SOFTWARE ENGINEERING

UNIT – 1

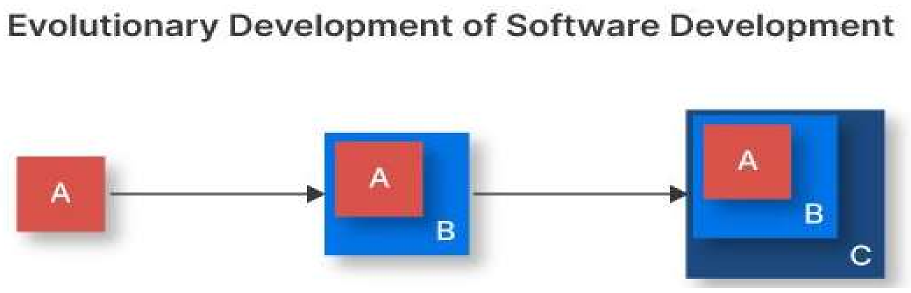
TOPIC – 6

PROCESS MODELS – EVOLUTIONARY PROCESS MODELS

1. Introduction to Evolutionary Process Models

- **What are Evolutionary Process Models?**

  Evolutionary process models are flexible software development approaches that allow continuous improvement and adaptation of the software product based on user feedback. Unlike rigid models like Waterfall, evolutionary models embrace change, enabling developers to modify the software at any stage to meet evolving needs.



Example: Imagine you are developing a shopping app. Initially, you release a basic version of the app with essential features like browsing and adding items to a cart. As users provide feedback, you keep updating and improving the app, adding features like payment options and user profiles over time.

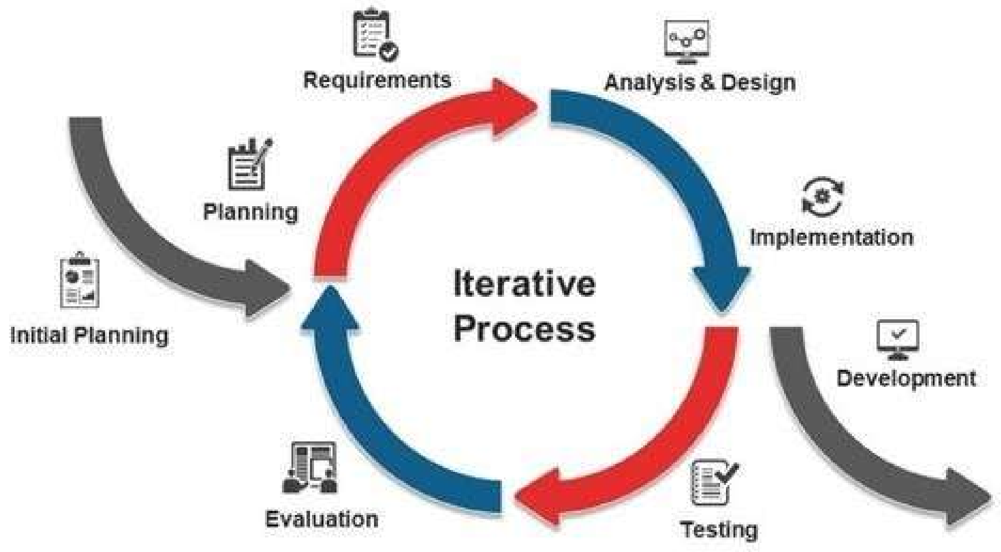
2. Iterative Model

- **Concept**:
  The Iterative Model is a development process where the system is built incrementally in small steps, with each iteration refining and improving the product based on user feedback.

Steps in the Iterative Model:

1. **Initial Planning**: Establish the overall goals, initial scope, and objectives for the project.
2. **Requirements**: Gather and define the detailed requirements for the specific iteration.
3. **Analysis & Design**: Create a design plan based on the requirements, focusing on how the system will be built.
4. **Implementation**: Write and integrate the code to develop the software for the current iteration.
5. **Development**: Ensure that the product is built according to the design and includes the necessary functionalities.
6. **Testing**: Validate the software through various tests, gather user feedback, and detect any issues.
7. **Evaluation**: Review the results of testing and user feedback to assess the quality of the iteration.
8. **Refinement (Planning for next iteration)**: Based on feedback, refine the design and plan for improvements in the next iteration.

The process repeats, continuously refining the system through each cycle of iteration until the software reaches its final form.

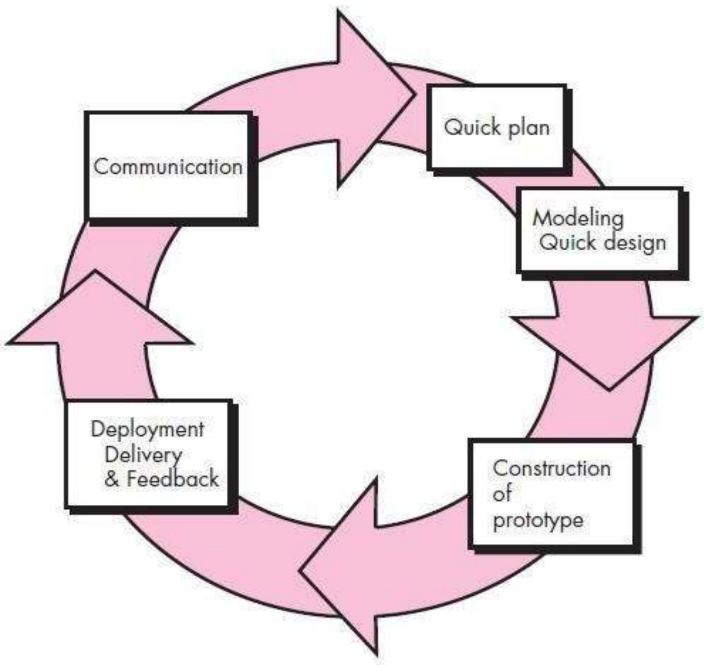


Example: Developing an e-learning platform by first launching basic courses. As time goes on, you add interactive features, more courses, and assessments, improving the platform with each iteration.

- **Advantages:**
    - **Early Problem Detection**: Issues are identified and addressed early in the development process.
    - **Continuous Feedback**: Regular feedback helps refine the product.
    - **Improved Quality**: The product improves with each iteration.
- **Disadvantages:**
    - **Time-Consuming**: Repeated cycles can take more time.
    - **Costly**: Multiple iterations and changes can increase development costs.
- **When to Use:** The Iterative Model is ideal when the project requires repeated refinements and when requirements are not fully understood from the start.

3. Prototype Model

- **Concept:**
  The Prototype Model involves building a simple, working version of the software (a prototype) to let users interact with it and provide feedback. The prototype helps clarify requirements and improve the final product before full-scale development.



Steps in the Prototype Model:

1. **Communication**: Engage with users to discuss and gather requirements for the system.
2. **Quick Plan**: Formulate a simple plan for building the prototype.
3. **Modelling/Quick Design**: Develop a basic model or quick design of the system, focusing on core functionalities.
4. **Construction of Prototype**: Build the prototype, creating an initial, simplified version of the software.
5. **Deployment, Delivery & Feedback**: Deploy the prototype to the users, delivering it for evaluation and collecting their feedback.
6. **Refinement through Communication**: Use the feedback gathered in the previous step to enhance and improve the prototype. This cycle is repeated until the system meets user expectations.

This iterative cycle continues until the final version of the system is developed.

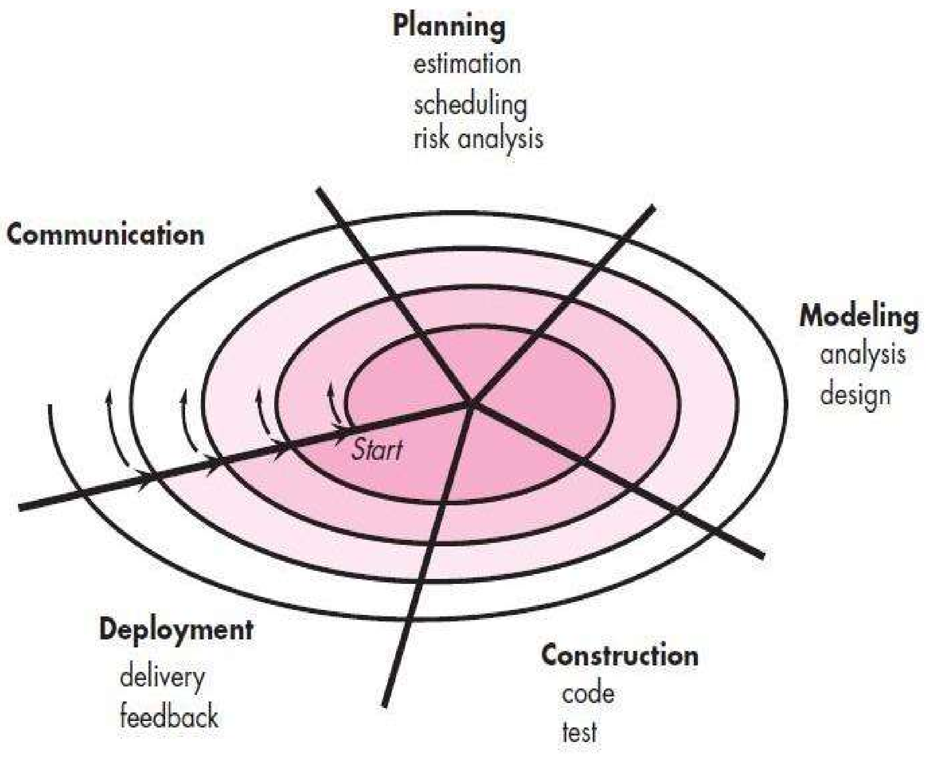
Example: Developing a mobile app for a new business. A simple version is quickly created and shown to users for feedback. Based on their input, the app is refined until it meets the users' needs.

- **Advantages:**
  - **Early Feedback**: Users can see and interact with the system early on.
  - **Flexible**: Easy to make changes during development.
  - **Risk Reduction**: Identifies potential problems or missing features early.
- **Disadvantages:**
  - **Time-Consuming**: Repeated revisions can take a lot of time.
  - **Incomplete System**: Users might confuse the prototype with the final product.
  - **Costly**: Multiple versions of the software can be expensive to develop.
- **When to Use:** The Prototype Model is best when users are unsure of their exact requirements, or when the project involves new or innovative features that need user validation.

4. Spiral Model

- **Concept:**

  The Spiral Model combines iterative development with systematic risk management. It is designed to manage risks by allowing the software to be developed in cycles, with each cycle addressing specific risks and adding new features.



Steps in the Spiral Model:

1. **Communication**: This initial step involves gathering and understanding requirements from users or stakeholders. It's crucial to ensure that everyone is on the same page before proceeding.
2. **Planning**: In this phase, you estimate the project cost, time, and resources. You also conduct risk analysis to identify potential challenges, such as technical difficulties, budget constraints, and market acceptance.
3. **Modelling**: This step focuses on the analysis and design of the system. You create models that represent the structure and behaviour of the system, which helps in visualizing the final product.

4. **Construction**: Here, the actual coding and testing take place. The system or its components are developed, tested, and refined based on the design created in the previous step.
5. **Deployment**: After construction, the system is delivered to the users. Feedback is gathered, which is essential for the next iteration of the spiral, ensuring continuous improvement of the system.

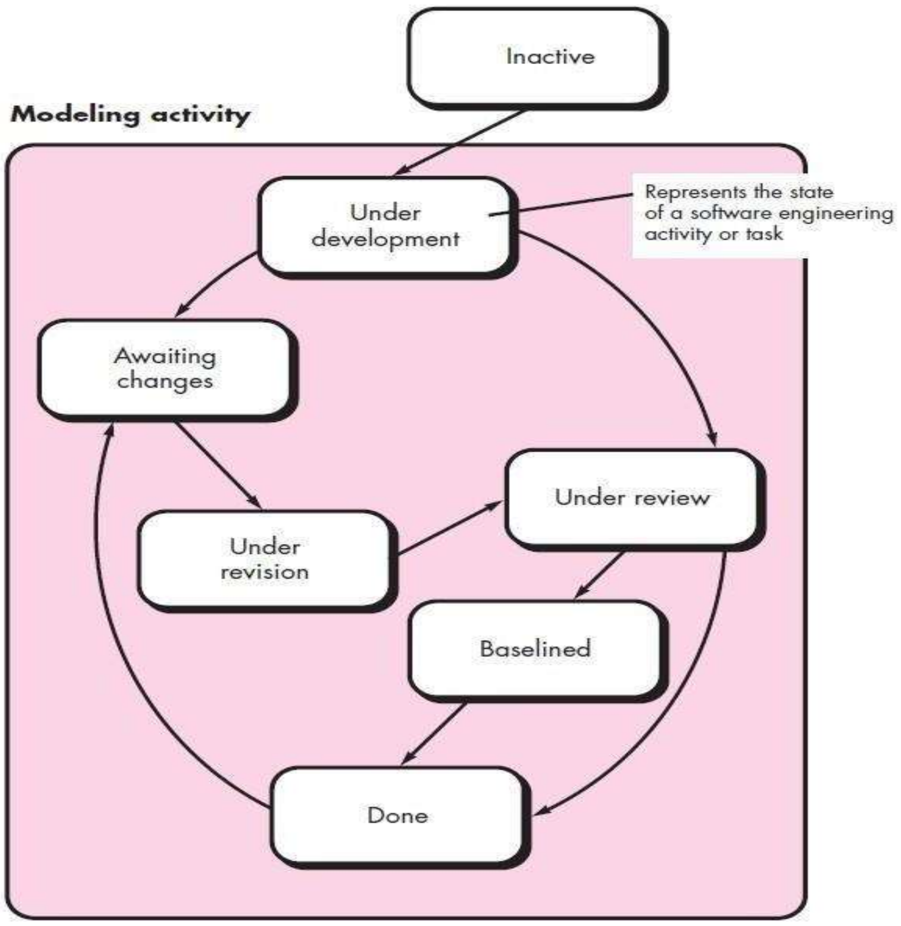
Each cycle of the spiral involves revisiting these steps with increasing detail and refinement, ultimately leading to the completed system.

Example: Developing a healthcare app. In the first cycle, you create basic features like patient registration. In the next cycle, you add appointment scheduling while assessing risks like data security and legal compliance.

- **Advantages:**
  - **Risk-Focused**: Identifies and mitigates risks early in the project.
  - **Flexible**: Allows for changes after each cycle based on feedback.
  - **Early Testing**: Parts of the system are tested after each phase.
- **Disadvantages:**
  - **Time-Consuming**: Multiple cycles can extend the development timeline.
  - **Expensive**: Revisiting phases and managing risks can increase costs.
  - **Complex**: Managing the project can be challenging, especially for large projects.
- **When to Use:** The Spiral Model is best for large, high-risk projects like banking software, healthcare systems, or mission-critical applications where managing risks such as security and timely delivery is crucial.

5. Concurrent Development Model

The Concurrent Development Model, also known as the Concurrent Engineering Model, allows multiple activities in software development to progress simultaneously rather than sequentially. This approach accommodates the natural overlaps in development tasks, enabling different components of the project to be in various stages at the same time.



Steps in the Concurrent Development Model:

- <u>**Parallel Activities:**</u>
  - In this model, tasks such as design, coding, and testing can occur in parallel. For example, while one team works on designing the user interface, another team could be coding the backend system, and yet another team might be testing modules that are already developed.
- <u>**State Progression:**</u>
  - Each task within the project can be in one of several states. The states in the diagram represent the lifecycle of a software engineering activity or task:
    - **Inactive**: The task has not yet started. It is planned, but no active work is being done.

- **Under Development**: The task is actively being worked on, whether it's coding, designing, or another activity.
- **Under Review**: The task is being evaluated for quality, completeness, and alignment with project goals. Reviews can involve code inspections, design walkthroughs, or test result assessments.
- **Awaiting Changes**: After review, the task may require modifications or corrections before it can proceed. It is on hold, waiting for the necessary changes to be made.
- **Under Revision**: The task is currently being revised based on feedback or new requirements. This involves making the necessary updates and corrections.
- **Baselined**: Once a task passes review and is approved, it is baselined. This means it is considered stable and set, serving as a reference point for further development or testing.
- **Done**: The task is completed, fulfilling all requirements and passing all necessary reviews. It is finalized and requires no further work.

Example:

Consider a project to develop a shopping app. The app's user interface (UI) is being designed while simultaneously, the backend payment system is being coded. At the same time, testing might already be underway for the basic functionalities. This parallel progression accelerates development by allowing multiple teams to work on different aspects of the project concurrently.

Advantages:

- **Faster Development**: By allowing multiple activities to occur simultaneously, the overall development process is faster, enabling quicker delivery of the product.
- **Flexibility**: The model's flexibility allows teams to adapt quickly to changes since various tasks are handled in parallel rather than sequentially.
- **Early Problem Detection**: Because different tasks progress simultaneously, issues in one area can be identified and addressed early, without waiting for a linear progression of tasks.

Disadvantages:

- **Complexity**: Managing multiple activities concurrently can be challenging, requiring careful coordination and oversight to ensure that tasks do not interfere with one another.
- **Coordination Required**: Effective communication and coordination between teams are essential to prevent confusion and ensure that all parallel tasks align with the overall project goals.
- **Risk of Overlap Issues**: If not managed properly, changes in one task may negatively impact other parallel tasks, leading to inconsistencies or integration problems.

When to Use:

The Concurrent Development Model is particularly useful for large, complex projects where different teams can work on various parts of the project simultaneously. It's ideal in scenarios where speed is critical, such as in the development of mobile apps, video games, or enterprise software. This model supports a dynamic development environment where tasks can evolve in parallel, adapting to new requirements as they arise.