Software Engineering & Product Management

Module 1

MODULE-1	SOFTWARE ENGINEERING – INTRODUCTION	22AIM51.1	8 Hours
Software Engineering – Definition, Software life cycle activities, Challenges in System Development, Software process models: Waterfall, Prototyping, spiral, and agile model, Software development methodology.			
Case Study	Investigate the Challenges of System Development, Compare any two Modern software development paradigms		

---

1) Key Principles of Software Engineering

Software Engineering is the process of designing, developing, testing, and maintaining software. It is a systematic and disciplined approach to software development that aims to create high-quality, reliable, and maintainable software.

1. Modularity: Breaking the software into smaller, reusable components that can be developed and tested independently.
2. Abstraction: Hiding the implementation details of a component and exposing only the necessary functionality to other parts of the software.
3. Encapsulation: Wrapping up the data and functions of an object into a single unit, and protecting the internal state of an object from external modifications.
4. Reusability: Creating components that can be used in multiple projects, which can save time and resources.
5. Maintenance: Regularly updating and improving the software to fix bugs, add new features, and address security vulnerabilities.
6. Testing: Verifying that the software meets its requirements and is free of bugs.
7. Design Patterns: Solving recurring problems in software design by providing templates for solving them.
8. Agile methodologies: Using iterative and incremental development processes that focus on customer satisfaction, rapid delivery, and flexibility.
9. Continuous Integration & Deployment: Continuously integrating the code changes and deploying them into the production environment.

2) Objectives of Software Engineering

1. Maintainability: It should be feasible for the software to evolve to meet changing requirements.
2. Efficiency: The software should not make wasteful use of computing devices such as memory, processor cycles, etc.
3. Correctness: A software product is correct if the different requirements specified in the SRS Document have been correctly implemented.
4. Reusability: A software product has good reusability if the different modules of the product can easily be reused to develop new products.

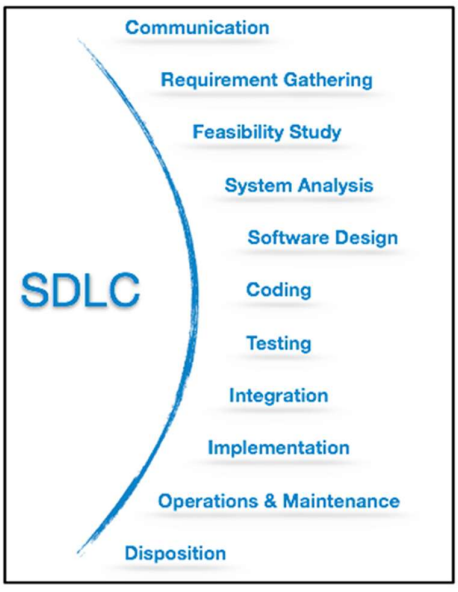
5. Testability: Here software facilitates both the establishment of test criteria and the evaluation of the software concerning those criteria.
6. Reliability: It is an attribute of software quality. The extent to which a program can be expected to perform its desired function, over an arbitrary time period.
7. Portability: In this case, the software can be transferred from one computer system or environment to another.
8. Adaptability: In this case, the software allows differing system constraints and the user needs to be satisfied by making changes to the software.
9. Interoperability: Capability of 2 or more functional units to process data cooperatively.

3) What Tasks do Software Engineers do?

- Requirement Analysis: Collaborating with stakeholders to understand and gather the requirements to design and develop software solutions.
- Design and Development: Creating well-structured, maintainable code that meets the functional requirements and adheres to software design principles.
- Testing and Debugging: Writing and conducting unit tests, integration tests, and debugging code to ensure software is reliable and bug-free.
- Code Review: Participating in code reviews to improve code quality, ensure adherence to standards, and facilitate knowledge sharing among team members.
- Maintenance: Updating and maintaining existing software systems, fixing bugs, and improving performance or adding new features.
- Documentation: Writing clear documentation, including code comments, API documentation, and design documents to help other engineers and future developers understand the system.

4) Software Development Life Cycle (SDLC)

SDLC framework includes the following steps:



1. Communication

The user contacts the company and tells what software they need.
User requests + discussion + written proposal.

2. Requirement Gathering

Developers collect all details about what the user wants.
Interviews, questionnaires, studying old systems, understanding user needs

3. Feasibility Study

Check if the project is possible — practically, technically, and financially.
Can we build it? Is it worth it?

4. System Analysis

Plan how to do the project — what resources, time, and team are needed.
Understand problems, define scope, and plan the roadmap.

5. Software Design

Convert requirements into a blueprint or design.
Create diagrams, data flow, database design, and system architecture.

6. Coding

Programmers write the actual code using suitable languages.
Convert design into working code.

7. Testing

Check for errors and fix them to make sure software works correctly.
Module testing → program testing → full product testing.

8. Integration

Combine all modules and connect software with databases or other systems.
Make all parts work together.

9. Implementation

Install the software for users and do final setup.
Deploy on user systems + test in real use.

10. Operation & Maintenance

Software is used by the client; developers fix bugs and update it as needed.
Support, updates, training, and documentation.

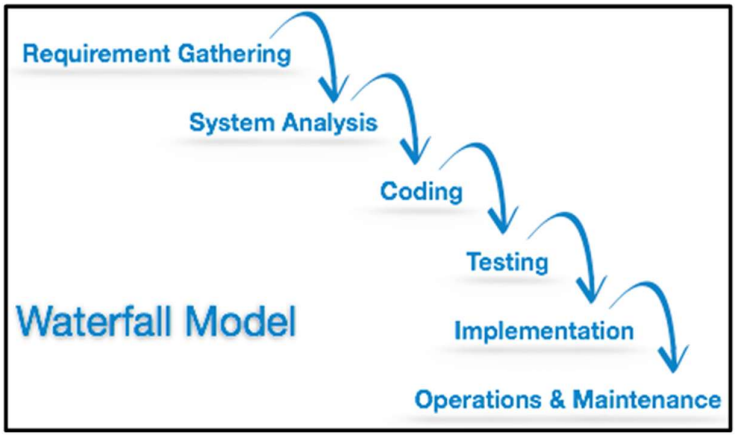
11. Disposition

When software becomes outdated, it's replaced or shut down.
Archive data, close the system properly.

✅ **Quick Tip to Remember Order:**
C R F S D C T I I O D
"Cool Rabbits Fly So Deep, Coding Takes Incredible Intelligent Operations Daily."

5) Waterfall Model

1. Waterfall model is the simplest model of software development.
2. It follows a **linear step-by-step approach** where each phase starts only after the previous one is completed.
3. Once a phase is finished, you **cannot go back** to make changes in the earlier stages.
4. It assumes that **each phase is completed perfectly** before moving to the next one.
5. The model works well when **requirements are clear and stable**.
6. It is best suited when **developers have experience with similar projects**.
7. The model is **not flexible** and **does not handle changes easily**.
8. Any error in the early phase can **affect the entire project**.



Advantages of Waterfall Model

1. Simple and easy to understand and use.
2. Phases are completed one at a time, making it well-structured.
3. Easy to manage due to its sequential nature.
4. Works well for small projects with clear and fixed requirements.
5. Each phase has specific deliverables and a review process.

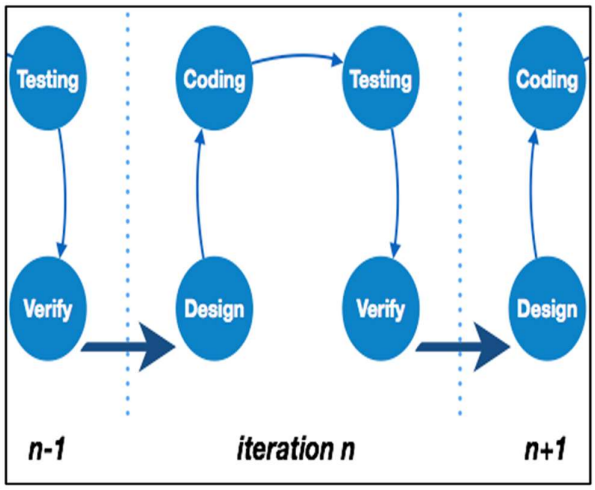
Disadvantages of Waterfall Model

1. Not suitable for complex or changing requirements.
2. Difficult to go back to a previous phase once it is completed.
3. Errors in early stages can cause major problems later.
4. Working software is only available at the end of the process.
5. Poor model for long and ongoing projects that need flexibility.

Example:
Developing a **library management system** where requirements are well-defined (like book issue, return, and user records).
Since the process is clear, each step can be done one by one — making the Waterfall model suitable.

5) Iterative Model

1. The Iterative Model develops software **in small parts (iterations or cycles)**.
2. Each iteration includes all steps of SDLC — **requirement, design, coding, testing, and implementation**.
3. In the **first cycle**, a simple version of the software is built.
4. In **each next cycle**, new features and improvements are added.
5. After every iteration, the product becomes **more complete and refined**.
6. It allows **risk management and feedback** after each iteration.
7. Easier to manage since development happens in smaller parts, but it **requires more time and resources**.



Example

Developing a **social media app**:

- **First iteration:** Basic user login and profile creation.
- **Second iteration:** Add friend list and messaging features.
- **Third iteration:** Add photo sharing and notifications.
  Each cycle adds new features until the full product is complete.

Advantages of Iterative Model

1. Early working version of the software is available in the first iteration.
2. Errors can be detected and corrected early through repeated testing.
3. Easier to manage smaller parts of the project in each cycle.
4. Feedback from users can be used to improve the next version.

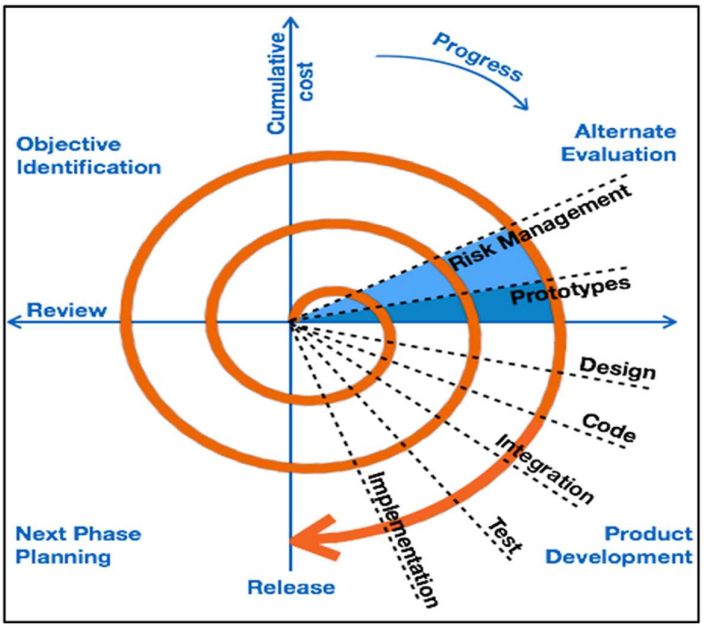
5. Reduces overall risk because each iteration is reviewed and improved.

Disadvantages of Iterative Model

1. Requires more time and resources due to repeated cycles.
2. Not suitable if requirements are not properly defined in the beginning.
3. System architecture may need major changes after many iterations.
4. Continuous involvement of users and developers is needed.
5. Managing too many iterations can become complex for large projects.

6) Spiral Model

1. The Spiral Model is a **combination of the Iterative Model and the Waterfall Model**.
2. Development is done in **cycles (spirals)** — each cycle includes planning, risk analysis, development, and evaluation.
3. **Risk analysis** is a major focus at every stage.
4. In each spiral, a **prototype** is developed and refined based on feedback.
5. After each cycle, the project moves to the next level with more detailed design and features.
6. Suitable for **large, complex, and high-risk projects**.



Developing an **online banking system**:

- First spiral: Identify requirements and create a prototype for account login.
- Second spiral: Add features for fund transfer and bill payments.
- Third spiral: Add security modules and mobile app features.
  At each stage, risks (like data security) are analyzed before moving ahead.

Advantages

1. Focuses on **risk identification and management**.
2. Provides **early prototypes** for user feedback.
3. Changes can be made after each iteration.
4. Suitable for **large and complex projects**.
5. Ensures **better project control** and gradual improvement.

Disadvantages

1. **Costly and time-consuming** due to repeated cycles.
2. **Requires expertise** in risk analysis and management.
3. Not suitable for **small or low-risk projects**.
4. Hard to manage if the number of cycles becomes large.
5. **Complex to implement** compared to simpler models.

7) Challenges in System Development

1. **Lack of Direction and Leadership**
    - Without proper guidance from project managers, teams lose focus.
    - Leads to confusion, poor coordination, and project failure.
2. **Difficulty in Estimating Time and Resources**
    - Hard to plan accurate timelines and budgets.
    - Causes project delays or cost overruns if not managed properly.
3. **Lack of Skilled Developers / Resources**
    - Shortage of experienced developers or required tools.
    - Increases workload and reduces software quality.
    - Solution: Training, outsourcing, or using open-source tools.
4. **Unclear Requirements**
    - Poorly defined goals or scope lead to confusion and rework.
    - Solution: Discuss with users, prepare prototypes, and get feedback early.
5. **Communication Barriers**
    - Miscommunication causes errors and misunderstandings.
    - Solution: Regular meetings, clear documentation, and open communication tools.
6. **Rigid Deadlines (Time Pressure)**
    - Unrealistic timelines reduce quality and increase stress.
    - Solution: Set realistic deadlines and allow buffer time for issues.
7. **Over-Complicated Projects**
    - Adding unnecessary features increases complexity.
    - Solution: Keep project scope clear and focus on essential features.
8. **Testing and Debugging Issues**
    - Finding and fixing errors takes time and effort.
    - Solution: Use test-driven development (TDD) and automated testing tools.
9. **Maintaining Competitiveness**
    - Rapidly changing technology makes software outdated quickly.
    - Solution: Stay updated with latest tools and invest in R&D.
10. **Wrong Development Methodology**

- Using an unsuitable model (like Agile, Waterfall, etc.) can harm progress.

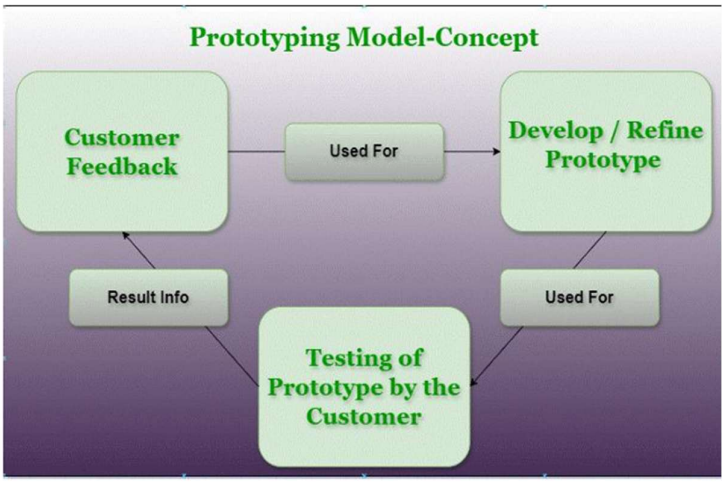
- Solution: Choose the model that fits the project's needs.

8) Prototyping Model

The **Prototyping Model** is a **software development approach** where a **prototype (sample model)** of the final product is built first.
It is shown to the customer for feedback, and the prototype is **refined repeatedly** until the user is satisfied.
Then, the final system is developed based on the approved prototype.

Phases of the Prototyping Model

1. **Requirement Gathering and Analysis**
   - Collect requirements through discussions, interviews, or questionnaires.
   - Understand what the customer expects from the system.
2. **Quick Design**
   - Create a simple preliminary design showing basic features and layout.
3. **Build Prototype**
   - Develop a small working model (basic version) based on the quick design.
4. **Initial User Evaluation**
   - Show the prototype to the user for feedback and suggestions.
5. **Refine Prototype**
   - Modify the prototype as per user feedback until the user is satisfied.
6. **Implement and Maintain**
   - Develop the **final system** based on the approved prototype.
   - Test, install, and maintain the system regularly.



Example:
Developing a **new mobile app** for a company where users are not sure what features they want — a prototype helps them visualize it.

Types of Prototyping

1. **Rapid Throwaway Prototyping**
   - Build a prototype quickly, gather feedback, then **discard it**.
   - Helps avoid design mistakes early.
2. **Evolutionary Prototyping**
   - Prototype is **continuously improved** until it becomes the final product.
   - Saves time and effort compared to throwaway prototyping.
3. **Incremental Prototyping**
   - System is divided into **small prototypes (modules)** developed separately.
   - Later, all modules are combined into the final system.
4. **Extreme Prototyping** (used in web apps)
   - Step 1: Create static web pages (HTML).
   - Step 2: Add simulated data.
   - Step 3: Implement real services.

9) Agile Model

- The **Agile Model** is a **software development approach** that focuses on **quick adaptation to changes** and **fast delivery** of working software.
- It is **a combination of iterative and incremental models** — development is done in small cycles called **iterations or sprints** (usually 1–4 weeks).
- The goal is to deliver a **working product quickly**, gather **feedback**, and make continuous improvements.

Phases of Agile Model

1. **Requirement Gathering**
   - Collect requirements by interacting with customers.
   - Identify goals, estimate effort and time, and check feasibility.
2. **Design the Requirements**
   - Create system architecture, UI wireframes, and UML diagrams.
   - Build small prototypes for user feedback.
3. **Construction / Iteration**
   - Develop the software in short cycles (1–4 weeks).
   - Each iteration produces a **working version** of the product with more features added gradually.
4. **Testing / Quality Assurance**
   - Test each iteration thoroughly using:
     - **Unit Testing:** Check individual code units.
     - **Integration Testing:** Test combined modules.
     - **System Testing:** Ensure the system meets all user needs.
5. **Deployment**
   - Release the working version to users after every iteration.
   - Deployment happens **regularly**, not just once.
6. **Feedback**
   - Gather customer feedback after each iteration.
   - Fix bugs, improve features, and plan the next sprint based on feedback.

Common Agile Methods

1. **Test-Driven Development (TDD):** Write tests before writing code.
2. **Behavior-Driven Development (BDD):** Focus on user behavior and interaction.
3. **Exploratory Testing:** Testers explore the software freely to find bugs.
4. **Acceptance Test-Driven Development (ATDD):** Developers, testers, and customers define acceptance criteria together.
5. **Extreme Programming (XP):** Emphasizes continuous feedback and frequent releases.
6. **Session-Based Testing:** Time-boxed testing sessions with structured documentation.
7. **Dynamic System Development Method (DSDM):** Framework for quick and controlled delivery.
8. **Crystal Methodologies:** Focus on people and communication rather than tools and rules.

Example

Developing a **food delivery app** using Agile:

- Each sprint delivers a part — e.g., login page, restaurant list, order system, payment, etc.
- After each release, user feedback is collected and improvements are made in the next sprint.

10) Software Development Methodologies