THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Design Engineering

Slide Set - 8

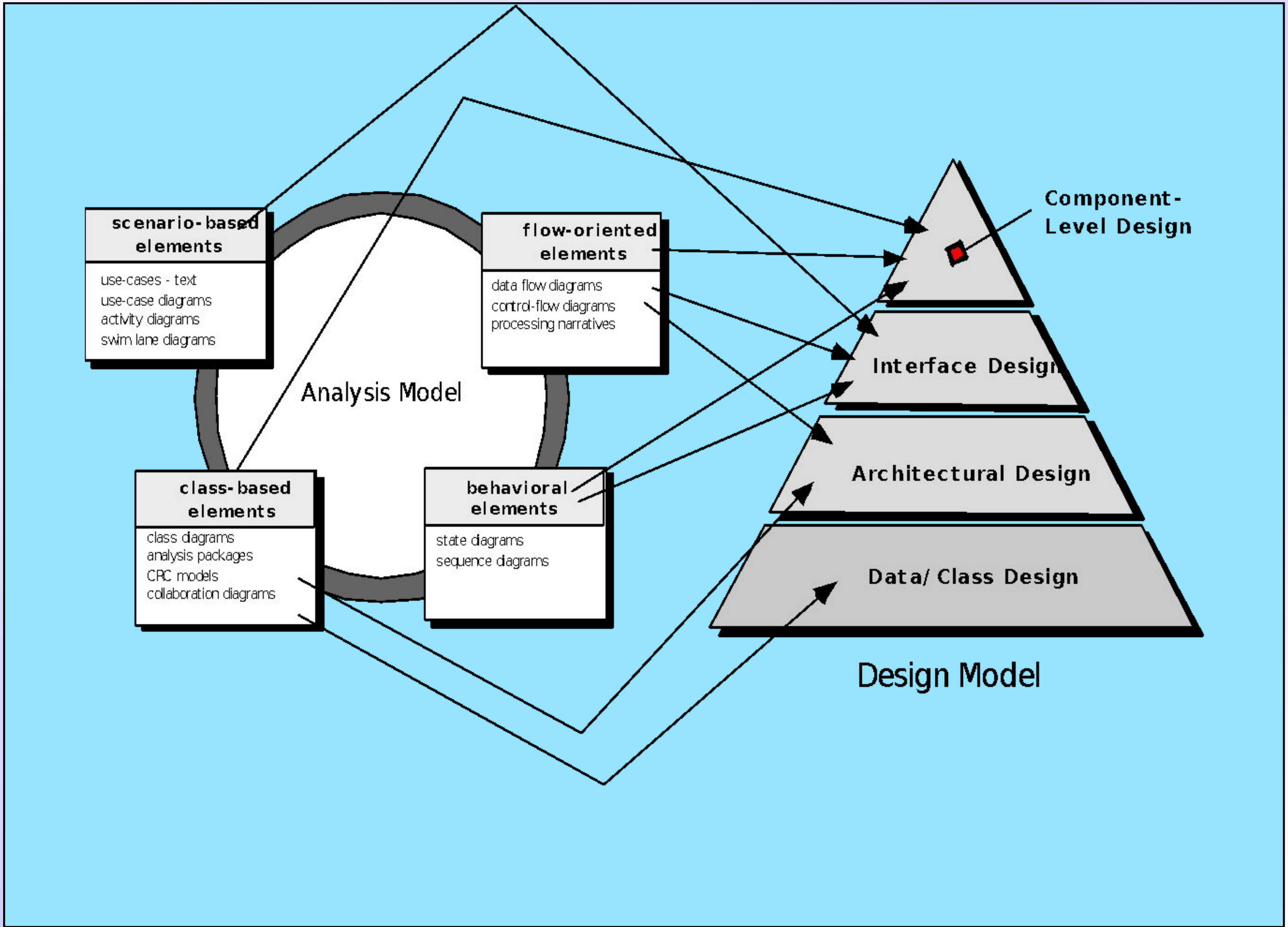
Organized & Presented By:
Software Engineering Team CSED
TIET, Patiala

From Analysis Model to Design Model

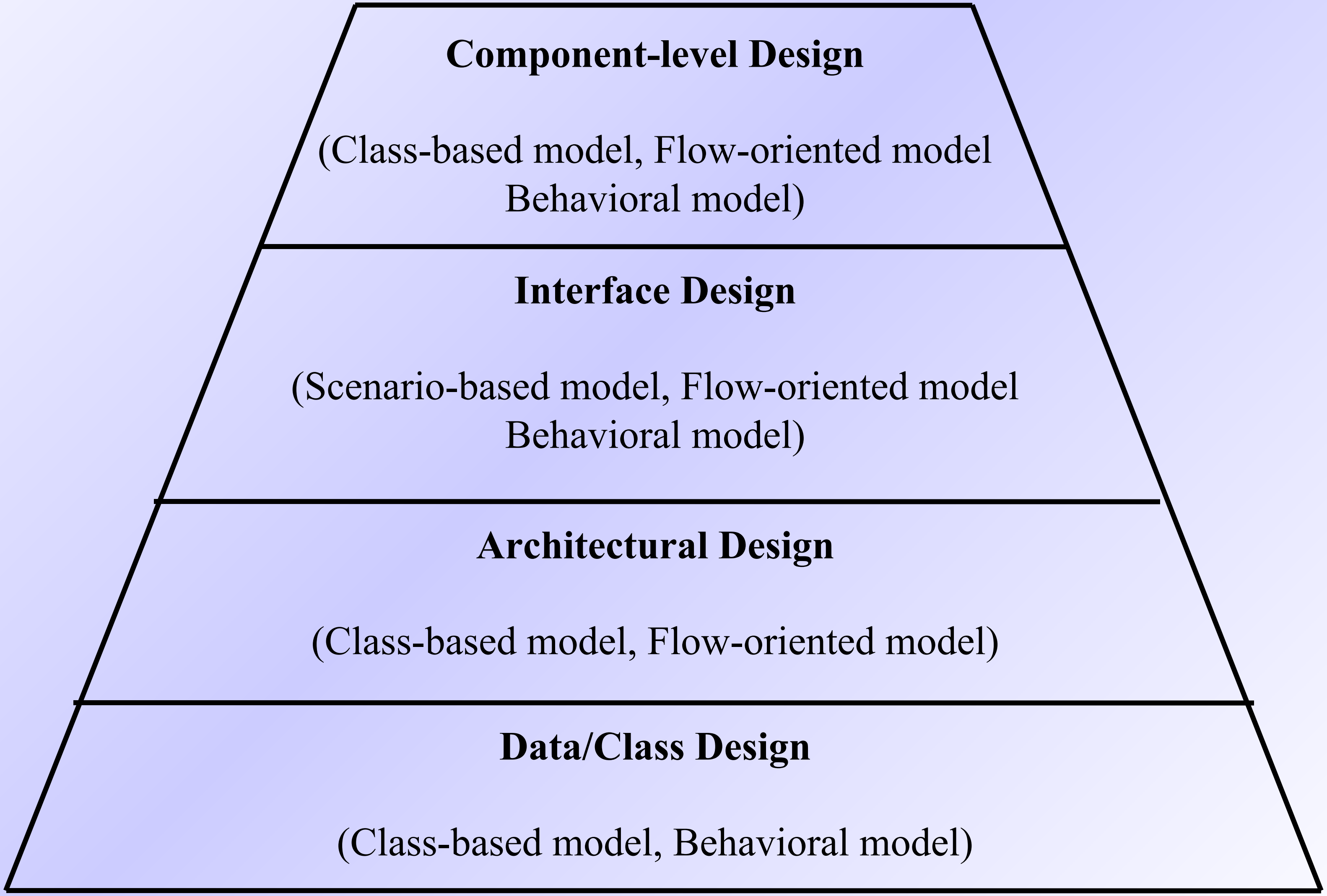
- Each element of the analysis model provides information that is necessary to create the four design models
  - The data/class design transforms analysis classes into design classes along with the data structures required to implement the software
  - The architectural design defines the relationship 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)

Task Set for Software Design

1) Examine the information domain model and design appropriate data structures for data objects and their attributes
2) Using the analysis model, select an architectural style (and design patterns) that are appropriate for the software
3) Partition the analysis model into design subsystems and allocate these subsystems within the architecture
    a) Design the subsystem interfaces
    b) Allocate analysis classes or functions to each subsystem
4) Create a set of design classes or components
    a) Translate each analysis class description into a design class
    b) Check each design class against design criteria; consider inheritance issues
    c) Define methods associated with each design class
    d) Evaluate and select design patterns for a design class or subsystem

(More on next slide)

Task Set for Software Design (continued)

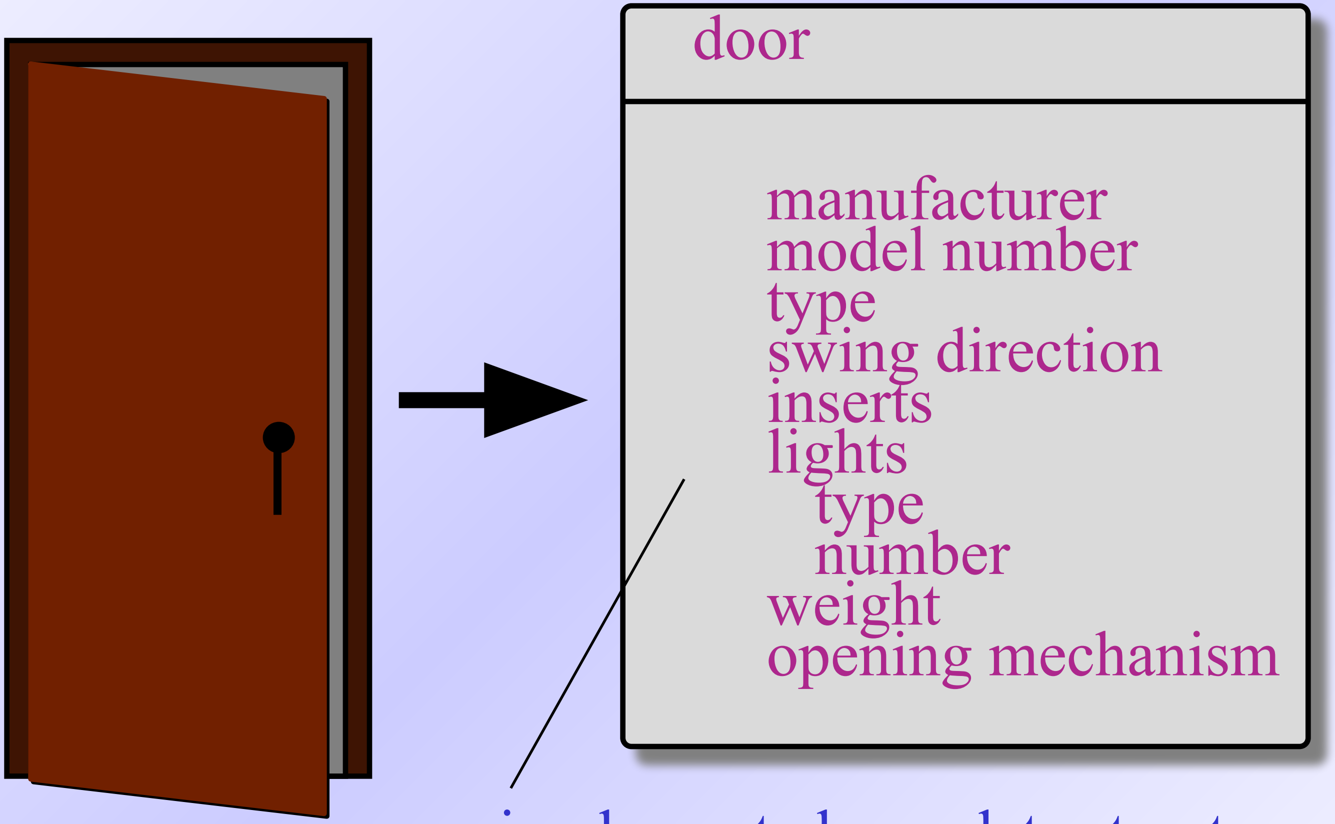
5) Design any interface required with external systems or devices
6) Design the user interface
7) Conduct component-level design
   a) Specify all algorithms at a relatively low level of abstraction
   b) Refine the interface of each component
   c) Define component-level data structures
   d) Review each component and correct all errors uncovered
8) Develop a deployment model
   - Show a physical layout of the system, revealing which components will be located where in the physical computing environment

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

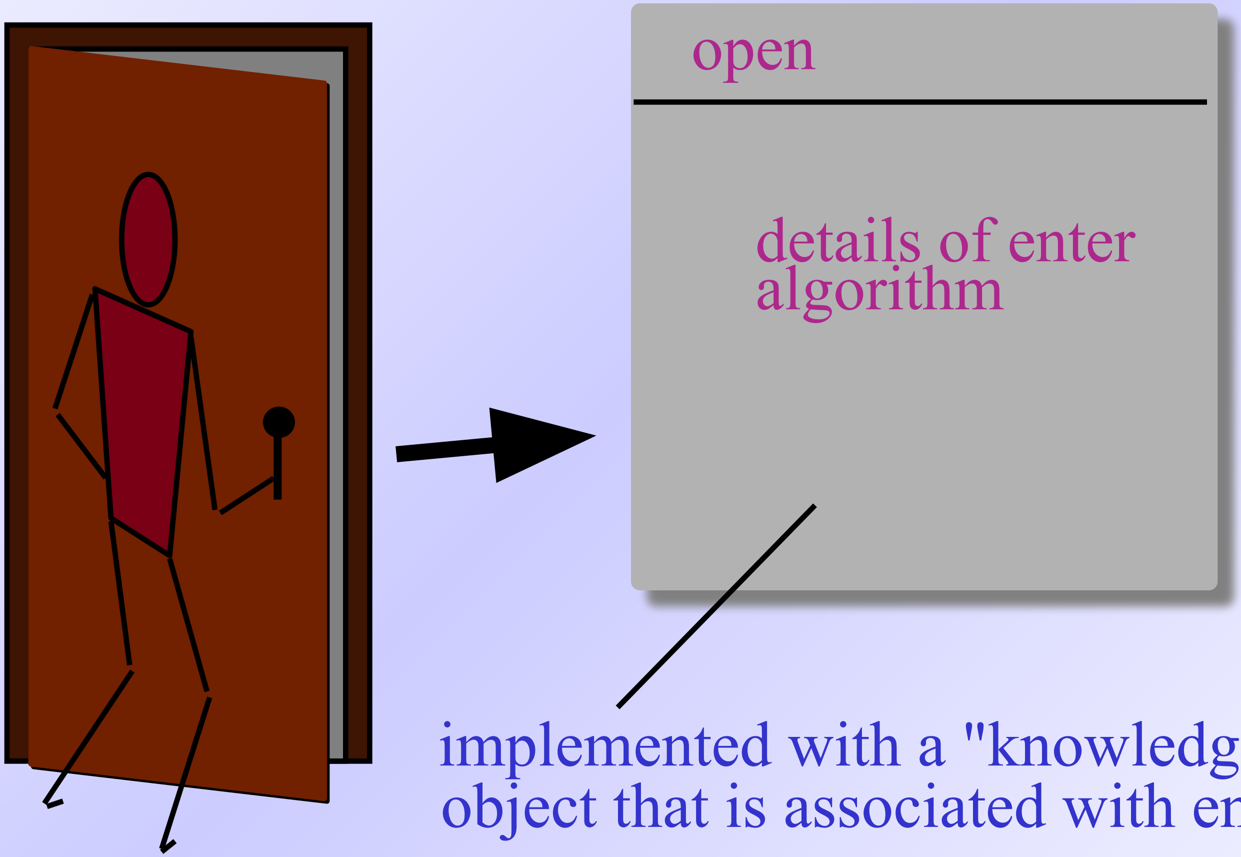


door

- manufacturer
- model number
- type
- swing direction
- inserts
- lights
  - type
  - number
- weight
- opening mechanism

implemented as a data structure

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 solves a particular design problem 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

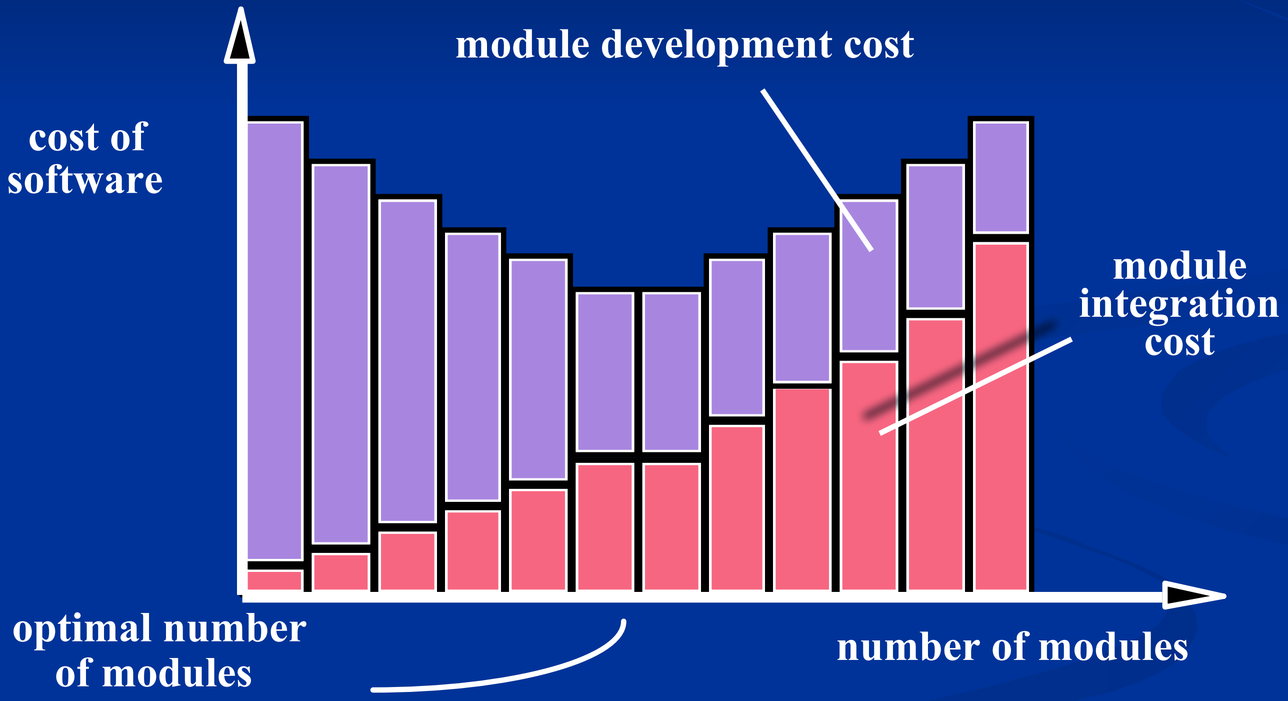
(more on next slide)

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?


module development cost
cost of software
module integration cost
optimal number of modules
number of modules