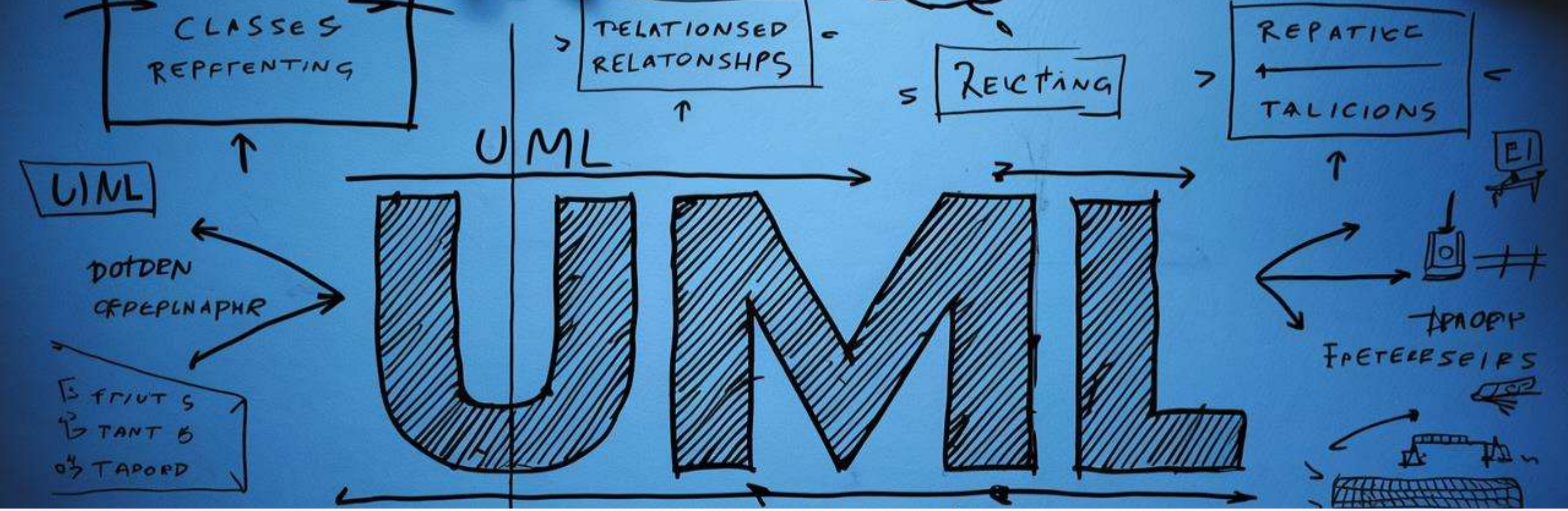


Object Oriented Analysis and Design using Unified Modeling Language

Er. Shiva Kunwar

Lecturer, GU

Lesson 1: Object Oriented Fundamentals (10hrs)

- Introduction
- Object Oriented Analysis and Design
- **Iterative development and unified process**
- Case Study
- **Understanding Requirements**
- Use Case modeling, Relating Use cases — include, extend and generalization
- Overview of the Unified Modeling Language: UML Fundamentals and Notations

OOAD Modeling

- Models helps to visualize, specify, construct and document the artifacts of software intensive system.
- OOAD modeling involves development of various diagrams that describes the system under consideration.
- It helps to manage complexity, understand requirements and properly derive the implementation.
-

OOAD Modeling Importance

- **Visualization, Specification, Construction, and Documentation**
  - OOAD models serve as visual representations that aid in understanding, specifying, constructing, and documenting software-intensive systems.
  - Visualizations help stakeholders, including developers and non-technical members, to grasp the system's structure and behavior.
- **Managing Complexity**
  - OOAD modeling plays a crucial role in managing the complexity of software systems.
  - By breaking down the system into comprehensible models, developers can focus on individual aspects without being overwhelmed by the entire system's intricacies.

OOAD Modeling Importance

- **Understanding Requirements:**
  - OOAD modeling contributes to a better understanding of system requirements.
  - It allows stakeholders to visualize how different components interact and ensures that the system aligns with the specified needs.
- **Deriving Implementation:**
  - OOAD models act as a bridge between high-level requirements and actual implementation.
  - They provide a blueprint for developers to translate design concepts into executable code.

OOAD Modeling Type: Conceptual Model

- Representation of Concepts:
  - Involves representing the concepts within the problem domain.
- Static Structure:
  - Depicts static structures where concepts are associated, have attributes, but no operations.
- Implementational Constraints:
  - Allows consideration of implementational constraints.
- Understanding Requirements:
  - Aids in understanding the problem requirements.

OOAD Modeling Type: Conceptual Model

- **Example: Online Shopping System**
- Representation of Concepts:
  - Concepts: Customer, Product, Order.
- Static Structure:
  - Customer has attributes (name, address).
- Product has attributes (name, price).
  - Order has attributes (order number, date).
- Implementational Constraints:
  - Consideration of constraints like payment methods.

OOAD Modeling Type: Structural Model

- Representation of Structure:
  - Illustrates the structure of the problem domain.
- Static Views:
  - Provides static views of the domain, focusing on entities, relationships, and their attributes.
- Understanding System Components:
  - Helps in understanding how different components are organized within the system.

OOAD Modeling Type: Structural Model

- **Example: Library Management System**
- Representation of Structure:
  - Entities: Book, Patron, Library.
- Static Views:
  - Illustrates how books are organized by genre, how patrons are registered, and how the library is structured.
- Understanding System Components:
  - Helps understand the relationships between entities (books, patrons, library).

OOAD Modeling Type: Behavioral Model

- Representation of Behavior:
  - Captures the dynamic behavior of the system.
- Dynamic Views:
  - Includes dynamic views of the domain, showcasing the flow of interactions between system components.
- Understanding System Dynamics:
  - Helps understand how components interact and behave over time.

OOAD Modeling Type: Behavioral Model

- **Example: ATM System**
- Representation of Behavior:
  - Captures how a user interacts with an ATM.
- Dynamic Views:
  - Shows the sequence of actions: insert card, enter PIN, select transaction.
- Understanding System Dynamics:
  - Helps understand the flow of interactions between the user and the ATM.

OOAD Modeling Type: Specification Model

- Software Abstractions:
  - Describes software abstractions with specifications and interfaces.
- No Commitment to Implementations:
  - Does not commit to specific implementations, focusing on defining what the system should do.
- Interface Definition:
  - Defines interfaces and specifications for components.

OOAD Modeling Type: Specification Model

- **Example: Email System**
- Software Abstractions:
  - Components: Email Server, User Interface.
- No Commitment to Implementations:
  - Describes the interfaces without specifying how the server stores emails.
- Interface Definition:
  - Defines methods for sending and receiving emails.

OOAD Modeling Type: Implementation Model

- Software with Implementations:
  - Describes the software with implementations using a particular technology.
- Technology-Specific Details:
  - Includes technology-specific details, providing a basis for coding.
- Bridge to Coding:
  - Acts as a bridge between higher-level design and actual coding.

OOAD Modeling Type: Implementation Model

- **Example: Online Reservation System**
- Software with Implementations:
  - Describes the technology-specific details.
- Technology-Specific Details:
  - Includes details on how reservations are stored in a database, implemented using a web framework.
- Bridge to Coding:
  - Serves as a guide for developers to implement the reservation system.

Iterative Development

- Iterative development is an approach to software development in which the development process is broken down into repeated cycles or iterations.
- Each iteration represents a complete development cycle, including **planning**, **analysis**, **design**, **implementation,** and **testing**.
- The iterative model contrasts with traditional linear development models, such as the Waterfall model, by allowing for flexibility, adaptability, and continuous improvement.

Iterative Development Characteristics

- **Incremental Progress:**
  - Development is divided into small, manageable increments or iterations.
  - Each iteration results in a potentially shippable product increment.
- **Feedback and Adaptability:**
  - Regular feedback from stakeholders, users, and testing is gathered after each iteration.
  - Adaptations and adjustments to the project are made based on feedback.

Iterative Development Characteristics

- **Repetition:**
  - The development process repeats through a series of iterations.
  - Each iteration builds upon the previous one, incorporating lessons learned and improvements.
- **Flexibility:**
  - Allows for changes in requirements, design, and functionality between iterations.
  - Emphasizes adaptability to evolving project needs.

Iterative Development Characteristics

- **Continuous Refinement:**
  - Continuous refinement of the product occurs throughout the development process.
  - Feedback from users and stakeholders guides ongoing improvements.
- **Risk Mitigation:**
  - Risks are addressed iteratively, reducing the likelihood of major issues emerging late in the development process.
  - Early identification and resolution of issues contribute to project success.

Iterative Development Steps

1. Planning:

- Define project objectives and scope.
- Identify key features and functionalities.
- Plan the initial iteration, breaking down the project into manageable tasks.
- Set goals, deliverables, and timelines for the first iteration.

Iterative Development Steps

2. Analysis:

- Conduct in-depth analysis of the identified features for the current iteration.
- Gather detailed requirements and user stories.
- Ensure a clear understanding of stakeholder needs.

Iterative Development Steps

3. Design:

- Design the architecture and high-level structure of the system.
- Create detailed design specifications for the features targeted in the current iteration.
- Focus on the design of specific features.

Iterative Development Steps

4. Implementation:

- Code the features based on the design specifications.
- Implement a subset of functionalities, keeping the scope limited to what can be achieved within the iteration.
- Leverage version control systems to manage code changes.

Iterative Development Steps

5. Testing:

- Conduct testing on the implemented features.
- Perform unit testing, integration testing, and other relevant testing types.
- Identify and fix defects.
- Ensure that the features meet specified requirements.

Object Oriented Development Lifecycle

- Object-oriented development is highly incremental
- First you perform analysis and then go to design
- Repeat again and again this each step until you get the result
- More rigorous process to do things right
- More time spent on gathering requirements, developing model
- Then finally turn the analysis model into design model
- The OODLC involves several phases, each focusing on different aspects of the development process.

Object Oriented Development Lifecycle

- **Requirements Analysis:**
- Objective:
  - Understand and gather detailed requirements for the software system.
- Activities:
  - Conduct interviews with stakeholders.
  - Identify system functionalities and user interactions.
  - Define use cases and user stories.

Object Oriented Development Lifecycle

- **System Design:**
- Objective:
  - Define the overall architecture and structure of the system.
- Activities:
  - Develop high-level design specifications.
  - Create class diagrams, defining relationships and dependencies.
  - Identify major system components and their interactions.

Object Oriented Development Lifecycle

- **Object Design:**
- Objective:
  - Specify the design of individual objects and their relationships.
- Activities:
  - Create detailed design specifications for each class.
  - Define attributes, methods, and their interactions.
  - Address issues like encapsulation and inheritance.

Object Oriented Development Lifecycle

- **Implementation:**
- Objective:
  - Transform the design into executable code.
- Activities:
  - Write code for classes and methods.
  - Implement relationships and interactions between objects.
  - Follow coding standards and best practices.

Object Oriented Development Lifecycle

- **Testing:**
- Objective:
  - Verify that the implemented code meets requirements and functions as intended.
- Activities:
  - Conduct unit testing for individual classes and methods.
  - Perform integration testing to ensure components work together.
  - Validate the system against the defined use cases.

Object Oriented Development Lifecycle

- **Deployment:**
- Objective:
  - Release the software to the end-users or deploy it in a production environment.
- Activities:
  - Prepare for deployment, including packaging and documentation.
  - Install the software on target environments.
  - Conduct user training and provide support.

Object Oriented Development Lifecycle

- **Maintenance and Evolution:**
- Objective:
  - Address issues, enhance features, and adapt to changing requirements over time.
- Activities:
  - Monitor system performance and address bugs.
  - Implement updates and enhancements.
  - Respond to user feedback and changing business needs.

Unified Process

- The Unified Process (UP) is an iterative and incremental software development methodology that provides a framework for efficiently managing and executing software projects.
- It is an adaptable and flexible process framework.
- It follows the principles of the Unified Modeling Language (UML).
- It is often associated with object-oriented development.
- UP emphasizes the creation and management of various artifacts, including use case models, class diagrams, sequence diagrams, and design specifications.
- Work products are created at each phase, providing documentation and guidance for the development team.


Modeling
Communication
Planning
Elaborations
Construction
Inception
Transition
Rational
Unified
Process
Release
Go no Go
Build
Testing
Deploy
Production
Feedback

Unified Process Phases

- The Unified Process (UP) is organized into four phases, each representing a distinct stage in the software development lifecycle.
- These phases are **Inception**, **Elaboration**, **Construction**, and **Transition**.
- The UP phases are characterized by specific objectives, activities, and milestones, and they follow an iterative and incremental approach.

Unified Process Phase: Inception

- **Objective:**
  - Understand the scope of the project and establish a vision.
- **Key Activities:**
  - Identify and define the initial requirements.
  - Outline the overall architecture and feasibility.
  - Identify major risks and potential solutions.
  - Develop a preliminary project schedule and estimate.

Unified Process Phase: Inception

- **Milestones:**
  - Vision: A clear understanding of the project scope and objectives.
  - Lifecycle Objectives: Defined goals for the project.
  - Risk Assessment: Identification of major risks and potential mitigation strategies.
  - Development Case: Initial project plan and schedule.
- **Outputs:**
  - Vision document.
  - Initial use cases.
  - Preliminary risk list.
  - Initial project plan.

Unified Process Phase: Elaboration

- **Objective:**
  - Refine the project vision, scope, and architecture.
- **Key Activities:**
  - Develop a more detailed use case model.
  - Refine the system architecture and design.
  - Address high-priority risks.
  - Develop a detailed project plan and estimate for the entire project.

Unified Process Phase: Elaboration

- **Milestones:**
  - Revised Vision: Adjusted based on feedback and more detailed analysis.
  - Baseline Architecture: A solid foundation for the system architecture.
  - Risk-Resolved Architecture: High-priority risks addressed.
  - Project Plan: Detailed plan for the entire project.
- **Outputs:**
  - Use case model.
  - Refined risk list.
  - Architectural prototype.
  - Revised project plan.

Unified Process Phase: Construction

- **Objective:**
  - Develop the system incrementally and iteratively.
- **Key Activities:**
  - Implement features based on the architecture.
  - Conduct regular testing and integration.
  - Continue to address and monitor risks.
  - Develop and release fully functional product increments.

Unified Process Phase: Construction

- **Milestones:**
  - First Operational Capability: Initial release of a functional system.
  - Architectural Complete: Key features and components implemented.
  - Beta Release: A version suitable for broader testing.
- **Outputs:**
  - Working software increments.
  - User documentation.
  - Test reports.
  - Release notes.

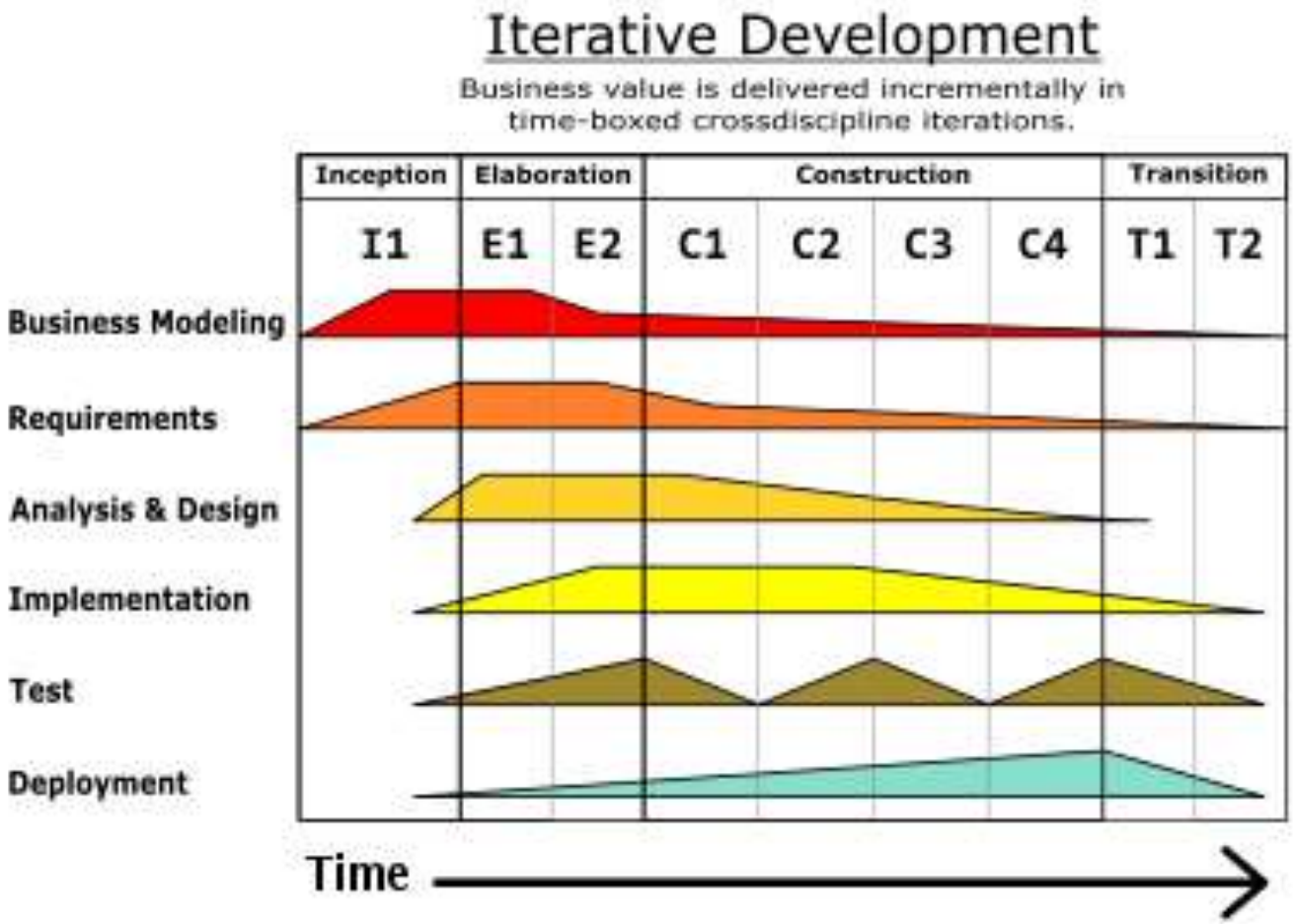
Unified Process Phase: Transition

- **Objective:**
  - Deploy the system for use by end-users.
- **Key Activities:**
  - Finalize testing and address any remaining issues.
  - Prepare for deployment and user training.
  - Release the system to end-users.
  - Conduct post-release support and monitoring.

Unified Process Phase: Transition

- **Milestones:**
  - Product Release: Deployment of the system for use.
  - Transition Assessment: Evaluation of the transition process.
  - Customer Feedback: Gather feedback from end-users.
- **Outputs:**
  - Deployed and operational system.
  - User training materials.
  - Support documentation.
  - Post-implementation review reports.

Unified Process



Understanding Requirements

- Requirement process is a systematic approach to finding, documenting, organizing, and tracking the users' needs and responses to a system's changing requirements.
- Requirements are the aspects that the system must follow.
- Clear and comprehensive requirements lay the foundation for designing and building a successful software system.
- A prime challenge of requirements work (fact-finding) is to find, communicate, and record what is really needed, in a form that clearly speaks to the client and development team members.

Requirements Types

- **Functional Requirement**
  - It describes the behavior of the system.
  - It includes user tasks that the system needs to support.
  - It is phrased as actions.
  - E.g., calculations, technical details, data manipulation and processing, etc.
- **Non-Functional Requirement**
  - It describes the properties of the system.
  - It is phrased as constraints or negative assertions.
  - also known as quality requirements, which impose constraints on the design or implementation (such as performance requirements, security, cost, and reliability).

Process of Understanding Requirements

- Requirement Elicitation Techniques:
  - **Interviews**: Talking with stakeholders.
  - **Workshops**: Collaborative sessions.
  - **Surveys**: Questionnaires for feedback.
  - **Observation**: Watching users interact with existing systems.
  - **Prototyping**: Creating mockups for validation.

Process of Understanding Requirements

- **Requirement Specification**:
- Documenting requirements using tools like:
  - **Use Case Diagrams**: Show interactions between actors and the system.
  - **User Stories**: Capture user interactions in a simple format.
  - **SRS (Software Requirements Specification)**: A formal document outlining all requirements.

Requirements Types - FURPS

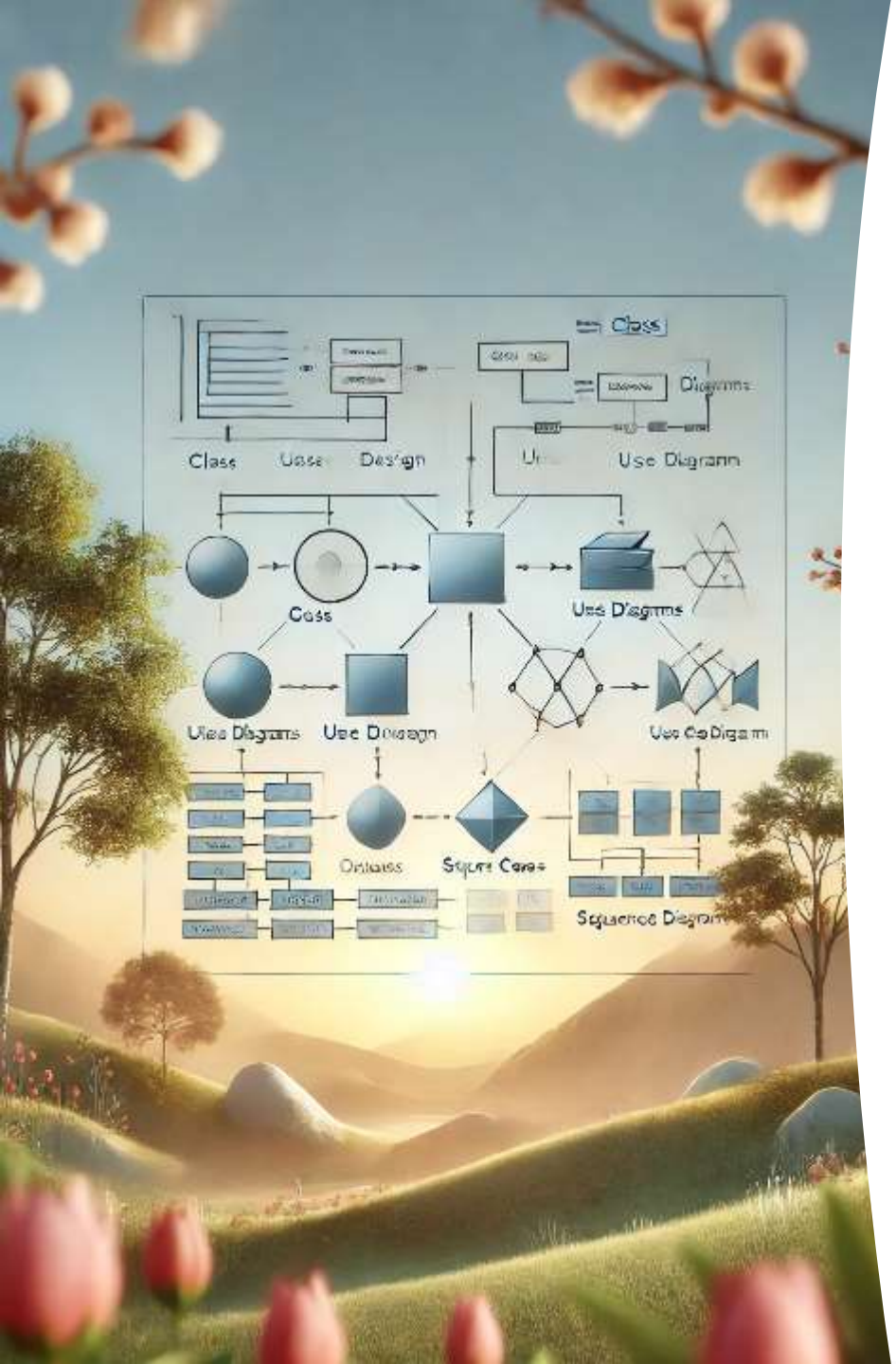
- An acronym representing a model for classifying software quality attributes (functional & non-functional requirements)
- First Developed at HP
- **Functionality**: features, capabilities.
- **Usability**: human factors, help, documentation.
- **Reliability**: frequency of failure, recoverability, predictability, security.
- **Performance**: response times, throughput, accuracy, availability, resource usage
- **Supportability**: adaptability, maintainability, internationalization, configurability.

Requirements Types - FURPS

- The "+" in FURPS+ indicates ancillary and sub-factors, such as:
- Implementation resource limitations, languages and tools, hardware, ...
- Operations system management in its operational setting.
- Packaging legal licensing and so forth

Requirement Elicitation Methods

- To identify all relevant requirements analysts must use a range of techniques.
- First, you must identify the information you need and develop a fact-finding plan.
- Fact finding can be done through Document Review, Observation, Questionnaire, Surveys, Sampling, Research, Task Analysis, Scenarios, Case study, etc.
- **Assignment**.



END OF LECTURE

SHIVA.KUNWAR@HOTMAIL.COM
+977-9819123654

PREVIEW FOR LECTURE 4

USE CASE MODELING