Process Framework in Software Engineering

(A **process framework** is just a **fixed set of steps** that helps people build **software** in an organized way.)

A **Process Framework** establishes the foundation for a complete software process.

It:

- Identifies a **small number of framework activities**
- Is applicable to **all software projects**, regardless of size or complexity
- Includes a set of **umbrella activities** that are applied throughout the entire software process

Each **framework activity** consists of a set of **software engineering actions**.

Each **software engineering action** (for example, design) contains a collection of related **tasks**, called **task sets**, which include:

- **Work tasks** – specific activities to be performed
- **Work products (deliverables)** – documents, models, code, etc.
- **Quality assurance points** – reviews, testing, validation
- **Project milestones** – checkpoints to measure progress

Generic Process Framework

The **Generic Process Framework** is applicable to the **vast majority of software projects**.

It consists of **five core framework activities**:

1. Communication

This activity focuses on **interaction with the customer and stakeholders**.

- Requirements gathering
- Understanding customer needs
- Clarifying expectations

Goal: To ensure a shared understanding of what the software should do.

2. Planning

This activity establishes a roadmap for the software project.

It includes:

- Identification of **technical tasks**
- Risk estimation
- Resource allocation
- Schedule preparation
- Identification of **work products**

Goal: To define *what will be done, how, when, and by whom*.

3. Modeling

This activity creates models that help developers and customers understand:

- Software requirements
- System design

It consists of **two software engineering actions**:

a) Analysis Action

Work tasks include:

- Requirements gathering
- Requirements elaboration
- Requirements negotiation
- Requirements specification
- Requirements validation

b) Design Action

Work tasks include:

- Data design
- Architectural design
- Interface design
- Component-level design

Goal: To translate requirements into a solution blueprint.

4. Construction

This activity involves:

- Code generation
- Unit testing and integration testing

Goal: To build the software and uncover defects in the code.

5. Deployment

In this activity:

- The software is delivered to the customer
- The customer evaluates the product
- Feedback is collected for improvement or evolution

Goal: To place the software into real use and refine it based on feedback.

Applicability

These five framework activities are suitable for:

- Small programs
- Large web applications
- Large, complex computer-based systems

umbrella Activities

Umbrella activities are applied **across all framework activities** throughout the software life cycle.

1. **Software Project Tracking and Control**
    - Monitors progress against the project plan
    - Ensures schedules are maintained
2. **Risk Management**
    - Identifies and assesses project and product risks
3. **Software Quality Assurance (SQA)**
    - Ensures software meets quality standards
4. **Formal Technical Reviews (FTRs)**
    - Detects and removes errors early
5. **Measurement**
    - Collects process, project, and product metrics
    - Helps meet customer needs
6. **Software Configuration Management (SCM)**
    - Manages changes throughout the software process
7. **Reusability Management**
    - Encourages reuse of existing components
8. **Work Product Preparation and Production**
    - Creation of documents, models, logs, reports, and lists

Capability Maturity Model Integration (CMMI)

CMMI represents a **process meta-model** that helps organizations **assess, improve, and standardize software processes**.
It evaluates **how well processes are defined, managed, measured, and optimized**.

Two Representations of CMMI

1. Continuous Model

- Each **process area is improved independently**
- Different process areas can have **different capability levels**
- Suitable when organizations want **flexible and focused improvement**

2. Staged Model

- Organization is evaluated as a **whole**
- All process areas together determine **one overall maturity level**
- Suitable for **benchmarking and comparison**

Capability Levels in CMMI

Each **process area** is assessed against **specific goals and practices** and rated from **Level 0 to Level 5**.

Level 0 – Incomplete

- Process is **not performed** or only partially performed
- Does **not meet the goals** of Level 1
- Outcomes are unpredictable or missing

➡ *Process is absent or ineffective*

Level 1 – Performed

- All **specific goals** of the process area are satisfied
- Required **work tasks** are performed
- Defined **work products** are produced

➡ *Work is done, but not managed*

Level 2 – Managed

- All Level 1 requirements are satisfied
- Work follows **organizational policies**
- Adequate **resources** are provided
- Stakeholders are **actively involved**
- Work tasks and products are **planned, monitored, controlled, and reviewed**

➡ *Work is planned and tracked*

Level 3 – Defined

- All Level 2 requirements are achieved
- Processes are **tailored from organizational standard processes**
- Tailoring follows **organizational guidelines**
- Process data, measures, and work products contribute to **organizational process assets**

➡ *Standard processes are used across the organization*

Level 4 – Quantitatively Managed

- All Level 3 requirements are achieved
- Processes are **measured and controlled quantitatively**
- **Quality and performance objectives** are defined
- Decisions are based on **metrics and statistical analysis**

➡ *Processes are controlled using data*

Level 5 – Optimized

- All Level 4 requirements are achieved
- Processes are **continuously improved**
- Quantitative methods are used to:
  - Adapt to changing customer needs
  - Improve process efficiency and effectiveness
- Focus on **innovation and defect prevention**

➡ *Processes are continuously optimized*

CMMI – Project Planning (PP)

In CMMI, **specific goals (SGs)** describe *what must be achieved*, and **specific practices (SPs)** describe *how those goals are achieved*.
Specific practices refine each goal into a **set of process-related activities** that help ensure effective project planning.

SG 1: Establish Estimates

This goal focuses on developing **realistic and reliable estimates** for planning the project.

SP 1.1 Estimate the Scope of the Project

- Identify the project boundaries
- Define deliverables and tasks

SP 1.2 Establish Estimates of Work Product and Task Attributes

- Estimate size, complexity, and required resources
- Consider constraints and dependencies

SP 1.3 Define Project Life Cycle

- Select an appropriate life cycle model
- Define project phases and milestones

SP 1.4 Determine Estimates of Effort and Cost

- Estimate required effort (person-hours)
- Calculate overall project cost

SG 2: Develop a Project Plan

This goal ensures a **comprehensive and executable project plan** is created.

SP 2.1 Establish the Budget and Schedule

- Allocate costs and resources

- Create timelines and deadlines

SP 2.2 Identify Project Risks

- Identify technical, schedule, and cost risks
- Plan risk mitigation strategies

SP 2.3 Plan for Data Management

- Decide how project data will be collected, stored, and maintained

SP 2.4 Plan for Needed Knowledge and Skills

- Identify required skills and training needs
- Plan for staffing and skill development

SP 2.5 Plan Stakeholder Involvement

- Identify stakeholders
- Define communication and involvement strategies

SP 2.6 Establish the Project Plan

- Document and finalize the complete project plan
- Ensure all elements are integrated

SG 3: Obtain Commitment to the Plan

This goal ensures that the **project plan is accepted and supported** by all relevant stakeholders.

SP 3.1 Review Plans That Affect the Project

- Review dependent or related plans
- Ensure alignment across teams

SP 3.2 Reconcile Work and Resource Levels

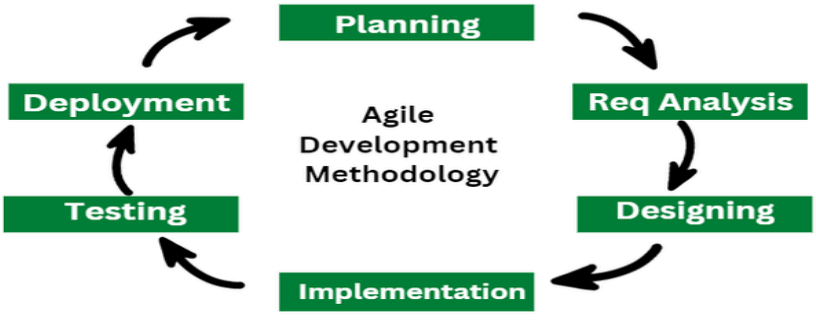
- Ensure available resources match planned work
- Resolve conflicts or shortages

SP 3.3 Obtain Plan Commitment

- Gain formal approval from management and stakeholders
- Ensure team commitment to execute the plan.

Agile Development Stages

The Agile (SDLC) breaks a project into six structured stages, enabling continuous improvement, faster delivery, and better adaptability to change.



Agile Development Stages

Planning

In this stage, the team creates a clear plan for how they'll build the software. They decide which features to focus on in each development cycle (called an iteration). Think of it like mapping out the journey of the project, so everyone knows what to expect and when things will be delivered.

Requirement Gathering

- In this stage, the project team identifies and documents the needs and expectations of various stakeholders, including clients, users, and subject matter experts.
- It involves defining the Project's Scope, objectives, and requirements.
- Establishing a budget and schedule.
- Creating a project plan and allocating resources.

Design

- Developing a high-level system architecture.
- Creating detailed specifications, which include data structures, algorithms, and interfaces.
- Planning for the software's user interface.

Development (Coding)

- Writing the actual code for the software.
- Conducting unit testing to verify the functionality of individual components.

Testing

This phase involves several types of testing:

- **Integration Testing:** Ensuring that different components work together.
- **System Testing:** Testing the entire system as a whole.
- **User Acceptance Testing**: Confirming that the software meets user requirements.
- **Performance Testing:** Assessing the system's speed, scalability, and stability.

Deployment

- Deploying the software to a production environment.
- Put the software into the real world where people can use it.
- Make sure it works smoothly in the real world.
- Providing training and support for end-users.

Review (Maintenance)

- Addressing and resolving any issues that may arise after deployment.
- Releasing updates and patches to enhance the software and address problems.

Key Agile Frameworks

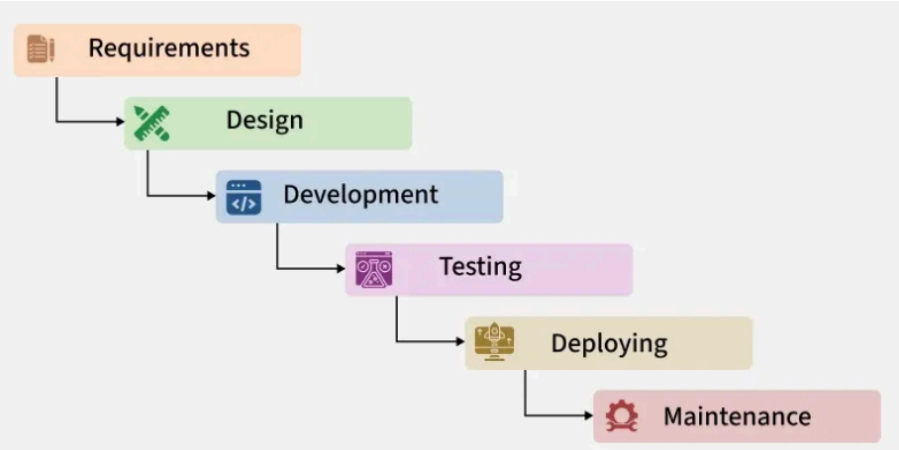
- **Scrum:** Structures work into fixed-length iterations called sprints (typically 1–4 weeks).
- **Kanban:** Visualizes work on a board to manage capacity and flow in real-time.
- **Extreme Programming (XP):** Focuses heavily on technical excellence and rapid feedback.

Benefits of Agile

- **Flexibility:** Teams can easily pivot based on new information.
- **Quality:** Frequent testing and reviews ensure a better end product.
- **Satisfaction:** High stakeholder involvement ensures the product meets user needs.
- **Speed:** Continuous, incremental releases reduce time-to-market.

Waterfall Model

The Waterfall Model is a traditional software development methodology that follows a linear, phase-by-phase approach, where each phase must be completed before moving to the next. It does not allow backtracking and permits only minimal changes once a phase is completed. The model is simple, well-structured, and easy to manage, offering high predictability and clearly defined milestones throughout the development process.



Phases of Waterfall Model

Classical Waterfall Model divides the life cycle into a set of phases. The development process can be considered as a sequential flow in the waterfall. The different sequential phases of the classical waterfall model are follow:

1. Requirements Analysis and Specification

This phase focuses on clearly understanding and documenting the customer's needs.

- **Requirement Gathering and Analysis:**
  Customer requirements are collected and carefully examined to remove errors, confusion, and inconsistencies.

- **Requirement Specification:**
  The approved requirements are documented in the **Software Requirement Specification (SRS)**, which serves as a formal agreement between the customer and the development team.

2. Design

In this phase, the requirements from the SRS are converted into a system design that can be implemented in code.

- **High-Level Design (HLD):**
  Defines the overall system architecture, major components, and their interactions.
- **Low-Level Design (LLD):**
  Provides detailed design of each component, including logic and data flow, to guide developers during coding.
- All designs are documented in the **Software Design Document (SDD)**.

3. Development

This phase involves converting the design into actual working software.

- Developers write source code based on the design documents.
- Suitable programming languages and tools are used.
- **Unit testing** is performed to verify that each module works correctly on its own.

4. Testing and Deployment

This phase ensures that the integrated software functions correctly and is successfully delivered for real world use.

Testing: After unit testing, software modules are integrated incrementally and tested at each stage. Once all modules are successfully integrated, full system testing is performed to ensure the system functions as expected.

Alpha Testing: Performed by the development team.
Beta Testing: Performed by selected end users.
Acceptance Testing: Performed by the customer to approve the software.

Deployment: After successful testing, the software is deployed to a live environment for end users. This phase includes environment setup, user training, and final checks to ensure smooth operation in real world conditions.

5. Maintenance

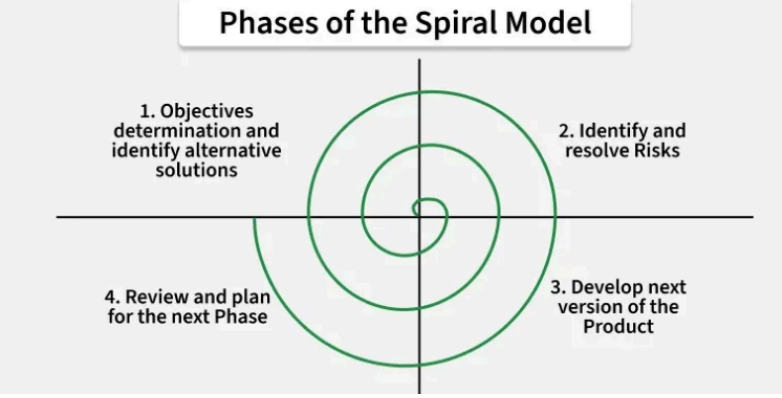
Maintenance ensures the software continues to function effectively after deployment.

- **Corrective Maintenance:** Fixes errors found after release.
- **Perfective Maintenance:** Enhances features based on user needs.

- **Adaptive Maintenance:** Updates software to work in new environments, such as new hardware or operating systems.

THE SPIRAL MODEL

The spiral model, originally proposed by Boehm, is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the **waterfall model.** The spiral model can be adapted to apply throughout the entire life cycle of an application, from concept development to maintenance. It is mainly used for risk handling in large and complex projects.



The Spiral Model is a risk-driven and iterative software development model. Development progresses in a spiral shape consisting of multiple loops, where each loop represents a complete development cycle.
The number of loops depends on project size, complexity, and risk.

Each loop includes:

- Planning
- Risk analysis
- Development
- Evaluation

Phases of the Spiral Model

Focus is on managing risk through multiple iterations of the software development process. Each phase of the Spiral Model is divided into four Quadrants.

1. Objectives Definition

- Project goals and requirements are identified
- Functional and non-functional requirements are analyzed
- Possible solutions are explored

2. Risk Analysis and Resolution

- Risks related to cost, schedule, performance, and technology are identified
- Best solution is selected
- Prototypes are built to reduce risks

3. Development and Testing

- Selected features are designed, developed, and tested
- The next working version of the software is created

4. Review and Planning

- Customers evaluate the current version
- Feedback is collected
- Planning for the next iteration begins

The next iteration of the spiral begins with a new planning phase, based on the results of the evaluation.