National University
Of Computer and Emerging Sciences

Software Analysis and Architecture

Software Development Methodologies

Sajid Anwer

Department of Software Engineering,
FAST-NUCES, CFD Campus

Lecture Objectives

- Software Design Methodologies
- Design Paradigms
- Object-oriented Design Approach

Software Design Methodologies

- In a Software development process, the Software Design Methodology (SDM) refers to:
  - *specific set* of procedures used to create a *conceptual design* for fulfilling the set of requirements.

- The choice of the SDM primarily depends upon several factors, namely,
  - the *type of the software* (such as standalone or distributed and networked; Strategic or operational etc.)
  - the *scope of the development project* (such as revamp of the existing system or new system, the number of modules involved, underlying complexity of the coding, system testing and implementation etc),
  - the *resources constraints* (such as time, money, expertise)

Software Design Methodologies

- Common software design approaches includes:
  - » Structured (Function-Oriented)
    - Process functions are identified
    - Process intensive tasks
  - » Object-Oriented
    - develop an object model of a system
    - To understand real-world entities and their relationship
  - » Data-Oriented (Data-structure-centered)
    - Entities are determined for each sub-system, then entity inter-relationships are examined to develop the additional entities needed to support the relationships.
    - Database and banking applications

Software Design Methodologies

- Component-based
  - » Divide the system into components
  - » For large systems that can be modularized.
- Formal Methods
  - » Requirements and programs are translated into *mathematical notation*
  - » For safety and security systems
  - » Expensive to implement

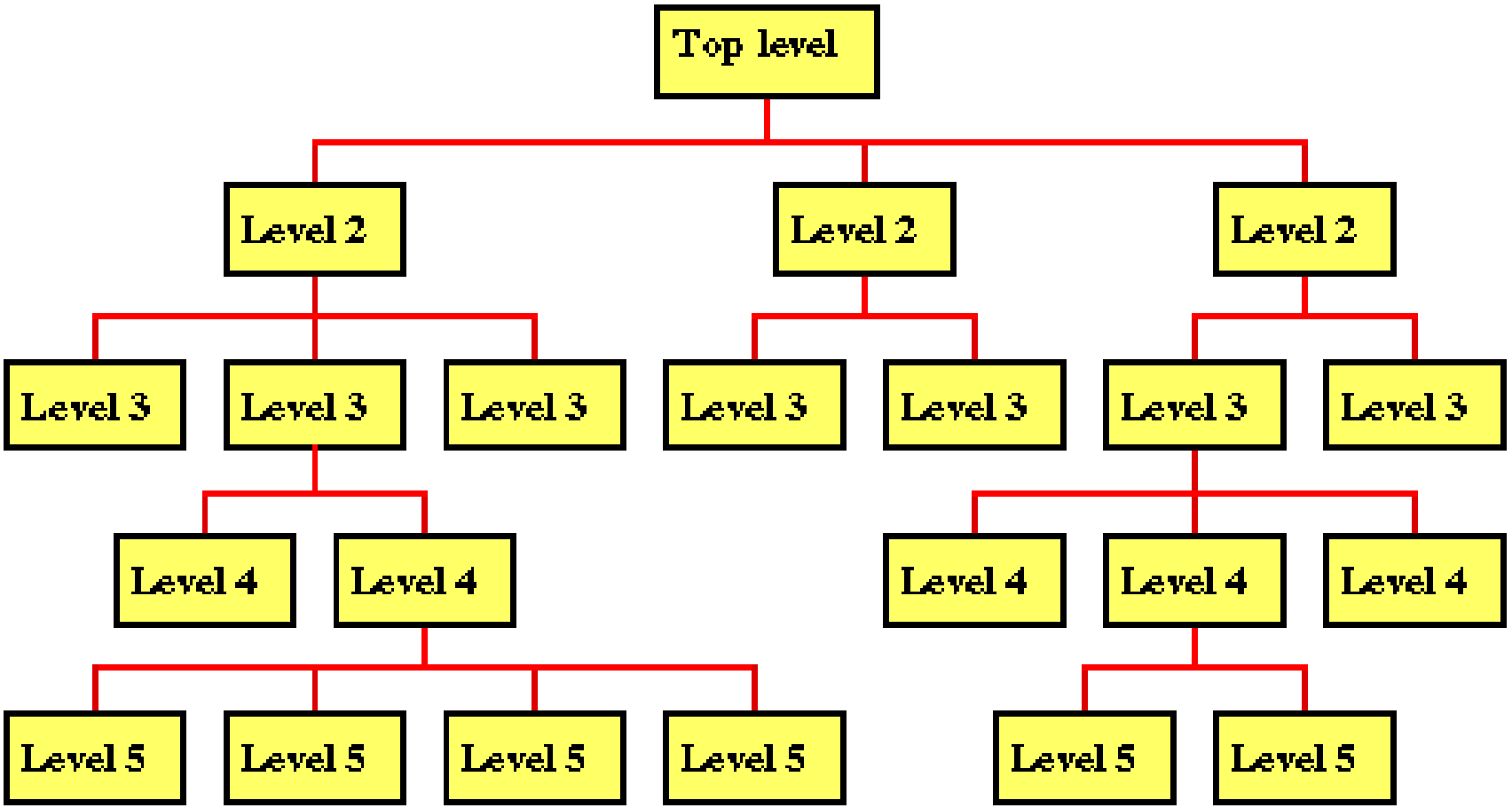
Software Design Paradigms

- Structured Design/Function Oriented/Component Design
- Object-Oriented Design

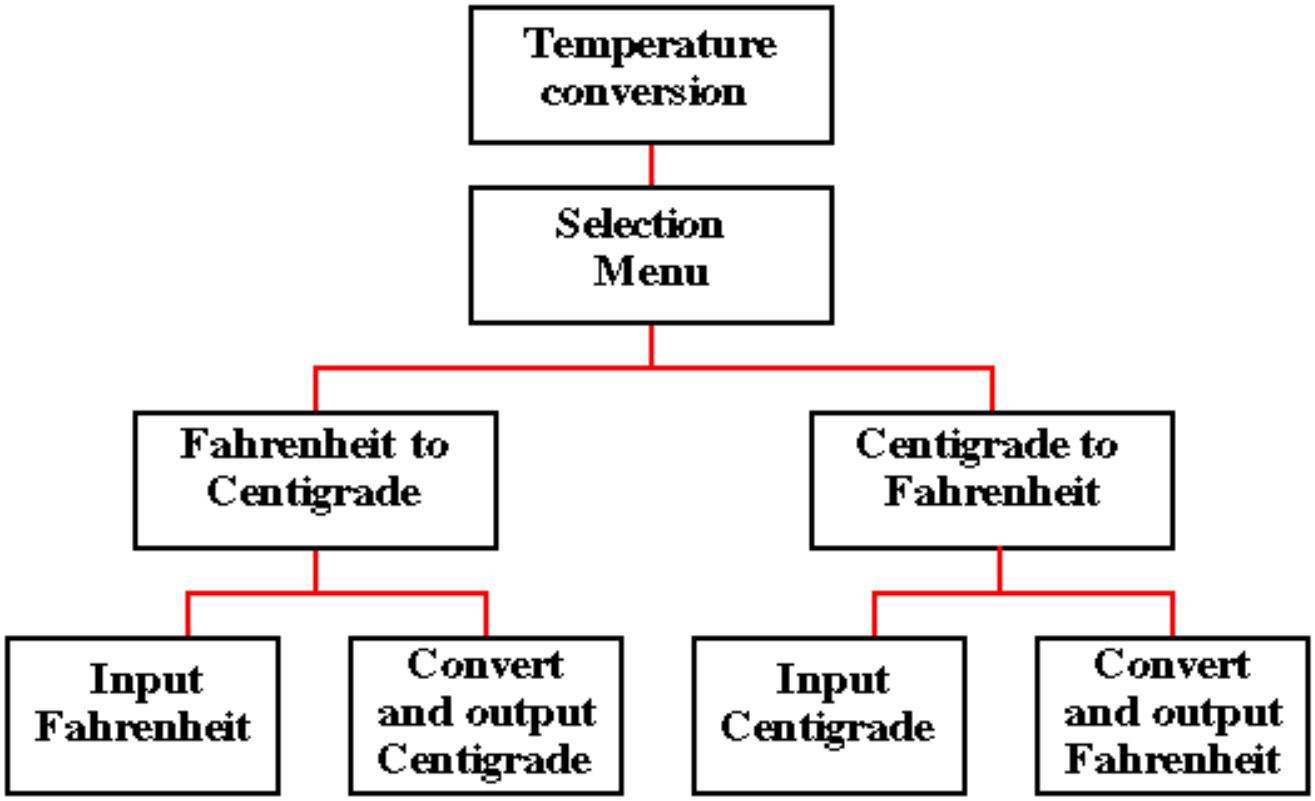
Software Design Paradigms - Structured

- Focus on *procedures and functions*.
- Design the system by *decomposing* it based on the processes and functions.
- *Top-down* algorithmic decomposition.
- This approach separates *data from procedures*.

Software Design Paradigms - Structured



Software Design Paradigms - Structured



Software Design Paradigms - Structured

- Drawbacks
  - » *Interdependencies* between various functions and processes.
  - » Cannot be *reused* easily, why?
  - » *Data* related to each function is not attached.

Software Design Paradigms – Object-Oriented Design (OOD)

- Bottom-up
  - » Describing the software solution in terms of *collaborating* objects, with responsibilities.
  - » Encapsulate data and procedures in objects and classes.
  - » Refinement in classes lead to a composed larger system.
- OO Design elements
  - » Objects,
  - » classes,
  - » encapsulation,
  - » States,
  - » inheritance,
  - » composition,
  - » polymorphism

Software Design Paradigms – OOD-Benefits

- Enjoys all the benefits of *Modular approach*
- Dependencies can be handled by finding the *commonalities* through inheritance and polymorphism.
- Naturalness
  - » OO paradigm models the *real world* better because everything in real world is an object.
- Reusability
  - » Using the existing classes or components in future design, without much effort.

Software Design Paradigms – OOD- Notation/Language

- UML is just a standard diagramming notation.
- It is just a tool, not a skill that is valuable in itself.
- Knowing UML helps you communicate with others in creating software, but the real work in this course is *learning* Object-Oriented Analysis and Design, not how to draw diagrams.
- The most important skill in Object-Oriented Analysis and Design is *assigning responsibilities* to objects.
- That determines how objects *interact* and what classes should perform what operations.

Software Design Paradigms – OOD- Notation/Language