size of the Project, Cost, Skill limitation, etc.

Example, in case of an unclear requirement, Spiral and Agile models are best to be used as the required change can be accommodated easily at any stage.

Waterfall model is a basic model and all the other SDLC models are based on that only.

3.3 Scrum Framework

Scrum is a framework that is used to implement agile development.

Scrum is a software product development strategy that organizes software developers as a team to reach a common goal — creating a ready-for-market product. It is a widely used subset of agile software development.

The word scrum also is used in rugby to define a play where players struggle against each to gain possession of the ball. The goal of a scrum in software development is to perform at a high-performing level like a rugby team does in a scrum.

How Scrum Works

In a rugby scrum, all the players literally put their heads together. When it comes to software development, a scrum can be characterized by developers putting their heads together to address complex problems.

- Scrum software development starts with a wish list of features — a.k.a. a product backlog.

The team meets to discuss:

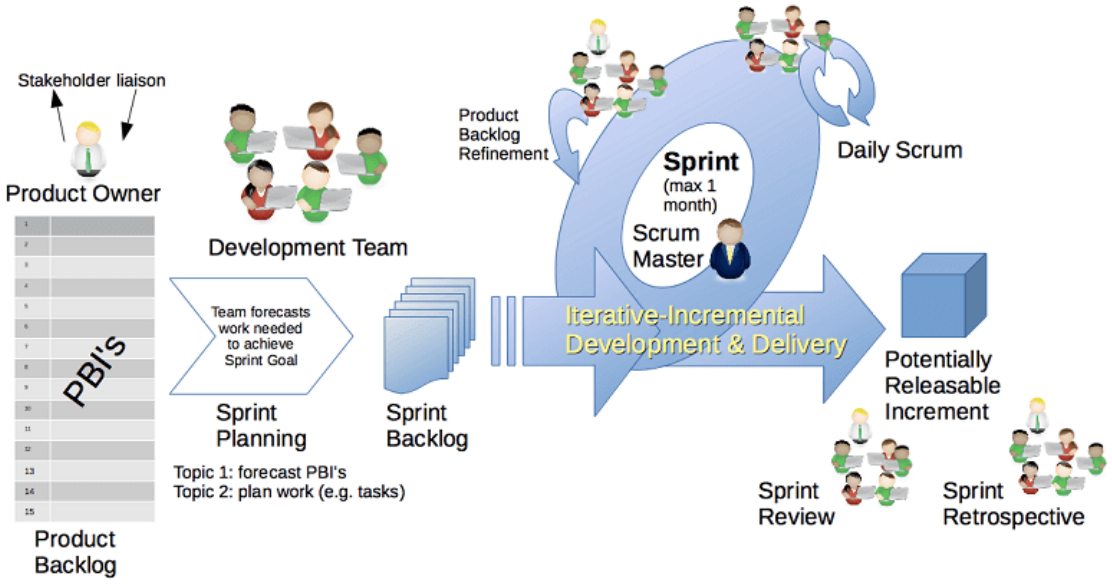
- The backlog.
- What still needs to be completed.
- How long it will take.

• Scrum relies on an agile software development concept called sprints:

- Sprints are periods of time when software development is actually done.
- A sprint usually lasts from one week to one month to complete an item from the backlog.
- The goal of each sprint is to create a saleable product.
- Each sprint ends with a sprint review.
- Then the team chooses another piece of backlog to develop — which starts a new sprint.
- Sprints continue until the project deadline or the project budget is spent.

• In daily scrums, teams meet to discuss their progress since the previous meeting and make plans for that day.

- The meetings should be brief — no longer than 15 minutes.
- Each team member needs to be present and prepared.
- The Scrum Master keeps the team focused on the goal.



Who is in the Scrum?

In rugby, the forwards are involved in the scrum. In software development, three roles are defined in the scrum framework:

The **scrum team** does the work. It is the individuals who are working together in the sprints to produce the products.

The **scrum master** is part of the scrum team makes sure the team works in compliance with the scrum rules. This is not a manager.

The **product owner** represents the customer. This role prioritizes the backlog and coordinates the scrum teamwork. The product owner is a role similar to project manager in more traditional project management frameworks.

Benefits of Scrum

Rugby players try to gain control of the ball in the scrum and move it downfield. Software developers use scrum to move their projects quickly. And the benefits trickle down to software developers:

- Developers who want the freedom to make decisions thrive in scrum teams. Team morale tends to be high.
- Each sprint produces a product that is ready for market even though the project is ongoing. The highest priority requirements are addressed first so a high-quality, low-risk product can be on the market.
- The incremental process shortens the time to market by about 30 percent to 40 percent. Because the product owner is part of the scrum team, requirements can be delivered as they are needed.
- Scrum projects often realize a higher return on investment (ROI). This is attributed to:
    - Decreased time to market.
    - Early and regular feedback that prompts course corrections early when they are less costly.
    - Defects that are fewer and less costly.
    - Projects failing early and quickly when it's less costly.
- Reviewing each sprint before the team moves on to the next sprint spreads testing throughout development.
- Project focus and goals can change with evolving business goals.

Disadvantages of Scrum

While a rugby scrum may get rough and bloody, software developers shouldn't have to worry about that. Nonetheless, scrum is not for all developer teams or software development projects. There are disadvantages to implementing scrum projects:

- There is a danger of scope creep if stakeholders keep adding functionality to the backlog. This could be encouraged by the fixed deadline.
- Scrum works best with small teams of experienced software developers. They need to be able to work quickly.
- Scrum teams do not work well when the scrum master micromanages their work.
- Losing any team members can hurt the progress of the project.

4 **Agile Vs. Scrum**

Agile	Scrum
Agile is a development methodology based on **iterative and incremental approach**.	Scrum is one of the implementations of agile methodology. In which **incremental builds** are delivered to the customer in every two to three weeks' time.
Agile software development has been widely seen as highly suited to environments which have **small but expert project** development team	Scrum is ideally used in the project where the requirement is **rapidly** changing.
In the Agile process, the **leadership** plays a vital role.	Scrum fosters a **self-organizing**, cross-functional team.
Compared to Scrum it is a more **rigid** method. So there is not much room for frequent changes.	The biggest advantage of Scrum is its **flexibility** as it quickly reacts to changes.
Agile involves **collaborations** and face-to-face interactions between the members of various cross-functional teams.	In Scrum, collaboration is achieved in **daily stand up meeting** with a fixed role assigned to scrum master, product owner, and team members.

Agile can require l**ots of up-front development process** and organizational change.	Not **too many changes** needed while implementing scrum process.
The agile method **needs frequent delivery** to the end user for their feedback.	In the scrum, after each sprint, a **build is delivered** to the client for their feedback.
In this method, each step of development like requirements, analysis, design, are **continually monitored** during the lifecycle.	A demonstration of the functionality is provided at the end of every sprint. So that **regular feedback** can be taken before next sprint.
Project head takes cares of all the tasks in the agile method.	There is no team leader, so the **entire team addresses the issues** or problems.
The Agile method **encourages feedback** during the process from the end user. In this way, the end product will be more useful.	**Daily sprint meeting** is conducted to review and feedback to decide future progress of the project.
Deliver and update the software on a regular basis.	When the team is done with the **current sprint activities**, the next sprint can be planned.
Design and execution should be kept **simple**.	Design and execution can be **innovative and experimental**.
In the Agile method, the priority is always to satisfy the customer by providing **continuous delivery** of valuable software.	**Empirical Process Control** is a core philosophy of Scrum based process.
Working software is the most **elementary measure** of progress.	Working software is **not an elementary measure**.
It is best to have **face-to-face communication**, and techniques like these should be used to get as close to this goal as possible.	Scrum team focus to deliver **maximum business value**, from beginning early in the project and continuing throughout.
Following are Agile principles: -Welcome changing requirements, even late in development. Agile processes allow change according to **customer's competitive advantage**. -Business people and developers will **work daily** throughout the project. -Attention to **technical excellence** and right design enhances agility -Agile team, work on to become more effective, for that they **adjust its behavior** according to the project.	Following are scrum principles: **-Self-organization:** This results in healthier shared ownership among the team members. It is also an innovative and creative environment which is conducive to growth. **-Collaboration:** Collaboration is another essential principle which focuses collaborative work. 1. Awareness 2. Articulation and 3. Appropriation. It also considers project management as a shared value-creation process with teams working together to offer the highest value. **-Time-boxing:** This principle defines how time is a limiting constraint in Scrum method. An important element of time-boxed elements are Daily Sprint planning

	and Review Meetings. **-Iterative Development:** This principle emphasizes how to manage changes better and build products which satisfy customer needs. It also defines the organization's responsibilities regarding iterative development.

Read attached Scrum Guide for further details

2016-Scrum-Guide-US.pdf

5 DATA FLOW DIAGRAM

A **data-flow diagram** (DFD) is a way of representing a **flow** of a **data** of a process or a system (usually an information system). The DFD also provides information about the outputs and inputs of each entity and the process itself. A **data-flow diagram** has no control **flow**, there are no decision rules and no loops.

6 CONTEXT DIAGRAM

A **context diagram**, sometimes called a level 0 data-flow **diagram**, is drawn in order to define and clarify the boundaries of the software system. It identifies the flows of information between the system and external entities. The entire software system is shown as a single process.

7 ACTIVITY DIAGRAM

An **activity diagram** is a behavioral **diagram** i.e. it depicts the behavior of a system. An **activity diagram** portrays the control flow from a start point to a finish point showing the various decision paths that exist while the **activity** is being executed.

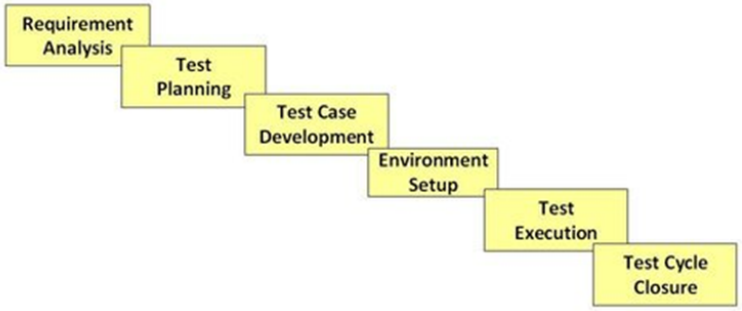
8 Software Testing Life Cycle (STLC)

8.1 What is STLC?

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

Contrary to popular belief, Software Testing is not a just a single activity. It consists of a series of activities carried out methodologically to help certify your software product.

8.2 Phases of STLC



Below are the phases of STLC:

- Requirement Analysis
- Test Planning
- Test case development
- Test Environment setup
- Test Execution
- Test Cycle closure

1. **Requirement Analysis**

During this phase, test team studies the requirements from a testing point of view to identify the testable requirements.

The QA team may interact with various stakeholders (Client, Business Analyst, Technical Leads, and System Architects etc.) to understand the requirements in detail.

Requirements could be either Functional (defining what the software must do) or Non Functional (defining system performance /security / availability)

Activities

- 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).

Deliverables

- RTM
- Automation feasibility report. (if applicable)

2. Test Planning

Typically, in this stage, a Senior QA manager will determine effort and cost estimates for the project and would prepare and finalize the Test Plan. In this phase, Test Strategy is also determined.

Activities

- Preparation of test plan/strategy document for various types of testing
- Test tool selection
- Test effort estimation
- Resource planning and determining roles and responsibilities.
- Training requirement

Deliverables

- Test plan /strategy document.

- Effort estimation document.

3. Test Case Development

This phase involves the creation, verification and rework of test cases & test scripts. Test data, is identified/created and is reviewed and then reworked as well.

Activities

- Create test cases, automation scripts (if applicable)
- Review and baseline test cases and scripts
- Create test data (If Test Environment is available)

Deliverables

- Test cases/scripts
- Test data

4. Test Environment Setup

Test environment decides the software and hardware conditions under which a work product is tested. Test environment set-up is one of the critical aspects of testing process and can be done in parallel with Test Case Development Stage. Test team may not be involved in this activity if the customer/development team provides the test environment in which case the test team is required to do a readiness check (smoke testing) of the given environment.

Activities

- Understand the required architecture, environment set-up and prepare hardware and software requirement list for the Test Environment.
- Setup test Environment and test data
- Perform smoke test on the build

Deliverables

- Environment ready with test data set up
- Smoke Test Results.

5. Test Execution

During this phase, the testers will carry out the testing based on the test plans and the test cases prepared. Bugs will be reported back to the development team for correction and retesting will be performed.

Activities

- Execute tests as per plan
- Document test results, and log defects for failed cases
- Map defects to test cases in RTM
- Retest the Defect fixes
- Track the defects to closure

Deliverables

- Completed RTM with the execution status
- Test cases updated with results
- Defect reports

6. Test Cycle Closure

Testing team will meet, discuss and analyze testing artifacts to identify strategies that have to be implemented in the future, taking lessons from the current test cycle. The idea is to remove the process bottlenecks for future test cycles and share best practices for any similar projects in the future.

Activities

- Evaluate cycle completion criteria based on Time, Test coverage, Cost, Software, Critical Business Objectives, Quality
- Prepare test metrics based on the above parameters.
- Document the learning out of the project
- Prepare Test closure report
- Qualitative and quantitative reporting of quality of the work product to the customer.
- Test result analysis to find out the defect distribution by type and severity.

Deliverables

- Test Closure report
- Test metrics

8.3 Difference between Test Plan and Test Strategy

Test strategy is a high level document which defines the approach for software testing. It is basically derived from the Business Requirement document. Test strategy is developed by project manager or business analyst. It is kind of static document which sets the standards for testing so not updated often.

Test plan is derived from SRS (*Software Requirement Specification*) which is prepared by test lead or manager. The main goal of test plan is to include all the details related to testing such as what to test, when to test, how to test and who will be the tester. Test plan is often not updated but if there is some new feature or change is introduced then it has to be updated accordingly.

Now, let's make a list of points which are included in both respectively.

Test strategy contains:

Scope and objective: The objective of the business and how much testing scope is there is defined under test strategy.

Business Issues: How much is the budget of the project, how much time is required for testing, how much resources are needed etc. are the part of business issues which needs to be considered before the actual testing starts.

Testing approach: What type of testing is needed (performance, load, stress, functional etc.) and whether the testing is only manual or automation or both are some of the crucial points which defines the testing approach.

Test deliverables: What are the documents required from the testing team, how they would keep the record of the testing cycles etc. will be included here.

Defect tracking approach: Which tool will be used for tracking the defects and how will the testing team communicate with the development team and how the flow would go for defects are decided at this point in test strategy.

Training: If there is some complex or new tool is introduced in the business then it is helpful if the team members are given proper training. What type of training and the responsible person to conduct such training is defined here.

Automation: If the project or business needs automation testing then the script language, tool used, reporting and code maintained is planned in test strategy.

Risks: Nobody can anticipate all the risks beforehand but obvious risks can be avoided and also solution (if risk occur) can be included in the document for future help.

Test plan contains:

Test plan ID: This is a unique ID which defines the test plan. It can be a number or name or mix of both, as per the convenience.

Test environment: This section defines what kind of environment is needed for the testing to carry out. For e.g. in device testing, usually a virtual set up is made to test emergency calling.

Features to be tested/Not tested: This will have all the details about the features which tester needs to test and what are the feature which are not tested (may be because it is not yet implemented or not tested for that particular release).

Entry/Exit criteria: These are the terms which define when to start or stop the testing. Standards will be defined under test strategy and followed by testers in test plan.

Status: Whether a test case is passed or failed or not tested, all these test results are included in

test plan with a proper reason.

Types of testing: The types of testing required such as regression, functional, non-functional, stress etc. are defined and then executed by the respective tester.

Brief Intro: Brief introduction is also included sometimes so that if any new member joins the team, he should get an idea how things work.

Basically, test plan is test strategy and test logistics (tools used, environment set up etc.). Strategy defines what approach should be there for testing and plan has all the details how those approach will be executed in a proper planned way. They both go hand in hand. Test plan will have all the names of the testers who tested a particular script and also it maintains cycle numbers so that if some feature fails in this cycle, previous cycle can be referred to see if that particular feature was passed or failed, in this way it is easy to get the root of the issue and hence becomes easy to resolve it.

As mentioned above. Test strategy should not be modified very often because it sets some standards for test plan and if standards are modified frequently then it becomes difficult to stick to a particular plan and changing plan frequently will affect the quality of the testing. Sometimes when small requirements are changed, we only need to update the test plan but test strategy remains the same.

8.4 Reviews

Reviews are the form of static testing. In software reviews people analyze the work product of projects such as requirements document, design document, test strategy, test plan in order to find out any defects in the documents.

Software Reviews, if done properly are the biggest and most cost effective contributor to product quality.

Review provides a powerful way to improve the quality and productivity of software development to recognize and fix their own defects early in the software development process.

Advantages of Reviews:-

1. Types of defects that can be found during static testing are: deviations from standards, missing requirements, design defects, non-maintainable code and inconsistent interface specifications.

2. Since static testing can start early in the life cycle, early feedback on quality issues can be established, e.g. an early validation of user requirements and not just late in the life cycle during acceptance testing.

3. By detecting defects at an early stage, rework costs are relatively low and thus a relatively cheap improvement of the quality of software products can be achieved.

4. The feedback and suggestions document from the static testing process allows for process improvement, which supports the avoidance of similar errors being made in the future.

Roles and Responsibilities in a Review

There are various roles and responsibilities defined for a review process. Within a review team, four types of participants can be distinguished: moderator, author, recorder, reviewer and manager. Let's discuss their roles one by one:-

1. The moderator: - The moderator (or review leader) leads the review process. His role is to determine the type of review, approach and the composition of the review team. The moderator also schedules the meeting, disseminates documents before the meeting, coaches other team members, paces the meeting, leads possible discussions and stores the data that is collected.

2. The author: - As the writer of the 'document under review', the author's basic goal should be to learn as much as possible with regard to improving the quality of the document. The author's task is to illuminate unclear areas and to understand the defects found.

3. The recorder: – The scribe (or recorder) has to record each defect found and any suggestions or feedback given in the meeting for process improvement.

4. The reviewer: - The role of the reviewers is to check defects and further improvements in accordance to the business specifications, standards and domain knowledge.

5. The manager :- Manager is involved in the reviews as he or she decides on the execution of reviews, allocates time in project schedules and determines whether review process objectives have been met or not.

Phases of a formal Review (Phases of Inspection)

A formal review takes place in a piecemeal approach which consists of 6 main steps. Let's discuss

about these phases one by one.

1. Planning

The review process for a particular review begins with a ‘request for review’ by the author to the moderator (or inspection leader). A moderator is often assigned to take care of the scheduling (dates, time, place and invitation) of the review. The project planning needs to allow time for review and rework activities, thus providing engineers with time to thoroughly participate in reviews. There is an entry check performed on the documents and it is decided that which documents are to be considered or not. The document size, pages to be checked, composition of review team, roles of each participant, strategic approach are decided into planning phase.

2. Kick-Off

The goal of this meeting is to get everybody on the same page regarding the document under review. Also the result of the entry and exit criteria is discussed. Basically, during the kick-off meeting, the reviewers receive a short introduction on the objectives of the review and the documents. Role assignments, checking rate, the pages to be checked, process changes and possible other questions are also discussed during this meeting. Also, the distribution of the document under review, source documents and other related documentation, can also be done during the kick-off.

3. Preparation

In this phase, participants work individually on the document under review using the related documents, procedures, rules and checklists provided. The individual participants identify defects, questions and comments, according to their understanding of the document and role. Spelling mistakes are recorded on the document under review but not mentioned during the meeting. The annotated document will be given to the author at the end of the logging meeting. Using checklists during this phase can make reviews more effective and efficient.

4. Review Meeting

This meeting typically consists of the following elements:-
-logging phase
-discussion phase
-decision phase.

During the logging phase the issues, e.g. defects, that have been identified during the preparation are mentioned page by page, reviewer by reviewer and are logged either by the author or by a scribe. This phase is for just jot down all the issues not to discuss them in detail. If an issue needs discussion, the item is logged and then handled in the discussion phase. A detailed discussion on whether or not an issue is a defect is not very meaningful, as it is much more efficient to simply log it and proceed to the next one.

The issues classified as discussion items will be handled during discussion phase. Participants can take part in the discussion by bringing forward their comments and reasoning. The moderator also paces this part of the meeting and ensures that all discussed items either have an outcome by the end of the meeting, or are defined as an action point if a discussion cannot be solved during the meeting. The outcome of discussions is documented for future reference.

At the end of the meeting, a decision on the document under review has to be made by the participants, sometimes based on formal exit criteria. The most important exit criterion is the average number of critical and major defects found per page. If the number of defects found per page exceeds a certain level, the document must be reviewed again, after it has been reworked. If the document complies with the exit criteria, the document will be checked during follow-up by the moderator or one or more participants. Subsequently, the document can leave or exit the review process.

5. Rework

Based on the defects detected and improvements suggested in the review meeting, the author improves the document under review. In this phase the author would be doing all the rework to ensure that defects detected should fixed and corrections should be properly implied. Changes that are made to the document should be easy to identify during follow-up, therefore the author has to indicate where changes are made.

6. Follow-Up

After the rework, the moderator should ensure that satisfactory actions have been taken on all logged defects, improvement suggestions and change requests. If it is decided that all participants will check the updated document, the moderator takes care of the distribution and

collects the feedback. In order to control and optimize the review process, a number of measurements are collected by the moderator at each step of the process. Examples of such measurements include number of defects found, number of defects found per page, time spent checking per page, total review effort, etc. It is the responsibility of the moderator to ensure that the information is correct and stored for future analysis.

8.4.1 Types of Review

1. Walkthrough (Informal Review)

A walkthrough is conducted by the author of the 'document under review' who takes the participants through the document and his or her thought processes, to achieve a common understanding and to gather feedback. This is especially useful if people from outside the software discipline are present, who are not used to, or cannot easily understand software development documents. The content of the document is explained step by step by the author, to reach consensus on changes or to gather information. The participants are selected from different departments and backgrounds if the audience represents a broad section of skills and disciplines, it can give assurance that no major defects are 'missed' in the walk-through. A walkthrough is especially useful for higher-level documents, such as requirement specifications and architectural documents.

The specific goals of a walkthrough are:-

- To present the document to stakeholders both within and outside the software discipline, in order to gather information regarding the topic under documentation.
- To explain and evaluate the contents of the document.
- To establish a common understanding of the document.
- To examine and discuss the validity of proposed solutions and the possible alternatives.

2. Technical review

A technical review is a discussion meeting that focuses on technical content of a document. It is led by a trained moderator, but also can be led by a technical expert. Compared to inspections, technical reviews are less formal and there is little or no focus on defect identification on the basis of referenced documents. The experts that are needed to be present for a technical review can be architects, chief designers and key users. It is often performed as a peer review without management participation.

The specific goals of a technical review are:

- Evaluate the value of technical concepts and alternatives in the product and project environment.
- Establish consistency in the use and representation of technical concepts.
- Ensuring at an early stage, that technical concepts are used correctly.
- Inform participants of the technical content of the document.

3. Inspection (Formal Review)

Inspection is the most formal review type. It is usually led by a trained moderator (certainly not by the author).The document under inspection is prepared and checked thoroughly by the reviewers before the meeting, comparing the work product with its sources and other referenced documents, and using rules and checklists. In the inspection meeting the defects found are logged. Depending on the organization and the objectives of a project, inspections can be balanced to serve a number of goals.

The specific goals of an Inspection are:

- Help the author to improve the quality of the document under inspection.
- Remove defects efficiently, as early as possible.
- Improve product quality, by producing documents with a higher level of quality.
- Create a common understanding by exchanging information among the inspection participants.

9 Introduction to Automation Testing

Manual Testing is performed by a human sitting in front of a computer carefully executing the test

steps.

Automation Testing 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 improved ROI of Test Automation. The goal of Automation is to reduce the number of test cases to be run manually and not to eliminate Manual testing altogether.

Benefits of automated testing

Following are benefits of automated testing:

- Increase productivity
- Saves money
- Increases software quality

9.1 Automation Testing

- Reduces testing time
- Support various applications
- Increase testing coverage
- Reduction of repetitive work
- Greater consistency
- Reliable in results
- Improves accuracy
- Test Frequently and thoroughly
- Early time to market

Why Automation Testing is important?

Suppose any software has come up with new releases and bug fixes, then how will you ensure about that the new released software with bug fixes has not introduced any new bug in previous working functionality? So it's better to test the software with old functionalities too. It is difficult to test manually all functionalities of the software every time with the addition of some bug fixes or new functionalities. So, it is better to test software every time by Automation testing technique using Automation Tool efficiently and effectively. It is effective in terms of cost, resources, Time etc.

Do automation testing at the time of lots of regression work– A web application where thousands of users access the application simultaneously. It is always difficult to think that how will you test such application and how to create those many users manually and simultaneously. So, it is better to go for Automation testing.

Automate your testing work when GUI is same but you have lot of often functional changes. That means test that application where codes changes / function changes frequently in particular GUI. More functional changes increase the testing work.

When to automate an application:

Do the Automation testing in the following scenario of the Software,

- Requirements do not change frequently
- Access the application for load and performance with many virtual users
- Steady Software with respect to manual testing
- Obtainability of time

- Huge and serious projects
- Projects that need to test the same areas often

Simple Steps to follow in Automation testing:

There are lots of helpful tools to write automation scripts, before using those tools it's better to identify the process which can be used to automate the testing,

- Identify areas within software to automate
- Choose the appropriate tool for test automation
- Write test scripts
- Develop test suits
- Execute test scripts
- Build result reports
- Find possible bugs or performance issue

Different Software testing tools Available in Automation Testing:

These are some tools which can be helpful in Automation testing,

- HP Quick Test Professional
- Selenium
- Visual Studio Test Professional
- WATIR
- IBM Rational Functional Tester
- TestComplete
- Testing Anywhere
- WinRunner
- LaodRunner
- SilkTest

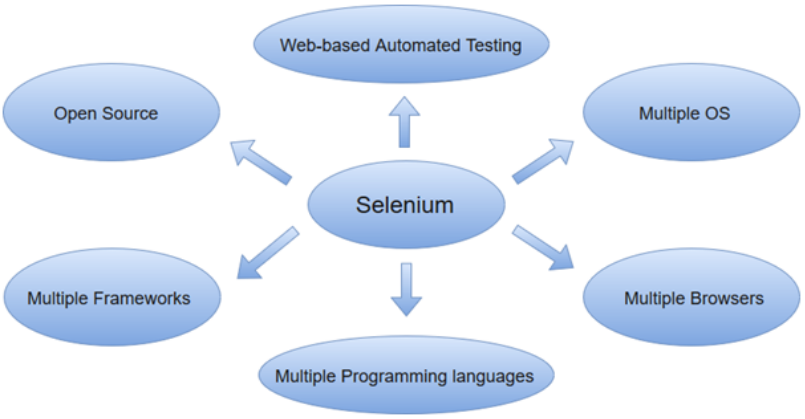
9.2 Introduction to Selenium

9.2.1 What is Selenium

Selenium is one of the most widely used open source Web UI (User Interface) automation testing suite. It was originally developed by Jason Huggins in 2004 as an internal tool at Thought Works. Selenium supports automation across different browsers, platforms and programming languages.

Selenium can be easily deployed on platforms such as Windows, Linux, Solaris and Macintosh. Moreover, it supports OS (Operating System) for mobile applications like IOS, windows mobile and android.

Selenium supports a variety of programming languages through the use of drivers specific to each language. Languages supported by Selenium include C#, Java, Perl, PHP, Python and Ruby. Currently, Selenium Web driver is most popular with Java and C#. Selenium test scripts can be coded in any of the supported programming languages and can be run directly in most modern web browsers. Browsers supported by Selenium include Internet Explorer, Mozilla Firefox, Google Chrome and Safari.



Selenium can be used to automate functional tests and can be integrated with automation test tools such as Maven, Jenkins, & Docker to achieve continuous testing. It can also be integrated with tools such as TestNG, & JUnit for managing test cases and generating reports.

Advantages of Selenium

- Selenium is pure open source, freeware and portable tool.
- Selenium supports variety of languages that include Java, Perl, Python, C#, Ruby, Groovy, Java Script, and VB Script. etc.
- Selenium supports many operating systems like Windows, Macintosh, Linux, Unix etc.
- Selenium supports many browsers like Internet explorer, Chrome, Firefox, Opera and Safari etc.
- Selenium can be integrated with TestNG testing framework for testing our applications and generating reports.
- Selenium supports very less CPU and RAM consumption for script execution.

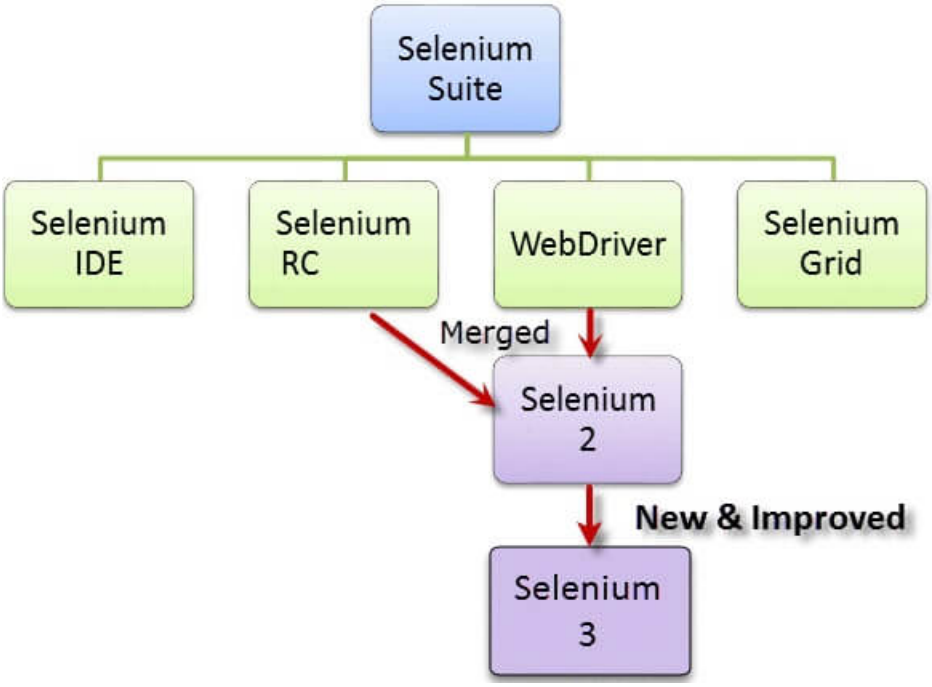
Disadvantages of Selenium

- Selenium needs good technical expertise. The resource should have good programming skills.
- Selenium only supports web based application and does not support windows based application.
- It is difficult to test Image based application.
- Selenium need outside support for report generation activity like TestNG or Jenkins.
- Selenium does not provide any built in IDE for script generation and it need other IDE like Eclipse for writing scripts.
- Selenium script creation time is bit high.

9.2.2 Selenium Components

Selenium is not just a single tool but a suite of software, each catering to different testing needs of an organization. It has four components.

- Selenium Integrated Development Environment (IDE)
- Selenium Remote Control (RC)
- WebDriver
- Selenium Grid



Selenium IDE

Selenium IDE (Integrated Development Environment) is an open source web automation testing tool under the Selenium Suite. It is a Firefox plugin that you can install as easily as you can with other plugins. Unlike Selenium WebDriver and RC, it does not require any programming logic to write its test scripts rather you can simply record your interactions with the browser to create test cases. Subsequently, you can use the playback option to re-run the test cases.

Selenium WebDriver

Selenium WebDriver is the most important component of Selenium Tool's Suite. In WebDriver, test scripts can be developed using any of the supported programming languages and can be run directly in most modern web browsers. Languages supported by WebDriver include C#, Java, Perl, PHP, Python and Ruby.

Before learning the concepts of Selenium WebDriver, you should be well versed with any of the supported programming languages. Currently, Selenium Web driver is most popular with Java and C#.

Selenium WebDriver is the most commonly used automation tool within Selenium tool suite, which automates the web application by speaking directly to the browser using the 'native' method for the browser and operating system.

Since Selenium WebDriver is the most popular and most used tool in the Selenium suite, it is referred simply as Selenium. So whenever someone mentions Selenium, it's most likely that they are talking about Selenium WebDriver.

Selenium Grid

Selenium Grid is a tool that allows you to run your test cases in parallel, that is, you can run different tests at the same time on different remote machines. This parallel execution would be especially helpful when you have a large number of test scripts to be executed.

Selenium RC

Selenium RC or Selenium Remote Control uses JavaScript based library to interact with web pages. While Selenium RC was a tremendous tool, it had its own drawbacks. Selenium RC is no longer in use now and is not actively supported by the Selenium Developers.

Selenium resources can be found at https://www.seleniumhq.org/

10 Basic Database Concepts

10.1 Relational Database Basics

SQL

SQL stands for Structured Query Language. It is the language used by relational database management systems (RDBMS) to access and manipulate data and to create structure and destroy databases and database objects.

Relational Databases

A relational database at its simplest is a set of tables used for storing data. Each table has a unique name and may relate to one or more other tables in the database through common values.

Tables

A table in a database is a collection of rows and columns. Tables are also known as entities or relations.

Rows

A row contains data pertaining to a single item or record in a table. Rows are also known as records or tuples.

Columns

A column contains data representing a specific characteristic of the records in the table. Columns are also known as fields or attributes.

Relationships

A relationship is a link between two tables (i.e. relations). Relationships make it possible to find data in one table that pertains to a specific record in another table.

Datatypes

Each of a table's columns has a defined datatype that specifies the type of data that can exist in that column. For example, the FirstName column might be defined as varchar (20), indicating that it can contain a string of up to 20 characters. Unfortunately, datatypes vary widely between databases.

Candidate Key

A Candidate key is an attribute or set of attributes that uniquely identifies a record. Among the set of candidate, one candidate key is chosen as Primary Key. So a table can have multiple candidate key but each table can have maximum one primary key.

Primary Keys

Most tables have a column or group of columns that can be used to identify records. For example, an Employees table might have a column called EmployeeID that is unique for every row. This makes it easy to keep track of a record over time and to associate a record with records in other tables.

Foreign Keys

Foreign key columns are columns that link to primary key columns in other tables, thereby creating a relationship. For example, the Customers table might have a foreign key column called SalesRep that links to EmployeeID, the primary key in the Employees table.

Alternate Key

Alternate keys are candidate keys that are not selected as primary key. Alternate key can also work as a primary key. Alternate key is also called "**Secondary Key**".

Unique Key

A unique key is a set of one or more attribute that can be used to uniquely identify the records in table. Unique key is similar to primary key but unique key field can contain a "**Null**" value but primary ke y doesn't allow "**Null**" value. Other difference is that primary key field contain a clustered index and unique field contain a non-clustered index.

Composite Key

Composite key is a combination of more than one attributes that can be used to uniquely identity each record. It is also known as "Compound" key. A composite key may be a candidate or primary key.

Super Key

Super key is a set of on e or more than one keys that can be used to uniquely identify the record in table. A Super key for an entity is a set of one or more attributes whose combined value uniquely identifies the entity in the entity set. A super key is a combine form of Primary Key, Alternate key and Unique key and Primary Key, Unique Key and Alternate Key are subset of super key.

Surrogate Key

Surrogate key is an artificial key that is used to uniquely identify the record in table. For example, in SQL Server or Sybase database system contain an artificial key that is known as "**Identity**". Surrogate keys are just simple sequential number. Surrogate keys are only used to act as a primary key.

Difference between primary key and unique constraints?

Primary key cannot have NULL value; the unique constraints can have NULL values. There is only one primary key in a table, but there can be multiple unique constrains.

Database Normalization

It is a process of analyzing the given relation schemas based on their functional dependencies and primary keys to achieve the following desirable properties:
1) Minimizing Redundancy
2) Minimizing the Insertion, Deletion, And Update Anomalies
Relation schemas that do not meet the properties are decomposed into smaller relation schemas that could meet desirable properties.

Employees' Skills

Employee ID	Employee Address	Skill
426	87 Sycamore Grove	Typing
426	87 Sycamore Grove	Shorthand
519	94 Chestnut Street	Public Speaking
519	96 Walnut Avenue	Carpentry

An **update anomaly**. Employee 519 is shown as having different addresses on different records.

Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201
424	Dr. Newsome	29-Mar-2007	?

An **insertion anomaly**. Until the new faculty member, Dr. Newsome, is assigned to teach at least one course, his or her details cannot be recorded.

Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201

A **deletion anomaly**. All information about Dr. Giddens is lost if he or she temporarily ceases to be assigned to any courses.

When an attempt is made to modify (update, insert into, or delete from) a relation, the following undesirable side-effects may arise in relations that have not been sufficiently normalized:

- **Update anomaly.** The same information can be expressed on multiple rows; therefore updates to the relation may result in logical inconsistencies. For example, each record in an "Employees' Skills" relation might contain an Employee ID, Employee Address, and Skill; thus a change of address for a particular employee may need to be applied to multiple records (one for each skill). If the update is only partially successful – the employee's address is updated on some records but not others – then the relation is left in an inconsistent state. Specifically, the relation provides conflicting answers to the question of what this particular employee's address is. This phenomenon is known as an update anomaly.
- **Insertion anomaly.** There are circumstances in which certain facts cannot be recorded at all. For example, each record in a "Faculty and Their Courses" relation might contain a Faculty ID, Faculty Name, Faculty Hire Date, and Course Code. Therefore, we can record the details of any faculty member who teaches at least one course, but we cannot record a newly hired faculty member who has not yet been assigned to teach any courses, except by setting the Course Code to null. This phenomenon is known as an insertion anomaly.
- **Deletion anomaly.** Under certain circumstances, deletion of data representing certain facts necessitates deletion of data representing completely different facts. The "Faculty and Their Courses" relation described in the previous example suffers from this type of anomaly, for if a faculty member temporarily ceases to be assigned to any courses, we must delete the last of the records on which that faculty member appears, effectively also deleting the faculty member, unless we set the Course Code to null. This phenomenon is known as a deletion anomaly.

Database Normalization
Normalization is a database design technique which organizes tables in a manner that reduces redundancy and dependency of data.

1st Normal Form
- Each table cell should contain a single value.
- Each record needs to be unique.

2nd Normal Form
- Be in 1NF
- Single Column Primary Key

3rd Normal Form
- Be in 2NF
- Has no transitive functional dependencies

4th Normal Form
- If no database table instance contains two or more, independent and multivalued data describing the relevant entity, then it is in 4th Normal Form.

5th Normal Form
- A table is in 5th Normal Form only if it is in 4NF and it cannot be decomposed into any number of smaller tables without loss of data.

View in SQL

A view is a virtual table based on the result-set of an SQL statement. We can create using create view syntax.

```
CREATE VIEW view_name AS
SELECT column_name(s)
FROM table_name
WHERE condition
```

1. Views can represent a subset of the data contained in a table; consequently, a view can limit the degree of exposure of the underlying tables to the outer world: a given user may have permission to query the view, while denied access to the rest of the base table.
2. Views can join and simplify multiple tables into a single virtual table
3. Views can act as aggregated tables, where the database engine aggregates data (sum, average etc.) and presents the calculated results as part of the data

Database Trigger
A Trigger is a code that associated with insert, update or delete operations. The code is executed

automatically whenever the associated query is executed on a table. Triggers can be useful to maintain integrity in database.

Stored Procedure
A stored procedure is like a function that contains a set of operations compiled together. It contains a set of operations that are commonly used in an application to do some common database tasks.

Database Indexes
A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and the use of more storage space to maintain the extra copy of data.

Relational Database Management System

A Relational Database Management System (RDBMS), commonly (but incorrectly) called a database, is software for creating, manipulating, and administering a database. For simplicity, we will often refer to RDBMSs as databases.

Characteristics of Database Management System

A database management system has following characteristics:

Data stored into Tables: Data is never directly stored into the database. Data is stored into tables, created inside the database. DBMS also allows to have relationships between tables which makes the data more meaningful and connected. You can easily understand what type of data is stored where by looking at all the tables created in a database.

Reduced Redundancy: In the modern world hard drives are very cheap, but earlier when hard drives were too expensive, unnecessary repetition of data in database was a big problem. But DBMS follows Normalization which divides the data in such a way that repetition is minimum.

Data Consistency: On Live data, i.e. data that is being continuously updated and added, maintaining the consistency of data can become a challenge. But DBMS handles it all by itself.

Support Multiple user and Concurrent Access: DBMS allows multiple users to work on it(update, insert, delete data) at the same time and still manages to maintain the data consistency.

Query Language: DBMS provides users with a simple Query language, using which data can be easily fetched, inserted, deleted and updated in a database.

Security: The DBMS also takes care of the security of data, protecting the data from un-authorized access. In a typical DBMS, we can create user accounts with different access permissions, using which we can easily secure our data by restricting user access.

DBMS supports **transactions**, which allows us to better handle and manage data integrity in real world applications where multi-threading is extensively used.

10.2 SQL Statements

Database Manipulation Language (DML)

DML statements are used to work with data in an existing database. The most common DML statements are:

- `SELECT`
- `INSERT`
- `UPDATE`
- `DELETE`

Database Definition Language (DDL)

DDL statements are used to structure objects in a database. The most common DDL statements are:

- `CREATE`
- `ALTER`
- `DROP`

Database Control Language (DCL)

DCL statements are used for database administration. The most common DCL statements are:

- `GRANT`
- `DENY (SQL Server Only)`

• REVOKE

The DELETE Statement is used to delete rows from a table.
To delete an employee with id 100 from the employee table, the sql delete query would be like,
DELETE FROM employee WHERE id = 100;
To delete all the rows from the employee table, the query would be like,
DELETE FROM employee;

The TRUNCATE command is used to delete all the rows from the table and free the space containing the table.
TRUNCATE TABLE table_name;

The SQL DROP command is used to remove an object from the database. If you drop a table, all the rows in the table is deleted and the table structure is removed from the database. Once a table is dropped we cannot get it back, so be careful while using DROP command. When a table is dropped all the references to the table will not be valid.

DROP TABLE table_name;
If a table is dropped, all the relationships with other tables will no longer be valid, the integrity constraints will be dropped, grant or access privileges on the table will also be dropped, if you want use the table again it has to be recreated with the integrity constraints, access privileges and the relationships with other tables should be established again. But, if a table is truncated, the table structure remains the same, therefore any of the above problems will not exist.

Writing my first query

Let's start by using the **surveys** table. Here we have data on every individual that was captured at the site, including when they were captured, what plot they were captured on, their species ID, sex and weight in grams.

Let's write an SQL query that selects only the year column from the surveys table. SQL queries can be written in the box located under the "Execute SQL" tab. Click 'Run SQL' to execute the query in the box.

```
SELECT year
FROM surveys;
```

We have capitalized the words SELECT and FROM because they are SQL keywords. SQL is case insensitive, but it helps for readability, and is good style.

If we want more information, we can just add a new column to the list of fields, right after `SELECT`:

```
SELECT year, month, day
FROM surveys;
```

Or we can select all of the columns in a table using the wildcard *

```
SELECT *
FROM surveys;
```

Unique values

If we want only the unique values so that we can quickly see what species have been sampled we use `DISTINCT`

```
SELECT DISTINCT species_id
FROM surveys;
```

If we select more than one column, then the distinct pairs of values are returned

```
SELECT DISTINCT year, species_id
FROM surveys;
```

Calculated values

We can also do calculations with the values in a query. For example, if we wanted to look at the mass of each individual on different dates, but we needed it in kg instead of g we would use

```
SELECT year, month, day, weight /1000.0
FROM surveys;
```

When we run the query, the expression `weight / 1000.0` is evaluated for each row and appended to that row, in a new column. Expressions can use any fields, any arithmetic operators (+, -, *, and /) and a variety of built-in functions. For example, we could round the values to make them easier to read.

```
SELECT plot_id, species_id, sex, weight, ROUND (weight / 1000.0, 2)
FROM surveys;
```

Challenge

Write a query that returns the year, month, day, species_id and weight in mg

SOLUTION

```
SELECT day, month, year, species_id, weight * 1000
FROM surveys;
```

Filtering

Databases can also filter data – selecting only the data meeting certain criteria. For example, let's say we only want data for the species *Dipodomys merriami*, which has a species code of DM. We need to add a `WHERE` clause to our query:

```
SELECT *
FROM surveys
WHERE species_id='DM';
```

We can do the same thing with numbers. Here, we only want the data since 2000:

```
SELECT * FROM surveys
WHERE year >= 2000;
```

We can use more sophisticated conditions by combining tests with `AND` and `OR`. For example, suppose we want the data on *Dipodomys merriami* starting in the year 2000:

```
SELECT *
FROM surveys
WHERE (year >= 2000) AND (species_id = 'DM');
```

Note that the parentheses are not needed, but again, they help with readability. They also ensure that the computer combines `AND` and `OR` in the way that we intend.

If we wanted to get data for any of the *Dipodomys* species, which have species codes `DM`, `DO`, and `DS`, we could combine the tests using OR:

```
SELECT *
FROM surveys
WHERE (species_id = 'DM') OR (species_id = 'DO') OR (species_id = 'DS');
```

Challenge

Write a query that returns the day, month, year, species_id, and weight (in kg) for individuals caught on Plot 1 that weigh more than 75 g

SOLUTION

```
SELECT day, month, year, species_id, weight / 1000.0
FROM surveys
WHERE plot_id = 1
AND weight > 75;
```

Building more complex queries

Now, lets combine the above queries to get data for the 3 *Dipodomys* species from the year 2000 on. This time, let's use `IN` as one way to make the query easier to understand. It is equivalent to saying `WHERE (species_id = 'DM') OR (species_id = 'DO') OR (species_id = 'DS')`, but reads more neatly:

```
SELECT *
FROM surveys
WHERE (year >= 2000) AND (species_id IN ('DM', 'DO', 'DS'));
```

We started with something simple, then added more clauses one by one, testing their effects as we went along. For complex queries, this is a good strategy, to make sure you are getting what you want. Sometimes it might help to take a subset of the data that you can easily see in a temporary database to practice your queries on before working on a larger or more complicated database.

When the queries become more complex, it can be useful to add comments. In SQL, comments are started by `--`, and end at the end of the line. For example, a commented version of the above query can be written as:

```
-- Get post 2000 data on Dipodomys' species
-- These are in the surveys table, and we are interested in all columns
SELECT * FROM surveys
-- Sampling year is in the column `year`, and we want to include 2000
WHERE (year >= 2000)
-- Dipodomys' species have the `species_id` DM, DO, and DS
AND (species_id IN ('DM', 'DO', 'DS'));
```

Although SQL queries often read like plain English, it is *always* useful to add comments; this is especially true of more complex queries.

Sorting

We can also sort the results of our queries by using `ORDER BY`. For simplicity, let's go back to the **species** table and alphabetize it by taxa.

First, let's look at what's in the **species** table. It's a table of the species_id and the full genus, species and taxa information for each species_id. Having this in a separate table is nice, because we didn't need to include all this information in our main **surveys** table.

```
SELECT *
FROM species;
```

Now let's order it by taxa.

```
SELECT *
FROM species
ORDER BY taxa ASC;
```

The keyword `ASC` tells us to order it in Ascending order. We could alternately use `DESC` to get descending order.

```
SELECT *
FROM species
ORDER BY taxa DESC;
```

`ASC` is the default.

We can also sort on several fields at once. To truly be alphabetical, we might want to order by genus then species.

```
SELECT *
FROM species
ORDER BY genus ASC, species ASC;
```

Challenge

Write a query that returns year, species_id, and weight in kg from the surveys table, sorted with the largest weights at the top.
SOLUTION

```
SELECT year, species_id, weight / 1000.0
FROM surveys ORDER BY weight DESC;
```

Order of execution

Another note for ordering. **We don't actually have to display a column to sort by it**. For example, let's say we want to order the birds by their species ID, but we only want to see genus and species.

```
SELECT genus, species
FROM species
WHERE taxa = 'Bird'
ORDER BY species_id ASC;
```

We can do this because **sorting occurs earlier in the computational pipeline** than field selection.

The computer is basically doing this:

1. Filtering rows according to `WHERE`
2. Sorting results according to `ORDER BY`
3. Displaying requested columns or expressions.

Clauses are written in a fixed order: `SELECT`, `FROM`, `WHERE`, then `ORDER BY`. It is possible to write a query as a single line, but for readability, we recommend to put each clause on its own line.

Challenge

Let's try to combine what we've learned so far in a single query. Using the surveys table write a query to display the three date fields, `species_id`, and weight in kilograms (rounded to two decimal places), for individuals captured in 1999, ordered alphabetically by the `species_id`. Write the query as a single line, then put each clause on its own line, and see how more legible the query becomes!
SOLUTION

```
SELECT year, month, day, species_id, ROUND(weight / 1000.0, 2)
FROM surveys
WHERE year = 1999
ORDER BY species_id;
```

GROUP BY:

The **GROUP BY clause** is a SQL command that is used to **group** rows that have the same values. The **GROUP BY clause** is used in the SELECT statement . Optionally it is used in conjunction with aggregate functions to produce summary reports from the database. That's what it does, summarizing data from the database.

Difference between HAVING and WHERE clauses

HAVING: is used to check conditions *after* the aggregation takes place.
WHERE: is used to check conditions *before* the aggregation takes place.
This code:

```
Select City, count(ContactAdd) as AddressCount from Address where State = 'MA' group by City having count(ContactAdd)>5
```

Gives you a table of cities in MA with more than 5 addresses and the number of addresses in each city.

10.3 SQL JOIN

SQL Join is used to fetch data from two or more tables, which is joined to appear as single set of data. It is used for combining column from two or more tables by using values common to both tables.

JOIN Keyword is used in SQL queries for joining two or more tables. A table can also join to itself, which is known as, Self Join.

Types of JOIN

Following are the types of JOIN that we can use in SQL:

• Cross

• Inner

• Outer

• Left

• Right

Cross JOIN or Cartesian product

This type of JOIN returns the Cartesian product of rows from the tables in Join. It will return a table which consists of records which combines each row from the first table with each row of the second table.

Cross JOIN Syntax is,

```
SELECT column-name-list
FROM
table-name1 CROSS JOIN table-name2
```

Example of Cross JOIN

Following is the class table,

ID	NAME
1	abhi
2	adam
4	alex

and the **class info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI

Cross JOIN query will be,

```
SELECT * FROM
class CROSS JOIN class_info;
```

The resultset table will look **like**,

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	1	DELHI
4	alex	1	DELHI
1	abhi	2	MUMBAI

2	adam	2	MUMBAI
4	alex	2	MUMBAI
1	abhi	3	CHENNAI
2	adam	3	CHENNAI
4	alex	3	CHENNAI

As you can see, this join returns the cross product of all the records present in both the tables.

INNER Join or EQUI Join

This is a simple JOIN in which the result is based on matched data as per the equality condition specified in the SQL query.

Inner Join Syntax is,

```
SELECT column-name-list FROM
table-name1 INNER JOIN table-name2
WHERE table-name1.column-name = table-name2.column-name;
```

Example of INNER JOIN

Consider a class table,

ID	NAME
1	abhi
2	adam
3	alex
4	anu

and the class_info table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI

Inner JOIN query will be,

```
SELECT * from class INNER JOIN class_info where class.id = class_info.id;
```

The result set table will look like,

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI

OUTER JOIN

Outer Join is based on both matched and unmatched data. Outer Joins subdivide further into,

1. Left Outer Join
2. Right Outer Join
3. Full Outer Join

LEFT Outer Join

The left outer join returns a resultset table with the matched data from the two tables and then the remaining rows of the left table and null from the right table's columns.

Syntax for Left Outer Join is,

```
SELECT column-name-list FROM
table-name1 LEFT OUTER JOIN table-name2
ON table-name1.column-name = table-name2.column-name;
```

To specify a condition, we use the ON keyword with Outer Join.

Left outer Join Syntax for Oracle is,

```
SELECT column-name-list FROM
table-name1, table-name2 on table-name1.column-name = table-name2.column-name(+);
```

Example of Left Outer Join

Here is the class table,

ID	NAME
1	abhi
2	adam
3	alex
4	anu
5	ashish

and the class_info table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI
7	NOIDA
8	PANIPAT

Left Outer Join query will be,

```
SELECT * FROM class LEFT OUTER JOIN class_info ON (class.id = class_info.id);
```

The resultset table will look like,

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI
4	anu	null	null
5	ashish	null	null

RIGHT Outer Join

The right outer join returns a resultset table with the matched data from the two tables being joined, then the remaining rows of the right table and null for the remaining left table's columns.

Syntax for Right Outer Join is,

```
SELECT column-name-list FROM
table-name1 RIGHT OUTER JOIN table-name2
ON table-name1.column-name = table-name2.column-name;
```

Right outer Join Syntax for Oracle is,

```
SELECT column-name-list FROM
table-name1, table-name2
ON table-name1.column-name(+) = table-name2.column-name;
```

Example of Right Outer Join

Once again the class table,

ID	NAME
1	abhi
2	adam
3	alex
4	anu
5	ashish

and the class_info table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI
7	NOIDA
8	PANIPAT

Right Outer Join query will be,

```
SELECT * FROM class RIGHT OUTER JOIN class_info ON (class.id = class_info.id);
```

The resultant table will look like,

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI
null	null	7	NOIDA
null	null	8	PANIPAT

Full Outer Join

The full outer join returns a resultset table with the matched data of two table then remaining rows of both left table and then the right table.

Syntax of Full Outer Join is,

```
SELECT column-name-list FROM
table-name1 FULL OUTER JOIN table-name2
ON table-name1.column-name = table-name2.column-name;
```

Example of Full outer join is,

The class table,

ID	NAME
1	abhi
2	adam
3	alex
4	anu
5	ashish

and the class_info table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI
7	NOIDA
8	PANIPAT

Full Outer Join query will be like,

```
SELECT * FROM class FULL OUTER JOIN class_info ON (class.id = class_info.id);
```

The resultset table will look like,

ID	NAME	ID	Address
1	Abhi	1	DELHI
2	Adam	2	MUMBAI
3	Alex	3	CHENNAI
4	Anu	null	null
5	Ashish	null	null
null	Null	7	NOIDA
null	Null	8	PANIPAT

11 Basic Programming Concepts

11.1 OOPs concepts in Java

Object-oriented programming System (OOPs) is a programming paradigm based on the concept of “objects” that contain data and methods. The primary purpose of object-oriented programming is to increase the flexibility and maintainability of programs. Object oriented programming brings together data and its behavior (methods) in a single location (object) makes it easier to understand how a program works.

11.1.1 What is an Object

Object: is a bundle of data and its behavior (often known as methods).

Objects have two characteristics: They have **states** and **behaviors**.

Examples of states and behaviors
Example 1:
Object: House
State: Address, Color, Area
Behavior: Open door, close door

So if I had to write a class based on states and behaviors of House. I can do it like this: States can be represented as instance variables and behaviors as methods of the class. We will see how to create classes in the next section of this guide.

```
class House {
  String address;
  String color;
  double area;
  void openDoor() {
    //Write code here
  }
  void closeDoor() {
    //Write code here
  }


}
```

Example 2:
Let's take another example.
Object: Car
State: Color, Brand, Weight, Model
Behavior: Break, Accelerate, Slow Down, and Gear change.

Note: As we have seen above, the states and behaviors of an object, can be represented by variables and methods in the class respectively.

Characteristics of Objects:

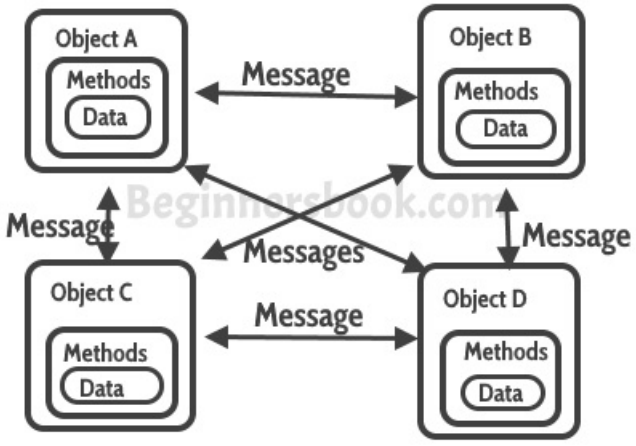
If you find it hard to understand Abstraction and Encapsulation, do not worry as I have covered these topics in detail with examples in the next section of this guide.

1. Abstraction
2. Encapsulation
3. Message passing

Abstraction: Abstraction is a process where you show only "relevant" data and "hide" unnecessary details of an object from the user.

Encapsulation: Encapsulation simply means binding object state(fields) and behaviour(methods) together. If you are creating class, you are doing encapsulation.

Message passing
A single object by itself may not be very useful. An application contains many objects. One object interacts with another object by invoking methods on that object. It is also referred to as **Method Invocation**. See the diagram below.



11.1.2 What is a Class in OOPs Concepts

A class can be considered as a blueprint using which you can create as many objects as you like. For example, here we have a class Website that has two data members (also known as fields, instance variables and object states). This is just a blueprint, it does not represent any website, however using this we can create Website objects (or instances) that represents the websites. We have created two objects, while creating objects we provided separate properties to the objects using constructor.

```
public class Website {
  //fields (or instance variable)
  String webName;
  int webAge;


  // constructor
  Website(String name, int age){
    this.webName = name;
    this.webAge = age;
  }
  public static void main(String args[]){
    //Creating objects
    Website obj1 = new Website("beginners book", 5);
    Website obj2 = new Website("google", 18);


    //Accessing object data through reference
    System.out.println (obj1.webName+" "+obj1.webAge);
    System.out.println (obj2.webName+" "+obj2.webAge);
  }
}
```

Output:

```
beginnersbook 5
google 18
```

What is a Constructor?

Constructor looks like a method but it is in fact not a method. It's name is same as class name and it does not return any value. You must have seen this statement in almost all the programs I have shared above:

```
MyClass obj = new MyClass();
```

If you look at the right side of this statement, we are calling the default constructor of class myClass to create a new object (or instance).

We can also have parameters in the constructor, such constructors are known as **parameterized constructors**.

Example of constructor

```
public class ConstructorExample {


  int age;
  String name;
```

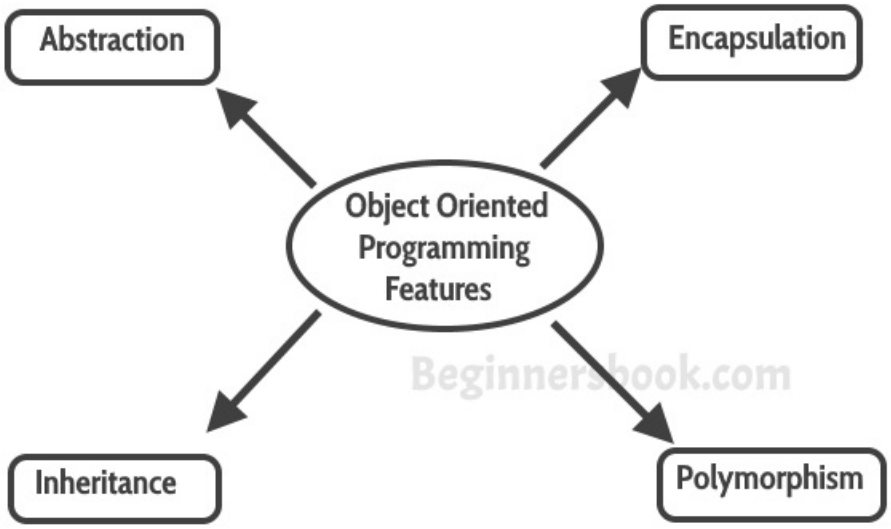
```
    //Default constructor
    ConstructorExample(){
            this.name="Chaitanya";
            this.age=30;
    }

    //Parameterized constructor
    ConstructorExample(String n,int a){
            this.name=n;
            this.age=a;
    }
    public static void main(String args[]){
            ConstructorExample obj1 = new ConstructorExample();
            ConstructorExample obj2 =
                            new ConstructorExample("Steve", 56);
            System.out.println(obj1.name+" "+obj1.age);
            System.out.println(obj2.name+" "+obj2.age);
    }
}
```

Output:

```
Chaitanya 30
Steve 56
```

11.1.3 Object Oriented Programming features



These four features are the main OOPs Concepts that you must learn to understand the Object Oriented Programming in Java

Abstraction

Abstraction is the concept of hiding the internal details and describing things in simple terms. There are many ways to achieve abstraction in object oriented programming, such as encapsulation and inheritance.

Abstraction is a process where you show only “relevant” data and “hide” unnecessary details of an object from the user. For example, when you login to your bank account online, you enter your user_id and password and press login, what happens when you press login, how the input data sent to server, how it gets verified is all abstracted away from the you.

Encapsulation

Encapsulation is the technique used to implement abstraction in object oriented programming.

Encapsulation is used for access restriction to a class members and methods.

Access modifier keywords are used for encapsulation in object oriented programming. For example, encapsulation in java is achieved using private, protected and public keywords.

private hides from other classes within the package. public exposes to classes outside the package. protected is a version of public restricted only to subclasses
in Java protected makes the method also accessible from the whole package.

Encapsulation simply means binding object state(fields) and behavior(methods) together. If you are creating class, you are doing encapsulation.

Encapsulation example in Java

How to
1) Make the instance variables private so that they cannot be accessed directly from outside the class. You can only set and get values of these variables through the methods of the class.
2) Have getter and setter methods in the class to set and get the values of the fields.

```
class EmployeeCount
{
  private int numOfEmployees = 0;
  public void setNoOfEmployees (int count)
  {
     numOfEmployees = count;
  }
  public double getNoOfEmployees ()
  {
     return numOfEmployees;
  }
}
public class EncapsulationExample
{
  public static void main(String args[])
  {
    EmployeeCount obj = new EmployeeCount ();
    obj.setNoOfEmployees(5613);
    System.out.println("No Of Employees: "+(int)obj.getNoOfEmployees());
   }
}
```

Output:

```
No Of Employees: 5613
```

The class EncapsulationExample that is using the Object of class EmployeeCount will not able to get the NoOfEmployees directly. It has to use the setter and getter methods of the same class to set and get the value.

So what is the benefit of encapsulation in java programming

Well, at some point of time, if you want to change the implementation details of the class EmployeeCount, you can freely do so without affecting the classes that are using it.

Inheritance

The process by which one class acquires the properties and functionalities of another class is called **inheritance.** Inheritance provides the idea of reusability of code and each sub class defines only those features that are unique to it, rest of the features can be inherited from the parent class.

1. Inheritance is a process of defining a new class based on an existing class by extending its common data members and methods.
2. Inheritance allows us to reuse of code, it improves reusability in your java application.
3. The parent class is called the **base class** or **super class**. The child class that extends the base class is called the derived class or **sub class** or **child class**.

Note: The biggest advantage of Inheritance is that the code in base class need not be rewritten in the child class.
The **variables** and **methods** of the base class can be used in the **child class** as well.

Syntax: Inheritance in Java

To inherit a class we use extends keyword. Here class A is child class and class B is parent class.

```
class A extends B
{
}
```

Inheritance Example

In this example, we have a parent class `Teacher` and a child class `MathTeacher`. In the `MathTeacher` class we need not to write the same code which is already present in the present class. Here we have college name, designation and does() method that is common for all the teachers, thus MathTeacher class does not need to write this code, the common data members and methods can inherited from the `Teacher` class.

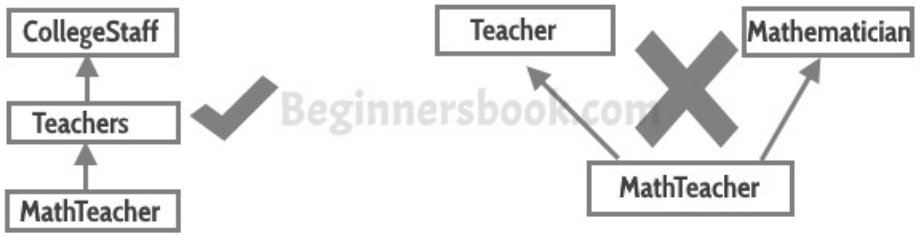
```
class Teacher {
  String designation = "Teacher";
  String college = "Beginnersbook";
  void does(){

            System.out.println("Teaching");
  }
}
public class MathTeacher extends Teacher{
  String mainSubject = "Maths";
  public static void main(String args[]){
    MathTeacher obj = new MathTeacher();
    System.out.println(obj.college);
    System.out.println(obj.designation);
    System.out.println(obj.mainSubject);
    obj.does();
  }
}
```

Output:

```
Beginnersbook
Teacher
Maths
Teaching
```

Note: Multi-level inheritance is allowed in Java but **not multiple inheritance**



Types of Inheritance:

Single Inheritance: refers to a child and parent class relationship where a class extends the another class.

Multilevel inheritance: refers to a child and parent class relationship where a class extends the child class. For example class A extends class B and class B extends class C.

Hierarchical inheritance: refers to a child and parent class relationship where more than one classes extends the same class. For example, class B extends class A and class C extends class A.

Multiple Inheritance: refers to the concept of one class extending more than one classes, which means a child class has two parent classes. Java doesn't support multiple inheritance, read more about it here.

Most of the new **OO languages** like Small Talk, Java, C# do not support Multiple inheritance. Multiple Inheritance is supported in C++.

Polymorphism

Polymorphism is the concept where an object behaves differently in different situations. In java, we use method overloading and method overriding to achieve polymorphism.
Polymorphism is a object oriented programming feature that allows us to perform a single action in different ways. For example, lets say we have a class `Animal` that has a

method `animalSound()`, here we cannot give implementation to this method as we do not know which Animal class would extend `Animal` class. So, we make this method abstract like this:

```
public abstract class Animal{
  ...
  public abstract void animalSound();
}
```

Now suppose we have two Animal classes `Dog` and `Lion` that extends `Animal` class. We can provide the implementation detail there.

```
public class Lion extends Animal{
...
   @Override
   public void animalSound(){
      System.out.println("Roar");
   }
}
```

and

```
public class Dog extends Animal{
...
   @Override
   public void animalSound(){
      System.out.println("Woof");
   }
}
```

As you can see that although we had the common action for all subclasses `animalSound ()` but there were different ways to do the same action. This is a perfect example of polymorphism (feature that allows us to perform a single action in different ways).

Types of Polymorphism

1) Static Polymorphism
2) Dynamic Polymorphism

Static Polymorphism:

Polymorphism that is resolved during compiler time is known as static polymorphism. Method overloading can be considered as static polymorphism example.
Method Overloading: This allows us to have more than one methods with same name in a class that differs in signature.

```
class DisplayOverloading
{
   public void disp( char c)
   {
       System.out.println(c);
   }
   public void disp(char c, int num)
   {
       System.out.println(c + " "+num);
   }
}
public class ExampleOverloading
{
  public static void main(String args[])
  {
     DisplayOverloading obj = new DisplayOverloading();
     obj.disp('a');
     obj.disp('a',10);
  }
}
```

Output:

```
a
a 10
```

When I say method signature I am not talking about return type of the method, for example if two methods have same name, same parameters and have different return type, then this is not a valid method overloading example. This will throw compilation error.

Dynamic Polymorphism

It is also known as Dynamic Method Dispatch. Dynamic polymorphism is a process in which a call to an overridden method (Method Overriding means method in the child class having same name and parameter as in parent class but different implementation) is resolved at runtime rather, thats why it is called runtime polymorphism.

Example

```
class Animal{
  public void animalSound(){
        System.out.println("Default Sound");
  }
}
public class Dog extends Animal{

  public void animalSound(){
        System.out.println("Woof");
  }
  public static void main(String args[]){
        Animal obj = new Dog();
        obj.animalSound();
  }
}
```

Output:

```
Woof
```

Since both the classes, child class and parent class have the same method `animalSound`. Which of the method will be called is determined at runtime by JVM.

Few more overriding examples:

```
Animal obj = new Animal();
obj.animalSound();
// This would call the Animal class method


Dog obj = new Dog();
obj.animalSound();
// This would call the Dog class method


Animal obj = new Dog();
obj.animalSound();
// This would call the Dog class method
```

Method Overloading vs Method Overriding

Overloading occurs when two or more methods in one class have the same method name but different parameters. **Overriding** means having two methods with the same method name and parameters (i.e., method signature). One of the methods is in the parent class and the other is in the child class.

Overriding

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }
}
class Hound extends Dog{
    public void sniff(){
        System.out.println("sniff ");
    }

    public void bark(){
        System.out.println("bowl");
    }
}
```

Same Method Name, Same parameter

Overloading

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }

    //overloading method
    public void bark(int num){
        for(int i=0; i<num; i++)
            System.out.println("woof ");
    }
}
```

Same Method Name, Different Parameter

OOPs interface vs abstract class

Interface	Abstract class
Interface support multiple implementations.	Abstract class does not support multiple inheritance.
Interface does not contain Data Member	Abstract class contains Data Member
Interface does not contain Constructors	Abstract class contains Constructors
An interface Contains only incomplete member (signature of member)	An abstract class Contains both incomplete (abstract) and complete member
An interface cannot have access modifiers by default everything is assumed as public	An abstract class can contain access modifiers for the subs, functions, properties
Member of interface can not be Static	Only Complete Member of abstract class can be Static

12 References

Following website can be visited for further details.

- https://www.softwaretestinghelp.com/
- https://www.softwaretestingmaterial.com/
- http://softwaretestingfundamentals.com
- http://www.professionalqa.com/
- https://www.testingexcellence.com/
- https://www.toolsqa.com/
- https://www.guru99.com/
- https://www.seleniumhq.org/
- http://www.softwaretestingclass.com/
- https://www.tutorialspoint.com/index.htm