[55958- Aman Goswami, Documentation – 2, Date - 12 Dec 2025]

Different types of Models

Agile Methodology

Agile Methodology is a modern and flexible way of developing software that focuses on delivering value to the customer quickly.

Instead of building the entire product at one time, Agile divides the work into small parts called iterations or sprints.

After each small delivery, the team collects feedback and improves the product. This helps teams adapt easily to change and deliver better-quality software in less time.

Agile Manifesto – What Agile Values

Agile is based on the **Agile Manifesto**, which tells us what is more important during development:

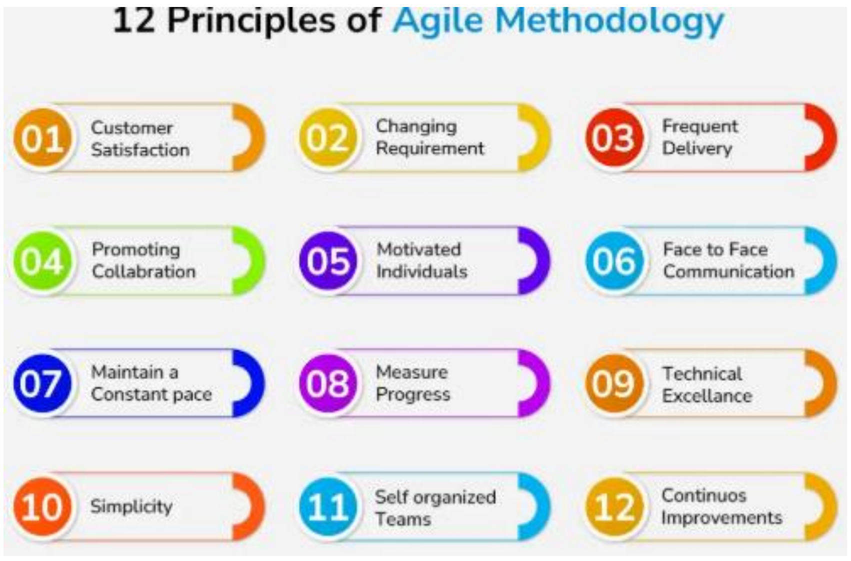
- **Individuals and interactions** are more important than strict processes and tools
- **Working software** is more valuable than heavy documentation
- **Customer collaboration** is better than fixed contracts
- **Responding to change** is better than blindly following a plan

This mindset helps teams stay flexible and focused on real business needs.

Key Principles of Agile

Agile has 12 principles, but the most important ones are:

- Deliver working software frequently
- Accept changes even in the later stages of development
- Business and development teams work closely on a daily basis
- Build projects around motivated and responsible individuals
- Maintain high code quality and technical excellence
- Regularly review work and improve continuously



Agile Development Life Cycle (Based on Methodology Phases)

1. Plan

In this phase, the team understands the requirements and decides what work will be done in the sprint.
The Product Owner prioritizes the backlog, and the team selects tasks based on their capacity.
The goal is to create a realistic and achievable plan.

2. Design

A simple and flexible design is prepared for the selected features.
This may include:

- UI mockups
- Basic architecture decisions
- Database or API changes

The design is kept light so it can change if needed.

3. Develop

The development team starts coding the planned features.
Agile encourages:

- Team collaboration
- Pair programming
- Continuous integration

- Daily stand-up meetings

The goal is to have working software by the end of the sprint.

4. Test

Testing is done within the same sprint, not at the end of the project.
QA tests the software for:

- Functionality
- Integration
- Usability
- Regression issues

If defects are found, they are fixed immediately.

5. Deploy

Once the feature is tested and approved, it is deployed to:

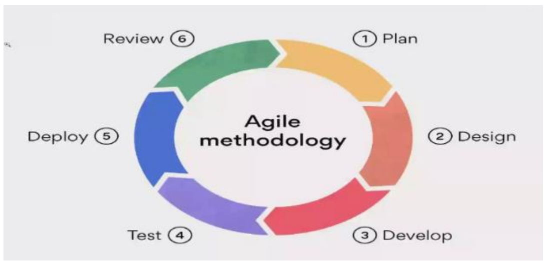
- Staging or
- Production environment

Agile teams deploy frequently—sometimes weekly or even daily—so users get benefits faster.

6. Review

At the end of the sprint, the team demonstrates the completed work to the Product Owner or client.
Feedback is collected and discussed.
This ensures the product is:

- Meeting expectations
- Moving in the right direction



Retrospective (Continuous Improvement)

After the review, the team conducts a Sprint Retrospective.
The team discusses:

- What went well
- What problems were faced
- What can be improved next sprint

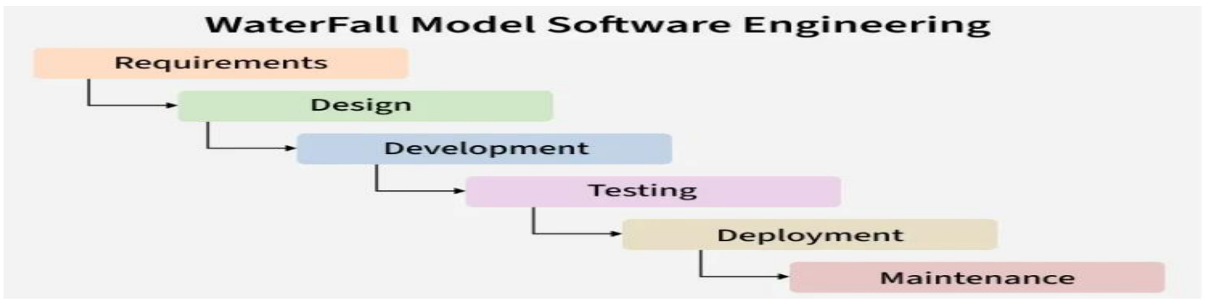
This helps the team learn from experience and improve continuously.

Waterfall Model

The Waterfall Model is one of the oldest and most traditional SDLC models used in software development.

It follows a linear, sequential flow, where each phase must be fully completed before moving on to the next.

Because the process flows downward step by step — like a waterfall — the model is named the Waterfall Model.



Phases of the Waterfall Model

1. Requirement Analysis

In this phase, the team collects **all requirements** from the client in complete detail.
A document is prepared that includes:

- What features the system should have
- How the system should behave
- Any limitations

- Performance needs

Since Waterfall does not allow changes later, requirements must be **final and clear** before moving ahead.

2. System Design

Here, the overall design of the system is created based on the requirements. This includes:

- System architecture
- Database design
- Flow diagrams

This design becomes the **blueprint** for developers.

3. Implementation (Development)

Developers start writing the actual code.
Each module is built separately, and after all modules are completed, they are combined to form the full system.

4. Integration & Testing

Now all modules are connected and tested together. Testing checks:

- Does the system work as expected?
- Are there any bugs?
- Do all parts work properly together?

Fixing problems at this stage is harder because changes affect earlier phases.

5. Deployment

After testing is complete, the software is delivered to the client.
This may include installing the system, giving training, and preparing real data.
Clients start using the software.

6. Maintenance

Even after deployment, small issues may come up.
The team handles:

- Bug fixes

- Updates
- Small improvements
- Performance fixes

Maintenance keeps the system stable over time.

Advantages of Waterfall Model	Disadvantages of Waterfall Model
Simple and easy to understand	Very difficult to make changes once a phase is completed
Each phase is clearly structured	Working software comes very late in the process
Strong and complete documentation	High risk if requirements are misunderstood
Easy to manage because steps follow a fixed order	Bugs found late are more expensive and time-consuming to fix
Suitable for small projects with fixed requirements	Not suitable for changing or complex projects

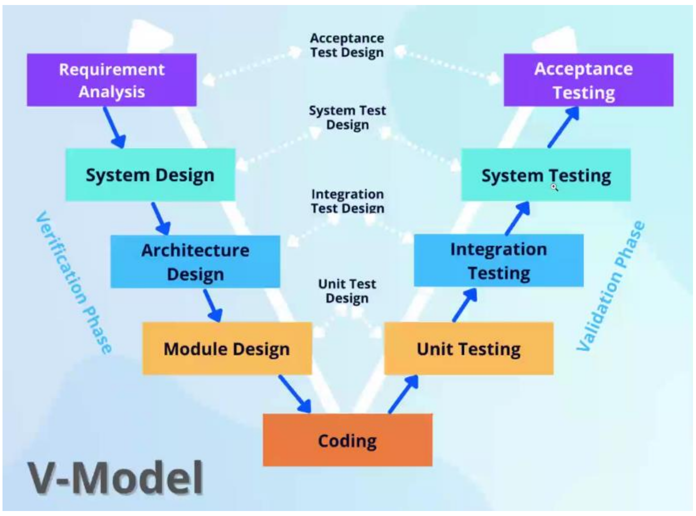
V- Model

The V-Model, also known as the Verification and Validation Model, is an SDLC approach where development and testing activities run in parallel.

It is called the V-Model because the process looks like the letter "V", where the left side represents verification (planning & design) and the right side represents validation (testing).

The main idea of the V-Model is that each development phase has a corresponding testing phase.

This helps detect mistakes early, making the model more reliable than Waterfall.



Why the V-Model Is Used

- Testing starts early, even before coding begins
- Reduces the chances of major defects at the end
- Ensures high quality because every stage is verified and validated
- Best suited for projects where requirements do not change frequently

Phases of the V-Model

The V-Model can be divided into two major sides:

Left Side → Verification (Planning & Design)

Right Side → Validation (Testing)

At the bottom of the V is Implementation (Coding).

Verification Phases (Left Side of the V)

1. Requirement Analysis

Here, the team gathers complete requirements from the client.
The output is the User Requirement Document (URD).
This phase matches with User Acceptance Testing (UAT) on the right side.

2. System Design

The system architecture is planned, including:

- Modules
- Database
- Interfaces
- Data flow

This corresponds to **System Testing** in the validation side.

3. High-Level Design (HLD)

Major components and their interactions are designed.
This includes:

- Module division
- Major functions

- Data tables
- High-level diagrams

This phase aligns with **Integration Testing**.

4. Low-Level Design (LLD)

This phase focuses on detailed design of each module.
It includes:

- Logic of each function
- Pseudocode
- Detailed workflow

This corresponds to **Unit Testing**.

Implementation Phase (Bottom of the V)

5. Coding (Development)

All modules are developed based on LLD.
Once coding is complete, the project moves to the validation side, where all testing phases begin.

Validation Phases (Right Side of the V)

6. Unit Testing

Tests each module individually to verify that the code works as expected.
Matches with the **Low-Level Design** phase.

7. Integration Testing

Tests how different modules work together.
Matches with the **High-Level Design** phase.

8. System Testing

Ensures the entire system functions correctly according to the design.
Matches with the **System Design** phase.

9. User Acceptance Testing (UAT)

Performed by the client to verify that:

- The system meets their needs

- All requirements are fulfilled
- The software is ready for real-world use

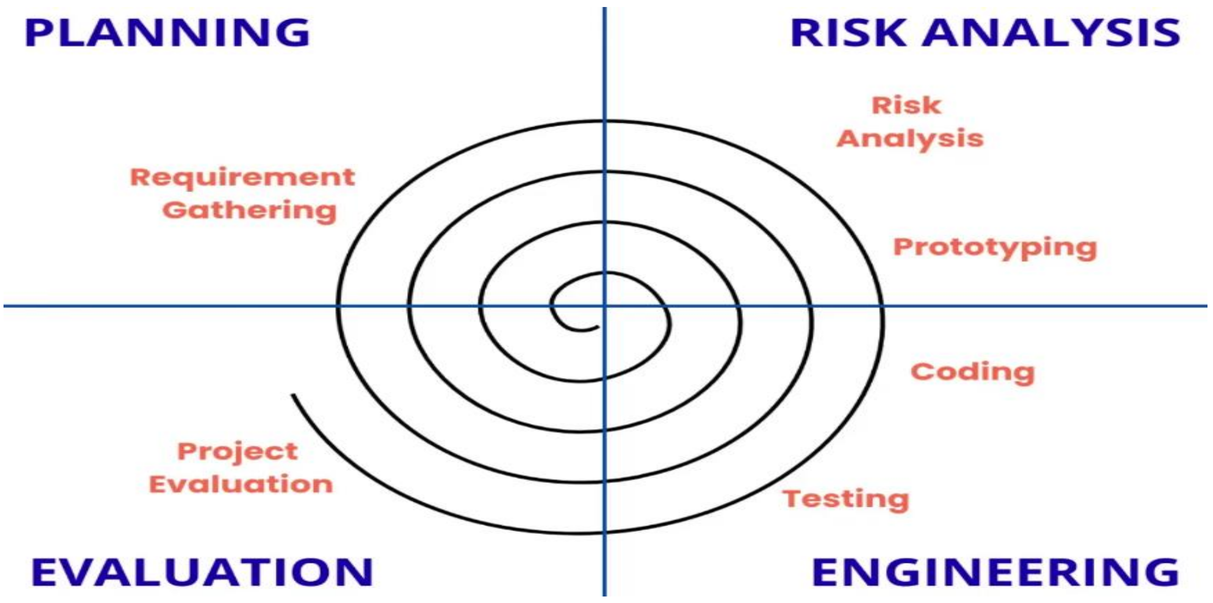
Matches with **Requirement Analysis**.

Spiral Model

The **Spiral Model** is an SDLC approach that combines the **iterative nature of Agile** with the **risk management focus of Waterfall**.

The project is developed in repeated cycles (called **spirals**), and each spiral includes **planning, risk analysis, development, and evaluation.**

This model is especially useful for **large, complex, and high-risk projects** where requirements may not be fully known at the beginning.



Phases of the Spiral Model

Each spiral cycle contains **four major phases**:

1. Planning

In this phase, the team identifies:

- Project objectives
- Requirements
- Features to be built in this spiral
- Constraints and priorities

The output is a plan for what will be developed in the current loop.

2. Risk Analysis (Most Important Phase)

Risk analysis is the heart of the Spiral Model.

The team identifies possible risks such as:

- Technical challenges
- Cost overruns
- Integration failures
- Security issues
- Requirement misunderstandings

To reduce these risks, the team may create:

- Prototypes
- Experiments
- Proof-of-concept models

This step ensures that problems are found early before they become costly.

3. Development

Once risks are handled, development begins.

Depending on the project, this phase may include:

- Designing
- Coding
- Testing
- Integrating modules

Small parts of the system are built in each spiral, and the software grows gradually.

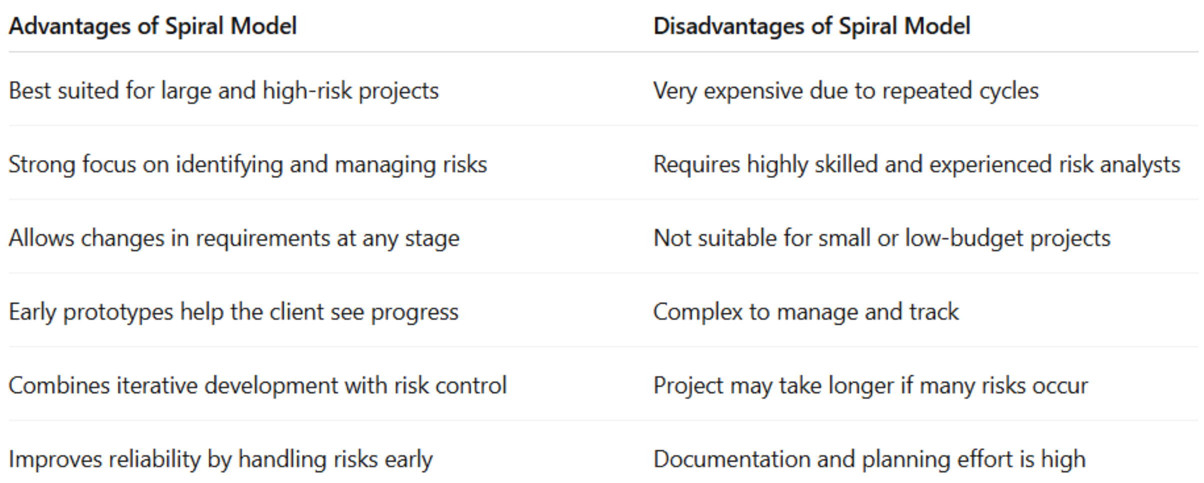
4. Evaluation

At the end of the spiral, the client and stakeholders review the progress. They check if the delivered features meet expectations and decide:

- What changes are needed

- What features should come in the next spiral
- Whether the project should continue or stop

This phase ensures the project stays aligned with real user needs.



Advantages of Spiral Model	Disadvantages of Spiral Model
Best suited for large and high-risk projects	Very expensive due to repeated cycles
Strong focus on identifying and managing risks	Requires highly skilled and experienced risk analysts
Allows changes in requirements at any stage	Not suitable for small or low-budget projects
Early prototypes help the client see progress	Complex to manage and track
Combines iterative development with risk control	Project may take longer if many risks occur
Improves reliability by handling risks early	Documentation and planning effort is high