Page No.
Date

Software Development Life cycle (SDLC)

- It is a structured process that defines the stages involved in developing software from planning to deployment and maintenance.
- It ensures the software is high quality, cost effective, and delivered on time.
- Phases :

1) Requirement Analysis
   Understanding what the client needs
2) Design
   Creating architecture and design documents
3) Implementation (coding)
   ~~Testing~~ Developers builds the actual product
4) Testing
   QA team tests the software for bugs or errors
5) Deployment
   Software is released to users
6) Maintenance
   Fixing issues and updating features post release.

Software Prototype

- It is an early model or sample of a system built to visualize, test, and refine requirements before the actual development begins.
- It helps users and developers understand how the final product will function.
- Types :

→ Throw away (Rapid) Prototype : discarded after getting feedback
→ Evolutionary Prototype : continuously improved until it becomes the final product.

- Advantages :
  - Clarifies requirements early
  - Reduces misunderstandings
  - Improves user involvement.

Process Model

- It is a framework that defines how SDLC phases are carried out. It represents the flow and order of development activities.
- Common Models :

1) Waterfall Model
   - Sequential, each phase must complete before the next starts.
   - It's simple and structured but inflexible changes are hard once a phase is complete.
   - Projects with well defined, stable requirements.

2) Iterative Model
   - Developed in repeated cycle (iterations) where each iteration add features, tests, and refines the system based on feedback.
   - It allows early working versions and adapts to change more easily than waterfall
   - Large projects where requirements evolve over

time.

3) Spiral Model
- Combines iterative development with risk management.
- Each spiral loop represents a phase: planning, risk analysis, development, evaluation
- Large, high risk projects where requirements are unclear or changing.

4) Rapid Application Development Model
- Emphasizes fast development and quick feedback through prototypes and component reusability.
- It involves user participation and minimal planning. Requirement planning → User Design → Construction → Cutover
- Projects with tight deadlines and clear requirements.

5) Test Driven Development
- It is a software development practice where tests are written before the actual code.
- Developers writes a failing tests then code just enough to make it pass, and finally refactor.
- Cycle: Red → Green → Refactor
- Benefit: Better code quality, Fewer bugs, Easier refactoring

6) V Model
- Verification: (define/check specification) → design → coding/Unit test →

- Validation : (Integrate / Integration / System test).

7) Agile Model
- Iterative and incremental approach focusing on collaboration, customer feedback, and flexibility.
- It delivers working software in small, frequent relases called sprints.
- Principles :
  - Costomer collaboration over contract negotiation
  - Responding to change over following a plan
  - Working software over documentation

8) Scrum (Agile FrameWork)
- Lightweight agile framework for managing and dereloping complex projects.
- Work is divided into sprints (typically 2-4 weeks).
- Key Events : Sprint planning, Daily Stand up, Sprint Review, Sprint Retrospective
- Key Roles :
  - Product Owner : Defines product goals
  - Scrum Master : Facilitates process, removes blockers
  - Development Team : Builds the product.

9) Capability Maturity Model
- Five maturity levels that measure an organization's software process capability and quality
- Levels :
  1) Initial :- Ad hoc, unpredictable process
  2) Repeatable :- Basic project management in place

3) Defined :- Standardized processes across projects.

4) Managed :- Quantitative metrics used for control

5) Optimizing :- Continuous process improvement

High Level Design

- It defines the overall architecture of the system. It breaks the system into modules or components and shows how they interact. It focuses on what each module does, not the internal details.
- Key aspects:
  - Architecture diagrams
  - Data flow between modules
  - Technology stack selection
  - Database design overview
- Examples: Choosing microservices architecture, APIs, database schema

Low Level Design

- It focuses on the internal logic of each module defined in HLD. It describes how the components are implemented.
- Key aspects:
  - Class diagrams and methods
  - Database tables and relationships

- Pseudocode or algorithms
- Interface and function definitions

• Example: Writing class level details for a "User Authentication" module.

Software Requirement Specification (SRS) —
It is a formal document that defines what the software must do. It describes all functional and non functional requirements in detail serving as a bridge between the client and the development team.

Contents of SRS:
1. Introduction (purpose, scope, definitions)
2. Functional requirements (features, use cases)
3. Non Functional requirements (performance, security, usability)
4. Constraints and assumptions
5. Acceptance criteria

Purpose: avoid misunderstandings, ensure clarity, and guide design and testing

Software Quality Assurance (SQA) —
It is a process oriented activity that ensures software development follows standard procedures and quality practices throughout the SDLC.
It aims to prevent defects by improving the development process itself.

Page No.
Date

Includes:
- Process audits and reviews
- Standards and guidelines enforcement
- Quality metrics and continuous Improvement

Goal: Ensure quality is built into the product, not just tested afterward.

Software Quality Control (SQC) —
It is a product oriented activity focused on detecting defects in the final product. It involves testing, inspection, and validation to verify that the software meets requirements.
Includes:
- Code reviews
- Testing (unit, integration, system, acceptance)
- Bug tracking

Goal: Identify and fix defects after the product is developed.

Verification
- It checks whether the product is being built correctly, it ensures the software meets design specifications and requirements
- Focus: Process oriented
- Question: "Are we building the product right?"
- Example: Reviewing design documents, code

Page No.
Date

Inspections, walkthroughs

Validation

- It checks whether the right product is built, it ensures the software actually meets user needs and expectations
- Focus: Product oriented
- Question: "Are we building the right product?"
- Example: Functional testing, user acceptance testing

Testing

It is the process of evaluating a software system to check whether it meets the specified requirements and is free of defects.

It helps ensure quality, reliability, and performance before release.

Types:

- Functional Testing
- Non Functional Testing
- Maintenance Testing

Page No.
Date

Functional Testing —

- It verifies that the software's features works as intended according to the requirements (from the SRS).
- It checks what the system does.
- Examples :
  - → Unit Testing - Tests individual components (by developers)
  - → Integration Testing - Tests combined modules for correct interaction
  - → System Testing - Verifies the entire system as a whole
  - → Acceptance Testing - validates the system against busines needs. (often done by clients)
- Tools : Selenium, JUnit, TestNG

Non Functional Testing —

- It checks the quality attributes of the system such as performance, security, usability, and reliability.
- It answers how well the system works, not what it does.
- Example :
  - Performance Testing
  - Load / Stress Testing
  - Security Testing
  - Compatibility and Scalability Testing

Page No.
Date

Static Testing

- It involves examining code, documents, or designs without executing the program.
- It's a verification activity to find errors early.
- Example: Reviews, walk throughs, code inspections
- Goal: Catch defects before runtime

Dynamic Testing

- It involves executing the software to check functional behavior and performance.
- It's validation activity that verifies the system works as a expected.
- Examples: Unit, Integration, System, Acceptance testing

Block Box Testing

- It evaluates the external behavior of the software without looking at the internal code or structure.
- Testers only know the inputs and expected outputs.
- Techniques:
  - Equivalence partitioning
  - Boundary value analysis
  - Decision tables

White Box Testing

- It involves the testing the internal logic, code, and structure of the program.
- Testers must understand the code and ensure all paths and conditions are covered.

Page No.
Date

- Techniques:
  - Statement coverage
  - Branch coverage
  - Path coverage.