Software Process Models

MCA SEMESTER 1 – SOFTWARE PROJECT MANAGEMENT- MODULE 2

Learning Objectives

- Understand traditional and modern software process models.
- Compare strengths and weaknesses of different models.
- Explore Agile, Scrum, and DevOps in practice.
- Apply models to real-world project scenarios.

What is a Process Model?

• Framework that describes the activities involved in software development.

• Ensures systematic, repeatable, and quality-driven development.

Why important?

- Reduces risks.

- Improves predictability.

- Supports planning & control.

Analogy: Building a House

- Design = Requirements
- Architect Plan = System Design
- Construction = Coding
- Inspection = Testing
- Handover = Deployment

☑ Key Takeaway: Process Models act as roadmaps – they enforce discipline, minimize surprises, and provide confidence to the team and client.

Waterfall Model

The **Waterfall Model** is the **oldest and simplest** software process model.

It follows a **linear and sequential flow**, meaning each phase must be completed before the next begins — like a waterfall flowing downwards.

Once you move to the next phase, you rarely go back.

Phases of the Waterfall Model

1.Requirement Analysis – Collect all user requirements.

2.System Design – Create architecture, database, and UI design.

3.Implementation (Coding) – Write the actual code.

4.Testing – Check for bugs and errors.

5.Deployment – Deliver the product to the customer.

6.Maintenance – Fix issues and provide updates after release.

Waterfall Model

Example

Suppose you're developing a **Payroll Management System** for a company:

- Step 1: Gather all rules for salary, bonuses, leaves.
- Step 2: Design input forms, database, and reports.
- Step 3: Code it.
- Step 4: Test thoroughly.
- Step 5: Deliver to HR.

Once the system is delivered, any major change (like new salary structure) would require starting over.

Waterfall Model

Advantages:

- Simple and easy to understand.
- Works well for **small, clearly defined projects**.
- Easy to manage — each phase has a fixed deliverable.

Limitations:

- No room for changes once development starts.
- Late discovery of issues (testing happens at the end).
- User feedback comes only after full development.

Prototyping Model

The **Prototype Model** is based on the idea of *"build something quickly -> get feedback -> improve it"*.

Instead of waiting till the end to show the full product, you create a **mock-up (prototype)** early in the process to help users visualize how the final software will look and behave.

It's a **trial-and-error model** — developers refine the prototype based on user feedback until it meets user expectations.

When to Use:

•When **requirements are unclear** or keep changing.

•When **user involvement** is high.

•Ideal for **UI-heavy applications** (like websites, apps, or dashboards).

Prototyping Model

Phases in the Prototype Model

1. **Requirement Identification:**
   Gather initial requirements from the user — not in full detail yet.

2. **Quick Design:**
   Create a **rough design** or mock interface (like a clickable UI, sample screens, or limited functions).

3. **Build Prototype:**
   Develop a working model demonstrating key features.

4. **User Evaluation:**
   Show the prototype to users → collect their feedback.

5. **Refine Prototype:**
   Modify and enhance based on feedback.

6. **Engineer Product:**
   Once users are satisfied, convert the final prototype into a fully functional system.

7. **Test & Implement.**

Prototyping Model

Imagine developing a **food delivery app** like *Swiggy*.

1. Users say they want to “order quickly and track live status.”
2. You create a **prototype** showing:
    1. Home screen with restaurant list
    2. “Add to cart” flow
    3. Live tracking page (dummy animation)
3. Users test it and suggest:
    1. Add “Filter by cuisine”
    2. Change color scheme
    3. Make “Order again” button visible on home
4. You refine and rebuild until users are happy.
5. Then, you build the full functional app.

Prototyping Model

Advantages

- Early user involvement ensures satisfaction.
- Detects missing or misunderstood requirements early.
- Saves time and cost of rework later.

Limitations

- Users might confuse prototype with final system.
- Frequent changes can cause project delays.
- Documentation might suffer due to quick iterations.

Spiral Model

The **Spiral Model** combines elements of the **Waterfall** and **Prototype** models, with a **strong focus on risk management**.

It looks like a **spiral** — each loop (iteration) represents one phase of the software.
Every loop includes:

1. Planning
2. Risk Analysis
3. Engineering
4. Evaluation

At the end of each loop, the project moves closer to completion.

Spiral Model

Phases of Spiral Model

1. **Planning Phase**
   Define objectives, alternatives, and constraints for this iteration.
2. **Risk Analysis Phase**
   Identify potential risks (technical, cost, schedule, etc.) and create strategies to handle them.
3. **Engineering Phase**
   Develop and test the product for that iteration.
4. **Customer Evaluation Phase**
   Get feedback and decide whether to continue to the next spiral loop.

Each loop adds more detail and features until the final system is complete.

Spiral Model

Advantages

- Excellent for **large, complex, and high-risk projects**.
- Early identification and handling of risks.
- Allows **continuous refinement** through feedback.
- Incorporates **prototyping and iterative development**.

Limitations

- Expensive due to repeated risk analysis.
- Requires **expertise** in risk assessment.
- Not ideal for small or low-budget projects.

Spiral Model

Feature	Prototype Model	Spiral Model
Focus	Rapid user feedback	Risk management & evolution
Type	Iterative	Evolutionary + Risk-driven
Best for	UI-heavy apps, unclear requirements	Complex, long-term, high-risk projects
User Involvement	High	Moderate–High
Cost	Moderate	High
Example	Food Delivery App	Banking / Defense Systems

Agile Development

The **Agile Model** is **iterative, flexible, and customer-focused**.

Instead of building the whole project at once, Agile breaks it into **small parts called "sprints"** (usually 1–4 weeks each).

At the end of each sprint, you deliver a **working product** and take **customer feedback** for improvement.

Agile values **individuals and interactions** over processes and documentation.

Agile Development

Phases (in every Sprint Cycle)

1. **Planning** – Define what to build in this sprint.
2. **Design & Development** – Build the planned features.
3. **Testing** – Check the new features immediately.
4. **Review** – Demonstrate to the customer and collect feedback.
5. **Next Sprint** – Refine, add more features, and repeat.

Example:

Suppose you're developing a **Food Ordering App**:

- **Sprint 1:** Create login + restaurant listing.
- **Sprint 2:** Add "Add to Cart" and "Checkout" features.
- **Sprint 3:** Integrate "Live Order Tracking."

At the end of each sprint, the client gets a working version to test.

Agile Development

Advantages

•Highly **flexible and adaptive** to change.
•Continuous **user feedback** and satisfaction.
•Early delivery of usable software.
•Encourages teamwork and transparency.

Limitations
•Needs experienced, disciplined teams.
•Hard to estimate total time and cost upfront.
•Requires active customer participation throughout.

Scrum Framework

Concept

Scrum is a **framework within the Agile model** — used to manage and deliver projects in short, time-boxed cycles called **Sprints** (usually 1–4 weeks).

It’s based on teamwork, accountability, and continuous improvement. Each sprint delivers a **usable product increment**, reviewed and improved before the next sprint.

Scrum Framework

Key Components of Scrum

👥 **Roles**

1. **Product Owner** – Represents the customer; defines *what* to build.
2. **Scrum Master** – Ensures the team follows Scrum rules and removes obstacles.
3. **Development Team** – Designers, developers, testers — they *build* the product.

🧱 **Artifacts**

1. **Product Backlog** – List of all features (like a "to-do" list for the project).
2. **Sprint Backlog** – Subset of backlog items selected for the current sprint.
3. **Increment** – The working version of the product delivered at the end of each sprint.

🔄 **Events**

1. **Sprint Planning** – Decide what to build this sprint.
2. **Daily Scrum (Stand-up)** – 15-minute daily meeting to discuss progress and issues.
3. **Sprint Review** – Show the work done to the Product Owner; get feedback.
4. **Sprint Retrospective** – Team reflects on what went well and what to improve next sprint.

Scrum Framework

Example

Let's say you're building a **Fitness App**.

- The **Product Backlog** includes: login, profile setup, live session link, reminders, diet tracker.
- In **Sprint 1**, the team picks "login + profile setup."
- Daily stand-ups ensure everyone's aligned.
- At the end of Sprint 1 → a working login + profile setup is demonstrated.
- In **Sprint 2**, add live sessions and notifications, and so on.

Each sprint ends with a working product update ready for users.

Scrum Framework

Advantages

- Continuous delivery of usable product.
- Easy to adapt to changing priorities.
- Promotes teamwork, transparency, and accountability.
- Regular feedback improves product quality.

Limitations

- Needs strong team coordination.
- Hard to use in very large or distributed teams.
- Scope changes too often if Product Owner is unclear.

DevOps Model

- Integrates Development & Operations.
- Goals: Continuous integration, continuous delivery (CI/CD).
- Advantages: Faster releases, fewer errors, better collaboration.

Tools: Jenkins, Docker, Kubernetes.

DevOps Model

Concept

DevOps = **Development + Operations**.
It's not just a model — it's a **culture and set of practices** that unify developers (who build software) and operations teams (who deploy and maintain it).

Goal: **Continuous Integration and Continuous Delivery (CI/CD)**
→ Code is written, tested, deployed, and monitored **automatically and continuously**.

It bridges the gap between **coding** and **deployment** — ensuring faster releases with fewer bugs.

DevOps Model

Phases of DevOps Cycle

1. **Plan:** Define new features, updates, or fixes.
2. **Code:** Developers write and push code to a shared repository (e.g., GitHub).
3. **Build:** Automated tools (like Jenkins) compile and package the code.
4. **Test:** Automated testing ensures quality before release.
5. **Deploy:** Software is automatically released to production (e.g., AWS, Azure).
6. **Operate:** Monitor system performance, uptime, and security.
7. **Feedback:** Collect user feedback and performance data to improve the next cycle.

This cycle repeats continuously — ensuring fast, stable, and reliable releases.

DevOps Model

Example

Imagine an **E-commerce Platform** like Amazon:

- A new “Wishlist” feature is planned → coded → tested → automatically deployed to production.
- DevOps tools (like **GitHub + Jenkins + Docker + AWS**) ensure:
  - Code merges automatically.
  - Testing runs instantly.
  - Deployment happens without downtime.
- Monitoring tools like **Prometheus or Grafana** track real-time performance.

This means new updates can go live **multiple times a day** — safely and fast.

DevOps Model

Advantages

- Faster release cycles.
- Fewer deployment failures.
- Continuous improvement through monitoring and feedback.
- Breaks the wall between development and operations teams.

Limitations

- Requires strong automation infrastructure.
- Cultural shift — dev and ops teams must truly collaborate.
- High initial setup cost.

Model Comparison

Model	Advantages	Disadvantages	Best Suited For
Waterfall	Simple, structured	Rigid, late testing	Stable requirements
Prototype	User feedback early	Scope creep	UI-heavy apps
Spiral	Handles risk	Expensive, complex	Large risky projects
Agile/Scrum	Flexible, fast delivery	Needs customer involvement	Dynamic projects
DevOps	Continuous delivery	Tool-heavy	Cloud/software products

Agile = Mindset of flexibility & collaboration.
Scrum = Structured way to apply Agile through sprints.
DevOps = Extends Agile into deployment — continuous delivery and monitoring.

Industry Example

Case: Mobile Banking App

- Waterfall for compliance features.

- Agile/Scrum for UI/UX features.

- DevOps for deployment pipeline.

Exercises & Activities

- Activity: Map a 'Food Delivery App' into Agile sprints.
- Group Discussion: Compare Spiral vs Agile for a healthcare system.

Quick Recap

- Process models guide software development.
- Waterfall, Prototype, Spiral → Traditional models.
- Agile, Scrum, DevOps → Modern models.
- Model selection depends on project type, size, and risk.

Sample Exam Questions

1. Explain the Spiral Model with a neat diagram.
2. What are the pillars of Scrum? Give an example.
3. Compare Agile and Waterfall with examples.
4. How does DevOps improve software delivery?