UML and introduction Slides:

- Definition and understanding of basic software engineering principles such as Abstraction, Modularisation/Decomposition, Coupling, Cohesion, Encapsulation, Inheritance, Polymorphism among others.
- Standard Patterns
- Anti-patterns
- Engineering Models and Modelling
- **Object Oriented Modelling**
    - *vs Data Oriented*
    - Abstraction
    - Encapsulation
    - Relationships
        - *Association, Aggregation, Inheritance etc.*
- UML
    - Definition
    - Use-case Diagram
    - **Class Diagram**
    - Sequence Diagram
    - *Statechart Diagram*
    - *Activity Diagram*
    - *Communication Diagram*
    - *Component Diagram*
- Static vs Dynamic Models (*classification of UML Diagrams*)
- Meta Models
- **UML Meta Model**
- **Class Diagram**
    - Components
    - Notations
    - Field and methods
    - Visibility notations
    - Interfaces
    - Inheritance (*Extension and Implementation*)
    - Specialisation/Generalisation
    - Realisation
- Association + *notations + multiplicity*
- Aggregation
- Composition

- Dependency *<<use>> + notation*
- Interaction Model (*Dynamic Modelling*)
    - Interaction
    - Why Interaction Models
- Sequence Diagram
    - Notations and Drawing
    - **UML 2.0 Sequence Diagram**
        - Notation
        - Fragment Types
        - Arrows

Refactoring:

- Lehman Belady Laws of SE
    - Continuing Changes
    - Increasing Complexity
- Why Refactor
- Design Smells - *causes (5)*
    - Viscosity - *Software, Environment*
    - Abstraction
    - Modularisation
    - Encapsulation
- Code Smells
    - Within Classes [11]
        - *Comments, Long Methods, Long Parameter List, Duplicate Code, Conditional Complexity, Combinatorial Explosion, Large Class, Uncommunicative name, Inconsistent Names, Dead Code, Speculative Generality.*
    - Between Classes[15]
        - *Common Interface, Primitive Obsession, Data Class, Data Clumps, Refused Bequest, Inappropriate Intimacy, Indecent Exposure, Feature Envy, Lazy Class, Message Chains, Middle Man, Divergent Change, Shotgun Surgery, Parallel Inheritance, Solution Sprawl*
- Principles of Refactoring
    - Don't refactor and extend at the same time.
    - Good tests in place before refactoring.
    - Small steps.
- When refactor? [2]

Anti-Patterns:

- Definition
- Purpose
- **Pattern vs Anti-Pattern**
- Examples(9) + Mini-AntiPatterns
    - *The Blob*, *Lava Flow*, *Functional Decomposition*, *Poltergeists*, *Golden Hammer*,
- **The Blob**
    - *what*, *why*
    - *Symptoms*
    - *Problems*
    - *causes*
    - *solution*
- **Lava Flow**
    - *what*
    - *Causes*
    - *Symptoms*
    - *Consequences*
    - *Solution*
- **Functional Decomposition**
    - *What*
    - *Causes*
    - *Symptoms*
    - *Consequences*
    - *Solution*
- **Poltergeists**
    - *What*
    - *Causes*
    - *Symptoms*
    - *Consequences*
    - *Solutions*
- **Cut and Paste Programming**
    - *What*
    - *Causes*
    - *Symptoms*
    - *Consequences*

    - *Solutions*
- **Golden Hammer**
    - *What*
    - *Causes*
    - *Symptoms*
    - *Consequences*
    - *Solutions*

- **Assigning Responsibilities**
    - Feature Identification
    - Stakeholder Identification
    - Feature Prioritization
    - **Use Case Diagram**
        - *Use Case Description*
    - Conceptual Classes
    - Identifying Attributes
    - Identifying Methods
    - Identifying Relationships

GRASP:

- Responsibility
    - *Action Oriented*
    - *Data Oriented*
- Balance between Extensibility and Reusability
- Patterns for Responsibility Assignment:
    - *High Cohesion*
    - *Information Expert*
    - *Creator*
    - *Low Coupling*
    - *Controller*
- **High Cohesion**
    - One general responsibility for a class.
    - Measure of how diverse an entity's features are.
    - Levels of cohesion : *very low*, *low*, *moderate*, *high*
    - *When not to follow*
- **Information Expert**
    - Information driven distribution of responsibility.
    - Problems with naive usage.

- **Creator**
    - A’s object created by B, given rule.
- **Low Coupling**
    - How
    - Why
    - Forms of coupling
- **Controller**
    - Systems responsibilities to one class.
    - Controller options
    - *Bloated Controllers -- flaws and errors*
- Avoid dumb objects
- Avoid God classes

Object Oriented Design Patterns:

- **Adapter Pattern**
    - Definition
    - Usage and benefits
    - When to use
    - *Object and Class Adapter Patterns*
- **Proxy Pattern**
    - Reasons for usage [3]
    - Goals
    - *Remote, Virtual, Protection*
    - *Other types [6]*
    - Consequences
- **Observer Pattern**
    - Definition
    - Observable
    - Observers
    - When to use
    - Consequences
- **Composite Pattern**
    - Basic idea, problem, solution
- ***Diagrams for each.***