DA-IICT

IT 314: Software Engineering

Object Design

Saurabh Tiwari

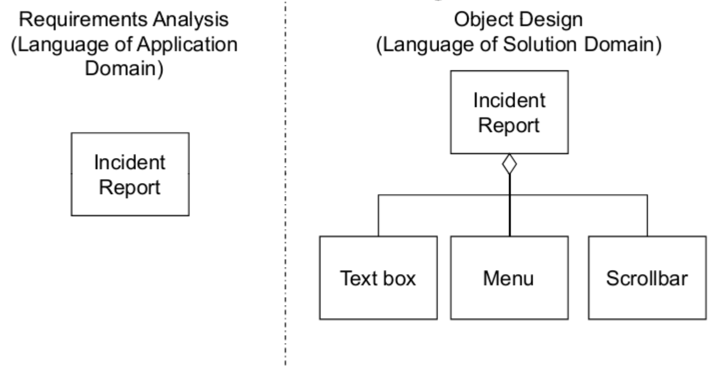
Object Design: Completing the Puzzle

- The pieces found during object design are:
  - New solution objects
  - Off-the-self-components and their adjustments
  - Design Patterns
  - Specification of subsystem interfaces and classes

Application domain vs. solution domain objects

- Application objects (also called domain objects) represent concepts of the domain that are relevant to the system
  - They are identified by the application domain specialists and by the end users
- Solution objects represent concepts that do not have a counterpart in the application domain,
  - They are identified by the developers
  - Examples: Persistent data stores, connection objects, user interface objects, data structures, middleware

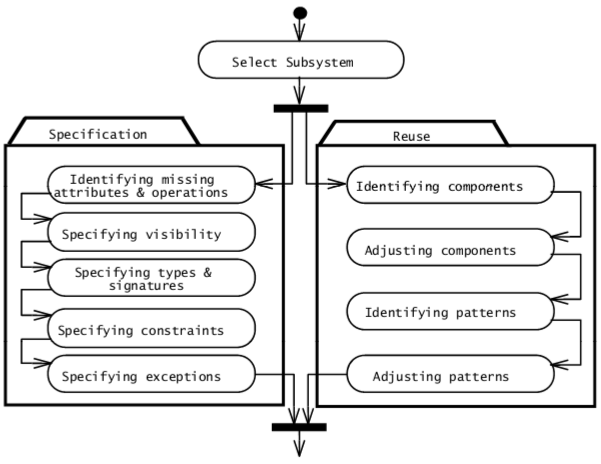
Application Domain vs Solution Domain Objects



Implementation of Application Domain Classes

- New objects are often needed during object design:
  - The use of design patterns introduces new classes
  - The implementation of algorithms may necessitate objects to hold values (Data Structures)
  - New low-level operations may be needed during the decomposition of high-level operations
- Example: The eraseArea() operation in a drawing program.
  - Conceptually very simple
  - Implementation
    - getArea() represented by pixels
    - repair () cleans up objects partially covered by the erased area
    - redraw() draws objects uncovered by the erasure
    - draw() paints pixels in background color not covered by other objects

Object Design Activities



Design Activities

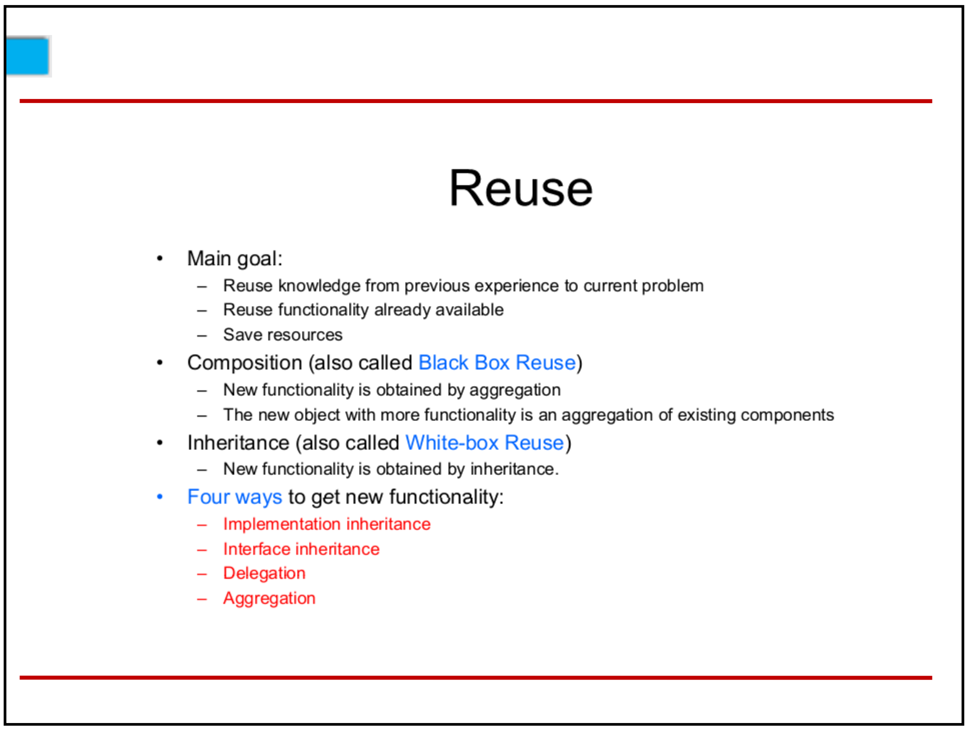
1. Reuse: Identification of existing solutions
   - Use of inheritance
   - Off-the-shelf components and additional solution objects
   - Design patterns
2. Interface specification
   - Describes precisely each class interface
3. Component/Object model restructuring
   - Transforms the object design model to improve its understandability and extensibility
4. Component/Object model optimization
   - Transforms the object design model to address performance criteria such as response time or memory utilization.

Component/ Object Design

Mapping Models to Code

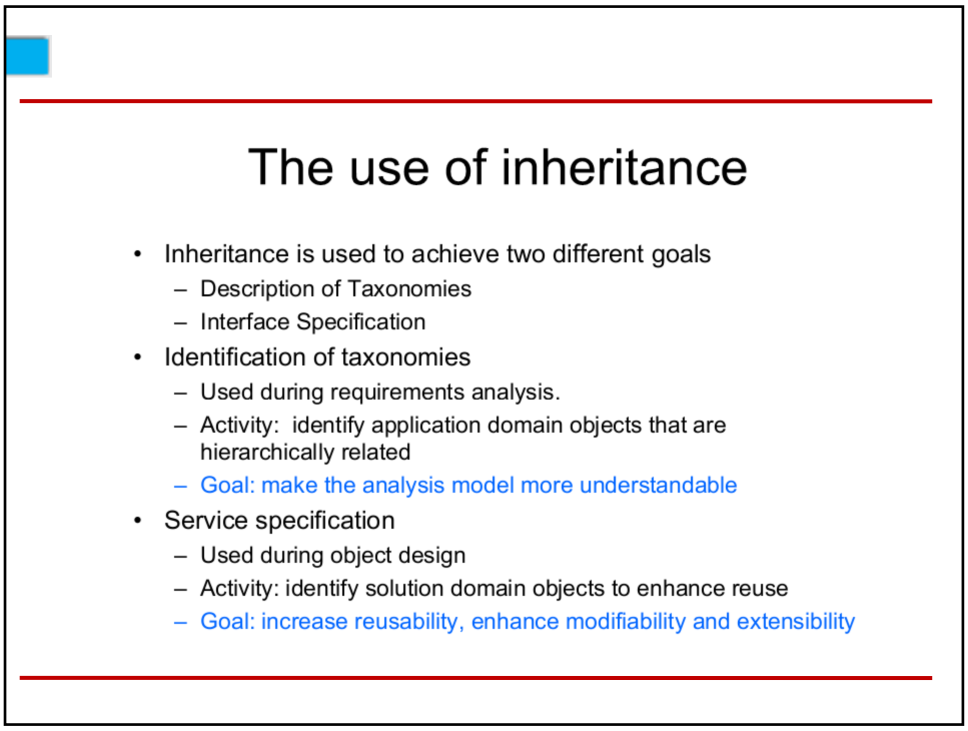
Component Selection

- Select existing
  - off-the-shelf class libraries
  - frameworks or
  - components
- Adjust the class libraries, framework or components
  - Change the API if you have the source code.
  - Use the adapter or bridge pattern if you don't have access
- Create a new Component (Architecture Driven Design)



Reuse

- Main goal:
  - Reuse knowledge from previous experience to current problem
  - Reuse functionality already available
  - Save resources
- Composition (also called Black Box Reuse)
  - New functionality is obtained by aggregation
  - The new object with more functionality is an aggregation of existing components
- Inheritance (also called White-box Reuse)
  - New functionality is obtained by inheritance.
- Four ways to get new functionality:
  - Implementation inheritance
  - Interface inheritance
  - Delegation
  - Aggregation

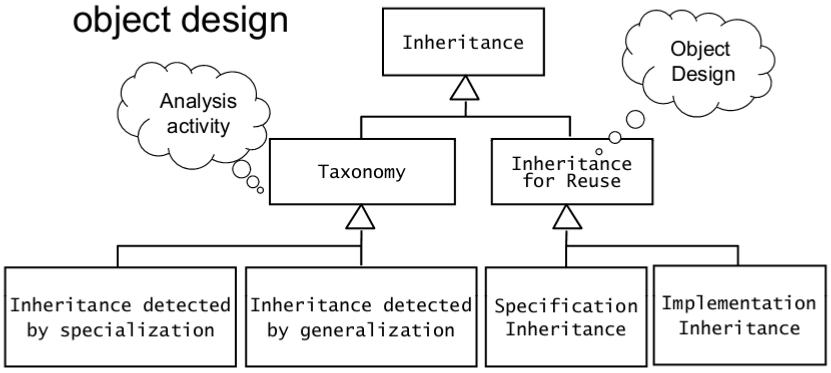


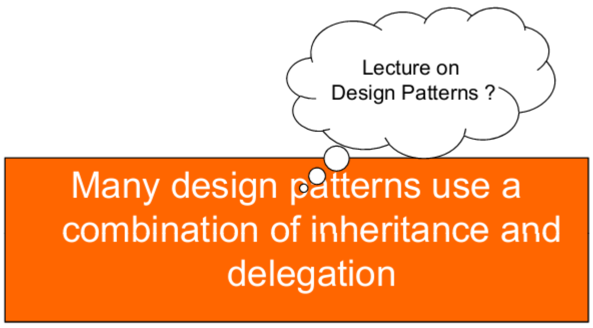
The use of inheritance

- Inheritance is used to achieve two different goals
  - Description of Taxonomies
  - Interface Specification
- Identification of taxonomies
  - Used during requirements analysis.
  - Activity: identify application domain objects that are hierarchically related
  - Goal: make the analysis model more understandable
- Service specification
  - Used during object design
  - Activity: identify solution domain objects to enhance reuse
  - Goal: increase reusability, enhance modifiability and extensibility

Metamodel for Inheritance

- Inheritance is used during analysis and object design


Inheritance
Object Design
Analysis activity
Taxonomy
Inheritance for Reuse
Inheritance detected by specialization
Inheritance detected by generalization
Specification Inheritance
Implementation Inheritance

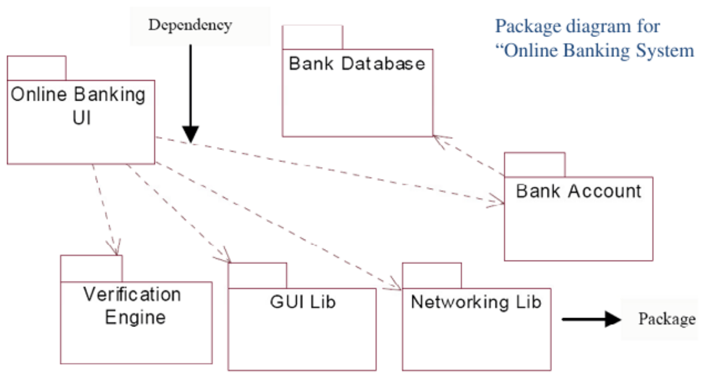


Many design patterns use a combination of inheritance and delegation

Package Diagram

- Structured organization of Code
- Grouping of related classes to help the software engineer to identify and to understand dependencies
- When to use?
  - Program Comprehension
  - Change Management

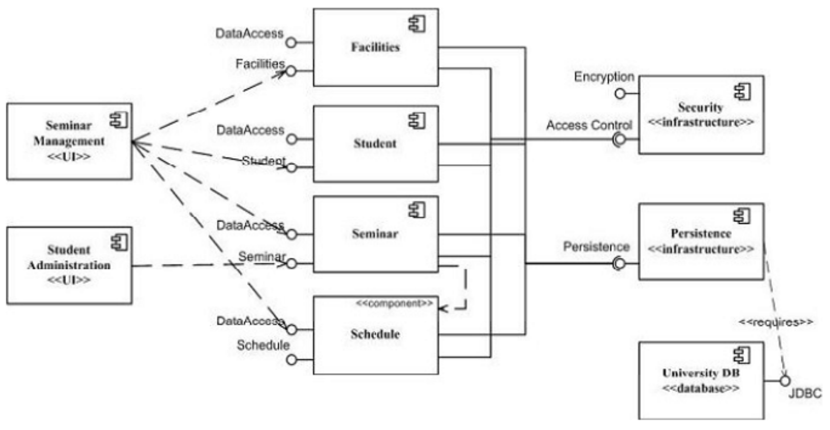
Package Diagram: An Example



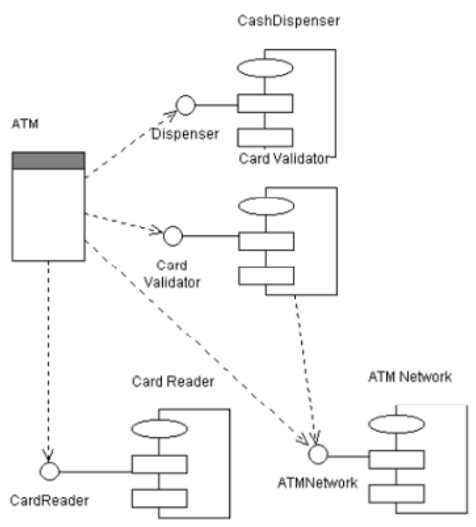
Component Diagram

- Depicts how components are wired together to form bigger component or system
- Component interacts with each other though interfaces
- Connect the required interface of one component with the provided interface of another component.
- Designed with an eye towards deployment

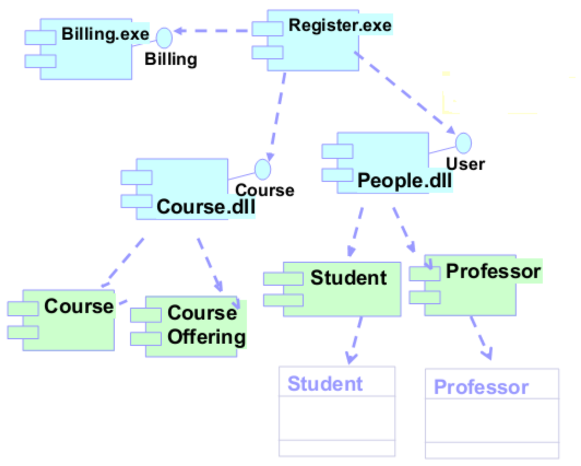
Component Diagram



Component Diagram

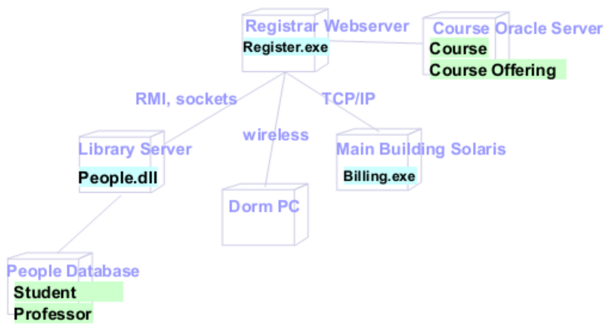


Component Diagram



Deployment Diagram

- shows the configuration of run-time processing elements and the software processes living on them.
- visualizes the distribution of components across the enterprise.



Summary

Design is the process of adding details to the requirements analysis and making implementation decisions

- An evolutionary activity

- Consists of
  - Sub-system Design (Choosing an Architecture)
  - Object Design (Solution domain)

Next Lectures…
Design Patterns