SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC) MODELS LECTURE 3

SDLC models define **how** the phases of software development (requirements → design → coding → testing → deployment → maintenance) are organized and executed. Different models suit different project types, complexity levels, budgets, timelines, and organizational cultures.

1. Waterfall Model

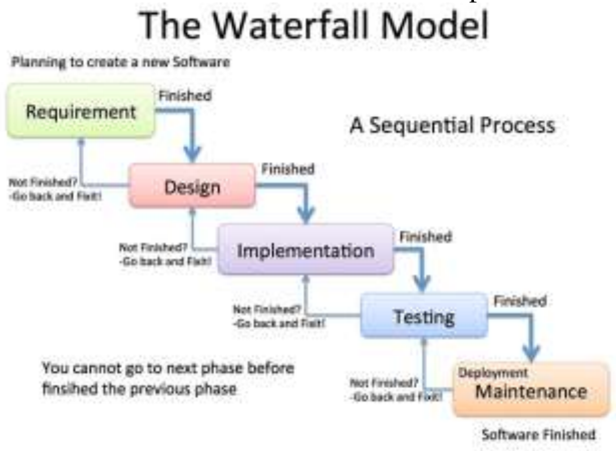
The Waterfall Model is one of the earliest and most widely known SDLC models. It follows a linear, sequential flow, where each phase must be completed before moving to the next. It represents a "waterfall" — once water flows down a step, it cannot go back up.

The Waterfall Model is a sequential SDLC approach where progress flows steadily down through phases like requirements, design, coding, testing, deployment, and maintenance. It is sometimes called the Linear Sequential Model.

Key Characteristics

Sequential: Each phase must be finished completely before the next one begins.
Highly structured: Documentation is formal and thorough.
Emphasis on early planning: Requirements and design are completed before coding starts.
Minimal user involvement during development: Users mainly provide requirements at the beginning.
Low flexibility: Changes are expensive once the project moves past early phases.

Phases of the Waterfall Model
The traditional Waterfall has six main phases:


The Waterfall Model
Planning to create a new Software
Requirement
Finished
A Sequential Process
Not Finished? -Go back and Fix!
Design
Finished
Not Finished? -Go back and Fix!
Implementation
Finished
Not Finished? -Go back and Fix!
Testing
Finished
You cannot go to next phase before finsihed the previous phase
Not Finished? -Go back and Fix!
Deployment
Maintenance
Software Finished

1. **Requirement Gathering and analysis** − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.

2. **System Design** − The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture. Define hardware requirements. Plan data structures, modules, interfaces.

   Two design levels:
   High-level design (HLD)
   - System architecture
   - Module breakdown
   - Data flow diagrams

   Low-level design (LLD)
   - Class diagrams
   - Database schema
   - Algorithms

3. **Implementation** − With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing. Translate requirements into a system architecture. Developers write code based on LLD and HLD. Code is developed module by module. Each module is developed independently. Coding standards and documentation are important

4. **Integration and Testing** − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures. Combine all modules, Perform full system testing
   Testing checks functional and non-functional requirements
   Common tests include: Unit, Integration, System, Acceptance

5. **Deployment of system** − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market. Install the software and carryout user training.

6. **Maintenance** − There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment. Types of maintenance include: Corrective (fixing errors), Adaptive (changes due to environment updates), Perfective (enhancements) and Preventive (avoiding future issues)

Advantages of Water fall Model
Some of the major advantages of the Waterfall Model are as follows −

- Simple and easy to understand and use

- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

Disadvantages

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.

2. V-Model (Verification and Validation Model)

The V-Model is an SDLC model that emphasizes the importance of testing at every stage of development. It extends the Waterfall model but introduces a parallel testing phase for each development activity. It is also known as the Validation and Verification Model. The V-Model is a sequential SDLC model where each development phase (left side of the “V”) has a corresponding testing phase (right side of the “V”). This means testing is planned early, not at the end of development.

Why It Is Called the V-Model

The process forms a V shape:

- Left Side → Verification phases
- Bottom → Coding
- Right Side → Validation phases

Each phase on the left has a matching test activity on the right.

Verification vs Validation

<u>Verification</u>

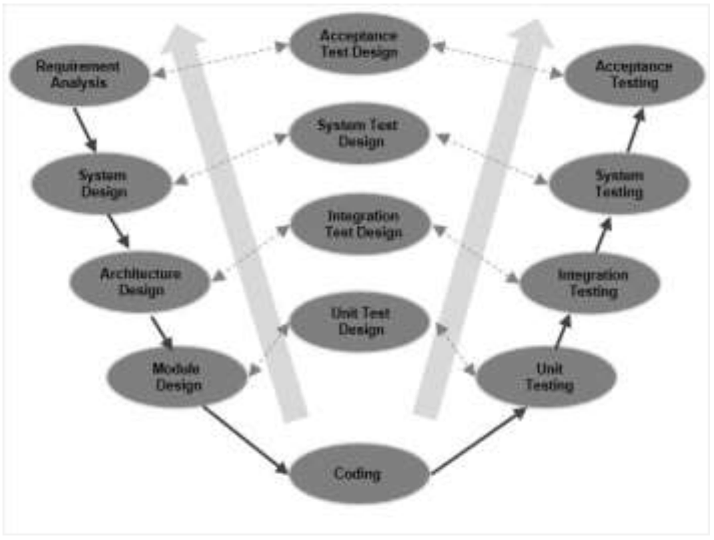
“Are we building the product right?”

Ensures the design meets requirements.

Validation

“Are we building the right product?”

Ensures the final product meets user expectations.



V-Model - Verification Phases

There are several Verification phases in the V-Model, each of these are explained in detail below.

Business Requirement Analysis

This is the first phase in the development cycle where the product requirements are understood from the customers perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirement. This is a very important activity and needs to be managed well, as most of the customers are not sure about what exactly they need. The acceptance test design planning is done at this stage as business requirements can be used as an input for acceptance testing.

System Design

Once you have the clear and detailed product requirements, it is time to design the complete system. The system design will have the understanding and detailing the complete hardware and communication setup for the product under development. The system test plan is developed based on the system design. Doing this at an earlier stage leaves more time for the actual test execution later.

Architectural Design

Architectural specifications are understood and designed in this phase. Usually more than one technical approach is proposed and based on the technical and financial feasibility the final decision is taken. The system design is broken down further into modules taking up different functionality. This is also referred to as High Level Design (HLD).

The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood and defined in this stage. With this information, integration tests can be designed and documented during this stage.

Module Design

In this phase, the detailed internal design for all the system modules is specified, referred to as Low Level Design (LLD). It is important that the design is compatible with the other modules in the system architecture and the other external systems. The unit tests are an essential part of any development process and helps eliminate the maximum faults and errors at a very early stage. These unit tests can be designed at this stage based on the internal module designs.

Coding Phase

The actual coding of the system modules designed in the design phase is taken up in the Coding phase. The best suitable programming language is decided based on the system and architectural requirements.

The coding is performed based on the coding guidelines and standards. The code goes through numerous code reviews and is optimized for best performance before the final build is checked into the repository.

Validation Phases

The different Validation Phases in a V-Model are explained in detail below.

Unit Testing

Unit tests designed in the module design phase are executed on the code during this validation phase. Unit testing is the testing at code level and helps eliminate bugs at an early stage, though all defects cannot be uncovered by unit testing.

Integration Testing

Integration testing is associated with the architectural design phase. Integration tests are performed to test the coexistence and communication of the internal modules within the system.

System Testing

System testing is directly associated with the system design phase. System tests check the entire system functionality and the communication of the system under development with external systems. Most of the software and hardware compatibility issues can be uncovered during this system test execution.

Acceptance Testing
Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. Acceptance tests uncover the compatibility issues with the other systems available in the user environment. It also discovers the non-functional issues such as load and performance defects in the actual user environment.

Advantages
- This is a highly-disciplined model and Phases are completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Simple and easy to understand and use.
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.

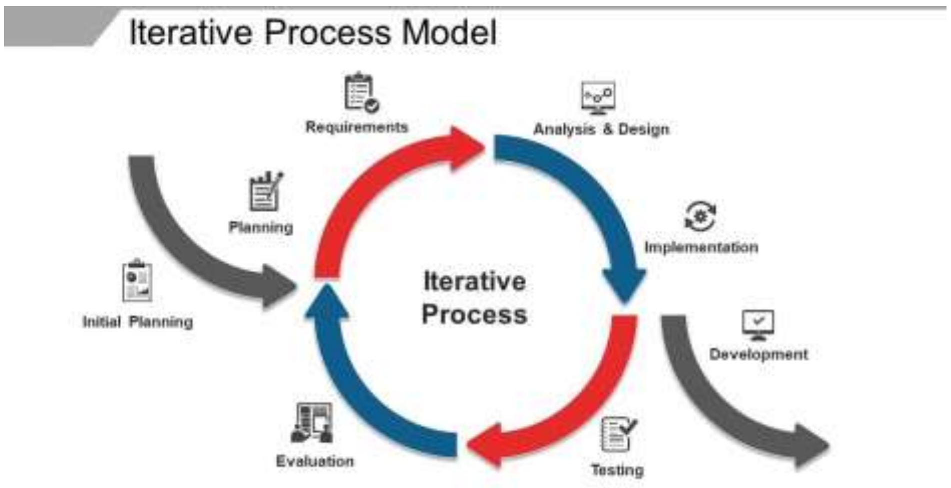
Disadvantages
- High risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.
- Once an application is in the testing stage, it is difficult to go back and change a functionality.
- No working software is produced until late during the life cycle.

3. Iterative Model
The Iterative Model is an SDLC model where software is developed through repeated cycles (iterations). Each iteration produces an improved or expanded version of the software until the final system is completed. It is a hybrid of Waterfall and prototyping, emphasizing continuous refinement.

The Iterative Model is a software development approach where:
- The system is developed step-by-step
- Each cycle includes planning, design, implementation, testing, and evaluation
- Each iteration produces a working version of the software
- Feedback drives improvement and new features
- It is ideal for projects with unclear or evolving requirements

Iterative and Incremental development is a combination of both iterative design or iterative method and incremental build model for development. "During software development, more than one iteration of the software development cycle may be in progress at the same time." This process may be described as an "evolutionary acquisition" or "incremental build" approach."

In this incremental model, the whole requirement is divided into various builds. During each iteration, the development module goes through the requirements, design, implementation and testing phases. Each subsequent release of the module adds function to the previous release. The process continues till the complete system is ready as per the requirement.



Phases of the Iterative Model

Each iteration includes all SDLC phases, but with smaller scope.

1. Planning
Identify which features will be developed in the iteration
Prioritize features based on business value
Set iteration goals

2. Requirements Analysis
Collect detailed requirements for the selected features
Requirements do not need to be fully complete initially
Requirements grow and evolve over iterations

3. Design
Create or refine system architecture
Design components and modules for this iteration
Update UI/UX designs

Produce updated class diagrams and database design

4. Implementation (Coding)
Developers code the features designed in this iteration
Code integrates with previous iterations
Version control (e.g., Git) is essential

5. Testing
Unit testing
Integration testing
Regression testing
System testing
Testing earlier iterations improves future design decisions.

6. Review & Evaluation
Evaluate the iteration with users/stakeholders
Gather feedback
Identify improvements or new requirements
Prepare for the next iteration

Advantages of the Iterative Model
1. Early working software delivered: Users see progress immediately.
2. Handles changing requirements easily: New features can be added in later iterations.
3. Early detection of problems: Each iteration includes testing and evaluation.
4. Reduces project risk: High-risk components addressed early.
5. Better user engagement: Feedback helps refine the system.
6. Supports complex and large projects: Architecture evolves to match needs.
7. Continuous improvement: Design, code, and requirements get better with each iteration.

Disadvantages of the Iterative Model
1. Requires active user involvement: Not all clients participate consistently.
2. Repeated cycles may increase cost: More meetings, designs, and testing cycles.
3. Scope creep risk: Frequent changes cause uncontrolled growth.
4. Requires strong planning & documentation: Otherwise iterations become chaotic.
5. Architecture becomes complex: New features must integrate with existing iterations.

Use this model when:

- Requirements are evolving
- User feedback is essential
- Project is large and complex
- Requirements cannot be fully known upfront
- Rapid prototyping needed
- High-risk systems

SPIRAL MODEL

The Spiral Model is a risk-driven, iterative SDLC model that combines elements of Waterfall, Iterative development, and Prototyping. It focuses on early identification and mitigation of risks in software projects.

The Spiral Model is an SDLC approach where software development progresses through a series of loops (spirals).

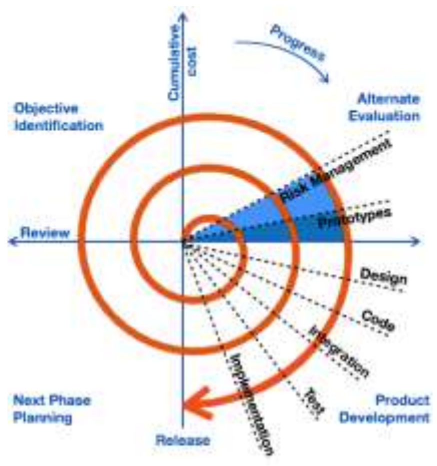
Each loop represents a development phase, and risk analysis is the central activity in every loop.

Why the Spiral Model Was Introduced

The Spiral Model was introduced by Barry Boehm to address the weaknesses of the Waterfall Model, especially:

- Late discovery of risks
- Inflexibility to change
- Poor handling of complex, high-risk projects

The Spiral Model ensures risks are addressed early and continuously.



Four Quadrants of the Spiral Model

Each spiral loop consists of the following four quadrants:

1. Planning Quadrant
Activities
Identify objectives for the phase
Define deliverables
Identify alternative approaches
Plan schedule and cost
Output

- Project plans
- Feature selection for the iteration

2. Risk Analysis Quadrant (Most Important)
Activities
Identify technical, cost, schedule, and usability risks
Analyze risk impact
Develop risk mitigation strategies
Build prototypes to reduce risks

Examples of Risks
New technology unfamiliar to team
Performance issues
Security vulnerabilities

User acceptance risks

Output

- Risk assessment report
- Prototypes or proof-of-concepts

3. Engineering Quadrant
Activities
Design system components
Code software modules

Perform unit and integration testing
Build functional software increments
Output

- Working software components
- Technical documentation

4. Evaluation Quadrant
Activities
Stakeholders review the developed increment
Collect user feedback
Decide whether to continue, modify, or terminate the project
Output

- Review feedback
- Approval to proceed to next spiral

Phases of the Spiral Model
Each spiral loop can represent:

- Concept development
- System design
- Implementation
- Testing
- Deployment

The number of spirals depends on:

- Project size
- Complexity
- Risk level

The advantages of the Spiral SDLC Model are as follows −

- Changing requirements can be accommodated.
- Allows extensive use of prototypes.
- Requirements can be captured more accurately.
- Users see the system early.
- Development can be divided into smaller parts and the risky parts can be developed earlier which helps in better risk management.

The disadvantages of the Spiral SDLC Model are as follows −

- Management is more complex.
- End of the project may not be known early.
- Not suitable for small or low risk projects and could be expensive for small projects.

- Process is complex

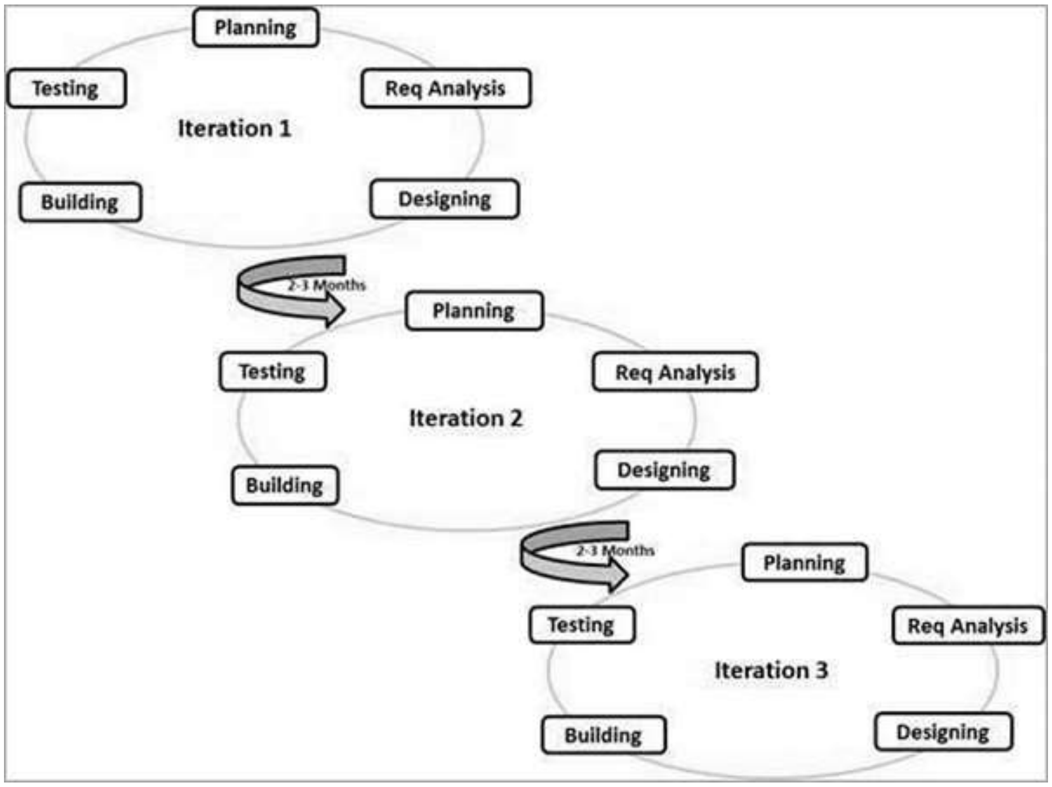
- Spiral may go on indefinitely.
- Large number of intermediate stages requires excessive documentation.
- Print Page

AGILE MODEL

The Agile Model is a flexible, iterative, and incremental software development approach that focuses on customer satisfaction, rapid delivery, collaboration, and adaptability to change.
The Agile Model is an SDLC approach where software is developed in small, time-boxed iterations called sprints, with continuous user feedback and frequent delivery of working software.

Agile emphasizes:

- People over processes
- Working software over documentation
- Customer collaboration over contracts
- Responding to change over following a plan



Following are the Agile Manifesto principles:

1. Individuals and interactions − In Agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.
2. Working software − Demo working software is considered the best means of communication with the customers to understand their requirements, instead of just depending on documentation.
3. Customer collaboration − As the requirements cannot be gathered completely in the beginning of the project due to various factors, continuous customer interaction is very important to get proper product requirements.

4. Responding to change − Agile Development is focused on quick responses to change and continuous development.

Popular Agile Frameworks

Scrum is an Agile framework used to manage and deliver software projects in an iterative and incremental manner.

A. Scrum

Time-boxed sprints (1–4 weeks):

Development work is divided into short, fixed-length cycles called sprints, during which a usable software increment is produced.

Well-defined roles and ceremonies:

Scrum has clear roles (Product Owner, Scrum Master, Development Team) and structured ceremonies such as Sprint Planning, Daily Stand-ups, Sprint Review, and Sprint Retrospective to ensure coordination and continuous improvement.

Most widely used Agile framework:

Scrum is the most popular Agile framework because it is simple to understand, adaptable, and effective for managing changing requirements in many types of projects.

B. Kanban

Kanban is an Agile framework that focuses on visualizing work and improving flow.

Visual workflow using boards:

Work items are displayed on a Kanban board (e.g., To Do → In Progress → Done), making progress and bottlenecks easy to see.

Limits work in progress (WIP):

The number of tasks in each stage is limited to prevent overload and improve efficiency.

Continuous delivery:

Work is completed and released continuously rather than in fixed-length iterations, allowing faster response to changes and customer needs.

C. Extreme Programming (XP)

Extreme Programming (XP) is an Agile framework that emphasizes high-quality software through strong engineering practices.

Focus on engineering practices:

XP prioritizes technical excellence to ensure reliable, maintainable, and clean code.

Pair programming:

Two developers work together at one workstation—one writes code while the other reviews—reducing errors and improving knowledge sharing.

Test-Driven Development (TDD):

Tests are written before the actual code, ensuring functionality meets requirements and defects are detected early.

D. Lean Software Development

Lean Software Development is an Agile framework that focuses on delivering maximum customer value while minimizing waste.

Eliminates waste:

Removes unnecessary activities such as excessive documentation, unused features, and delays.

Maximizes value:

Prioritizes features that provide the highest business value to customers.

Continuous improvement:
Encourages ongoing evaluation and refinement of processes to improve efficiency and quality.

The advantages of the Agile Model are as follows −

- Is a very realistic approach to software development.
- Promotes teamwork and cross training.
- Functionality can be developed rapidly and demonstrated.
- Resource requirements are minimum.
- Suitable for fixed or changing requirements
- Delivers early partial working solutions.
- Good model for environments that change steadily.
- Minimal rules, documentation easily employed.
- Enables concurrent development and delivery within an overall planned context.
- Little or no planning required.
- Easy to manage.
- Gives flexibility to developers.

The disadvantages of the Agile Model are as follows −

- Not suitable for handling complex dependencies.
- More risk of sustainability, maintainability and extensibility.
- An overall plan, an agile leader and agile PM practice is a must without which it will not work.
- Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
- Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
- There is a very high individual dependency, since there is minimum documentation generated.

- Transfer of technology to new team members may be quite challenging due to lack of documentation.