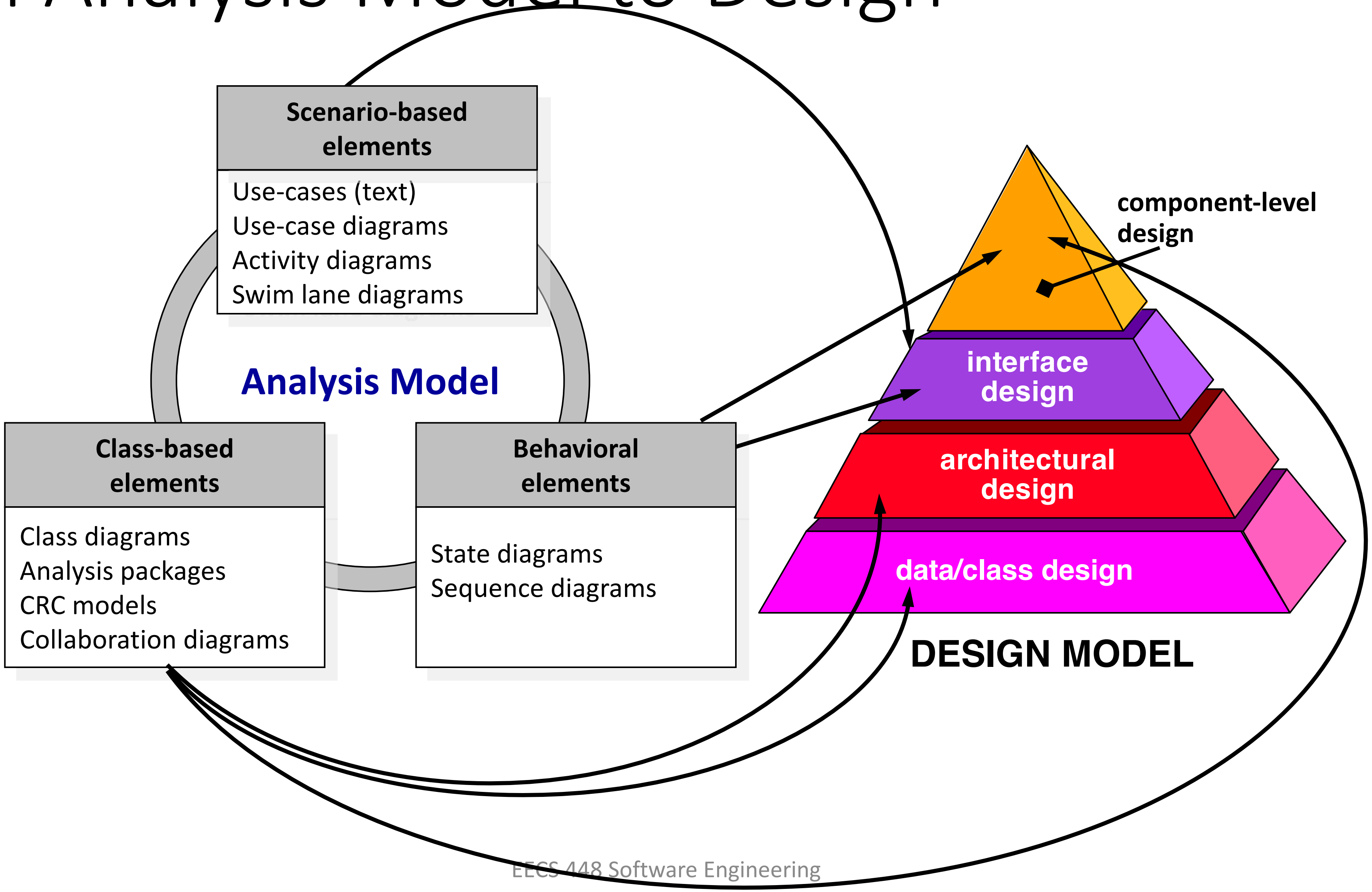
Design Engineering

Prof. Alex Bardas

From Analysis Model to Design

- In previous stages, we focused on obtaining
  - Requirements
  - An analysis model
- Now, we set the stage for construction
  - Choose elements to form alternative design solutions
  - Follow design models for *architectural*, *interface*, *component-level* and *data/class* design
  - An iterative process from a high level of abstraction to lower levels of abstraction through refinements

From Analysis Model to Design



Quality Attributes

The FURPS Quality Attributes [Hewlett-Packard]

- *Functionality:* evaluate the feature set, capability of the program, generality of the functions, and the overall security
- *Usability:* overall aesthetics, consistency and documentations
- *Reliability:* accuracy of output, frequency and severity of failure, mean time to failure, ability to recover and predictability
- *Performance:* processing speed, response time, resource consumption, throughput, and efficiency
- *Supportability:* overall maintainability in terms of extensibility, adaptability, serviceability, testability, compatibility, configurability

Software Quality

- Quality of design should be assessed with technical reviews during the iterative process
- A good design has three **characteristics**
  1. The design must implement all of the explicit requirements contained in the requirements model, and it must accommodate all of the implicit requirements desired by the customer.

Software Quality

- Quality of design should be assessed with technical reviews during the iterative process
- A good design has three **characteristics**

  2. The design must be a readable, understandable guide for those who generate code and for those who test and subsequently support the software.

Software Quality

- Quality of design should be assessed with technical reviews during the iterative process
- A good design has three **characteristics**
  3. The design should provide a complete picture of the software, addressing the data, functional, and behavioral domains from an implementation perspective.

Quality Guidelines

Technical criteria for good design:

- A design should exhibit an architecture that

  (1) Is created using recognizable architectural styles or patterns

  (2) Is composed of components that exhibit good design characteristics

  (3) Can be implemented in an evolutionary fashion

Quality Guidelines

Technical criteria for good design:

- A design should be modular
  - ➢The software should be logically partitioned into elements or subsystems
- A design should contain distinct representations of data, architecture, interfaces, and components.
- A design should lead to data structures that are appropriate for the classes to be implemented and are drawn from recognizable data patterns.
- A design should lead to components that exhibit independent functional characteristics.

Quality Guidelines

Technical criteria for good design:

- A design should lead to interfaces that reduce the complexity of connections between components and with the external environment.
- A design should be derived using a repeatable method that is driven by information obtained during software requirements analysis.
- A design should be represented using a notation that effectively communicates its meaning.

Fundamental Design Concepts

- **Abstraction**—data, procedure, control
- **Architecture**—the overall structure of the software
- **Patterns**—“conveys the essence” of a proven design solution
- **Separation of concerns**—any complex problem can be more easily handled if it is subdivided into pieces
- **Modularity**—compartmentalization of data and function
- **Information hiding**—controlled interfaces
- **Functional independence**—single-minded function and low coupling
- **Refinement**—elaboration of detail for all abstractions
- **Aspects**—a mechanism for understanding how global requirements affect design
- **Refactoring**—a reorganization technique that simplifies the design
- **OO design concepts**
- **Design Classes**—provide design details that will enable analysis classes to be implemented

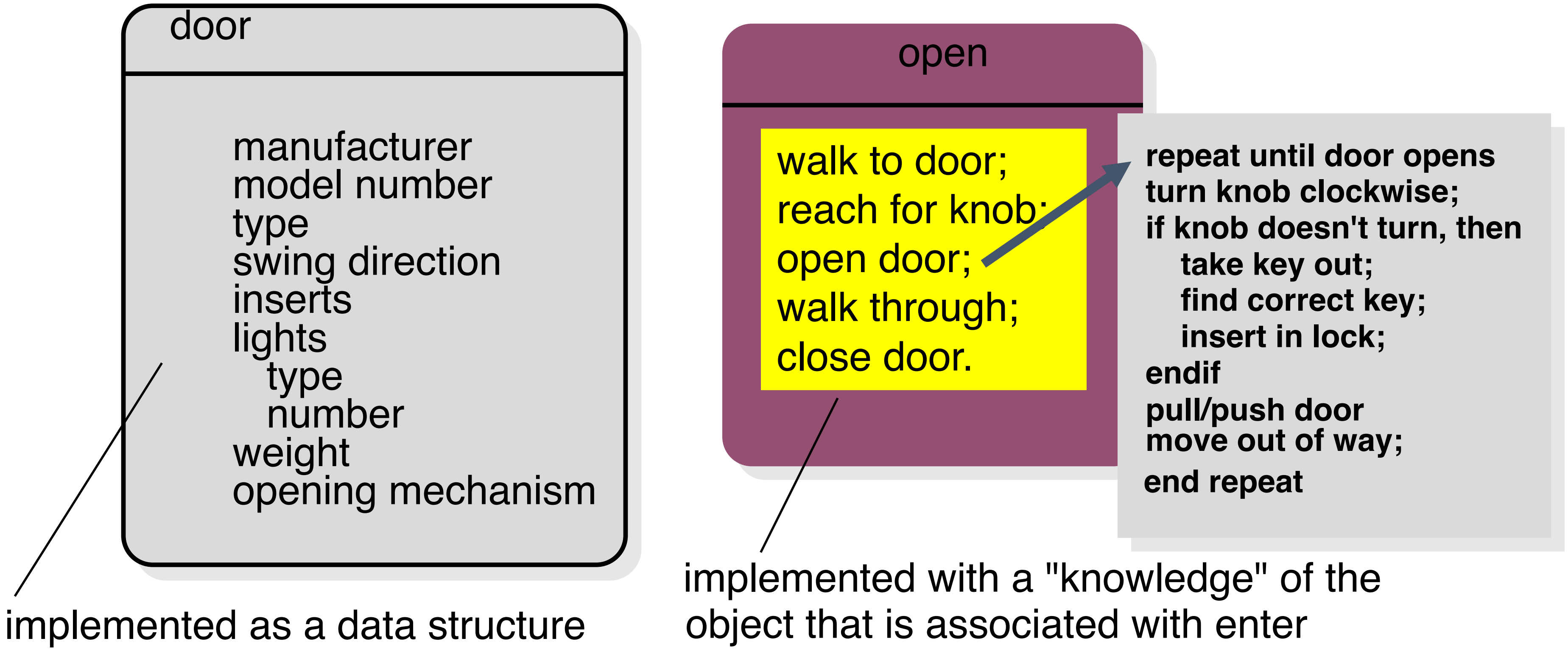
Fundamental Design Concepts

1. Abstraction

- Problem and solution can be understood at different levels of abstraction
  - Procedural abstraction: a sequence of instructions of a specific and limited function, with details of the function suppressed
  - Data abstraction: a collection of data that describes a data object
- Refinement – a top-down design strategy

An abstraction example

- "open the door"



Fundamental Design Concepts

2. Architecture

- "The overall structure of the software and the ways in which that structure provides conceptual integrity for a system. "
  - The structure of modules and data
- Structural properties: components of a system and interactions
- Extra-functional properties: non-functional requirements
- Families of related systems: repeatable patterns that are commonly encountered in the design of families of similar systems

Fundamental Design Concepts

3. Patterns

- A design pattern describes a design structure that solves *a particular design problem within a specific context and amid a set of forces*
- Should provide a description about in case
  - The pattern is applicable to the current work
  - The pattern can be reused
  - The patter can serve as a guide for developing similar but functionally different patterns

Design Pattern Template

- **Pattern name** describes the essence of the pattern
- **Intent** describes the pattern and what it does
- **Also-known-as** synonyms for the pattern
- **Motivation** provides an example of the problem
- **Applicability** specific design situations in which the pattern is applicable
- **Structure** classes that are required to implement the pattern
- **Participants** responsibilities of the classes
- **Collaborations** how the participants collaborate to carry out their responsibilities
- **Consequences** describes the “design forces” that affect the pattern and the potential trade-offs
- **Related patterns** cross-references related design patterns

Fundamental Design Concepts

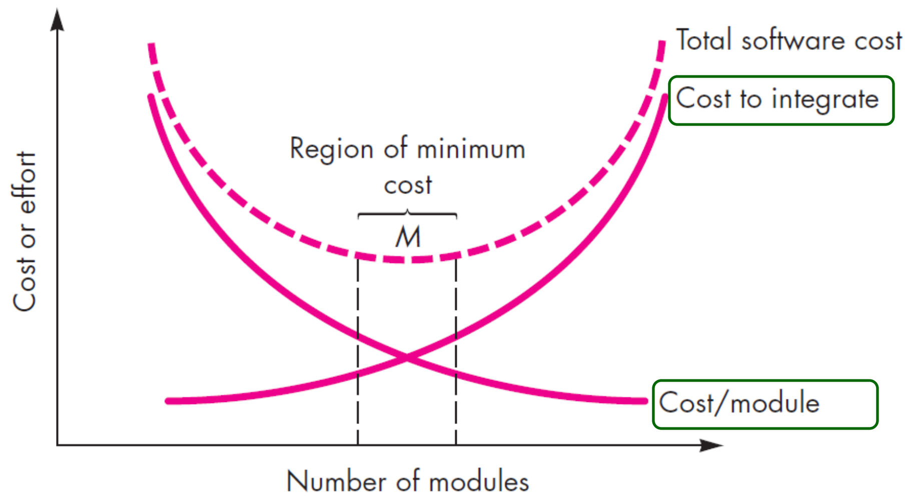
4. Separation of Concerns

- "Concern": a feature specified by the requirement model
- A divide-and-conquer strategy
  - If solving the combined problem requires more efforts than the sum of solving two individual problems independently
- Lead to software **modularity**, **functional independence**, **refinement**, and **aspects**

Fundamental Design Concepts

5. Modularity

- Helps development, increments and changes, testing and debugging, long-term maintenance
- *What is the "right" number and size of the modules?*
- As the number of modules grows, the effort associated with integration also grows



Fundamental Design Concepts

6. Information Hiding

- How to decompose software into modules?
- Define and enforce **access constraints**

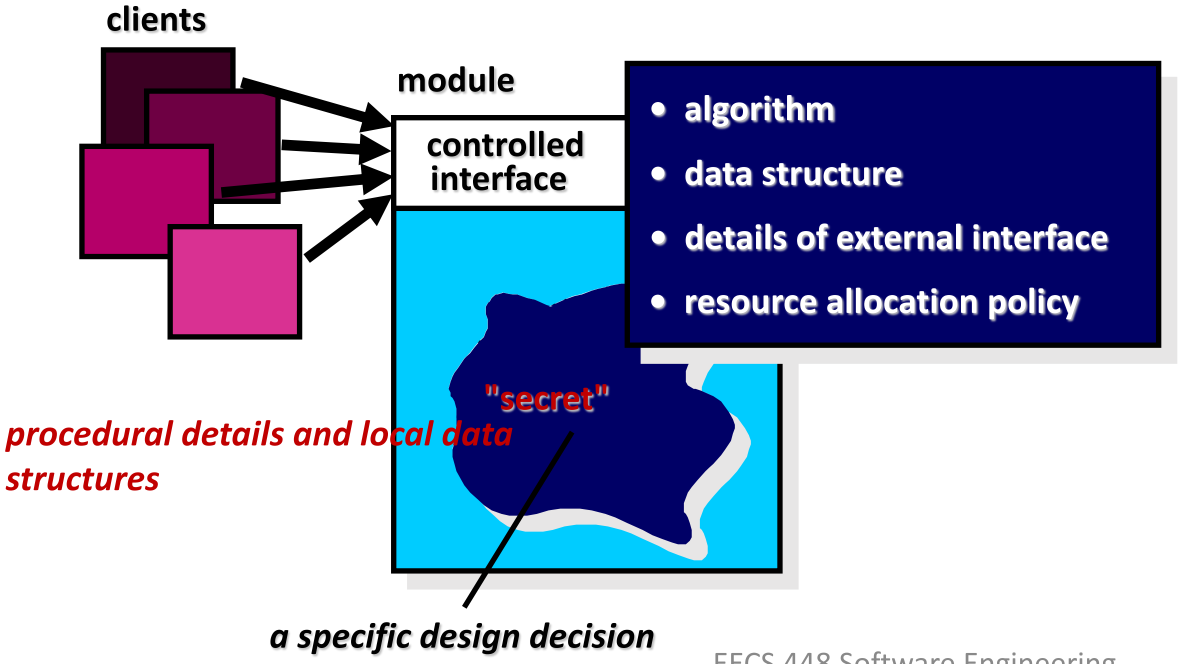
"information within a module is inaccessible to other modules that have no need for such information"

- So, most data and procedural details are hidden from other modules
- Leads to encapsulation – an attribute of high quality design

Fundamental Design Concepts

7. Functional Independence

- Each module addresses a specific subset of requirements
- Each module has a simple interface



✓reduces the likelihood of "side effects"

✓discourages the use of global data

✓limits the global impact of local design decisions

✓emphasizes communication through controlled interfaces

Fundamental Design Concepts

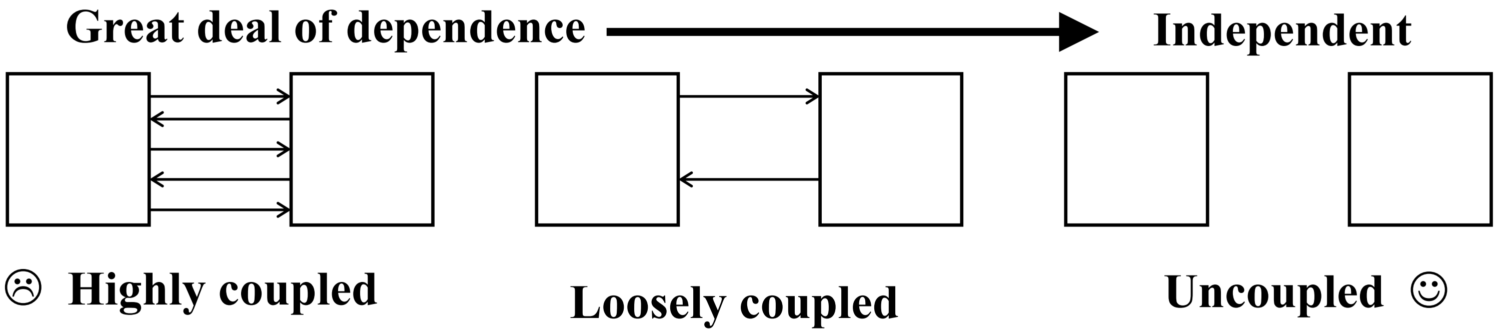
7. Functional Independence

- Two criteria: **Cohesion** and **Coupling**
- Cohesion is an indication of the relative functional strength of a module
  - A cohesive module performs a single task: requires little interaction with components in other parts
  - Avoid modules that perform many unrelated functions

Fundamental Design Concepts

7. Functional Independence

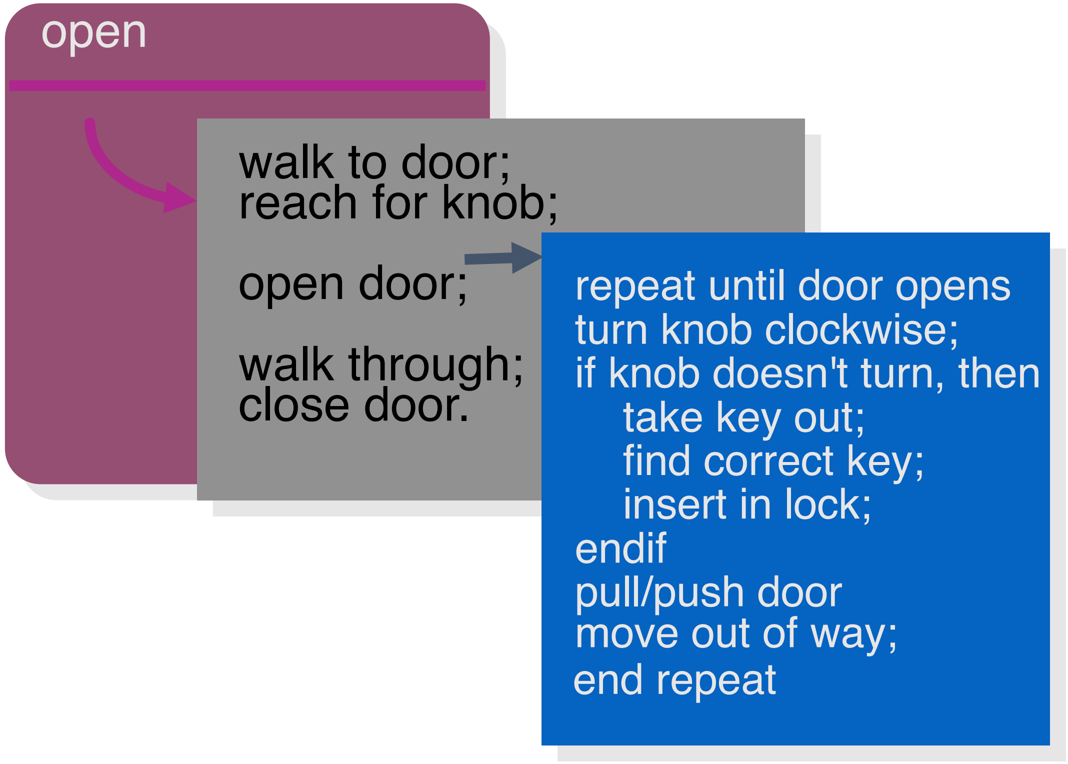
- Two criteria: **Cohesion** and **Coupling**
- Coupling is an indication of the relative interdependence among modules
  - Relies on the interface complexity between modules
  - Goal: as loose as possible = as independent as possible



Fundamental Design Concepts

8. Refinement

- "Elaboration"
- A top-down design strategy successively refining levels of procedural details
- As design progresses, refinement reveals the low-level details



Fundamental Design Concepts

9. Aspects

- "Concerns": features requirements, data structures, use cases, quality issues, variants, collaborations, patterns, …
- However, some concerns cannot be easily compartmentalized and span over the entire system
  - A requirement could crosscut another
- An aspect is a cross-cutting concern

Aspects - An Example

- Consider two requirements for the *SafeHomeAssured.com* WebApp
  - Requirement A states that *a registered user could access videos from cameras placed throughout a space*
  - Requirement B is a generic security requirement that states that *a registered user must be validated prior to using SafeHomeAssured*
    - It’s applicable for all functions that are available to registered users.
  - B* cross-cuts A*, where A* and B* are design representations of concerns

Fundamental Design Concepts

10. Refactoring

- A reorganization technique
  - Simplifies the code of a component without changing its function
  - “Improves the internal structure of a design (or source code) without changing its external functionality or behavior” (Chapter 5 – Agile Development)
- In refactoring, examine the existing design for
  - Redundancy
  - Unused design elements
  - Inefficient or unnecessary algorithms
  - Poorly constructed or inappropriate data structures
  - Any other design failure that can be corrected to yield a better design

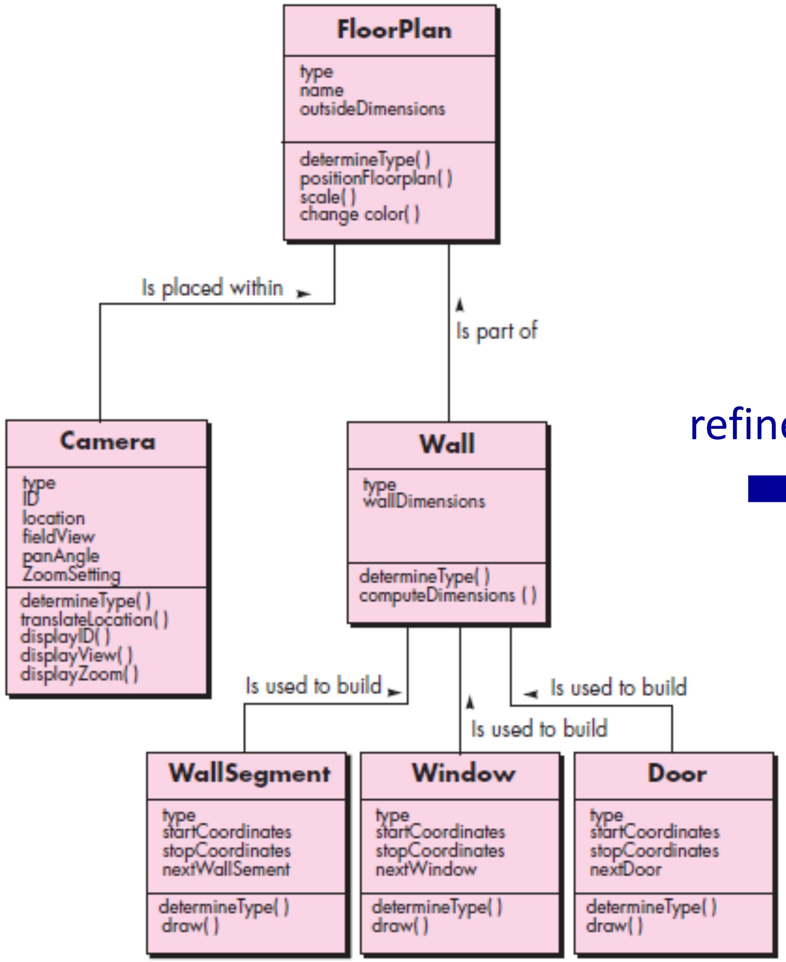
OO Design Concepts

- message <parameters>
  - Stimulate behavior to occur in the receiving object
- inheritance
  - All responsibilities of a superclass are immediately inherited by all subclasses
- polymorphism
  - Reduces effort required to extend the existing design

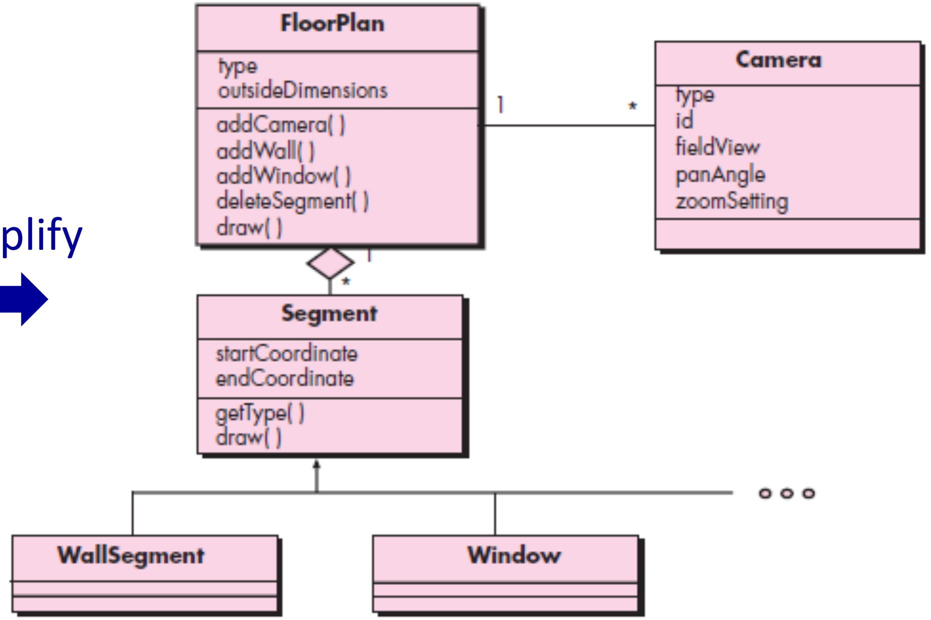
Design Classes

- A set of design classes
    1. User interface classes
    2. Business domain (BD) classes
    3. Process classes – lower-level business abstractions required to fully manage the BD classes
    4. Persistent classes – database
    5. System classes – management and control functions
- High cohesion & low coupling
- Complete & primitive

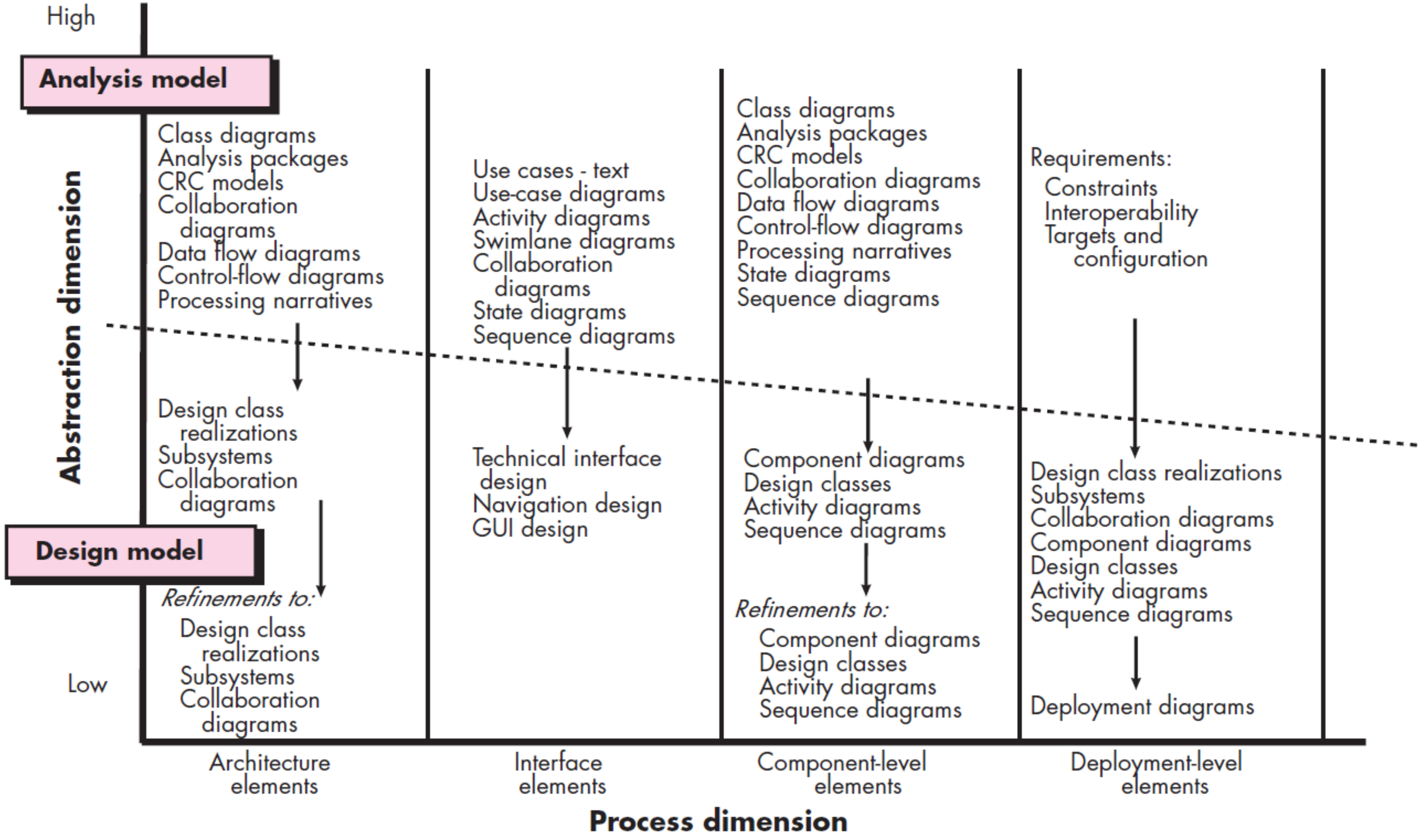
From Analysis Classes to Design Classes



refine, simplify



Design model: two dimensions



References

- Prof. Fengjun Li's EECS 448 Fall 2015 slides
- This slide set has been extracted and updated from the slides designed to accompany *Software Engineering: A Practitioner's Approach, 8/e* (McGraw-Hill 2014) by Roger Pressman