# Week 11 Component-Level Design

- **Definition**
  - Component-level design defines the data structures, algorithms, interface characteristics and communication mechanisms allocated to each software component
  - A component-level design can be represented using some intermediate representations (e.g. graphical, tabular, or text-based) that can be translated into source code
- **Software Component**
  - Definition
    - A software component is a modular building block for computer software
  - Usage
    - It can be used to review for correctness and consistency with other components
    - It can be used to access whether data structure, interfaces and algorithms will work
    - It should provide sufficient information to guide implementation
  - Three different views
    - Object-oriented view
      - A component is a set of collaborating classes
      - Mode detailed attribute information required in the objects
    - Conventional view
      - A component is a functional element of a program that incorporates --
        - Processing logic
        - The internal data structures
        - An interface that enables the component to be invoked
      - Each module is elaborated -- into functions
    - Process view
      - How the system is built from existsing components
- **Component-level Design Process**
  - 1 Identify all design classes corresponding to the problem domain
  - 2 Identify all design classes corresponding to the infrastructure domain
    - Such as GUI components, OS components, data management components
  - 3 Elaborate all design classes that are not acquired as reusable components
    - Specify message details when classes or components collaborate
    - Identify appropriate interfaces for each component
    - Elaborate attributes and define data types and data structures required to implement them
    - Describe processing flow within each operation in detail
  - 4 Describe persistent data sources (databases and files) and identify the classes required to manage them
  - 5 Develop and elaborate behavioral representations for a class or component
  - 6 Elaborate deployment diagrams to provide additional implementation in detail
  - 7 Refractor every component-level design representation and always consider alternatives
- **Design and implementation**
  - Recall: A stage in software process
  - Object-oriented design using UML
    - Define the context and the external interactions with the system

      - System context
        - A structural model (e.g. class diagram) that demonstrates the other systems in the environment of the system being developed
      - Interaction model
        - A dynamic model (e.g. a use case diagram + structured natural language description)
- Implementation
  - Reuse
    - Reuse levels
      - Abstraction level
      - Object level
      - Component level
      - System level
        - Reuse the entire application systems
    - Reuse costs
      - In buying reusable software
      - In adapting and configuring the reusable software components
      - In integrating reusable software elements
  - Configuration management
    - Managing a changing software system -- different versions
  - Host-target developement
    - Development platform and execution platform
  - Open source development
    - An approach to software development in which the source code of a software system is published and volunteers are invited to participate in the development process
    - Fundamental principle: source code should be freely available