Project Announcement

- Each team leader should send me an e-mail saying who your team leader is by tonight (April 24)
- 1st deadline is extended to May 1st (next Tuesday)
  - Team leader should submit a complete document both in hardcopy and softcopy
  - Do not forget to hand in "who did what" list
- Your requirement document should be solidly organized
  - the requirement document needs to be conveniently referred/traced back during your subsequent design/implement process
    - Table of Contents
    - Overview
  - For the same reason, indicate SEPA pages based on which you describe use-cases
- Review meeting of your requirement document will be TBD.

Chapter 9
Design Engineering

Moonzoo Kim
CS Division of EECS Dept.
KAIST
moonzoo@cs.kaist.ac.kr
http://pswlab.kaist.ac.kr/courses/cs550-07

Roadmap of SEPA covered in CS550

- **March : Ch 1 – Ch 5**
  - 1. Intro to SE
  - 2. A Generic View of Process
  - 3. Process Models
  - 4. An Agile View of Process
  - 5. SE Practice
  - 6. System Engineering

- **April: Ch 7- Ch 9**
  - 7. Requirement Engineering
    - Req. eng tasks
    - Req. elicitation
    - Developing use-cases
    - Building the analysis model
  - 8. Building the Analysis Model
  - 9. Design Engineering

  SafeHome Project
  - Use-case diagram
  - Use-cases
  - Activity diagram

- **May : Ch 10 – Ch 14**
  - 10. Creating an Architectural Design
  - 11. Modeling Component-Level Design
  - 12. Performing UI Design
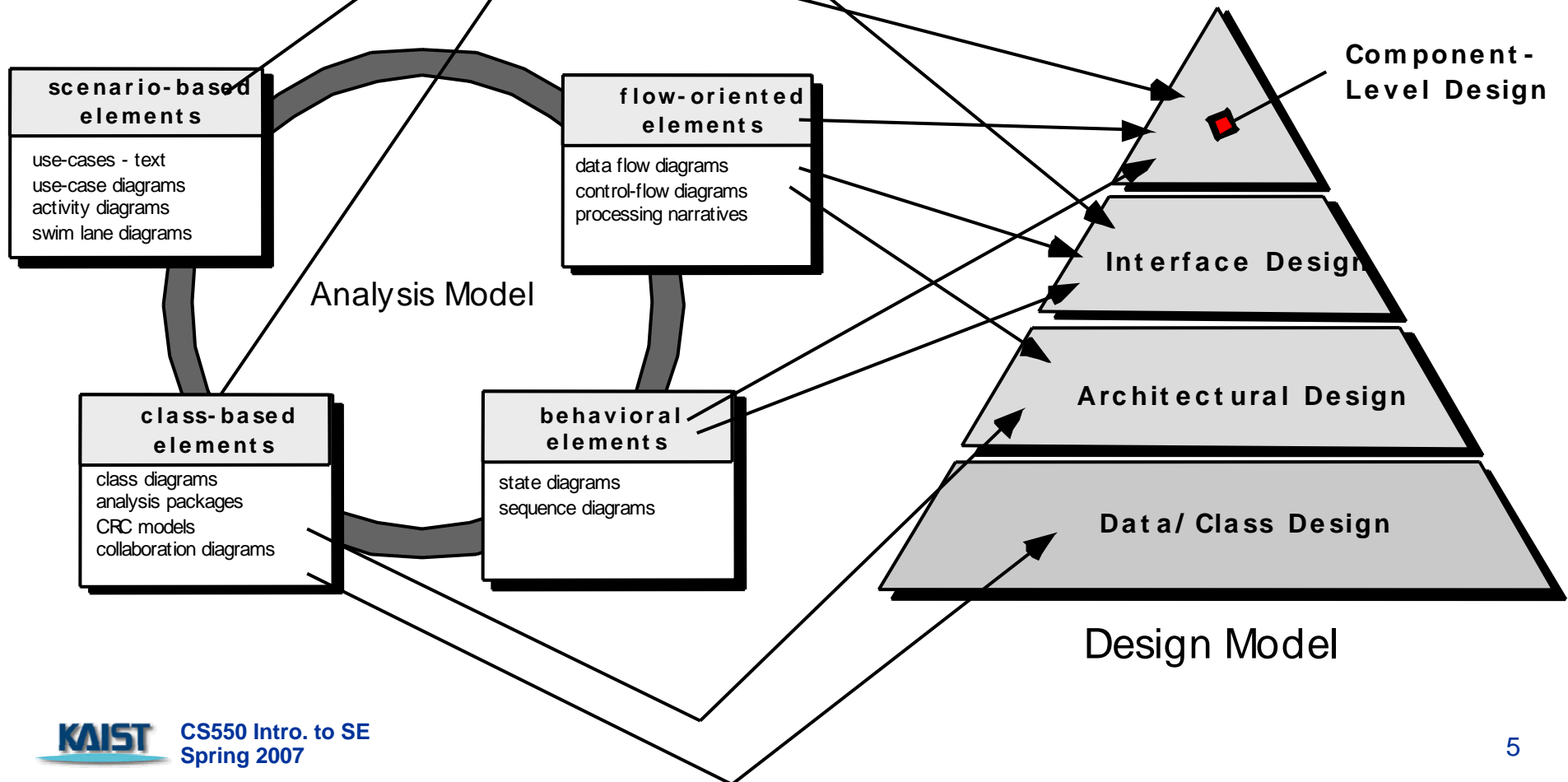 - 13. Testing Strategies
  - 14. Testing Tactics

  SafeHome Project
  - Class diagram
  - CRC cards
  - Sequence diagram
  - State diagram

Overview of Ch 9. Design Engineering

- 9.1 Design within the Context of SE
- 9.2 Design Process and Design Quality
- 9.3 Design Concepts
    - Abstraction
    - Architecture
    - Patterns
    - Modularity
    - Information Hiding
    - Functional Independence
    - Refinement
    - Refactoring
    - Design Classes
- 9.4 Design Model
    - Data Design Elements
    - Architectural Design Elements
    - Interface Design Elements
    - Component-level Design Elements
    - Deployment-level Design Elements
- 9.5 Pattern-based SW Design
    - Describing a Design Pattern
    - Using Patterns in Design
    - Frameworks

Analysis Model -> Design Model

scenario-based elements
- use-cases - text
- use-case diagrams
- activity diagrams
- swim lane diagrams

flow-oriented elements
- data flow diagrams
- control-flow diagrams
- processing narratives

class-based elements
- class diagrams
- analysis packages
- CRC models
- collaboration diagrams

behavioral elements
- state diagrams
- sequence diagrams

Analysis Model

Component-Level Design

Interface Design

Architectural Design

Data/ Class Design

Design Model

Design and Quality

- the design must implement all of the explicit requirements contained in the analysis model, and it must accommodate all of the implicit requirements desired by the customer.

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

- the design should provide a complete picture of the software, addressing the data, functional, and behavioral domains from an implementation perspective.

Quality Guidelines

1. A design should exhibit an architecture which
   1. has been 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
2. A design should be modular
3. A design should contain distinct representations of data, architecture, interfaces, and components.
4. A design should lead to data structures that are
   1. appropriate for the classes to be implemented
   2. drawn from recognizable data patterns.
5. A design should lead to components that exhibit independent functional characteristics.
6. A design should lead to interfaces that reduce the complexity of connections between components and with the external environment.
7. A design should be derived using a repeatable method that is driven by information obtained during software requirements analysis.
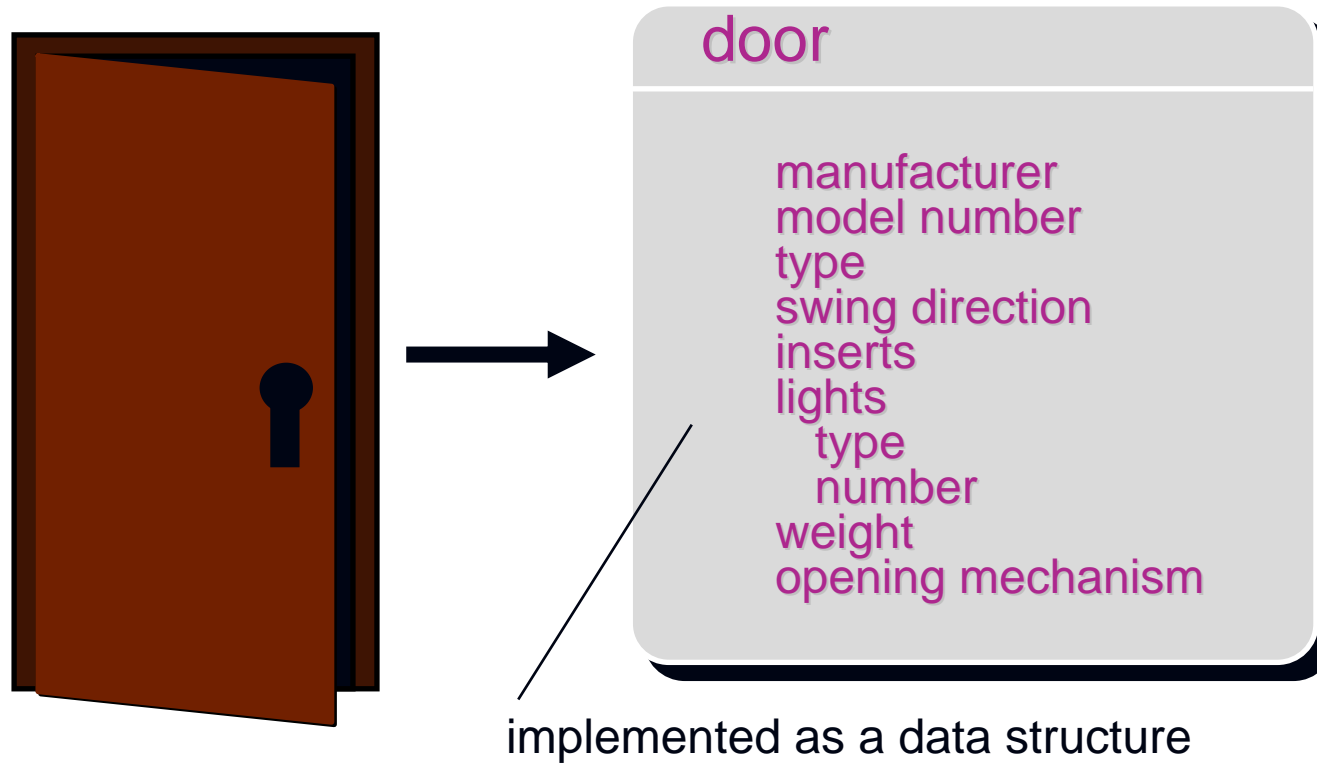8. A design should be represented effectively for communicating its meaning.

Quality Attributes – FURPS [GRA87]

- **Functionality**
  - Assessed by evaluating feature set and capabilities of the program and generality of the functions that are delivered

- **Usability**
  - Assessed by considering human factors, overall aesthetics

- **Reliability**

- **Performance**

- **Supportability**
  - Maintainability
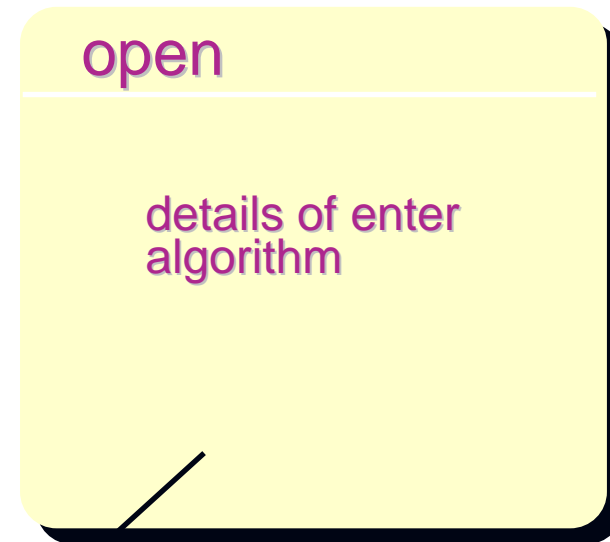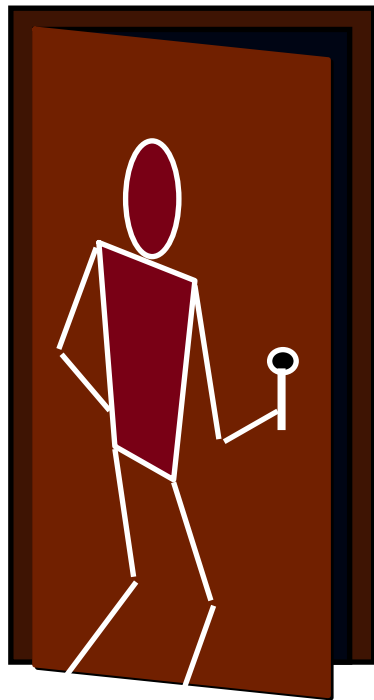 - Compatibility, ease of configuration, ease of installation, etc

Fundamental SW Design Concepts

- **Abstraction**
  - data, procedure
- **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

open

details of enter
algorithm

implemented with a "knowledge" of the

Design Patterns

- The best designers in any field have an uncanny ability to see
    - patterns that characterize a problem
    - patterns that can be combined to create a solution
- A design pattern may also consider a set of design forces.
    - *Design forces* describe non-functional requirements (e.g., ease of maintainability, portability) associated the software for which the pattern is to be applied.
- The pattern characteristics (classes, responsibilities, and collaborations) indicate the attributes of the design that may be adjusted to enable the pattern to accommodate a variety of problems.
- Levels of abstraction
    - Architectural patterns
    - Design patterns
    - Idioms (coding patterns)

Design Patterns Template

- *Pattern name*
  - describes the essence of the pattern in a short but expressive name
- *Intent*
  - describes the pattern and what it does
- *Motivation*
  - provides an example of the problem
- *Applicability*
  - notes specific design situations in which the pattern is applicable
- *Structure*
  - describes the classes that are required to implement the pattern
- *Participants*
  - describes the responsibilities of the classes that are required to implement the pattern
- *Collaborations*
  - describes how the participants collaborate to carry out their responsibilities
- *Related patterns*—cross-references related design patterns

Modular Design

easier to build, easier to change, easier to fix ...
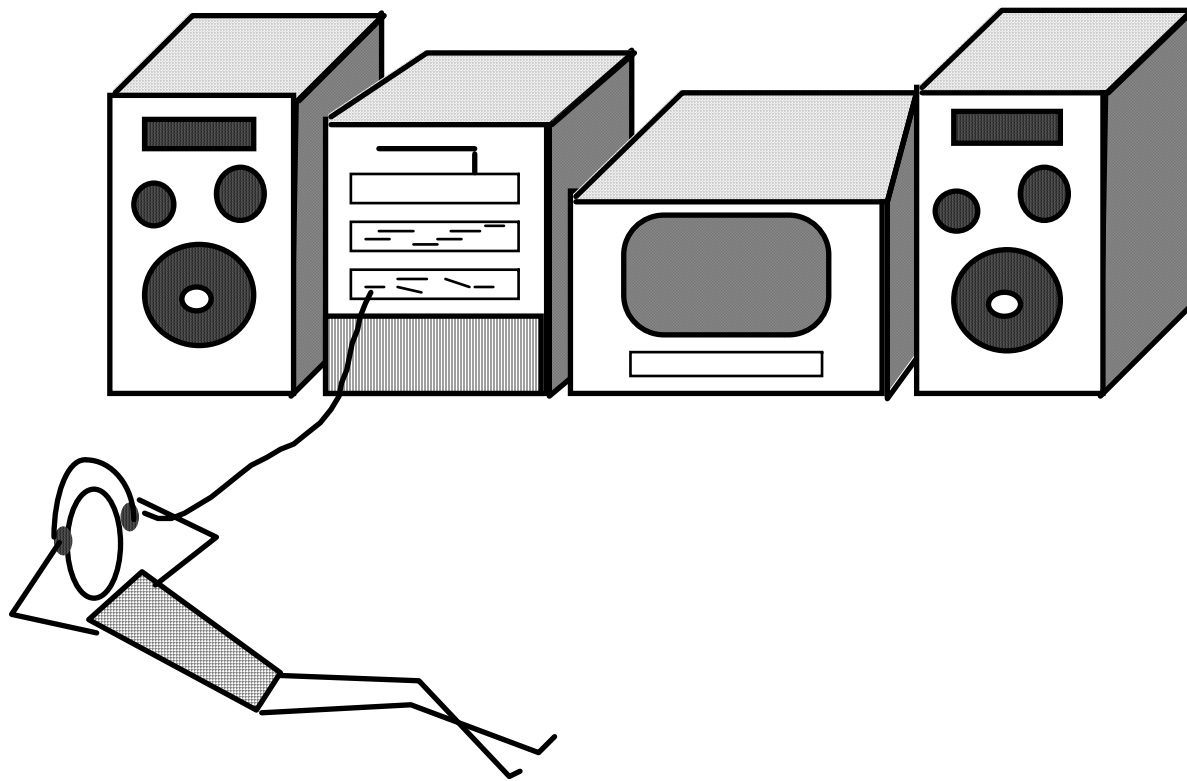
Modularity: Trade-offs

*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

Information Hiding

module

controlled interface

clients

- algorithm
- data structure
- details of external interface
- resource allocation policy

"secret"

a specific design decision
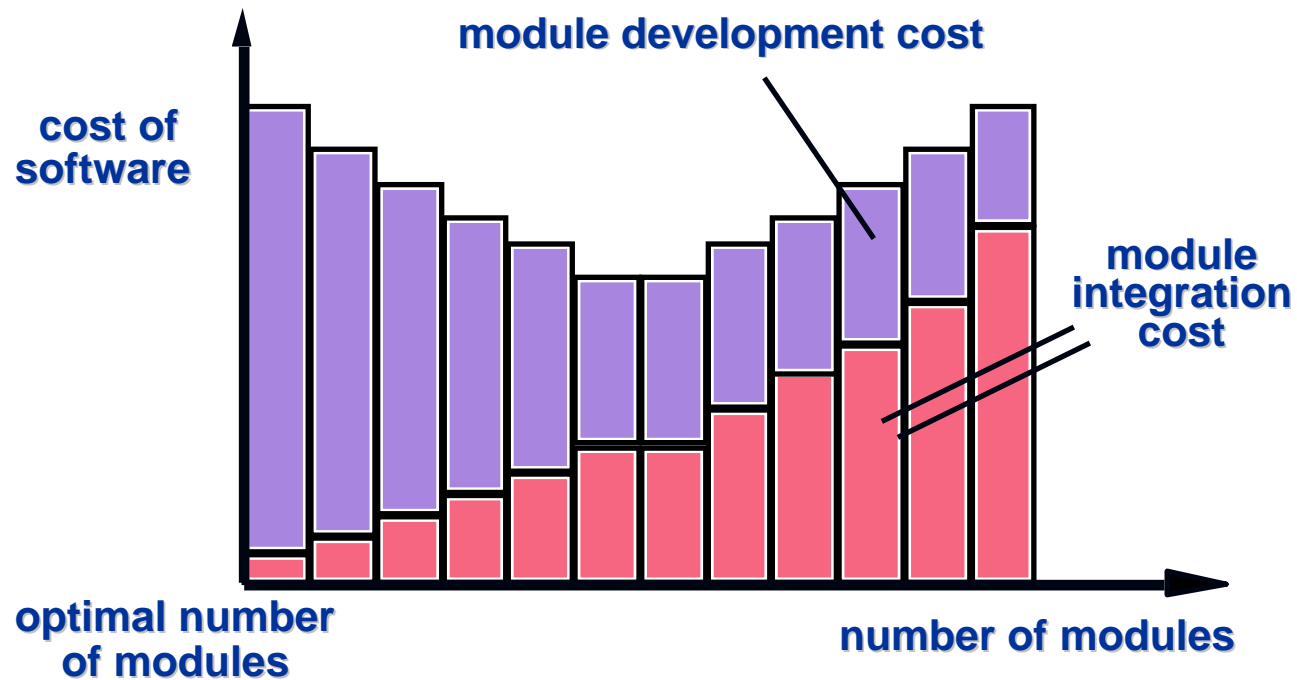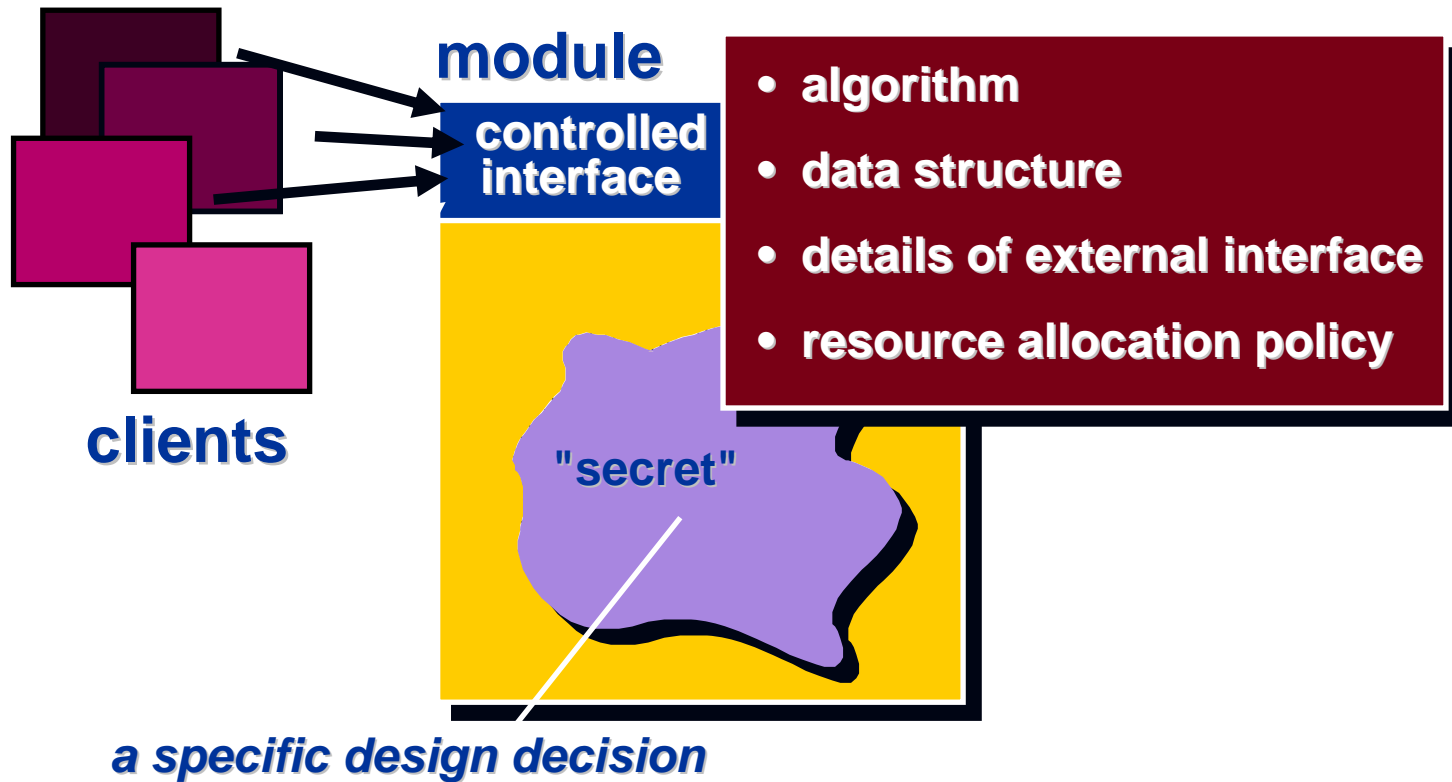
Why Information Hiding?

- reduces the likelihood of "side effects"
- limits the global impact of local design decisions
- emphasizes communication through controlled interfaces
- discourages the use of global data
- leads to encapsulation—an attribute of high quality design
- results in higher quality software

Stepwise Refinement

open

walk to door;
reach for knob;

open door;

walk through;
close door.

repeat until door opens
turn knob clockwise;
if knob doesn't turn, then
   take key out;
   find correct key;
   insert in lock;
endif
pull/push door
move out of way;
end repeat
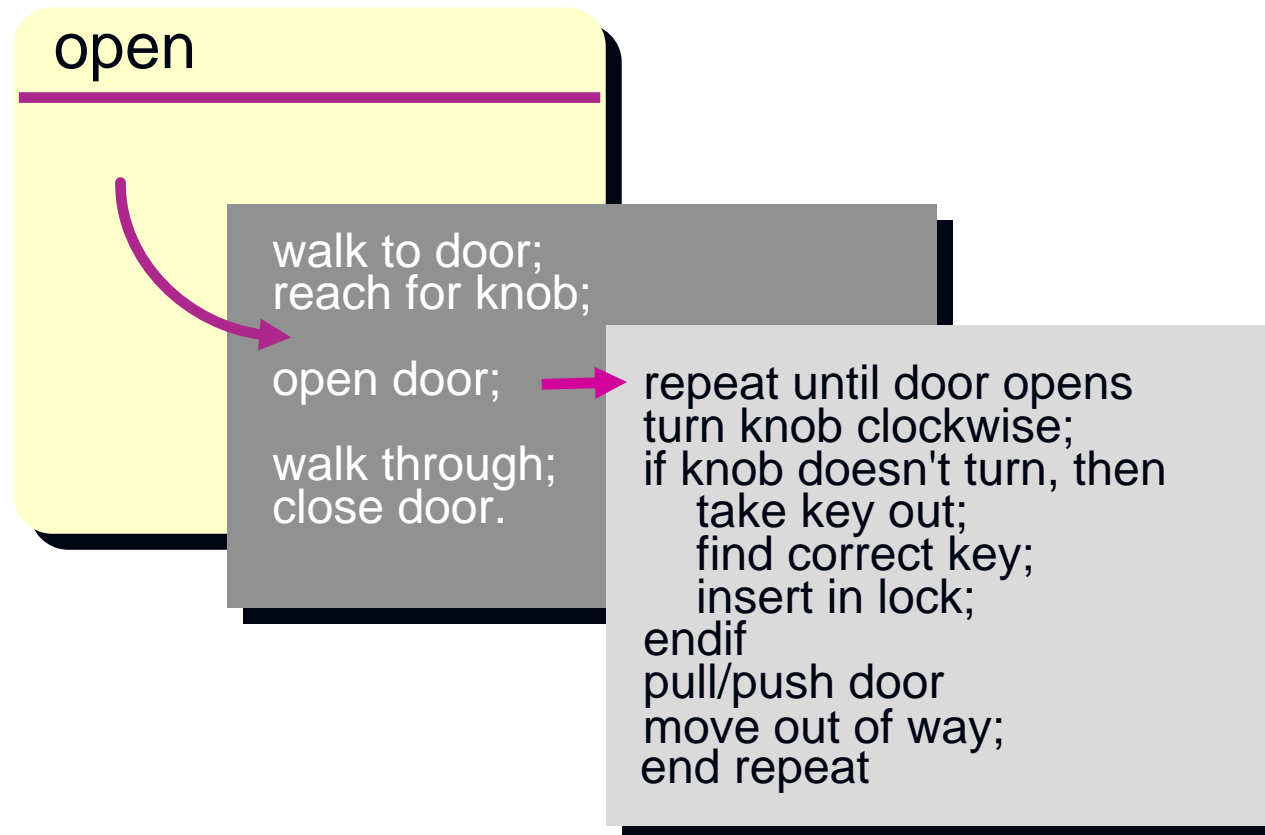
Refactoring

- Fowler [FOW99] defines refactoring in the following manner:
    - "Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code [design] yet improves its internal structure."

- When software is refactored, the existing design is examined for
    - redundancy
    - unused design elements
    - inefficient or unnecessary algorithms
    - poorly constructed or inappropriate data structures
    - or any other design failure that can be corrected to yield a better design.
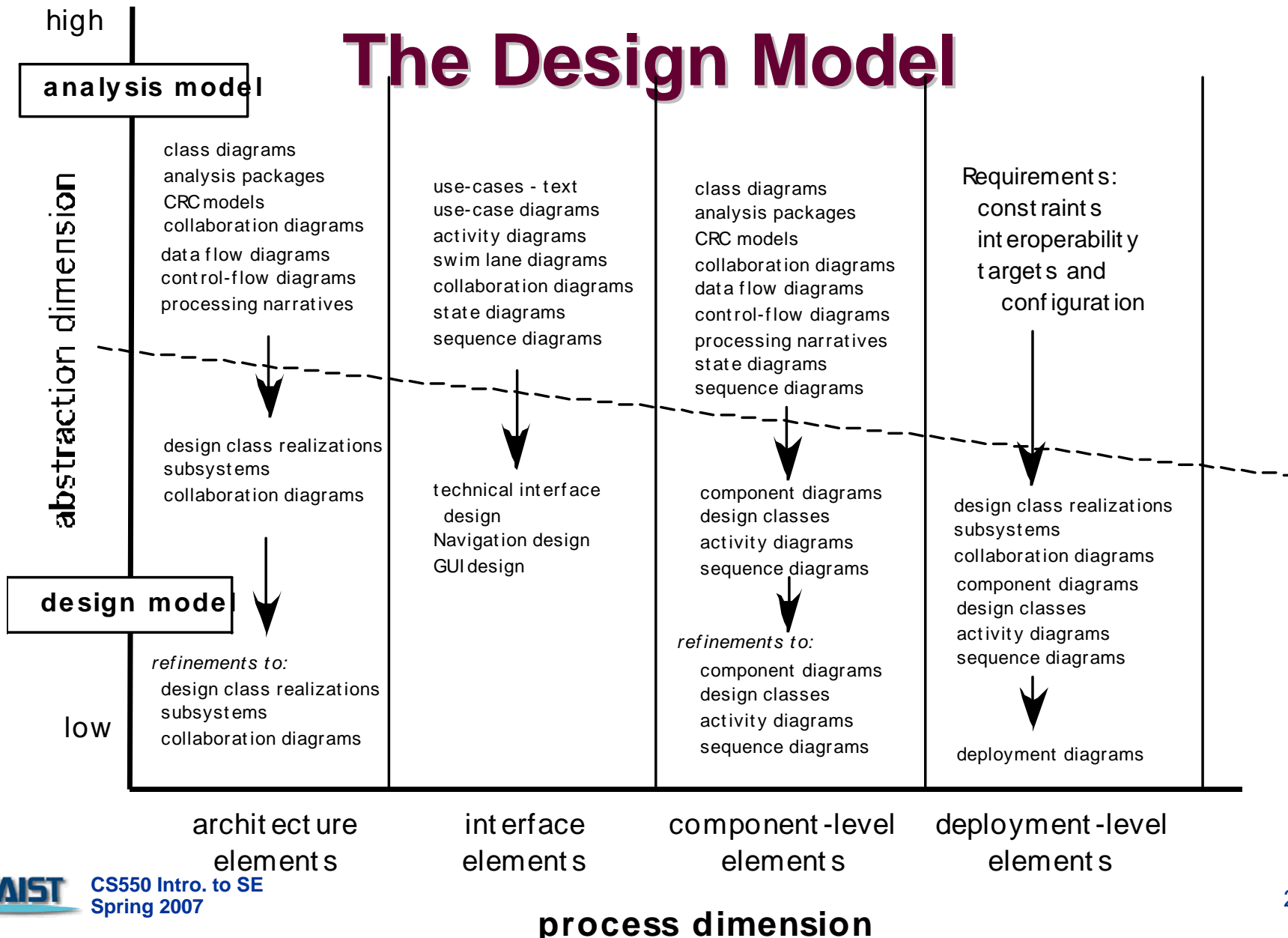
Functional Independence

COHESION - the degree to which a module performs one and only one function.

COUPLING - the degree to which a module is "connected" to other modules in the system.
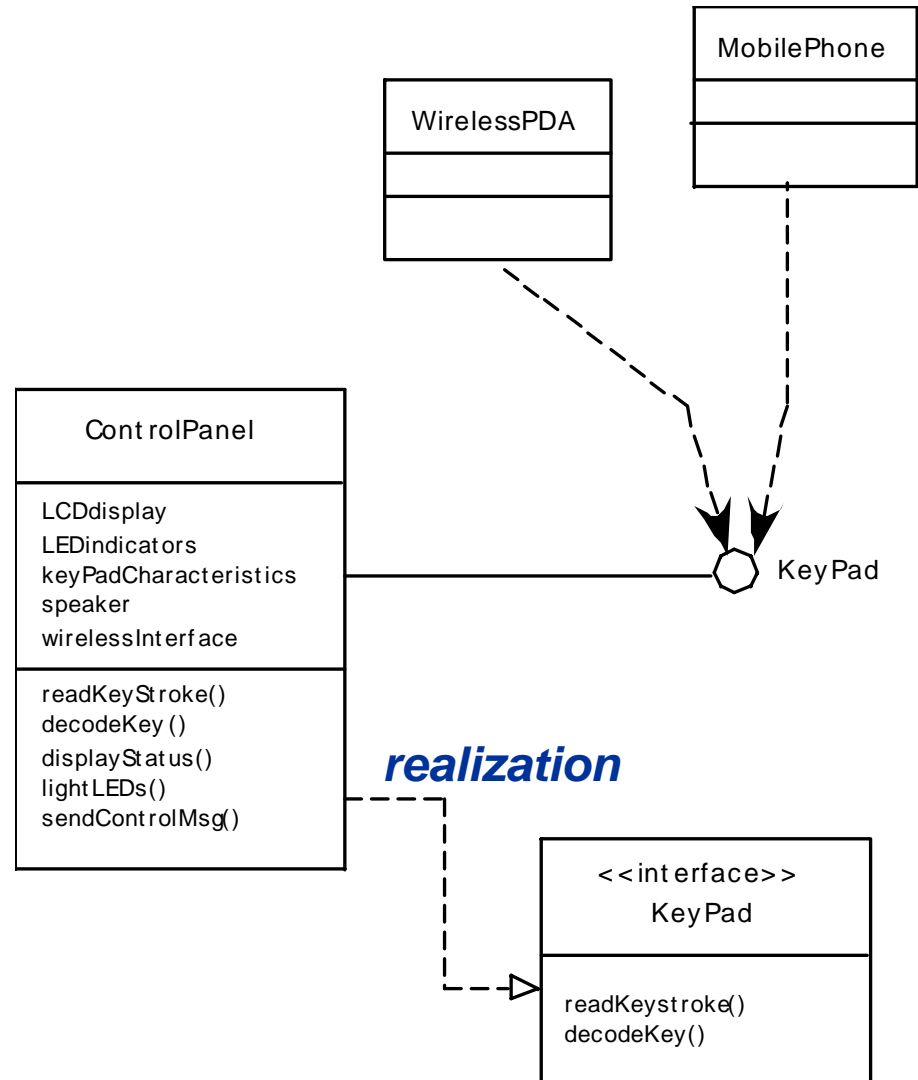
The Design Model

high

| analysis model |

abstraction dimension

class diagrams
analysis packages
CRC models
collaboration diagrams
data flow diagrams
control-flow diagrams
processing narratives

use-cases - text
use-case diagrams
activity diagrams
swim lane diagrams
collaboration diagrams
state diagrams
sequence diagrams

class diagrams
analysis packages
CRC models
collaboration diagrams
data flow diagrams
control-flow diagrams
processing narratives
state diagrams
sequence diagrams

Requirements:
constraints
interoperability
targets and
configuration

design class realizations
subsystems
collaboration diagrams

technical interface
design
Navigation design
GUI design

component diagrams
design classes
activity diagrams
sequence diagrams

design class realizations
subsystems
collaboration diagrams
component diagrams
design classes
activity diagrams
sequence diagrams

| design model |

refinements to:
design class realizations
subsystems
collaboration diagrams

refinements to:
component diagrams
design classes
activity diagrams
sequence diagrams

deployment diagrams

low

architecture
elements

interface
elements

component-level
elements

deployment-level
elements
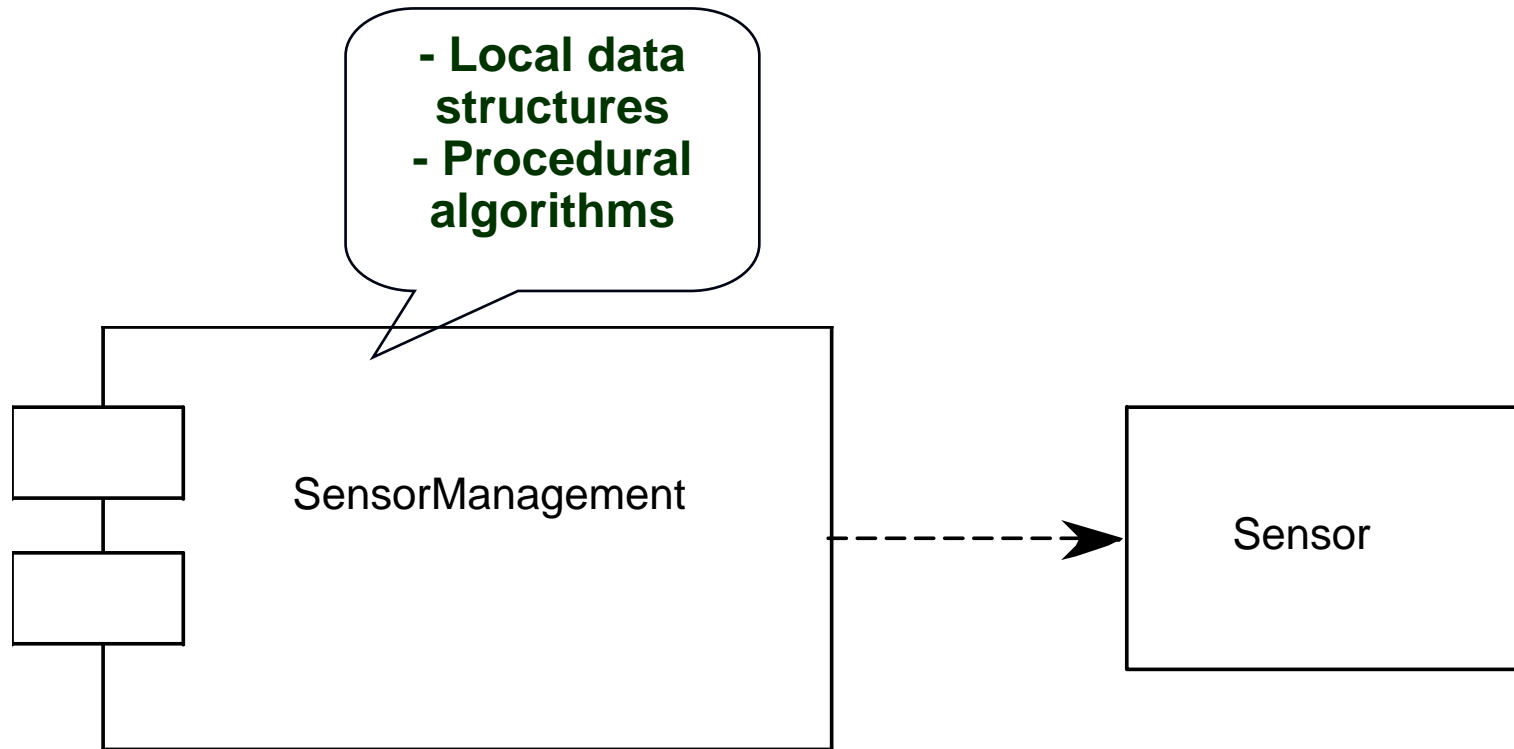
process dimension

Design Model Elements

- **Data elements**
  - a model of data and/or information that is represented at a high level of abstraction (the customer/user's view of data)
  - Refined into more implementation-specific representations
- **Architectural elements**
  - Application domain
  - Analysis classes, their relationships, collaborations and behaviors are transformed into design realizations
  - Patterns and "styles" (Chapter 10)
- **Interface elements**
  - the user interface (UI)
  - external interfaces to other systems, devices, networks or other producers or consumers of information
  - internal interfaces between various design components
- **Component elements**
- **Deployment elements**

Interface Elements

Component Elements

- Local data structures
- Procedural algorithms

SensorManagement

Sensor

Deployment Elements