SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Lecture 2: Software Development Models

IT2030 – Software Engineering

Lesson Learning Outcomes

By the end of this lecture, students will be able to:

- Explain the concept of the Software Development Life Cycle (SDLC)
- Differentiate between SDLC and Software Process Models
- Describe key characteristics, advantages, and limitations of various process models
- Identify suitable models for different types of software projects
- Apply understanding of process models to real-world scenarios

SDLC vs. Software Process Models

What is SDLC (Software Development Life Cycle)?

SDLC is a step-by-step process used to develop software. It describes the main phases involved in creating a software product

***Quick Question**: What are the phase in SDLC?*

What is a Software Process Model?

- A **Software Process Model** is a structured method or strategy that guides how the SDLC phases are carried out.
- It provides a framework or roadmap for organizing and managing the tasks in SDLC.

Examples of Software Process Models:

- **Waterfall –** Each phase is completed one after another
- **Spiral –** Combines design and prototyping in a repeated cycle
- **Agile –** Focuses on small, fast iterations with regular feedback

How Are They Connected?

SDLC	Software Process Models
Defines **what phases** to follow	Defines **how** to execute those phases
A general structure	A specific implementation
Same phases used across models	Each model **organizes, iterates, or emphasizes** phases differently

Example:

- Waterfall model: executes SDLC **sequentially**
- Agile model: executes SDLC in **iterations (sprints)**

Evolution of Software Process Models

- From Traditional to Modern Approaches:

Model	Description
Waterfall Model	Linear and phase-based development model
V - Model	Emphasizes validation and testing at each phase
Incremental Model	Software delivered in small parts or versions
Prototyping Model	Early versions built to refine requirements
Spiral Model	Iterative model with risk assessment in each cycle
Agile (Scrum / Kanban)	Iterative, collaborative, and flexible approach

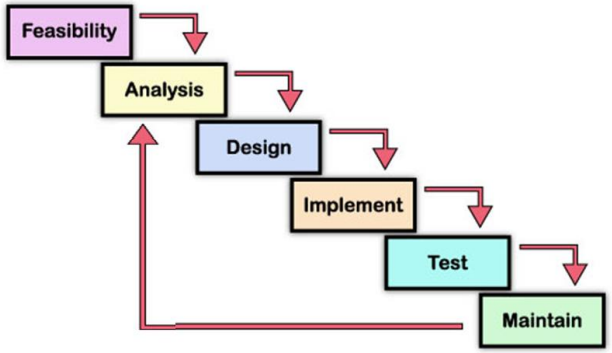
Waterfall Model

Waterfall Model

A **linear and step-by-step** process where each phase is completed before moving to the next.

Phases:

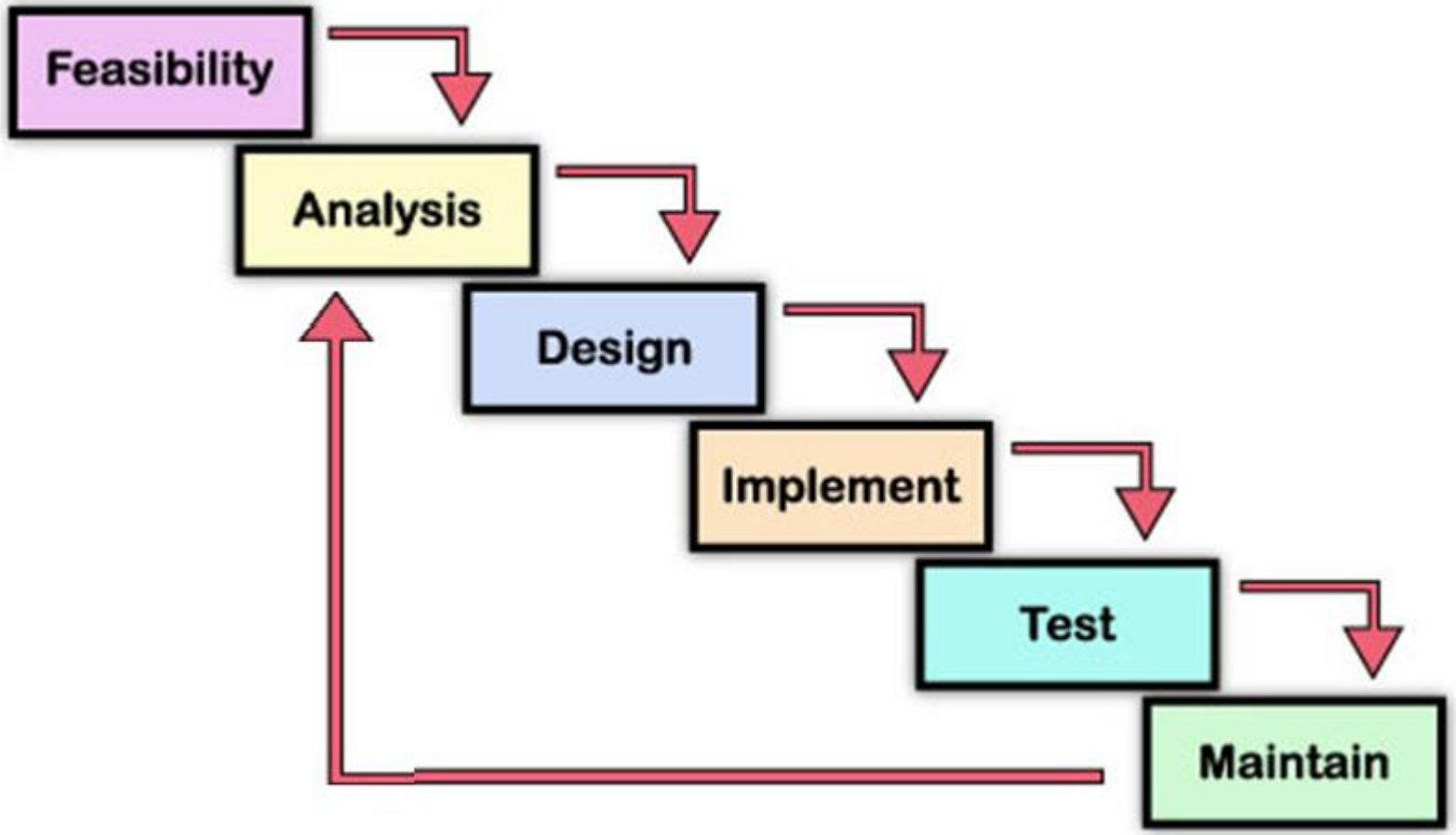
Requirements → Design → Implementation → Testing → Deployment → Maintenance



Key Features:

- Simple and easy to understand
- Each phase has clear goals and deliverables
- Not suitable for projects where requirements may change later

Waterfall Model

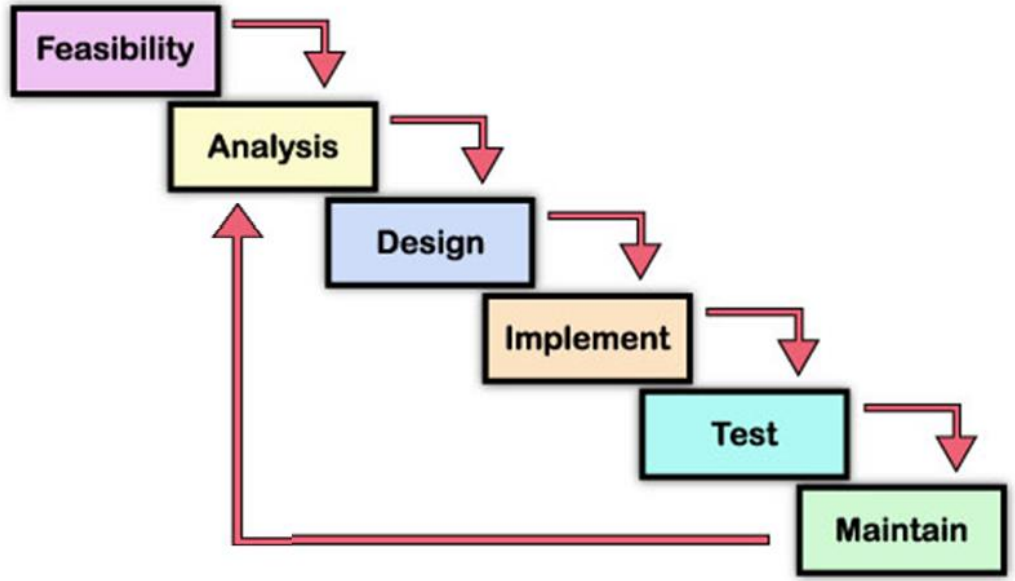


Waterfall Model

Example:

Used for systems like **Payroll** or **HR Management**, where the requirements are stable and well-defined.

- **Best Use Case:**
  When the project requirements are **clearly known in advance** and unlikely to change.



Iterative Waterfall Model

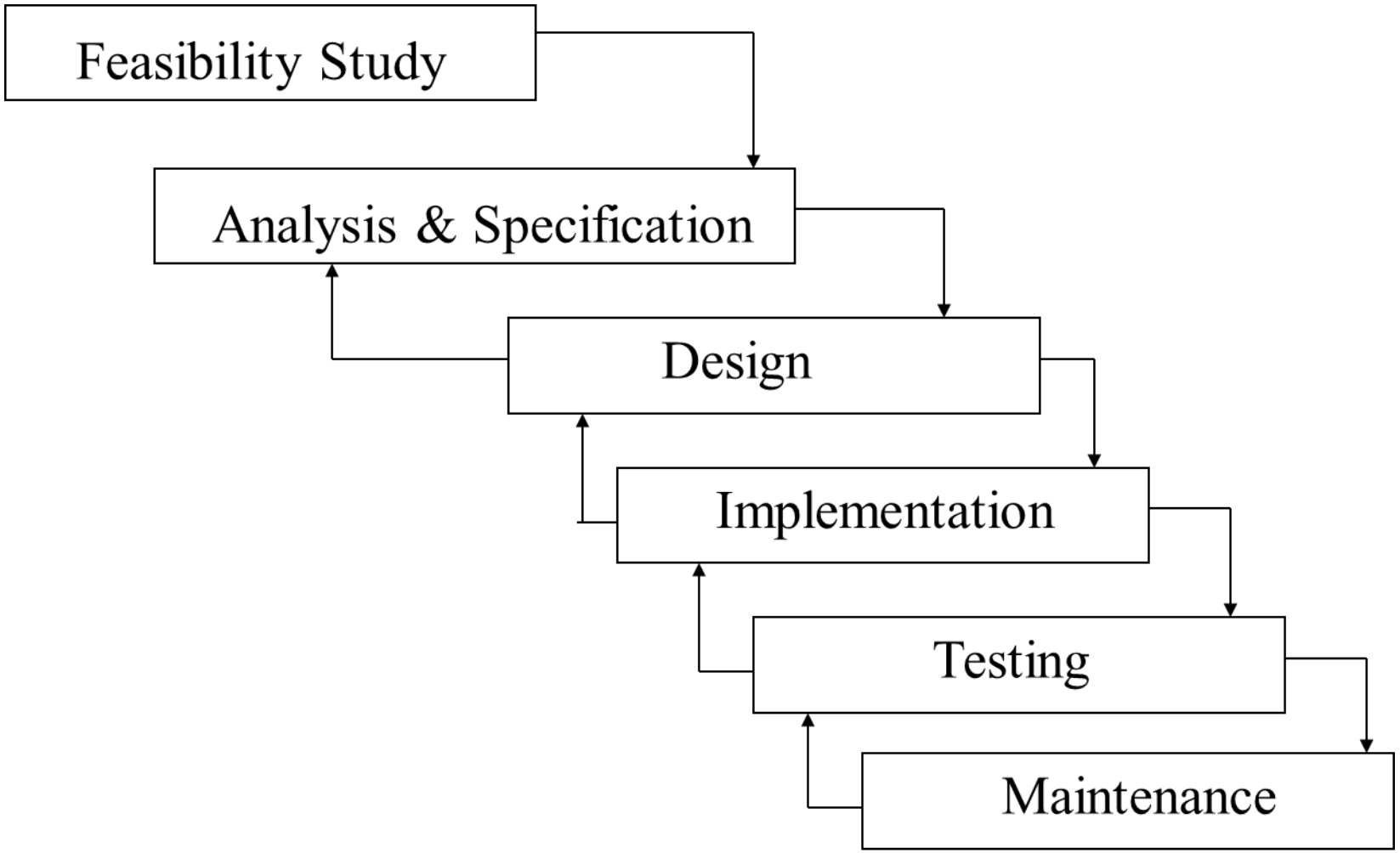
Iterative Waterfall Model

A **modified version** of the traditional Waterfall Model that allows **feedback and revisions** between phases.

Key Features:

- Follows the same step-by-step phases as the classic Waterfall (Requirements → Design → Implementation → Testing → Deployment)
- But now, **teams can go back** to earlier phases if needed (e.g., If problems are found during Testing, they can return to the Design phase to fix them)

Iterative Waterfall Model



Iterative Waterfall Model

- This **reduces the rigid structure** of the original Waterfall model
- Allows for **corrections and improvements** before the final product is delivered
- **Benefit:**
  More flexible and realistic for real-world projects where issues may only appear in later stages.

Iterative Waterfall Model

When to Use the Iterative Waterfall Model

This model is suitable when:

- **Requirements are mostly clear**, but **some minor changes** may be needed during development
- **Client feedback is available** at the end of each phase
- The project is a **medium-sized system** with **moderate risk** (not too complex, but not too simple either)

Activity 1

Think of an example system (e.g., student record system).

- Would you use Waterfall or Iterative Waterfall for it?
- Justify your answer with 1 reason.

V-Model (Validation and Verification Model)

The V-Model is a **variation of the Waterfall Model** that focuses on **testing at every stage of development**.

Key Features:

- Development happens on the **left side** of the "V"
- Testing and validation happen on the **right side**, matched to each development phase
- For every development step, there is a **corresponding test step** (e.g., Requirements → Acceptance Testing)

V-Model

Why it's useful:

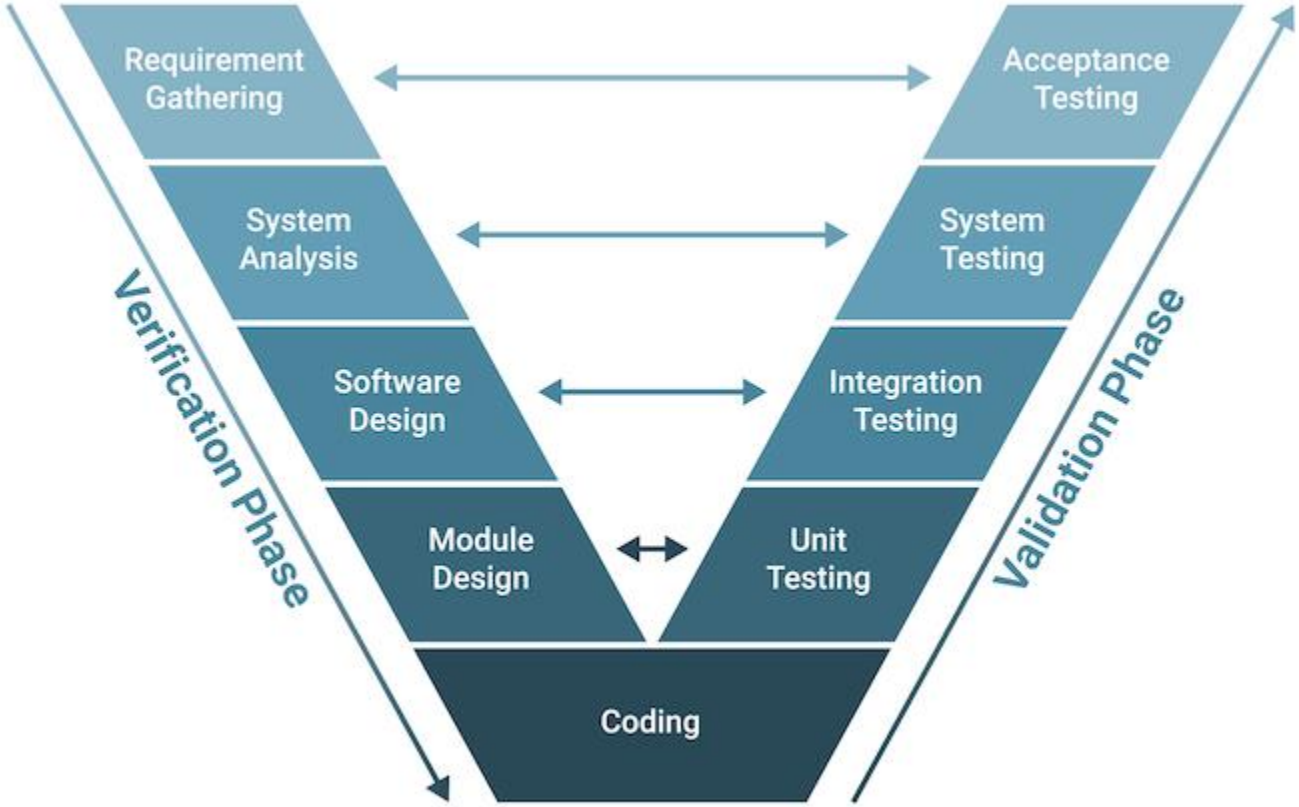
This model helps **catch errors early** by planning tests as soon as each phase begins.

Common Use:

Ideal for **safety-critical systems** like:

- Medical devices
- Automotive systems

V-Model


Requirement Gathering
System Analysis
Software Design
Module Design
Coding
Unit Testing
Integration Testing
System Testing
Acceptance Testing
Verification Phase
Validation Phase

Prototyping Model

- A method where we build a **quick and rough version** of the system (a prototype) to show users early.
  They test it and give feedback, so we can improve the design **before building the final system**.

How it Works:

1. Gather basic requirements
2. Build a quick prototype
3. Show it to the user
4. Get feedback and improve it
5. Once users are happy → build the real system

Prototyping Model

Why Use It?

- Helps understand what the user really wants
- Saves time by avoiding big mistakes
- Great for projects where requirements are unclear or may change

Example:
Making a **university mobile app** – build a sample screen, get student feedback, and improve it before full development

Incremental Model

In this model, the **system is developed step by step** in small parts called **increments**.

Key Features:

- The system is built and delivered in **smaller functional units**, not all at once
- Each increment adds **new features** to the system
- Every increment goes through the **entire SDLC**:
  Requirements → Design → Implementation → Testing → Deployment
- **User feedback** from earlier increments can help improve future ones

Why use it?
It allows early delivery of useful features and supports ongoing improvements.

When to Use the Incremental Model

Use this model when:

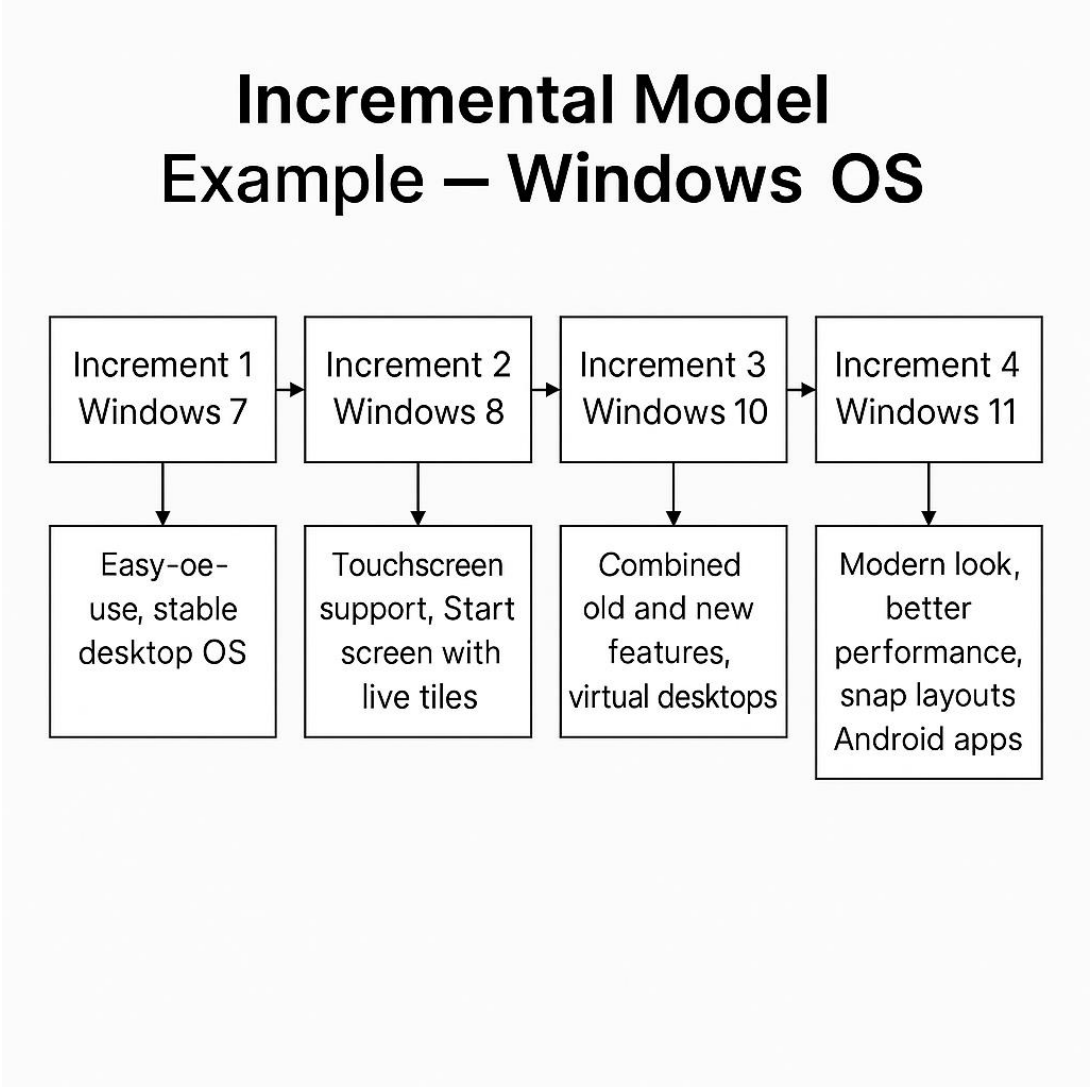
- **Working on very large projects.**
- **Most requirements are known**, but the full system needs to be developed **in phases**
- You need **quick delivery** of working features and **early user feedback**
- You want to **spread out the risk** across multiple stages of development

Example – Windows OS (Version-Based Increments)

The Windows operating system was developed and improved in steps:

- **Increment 1: Windows 7**
- **Increment 2: Windows 8**
- **Increment 3: Windows 10**
- **Increment 4: Windows 11**

Incremental Model



Spiral Model

The Spiral Model is a **risk-focused and iterative** software development approach.
It combines features of both the **Waterfall Model** and **Prototyping Model**.

- Each **loop** in the spiral represents a phase of the project. The project moves through these loops repeatedly, improving the product each time.

Spiral Model

Structure of Each Spiral Loop:

1. Planning

- Set goals, define system requirements
- Identify constraints and possible solutions

2. Risk Analysis

- Identify potential risks (e.g., cost, time, technical challenges)
- Analyze and take steps to reduce or eliminate those risks

3. Engineering

- Design, build, and test part of the system (a prototype or increment)

4. Evaluation

- Customer reviews the work done
- Based on feedback, the next loop (iteration) is planned

When to Use the Spiral Model

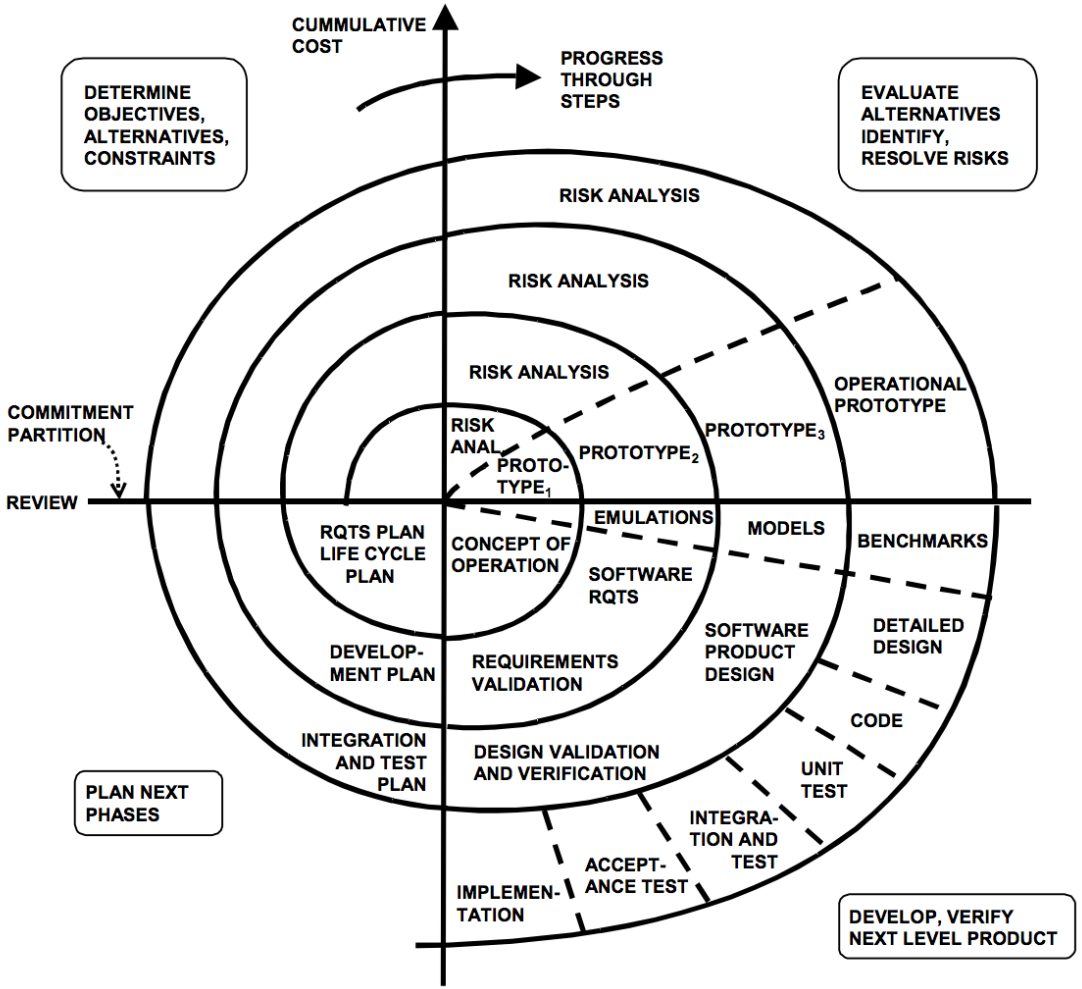
Use this model when:

- The project is **large, complex, or high-risk**
- **Requirements are not fully known** at the beginning and may change over time
- The project needs **frequent reviews, testing, and updates**
- You want to **identify and reduce risks early**

Example:
Defense or aerospace systems, where:

- Mistakes can be **very costly or dangerous**
- Requirements often **change as the system develops**

Spiral Model



Agile Model

An **iterative and incremental** approach to software development that focuses on:

- **Team collaboration**
- **Customer involvement and feedback**
- **Flexibility to change**

Key Features:

- Software is built and delivered in **small usable parts** called **sprints** (usually 1–4 weeks)
- Each sprint delivers a **working product**
- Encourages **continuous improvement** and quick responses to change
- Based on the **Agile Manifesto (2001)** – values **people and interactions** over rigid processes

Agile Model

Example Uses:

- Mobile Applications
- Cloud-based SaaS Products

Best for:
Projects where **requirements may change frequently** and **fast delivery** is important

Agile and Its Common Frameworks

Agile is not one specific method.
It is a **philosophy** based on a set of **values and principles** (from the Agile Manifesto).

To apply Agile in real projects, teams use different **frameworks or methods**.

The two most popular Agile frameworks are:

- **Scrum** – Focuses on working in fixed-length cycles called sprints (usually 2–4 weeks)
- **Kanban** – Focuses on visualizing tasks on a board and managing continuous flow of work

Each framework helps teams **stay flexible**, **collaborate**, and **deliver faster**.

Scrum – Agile Framework – Will discuss in lecture 3 in detail

- Uses **short, fixed-length cycles** called **sprints**
- Focuses on **teamwork**, **feedback**, and **continuous delivery**

Key Roles:

- **Product Owner** – Chooses what to build
- **Scrum Master** – Guides the team
- **Team** – Builds the product

Main Elements:

- **Backlogs** – Lists of work to do
- **Increment** – Working part of the product

Key Meetings:

- **Sprint Planning**, **Daily Stand-up**, **Sprint Review**, **Retrospective**

Model Comparison

Model	Approach	Delivery Style	Best For	Drawbacks
Waterfall	Linear	One final delivery	Simple, well-defined projects	No feedback until the end
Iterative Waterfall	Phased + Feedback	One final delivery	Moderate complexity projects	Still limited flexibility
V-Model	Validation-based	One final delivery	Safety-critical systems (e.g. avionics)	Costly to fix late changes
Prototyping	Build quick model, get feedback, refine	Evolves through user feedback	Unclear or changing requirements	Prototype may be mistaken as final; scope creep
Incremental	Phased	Partial releases	Evolving requirements	Requires stable architecture upfront
Spiral	Risk-driven	Incremental	High-risk, large-scale projects	Complex and expensive
Scrum (Agile)	Sprint-based	Every sprint	Teams needing structure and roles	Rigid schedules and roles
Kanban (Agile)	Flow-based	Continuous	Maintenance and support workflows	May lack clear deadlines

How to Choose a Model?

Factor	Why It Matters
1. Fixed vs. Changing Requirements	Stable requirements suit Waterfall; changing needs call for Agile or Spiral
2. Project Size & Complexity	Large, complex systems need Spiral or V-Model for structure and risk control
3. Risk Level	High-risk projects need risk assessment → Spiral is ideal
4. Time to Market Requirements	Need fast delivery? Choose Agile, Scrum, or Incremental
5. Team Style	Structured teams suit Scrum; flexible teams may prefer Kanban
6. User Feedback	If frequent feedback is critical, Agile-based models are most suitable

Activity

Scenario: You must rapidly deploy a nationwide **COVID-19 vaccination appointment system** while the pandemic is still spreading.

Q: **Which SDLC model would you adopt to deliver usable features quickly, and why?**

SE2030 – Group Assignment Briefing

Your **lecturer will guide you through** the details of the group assignment, including:

- Assignment objectives
- Key milestones and deadlines
- Evaluation criteria and expectations

References

- Pressman & Maxim – Software Engineering: A Practitioner's Approach
  - Chapter 2: Software Process Models
  - Chapter 4: Agile Development
  - Chapter 12: Project Management and DevOps
- SWEBOK v3.0 – Guide to the Software Engineering Body of Knowledge
  - Chapter 2: Software Life Cycle Processes
  - Chapter 3: Software Engineering Models and Methods
- Agile Alliance – Agile Manifesto (2001)
- IEEE Software Engineering Standards – IEEE Std 730, IEEE Std 828

Thank You!