Explain the SDLC with its phases and diagrams

Software Development Life Cycle (SDLC) is a framework that defines the steps involved in the development of software at each phase. It covers the detailed plan for building, deploying and maintaining the software.

SDLC defines the complete cycle of development i.e., all the tasks involved in planning, creating, testing, and deploying a Software Product. Every phase of the SDLC life Cycle has its own process and deliverables that feed into the next phase.

Software Development Life Cycle Process :

SDLC is a process that defines the various stages involved in the development of software for delivering a high-quality product. SDLC stages cover the complete life cycle of a software i.e. from inception to retirement of the product.

Adhering to the SDLC process leads to the development of the software in a systematic and disciplined manner.

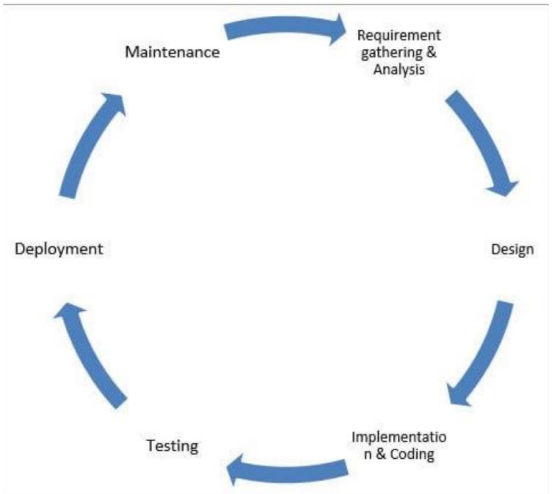
Purpose:

Purpose of SDLC is to deliver a high-quality product which is as per the customer's requirement. SDLC has defined its phases as, Requirement gathering, Designing, Coding, Testing, and Maintenance. It is important to adhere to the phases to provide the Product in a systematic manner. For Example, A software has to be developed and a team is divided to work on a feature of the product and is allowed to work as they want. One of the developers decides to design first whereas the other decides to code first and the other on the documentation part.

This will lead to project failure because of which it is necessary to have a good knowledge and understanding among the team members to deliver an expected product.

SDLC Cycle

SDLC Cycle represents the process of developing software. Below is the diagrammatic representation of the SDLC cycle:



SDLC Phases

The entire SDLC process divided into the following SDLC steps:

Phase 1: Requirement collection and analysis

Phase 2: Feasibility study

Phase 3: Design

Phase 4: Coding

Phase 5: Testing

Phase 6: Installation/Deployment

Phase 7: Maintenance

Phase 1: Requirement collection and analysis

The requirement is the first stage in the SDLC process. It is conducted by the senior team members with inputs from all the stakeholders and domain experts in the industry. Planning for the quality assurance requirements and recognition of the risks involved is also done at this stage. This stage gives a clearer picture of the scope of the entire project and the anticipated issues, opportunities, and directives which triggered the project. Requirements Gathering stage need teams to get detailed and precise requirements. This helps companies to finalize the necessary timeline to finish the work of that system.

Phase 2: Feasibility study

Once the requirement analysis phase is completed the next sdlc step is to define and document software needs. This process conducted with the help of 'Software Requirement Specification' document also known as 'SRS' document. It includes everything which should be designed and developed during the project life cycle. There are mainly five types of feasibilities checks:

Economic: Can we complete the project within the budget or not?

Legal: Can we handle this project as cyber law and other regulatory framework/compliances.

Operation feasibility: Can we create operations which is expected by the client?

Technical: Need to check whether the current computer system can support the software

Schedule: Decide that the project can be completed within the given schedule or not.

Phase 3: Design

In this third phase, the system and software design documents are prepared as per the requirement specification document. This helps define overall system architecture. This design phase serves as input for the next phase of the model. There are two kinds of design documents developed in this phase:

High-Level Design (HLD) : Brief description and name of each module

An outline about the functionality of every module

Interface relationship and dependencies between modules

Database tables identified along with their key elements

Complete architecture diagrams along with technology details

Low-Level Design (LLD): Functional logic of the modules

Database tables, which include type and size

Complete detail of the interface

Addresses all types of dependency issues

Listing of error messages

Complete input and outputs for every module

Phase 4: Coding

Once the system design phase is over, the next phase is coding. In this phase, developers start build the entire system by writing code using the chosen programming language. In the coding phase, tasks are divided into units or modules and assigned to the various developers. It is the longest phase of the Software Development Life Cycle process. In this phase, Developer needs to follow certain predefined coding guidelines. They also need to use programming tools like compiler, interpreters, debugger to generate and implement the code.

Phase 5: Testing

Once the software is complete, and it is deployed in the testing environment. The testing team starts testing the functionality of the entire system. This is done to verify that the entire application works according to the customer requirement.

During this phase, QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and send back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the business needs of that system.

Phase 6: Installation/Deployment

Once the software testing phase is over and no bugs or errors left in the system then the final deployment process starts. Based on the feedback given by the project manager, the final software is released and checked for deployment issues if any.

Phase 7: Maintenance

Once the system is deployed, and customers start using the developed system, following 3 activities occur

Bug fixing – bugs are reported because of some scenarios which are not tested at all

Upgrade – Upgrading the application to the newer versions of the Software

Enhancement – Adding some new features into the existing software The main focus of this SDLC phase is to ensure that needs continue to be met and that the system continues to perform as per the specification mentioned in the first phase

What is the classification of the software requirements? Explain with example.

Classification of software requirements

Software requirements is a field within software engineering that deals with establishing the needs of stakeholders that are to be solved by software. **Software system requirements are often classified as functional requirements, non – functional requirements or domain requirements.**

1) **Functional requirements**: In software engineering, a functional requirement defines a system or its component. It describes the functions a software must perform. A function is nothing but inputs, its behavior, and outputs. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform. In some cases, the functional requirements may also explicitly state what the system should not do. These requirements depend on the type of software being developed, the expected users of the software and the general approach taken by the organization when writing requirements.

   Functional requirements in software engineering help you to capture the intended behavior of the system. This behavior may be expressed as functions, services or tasks or which system is required to perform. **Functional requirements indicate what a system must do.**

2) **Non – functional requirements** - A non-functional requirement defines the quality

attribute of a software system. They represent a set of standards used to judge the specific operation of a system. Example, how fast does the website load? A non-functional requirement is essential to ensure the usability and effectiveness of the entire software system. Failing to meet non-functional requirements can result in systems that fail to satisfy user needs. On-functional Requirements allows you to impose constraints or restrictions on the design of the system across the various agile backlogs. Example, the site should load in 3 seconds when the number of simultaneous users is > 10000. Description of non-functional requirements is just as critical as a functional requirement. **Non-functional requirements support them and determine how the system must perform.**

Types of non – functional requirements

These are constraints on the services or functions offered by the system. They include timing constraints, constraints on the development process and standards. Non – functional requirements often apply to the system as a whole. The types of non – functional requirements are:

1) **Product requirements**: Requirements which specify that the delivered product must behave in a particular way e.g., execution speed, reliability, etc.

2) **Organizational requirements**: Requirements which are a consequence of organizational policies and procedures e.g., process standards used, implementation requirements, etc.

External requirements: Requirements which arise from factors which are external to the system and its development process e.g., interoperability requirements, legislative requirements, etc

Metrics of specifying Non – functional requirements	
Speed	Processed transactions/Event response time, Screen refresh time.
Size	K bytes, Number of RAM chips
Ease of use	Training time, Number of help frames
Reliability	Mean time to failure, Portability of unavailability, Rate of failure occurrence, viability
Robustness	Time to restart after failure, Percentage of events causing failure, Portability of data corruption on failure.
Portability	Percentage of target-dependent statements, Number of target systems

What are the fundamental activities of software process?

Software processes

Software Engineering is defined as the systematic approach to the development, operation, maintenance and retirement of software. The systematic approach must help to achieve a high quality and productivity (Q&P) of software. A software process (also known as software methodology) is a set of related activities that leads to the production of the software. These activities may involve the development of the software from the scratch, or, modifying an existing system.

A software process specifies the abstract set of activities that should be performed to go from user needs to final product. The actual act of executing the activities for some user needs is a software project and all the outputs that are produced while the activities are being executed are the products.

Software process activities

Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.

The four basic process activities of specification, development, validation and evolution are organized differently in different development processes. In the waterfall model, they are organized in sequence, whereas in incremental development they are interleaved.

1) Software specification

The process of establishing what services are required and the constraints on the system's operation and development.

Requirements engineering process:

Feasibility study: is it technically and financially feasible to build the system?

Requirements elicitation and analysis: what do the system stakeholders require or expect from the system?

Requirements specification: defining the requirements in detail

Requirements validation: checking the validity of the requirements

2) Software design and implementation

The process of converting the system specification into an executable system.

Software design: design a software structure that realizes the specification;

Implementation: translate this structure into an executable program;

The activities of design and implementation are closely related and may be interleaved.

Design activities include:

Architectural design: identify the overall structure of the system, the principal components (sometimes called sub-systems or modules), their relationships and how they are distributed. **Interface design:** define the interfaces between system components.

Component design: take each system component and design how it will operate.

Database design: design the system data structures and how these are to be represented in a database.

3) Software validation

Verification and validation (V & V) is intended to show that a system conforms to its

specification and meets the requirements of the system customer.

Validation: are we building the right product (what the customer wants)?

Verification: are we building the product, right?

V & V involves checking and review processes and system testing. System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.

Testing is the most commonly used V & V activity and includes the following stages:

Development or component testing: individual components are tested independently; components may be functions or objects or coherent groupings of these entities.

System testing: testing of the system as a whole, testing of emergent properties is particularly important.**Acceptance testing**: testing with customer data to check that the system meets the customer's needs.

4) Software evolution

Software is inherently flexible and can change. As requirements change through changing business circumstances, the software that supports the business must also evolve and change. Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new**.**