설계 개념

한국항공대학교 소프트웨어학과 지승도교수
R.S. Pressman

Design within the Context of Software Engineering

- Software design
  - Technical kernel of software engineering
  - The last software engineering action within the modeling activity
  - Sets the stage for construction

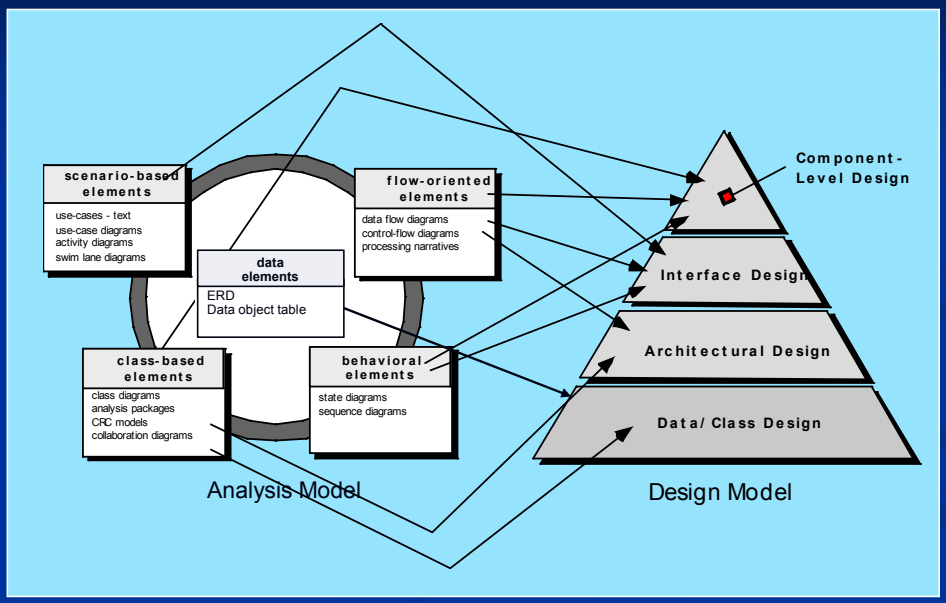
  ➔ Data/class design
    - Transforms analysis-class model into design class realizations and required data structure to implement the software

  ➔ Architectural design
    - Structural elements of the software

  ➔ Interface design
    - Describe how the software communicates with systems that interoperate with it, and with humans who use it

  ➔ Component-level design
    - Architecture into a procedural description of software component
    - Class based models, flow models, behavior models serve as the basis

Analysis Model ➔ Design Model



Design Process

- Software design is an iterative process through which requirements are translated into a "blueprint" for constructing the software
- Three characteristics for a good design
  - 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

Fundamental Design Concepts

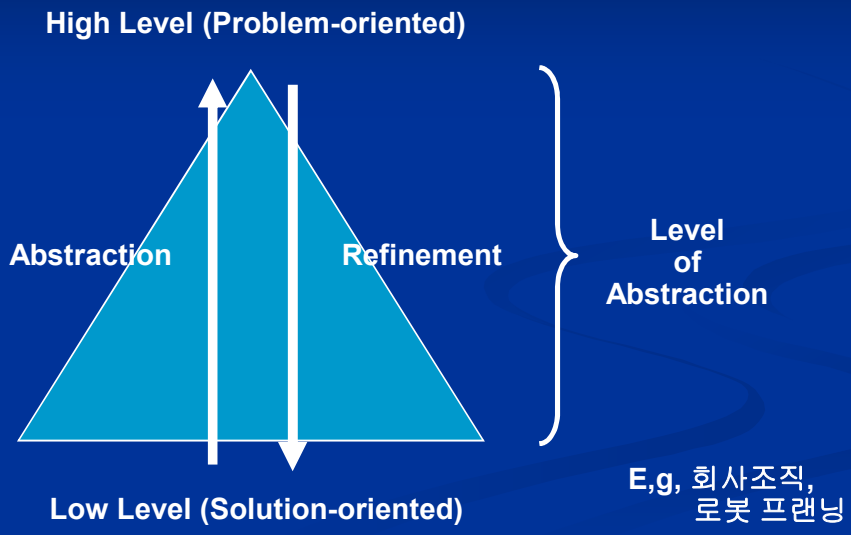
- Abstraction — data, procedure, control
- Architecture — the overall structure of the software
- Patterns — "conveys the essence" of a proven design solution
- Modularity — compartmentalization of data and function
- Information hiding — controlled interfaces
- Functional independence — single-minded function and low coupling
- Refinement — elaboration of detail for all abstractions
- Refactoring — a reorganization technique that simplifies the design
- Design class — detail class to be implemented

Design Concept: Abstraction

- Level of Abstraction
- At the highest levels of abstraction, a solution is stated in broad terms using the language of the problem environment
- At lower levels of abstraction, a more detailed description of the solution is provided
- Procedural abstraction refers to a sequence of instructions that have a specific and limited function
- Data abstraction is a named collection of data that describes a data object

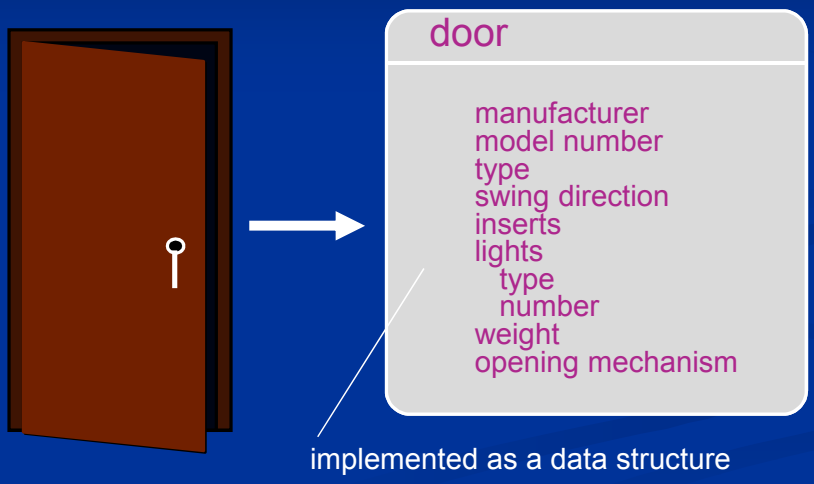
60조 세포
(몸)

Design Concept: Abstraction

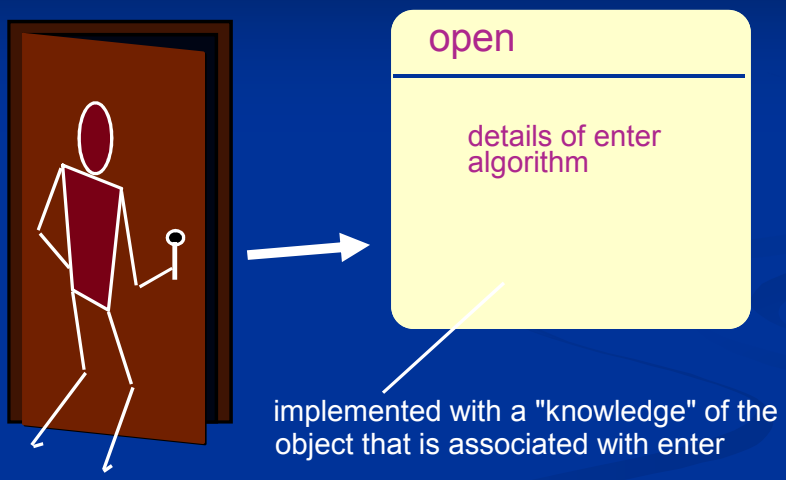


E,g, 회사조직,
로봇 프랜닝

Data Abstraction



Procedural Abstraction



Design Concept: Architecture

- Software architecture is

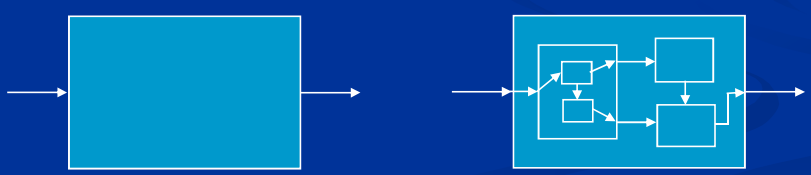
✓ the structure of components, the manner in which these component interact and
✓ the structure of data that are used by the component

Design Concept: Patterns

- A design structure that solves a particular design problem within a specific context and amid "forces" that may have an impact on the manner in which the pattern is applied and used
- Enable to determine;
  - (1) Whether the pattern is applicable to the current work
  - (2) Whether the pattern can be reused
  - (3) Whether the pattern can serve as a guide for developing a similar but functionally or structurally different pattern

Design Concept: Modularity

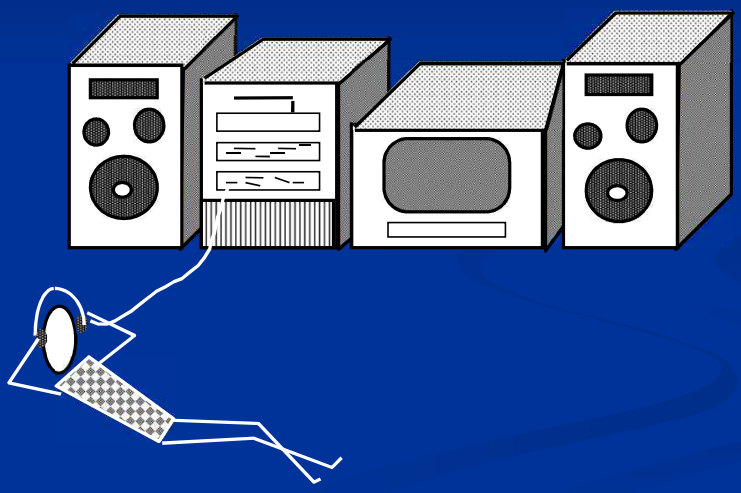
- Software architecture and design patterns embody modularity
- Software is divided into separately named and addressable component, sometimes called modules that are integrated to satisfy problem requirements
- "Divide and conquer"



"Monolithic" "Modular"

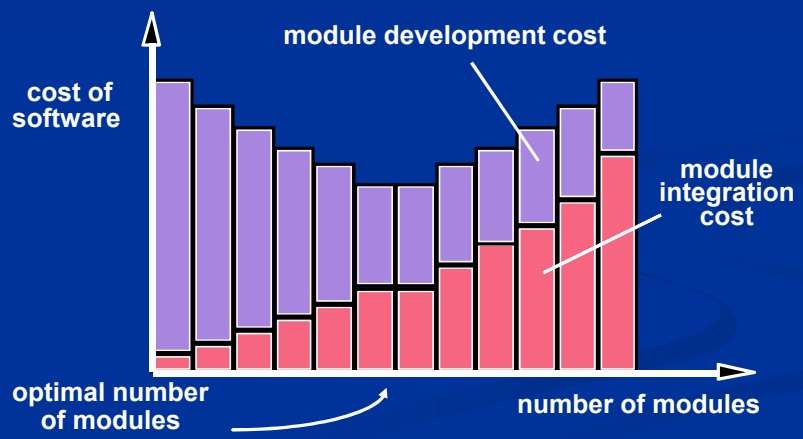
Modular Design

easier to build, easier to change, easier to fix ...

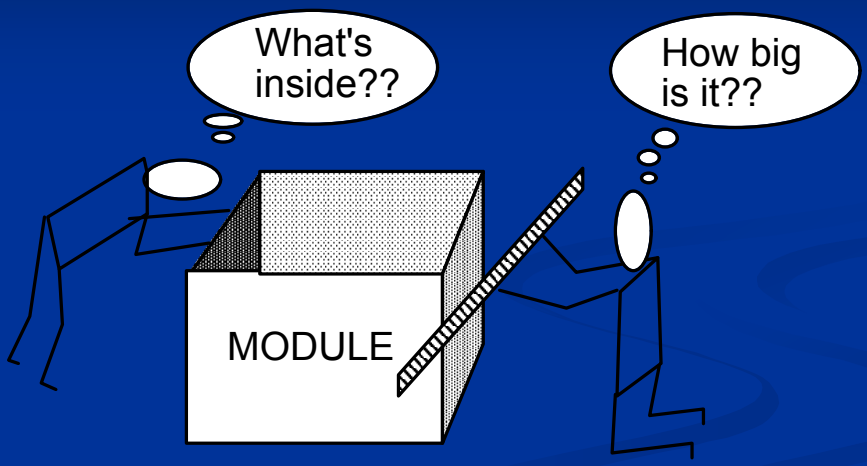


Modularity: Trade-offs

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



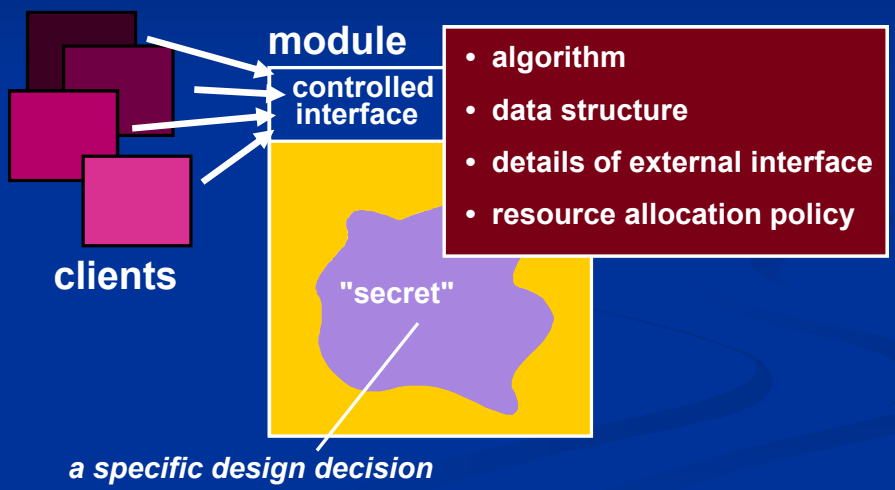
Sizing Modules: Two Views



Design Concept: Information hiding

- Modules should be "characterized by design decisions that hides from all other"
- Implies that effective modularity can be achieved by defining a set of independent modules that communicate with one another only that information necessary to achieve software function
- Use of information hiding
  - Provides the greatest benefits when modifications are required during testing and later, during software maintenance
  - Because inadvertent errors introduced during modification are less likely to propagate to other locations within the software

Information Hiding



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

Design Concept: Functional Independence

- Direct outgrowth of modularity and the concepts of abstraction and information hiding
- Independence is assessed using two qualitative criteria: cohesion and coupling.
  - Cohesion is an indication of the relative functional strength of a module.
  - Coupling is an indication of the relative interdependence among modules

Functional Independence

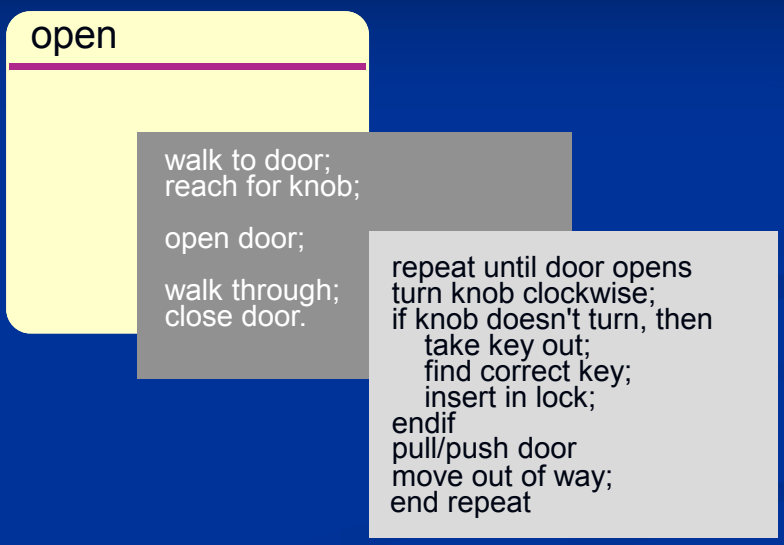
COHESION - the degree to which a module performs one and only one function.

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

Design Concept: Refinement

- Stepwise refinement
  - Top-down design strategy
  - A program is developed by successively refining levels of procedural detail
- Refinement is actually a process of elaboration
- Abstraction and refinement are complementary concepts
- Abstraction enables a designer to specify procedure and data and yet suppress low-level details
- Refinement helps the designer to reveal low-level details as design progresses

Stepwise Refinement



Design Concept: Refactoring

- Reorganization technique that simplified the design of a component without changing its function or behavior
- 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.

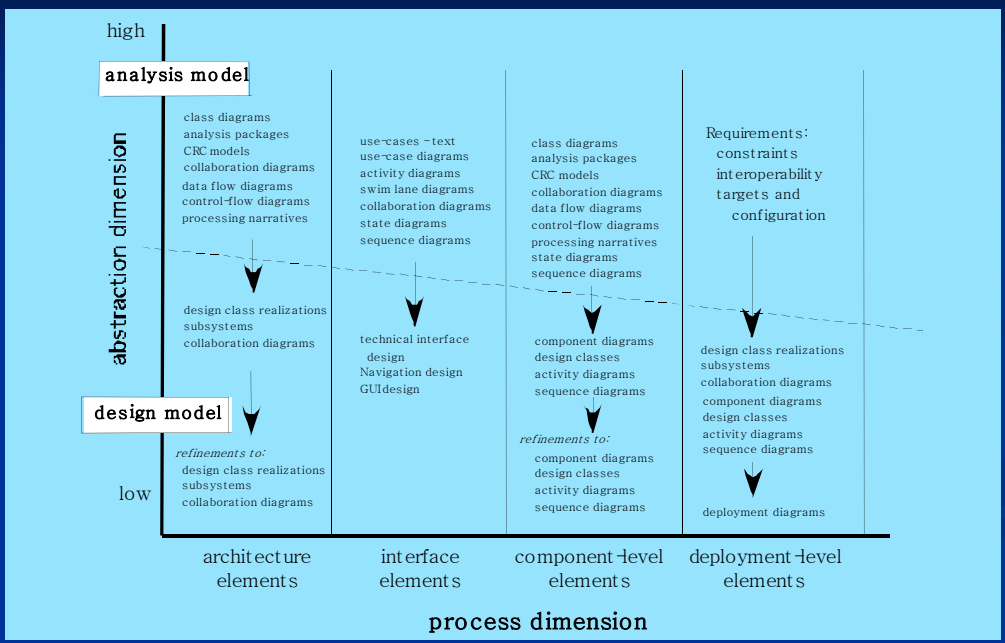
Design Concept: Design classes

- Refine the analysis classes by providing design detail that will enable the classes to be implemented
- Create a new set of design classes that implement a software infrastructure to support the business solution
  - Complete and sufficient
    - Should be the complete encapsulation of all attributes and methods that can reasonably be expected
  - Primitiveness
    - Methods associated with a design class should not provide another way to accomplish the same thing
  - High Cohesion
    - A cohesion design class has a small, focused set of responsibilities and single-mindedly applies attributes and methods to implement those responsibilities
  - Low coupling
    - Collaborate with one another
    - Should be kept to an acceptable minimum

The Design Model

- Process dimension indicates the evolution of the design model as design tasks are executed as part of the software process
- Abstraction dimension represents the level of detail as each element of the analysis model is transformed into a design equivalent and then refined iteratively

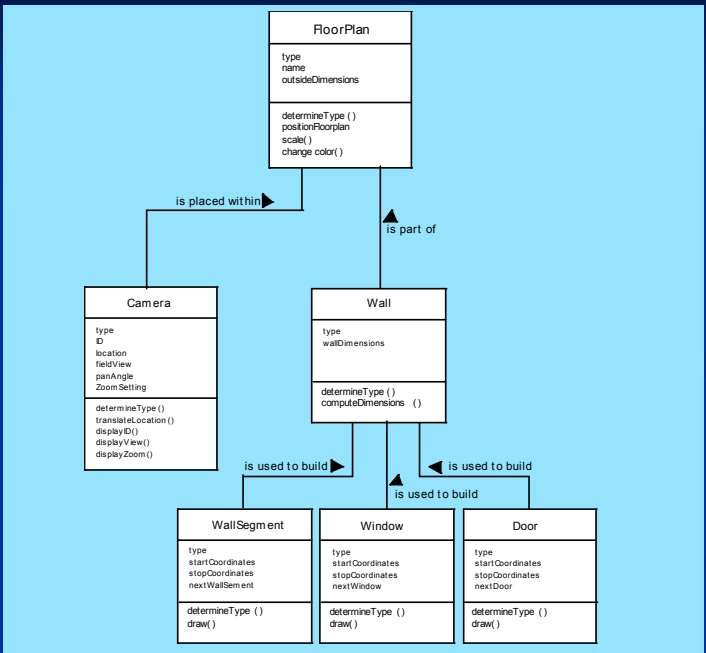
The Design Model



Design Model Elements

- Data elements
  - Data model --> data structures
  - Data model --> database architecture
- Architectural elements
  - Application domain
  - Analysis classes, their relationships, collaborations and behaviors are transformed into design realizations
  - Patterns and “styles”
- 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

Architecture Elements



Interface Elements

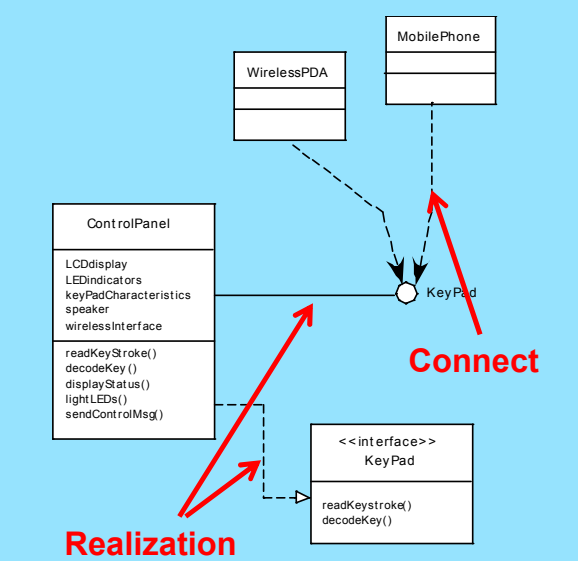
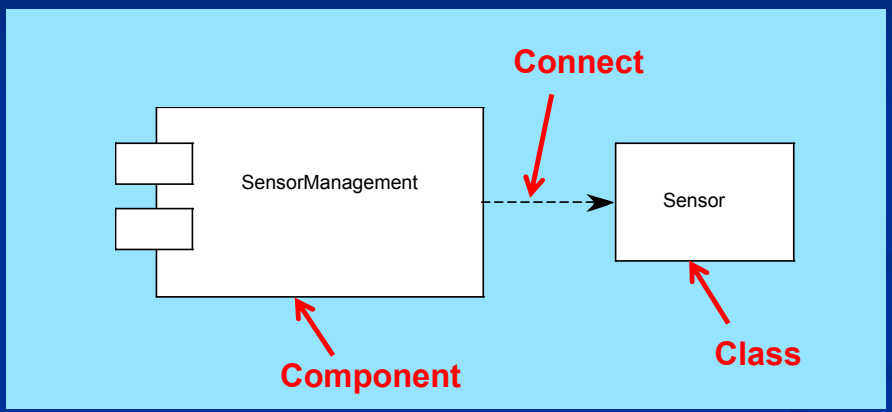


Figure 9.6 UML interface representation for **ControlPanel**

Component Elements



Deployment Elements

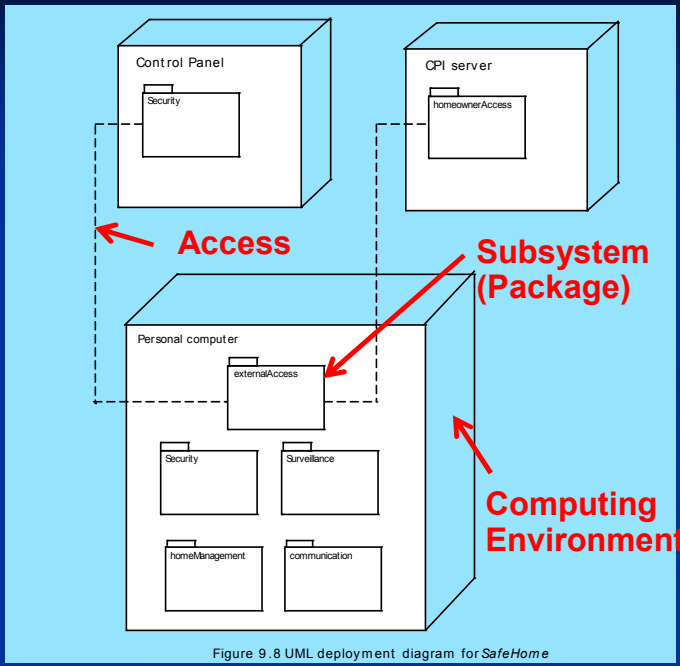


Figure 9.8 UML deployment diagram for *SafeHome*