Design Engineering

Slide Set - 8

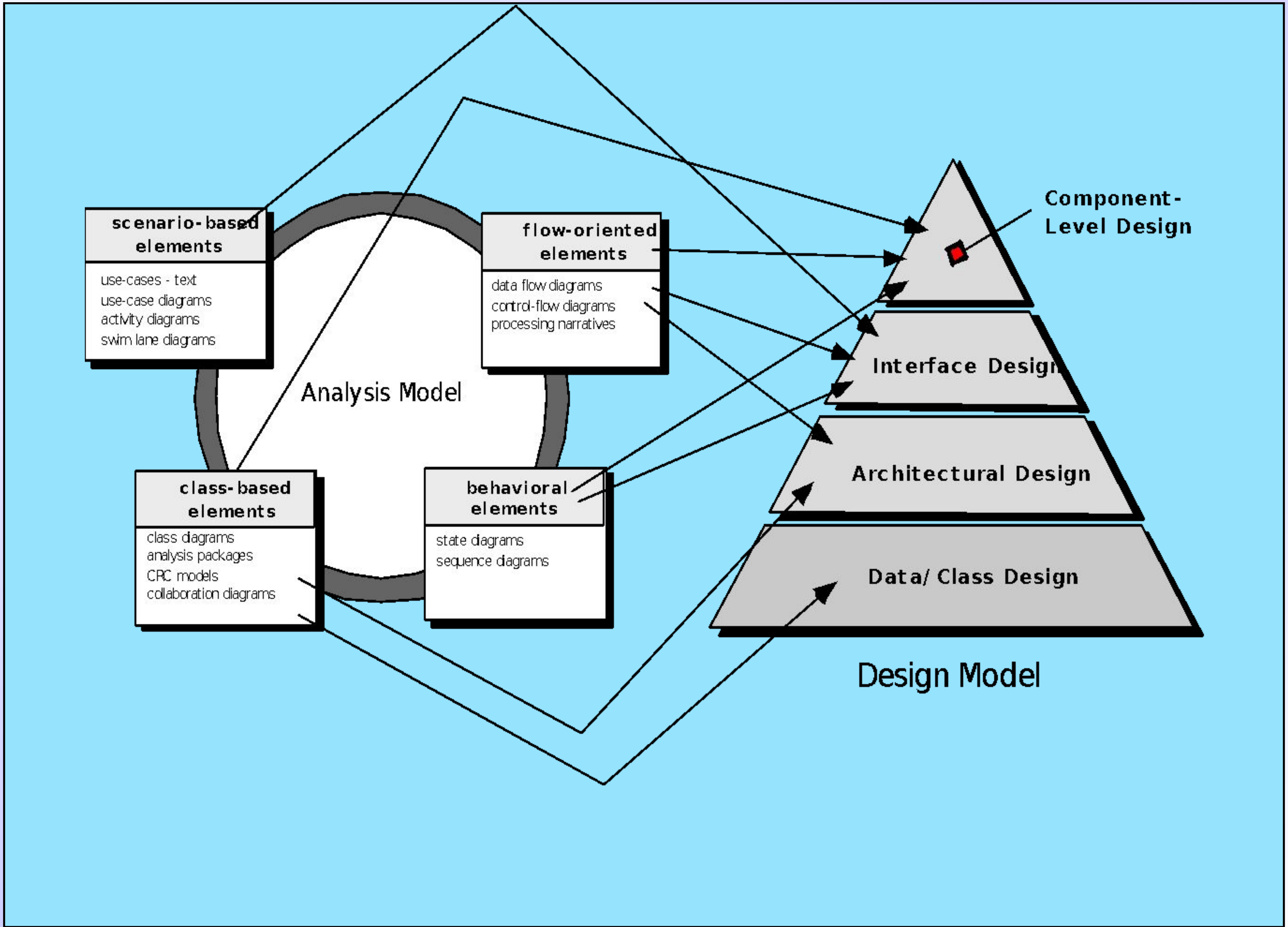
Organized & Presented By:
Software Engineering Team CSED
TIET, Patiala

From Analysis Model to Design Model

- Each element of the analysis model provides information that is necessary to create the four design models
  - The data/class design transforms analysis classes into design classes along with the data structures required to implement the software
  - The architectural design defines the relationship between major structural elements of the software; architectural styles and design patterns help achieve the requirements defined for the system
  - The interface design describes how the software communicates with systems that interoperate with it and with humans that use it
  - The component-level design transforms structural elements of the software architecture into a procedural description of software components

(More on next slide)

Analysis Model -> Design Model



From Analysis Model to Design Model (continued)

Component-level Design

(Class-based model, Flow-oriented model
Behavioral model)

Interface Design

(Scenario-based model, Flow-oriented model
Behavioral model)

Architectural Design

(Class-based model, Flow-oriented model)

Data/Class Design

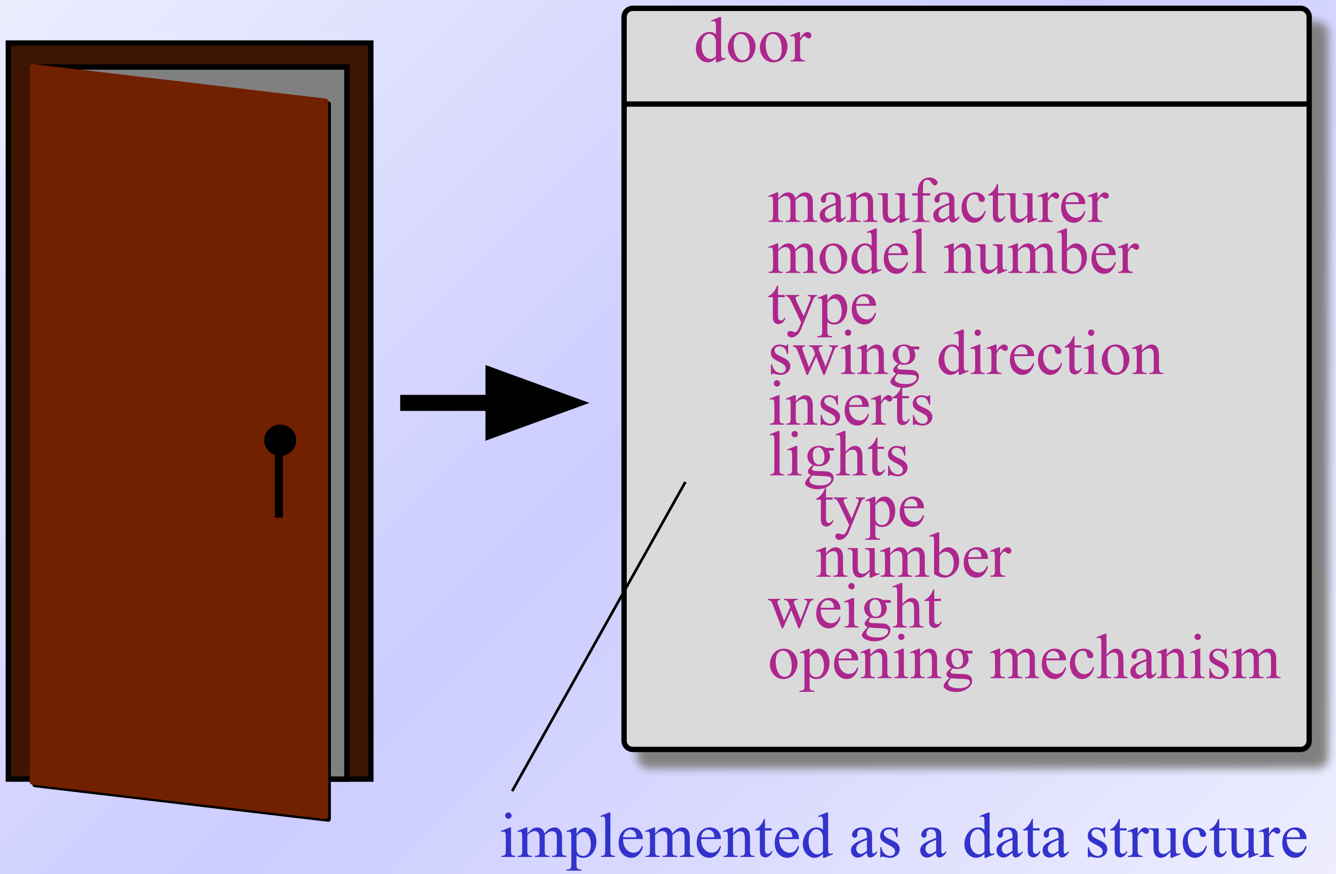
(Class-based model, Behavioral model)

Design Concepts

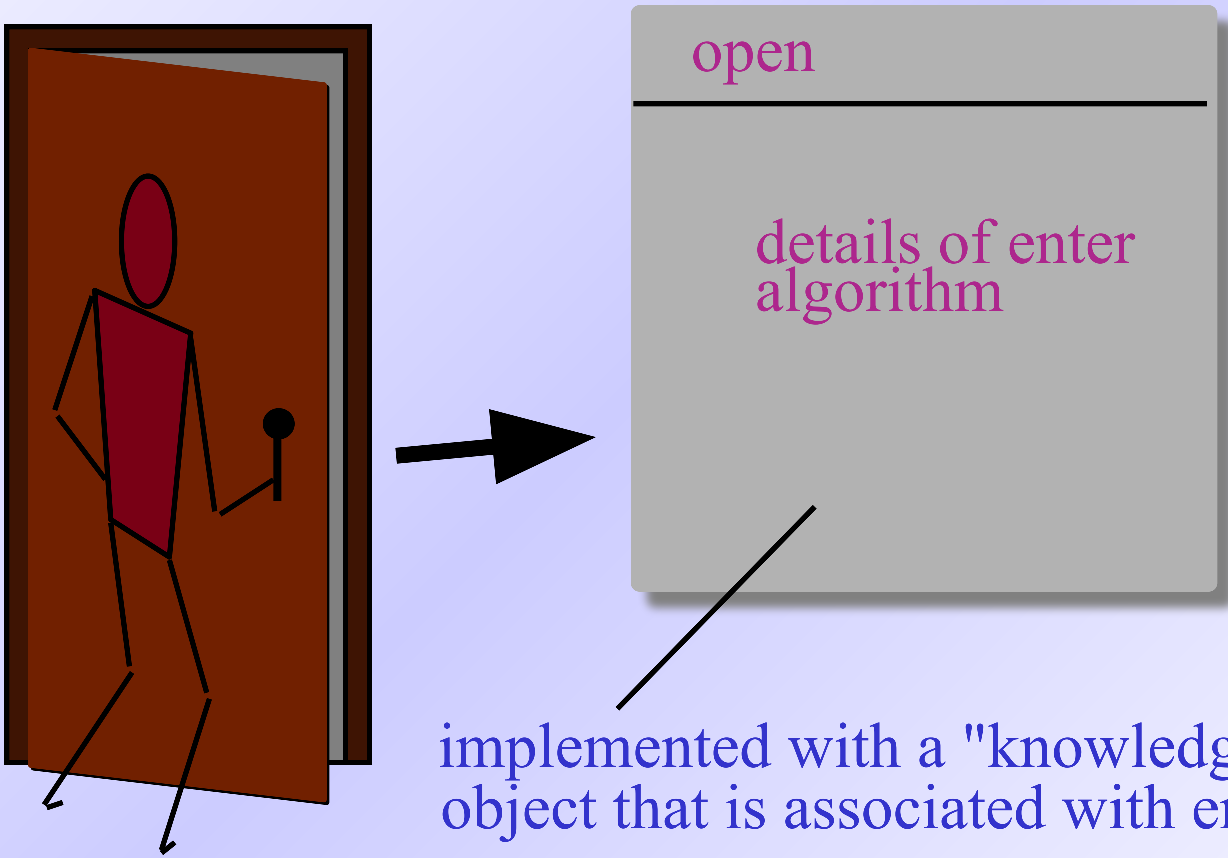
Fundamental Concepts

- abstraction—data, procedure, control
- architecture—the overall structure of the software
- patterns—”conveys the essence” of a proven design solution
- modularity—compartmentalization of data and function
- hiding—controlled interfaces
- Functional independence—single-minded function and low coupling
- Refinement—elaboration of detail for all abstractions
- Refactoring—a reorganization technique that simplifies the design

Data Abstraction



Procedural Abstraction



implemented with a "knowledge" of the object that is associated with enter

Design Concepts

- Abstraction
  - Procedural abstraction – a sequence of instructions that have a specific and limited function
  - Data abstraction – a named collection of data that describes a data object
- Architecture
  - The overall structure of the software and the ways in which the structure provides conceptual integrity for a system
  - Consists of components, connectors, and the relationship between them
- Patterns
  - A design structure that <u>solves a particular design problem</u> within a specific context
  - It provides a description that enables a designer to determine whether the pattern is applicable, whether the pattern can be reused, and whether the pattern can serve as a guide for developing similar patterns

(more on next slide)

Design Concepts (continued)

- Modularity
  - Separately named and addressable components (i.e., modules) that are integrated to satisfy requirements (divide and conquer principle)
  - Makes software intellectually manageable so as to grasp the control paths, span of reference, number of variables, and overall complexity
- Information hiding
  - The designing of modules so that the algorithms and local data contained within them are inaccessible to other modules
  - This enforces access constraints to both procedural (i.e., implementation) detail and local data structures
- Functional independence
  - Modules that have a "single-minded" function and an aversion to excessive interaction with other modules
  - High cohesion – a module performs only a single task
  - Low coupling – a module has the lowest amount of connection needed with other modules

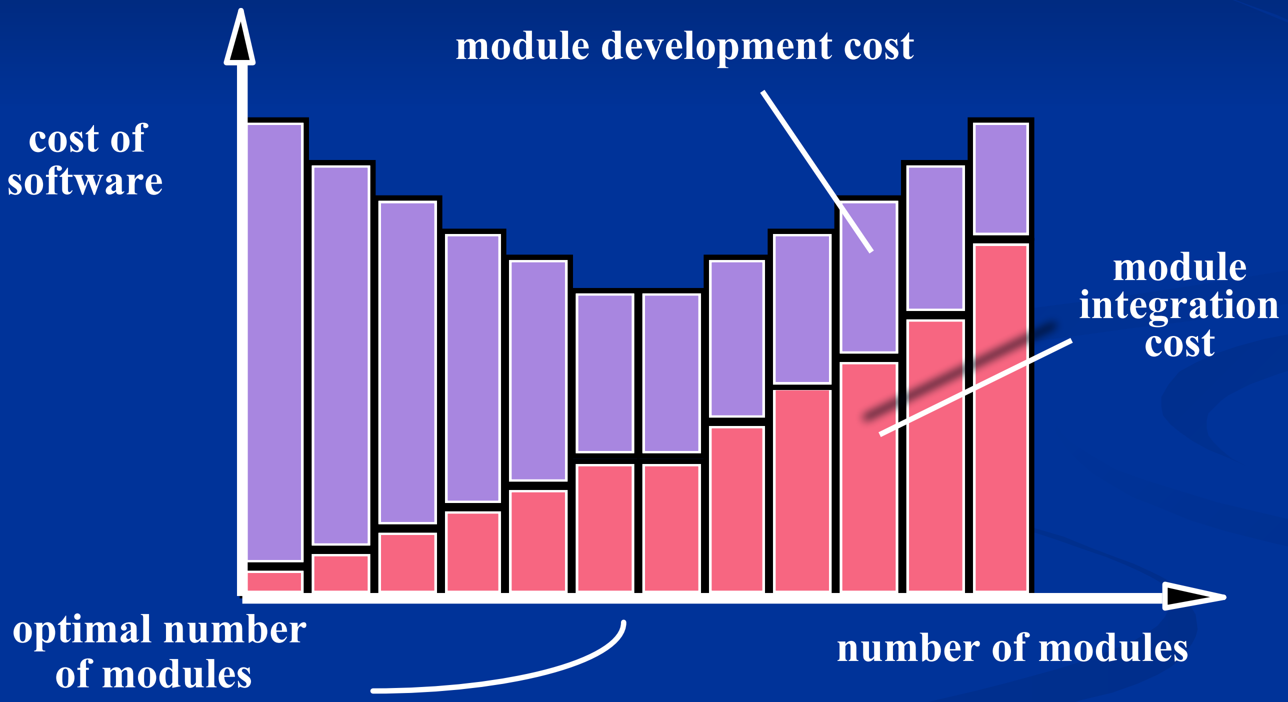
(more on next slide)

Design Concepts (continued)

- Stepwise refinement
  - Development of a program by successively refining levels of procedure detail
  - Complements abstraction, which enables a designer to specify procedure and data and yet suppress low-level details
- Refactoring
  - A reorganization technique that simplifies the design (or internal code structure) of a component without changing its function or external behavior
  - Removes redundancy, unused design elements, inefficient or unnecessary algorithms, poorly constructed or inappropriate data structures, or any other design failures
- Design classes
  - Refines the analysis classes by providing design detail that will enable the classes to be implemented
  - Creates a new set of design classes that implement a software infrastructure to support the business solution

Modularity: Trade-offs

What is the "right" number of modules for a specific software design?



Separation of concerns (SoC)

- SoC is a fundamental principle in software engineering that revolves around breaking down complex software systems into smaller, more manageable parts.
- Each of these parts addresses a single concern, which is essentially a specific aspect of functionality.
- By separating these concerns, developers can create well-organized and maintainable code.
- Key points of SoC:

a. **Modularization:** The system is divided into modules, each with a well-defined purpose and functionality. These modules have minimal overlap in their responsibilities.

b. **Encapsulation:** Each module encapsulates its data and logic, hiding the internal workings from other parts of the system. This allows for changes to be made within a module without affecting other parts.

Object-Oriented Design (OOD)	Function-Oriented Design (FOD)	Service-Oriented Design (SOD)
• Objects encapsulate data (attributes) and the code that manipulates it (methods). Imagine objects as real-world things like a car (data: color, make, model; methods: accelerate, brake, turn). • Reusability. Objects can be grouped into classes, and inherit properties and functionalities from parent classes. This promotes code reuse and reduces redundancy.	• Functions are independent, self-contained blocks of code that perform specific tasks or operations. • Breaking down a program into smaller, manageable functions. Each function has a defined input and output, making it easier to understand, test, and reuse.	• Services are self-contained units that provide specific functionalities through well-defined interfaces. Think of services as mini-applications that can be accessed by other applications. • Loose coupling and network communication. Services are designed to be independent and communicate with each other over a network using standardized protocols. This allows for flexibility and scalability.

Detailed Design: Key Activities

- **Module Decomposition:** The high-level components identified in the earlier design phase are further broken down into smaller, more manageable modules.
- **Functional Allocation:** Each module is assigned specific functionalities and responsibilities. This ensures clear ownership and avoids redundancy.
- **Data Design:** The data structures used by each module are defined. This includes data types, storage mechanisms, and how data will be passed between modules.
- **Algorithm Selection:** The specific algorithms that will be used to implement the functionalities within each module are chosen. Factors like efficiency and suitability for the task are considered.
- **User Interface (UI) Design:** If the software has a user interface, detailed design may involve defining the UI elements, their interactions, and the data flow between the UI and the backend modules.

Deliverables of Detailed Design

- **Module Specifications:** Detailed documents outlining the functionalities, interfaces, data structures, and algorithms used by each module.
- **Data Flow Diagrams (DFDs):** Visual representations of how data flows through the system, showing interactions between modules and external entities.
- **Sequence Diagrams:** Illustrate the sequence of interactions between objects or modules during a specific operation.
- **Class Diagrams (OOD):** In object-oriented design, class diagrams depict the classes, their attributes, methods, and relationships between them.
- **Pseudocode:** An informal language resembling actual code that outlines the logic and algorithms used in specific sections.