CSE4006 Software Engineering

07. Design Engineering

기초 블루프린트 단계

Scott Uk-Jin Lee

Department of Computer Science and Engineering
Hanyang University ERICA Campus

1st Semester 2015

lab(se);

Software Design

- Encompasses :
  - **design principles**: establish and override philosophy that guides the designer as the work is performed
  - **design concepts**: must be understood before the mechanics of design practice are applied
  - **design practices**: change continuously as new methods, better analysis, and broader understanding evolve
- Good software design should exhibit :
  - **Firmness**: should not have any bugs that inhibit its function
  - **Commodity**: should be suitable for the purposes for which it was intended 상품성
  - **Delight**: the experience of usage should be pleasurable

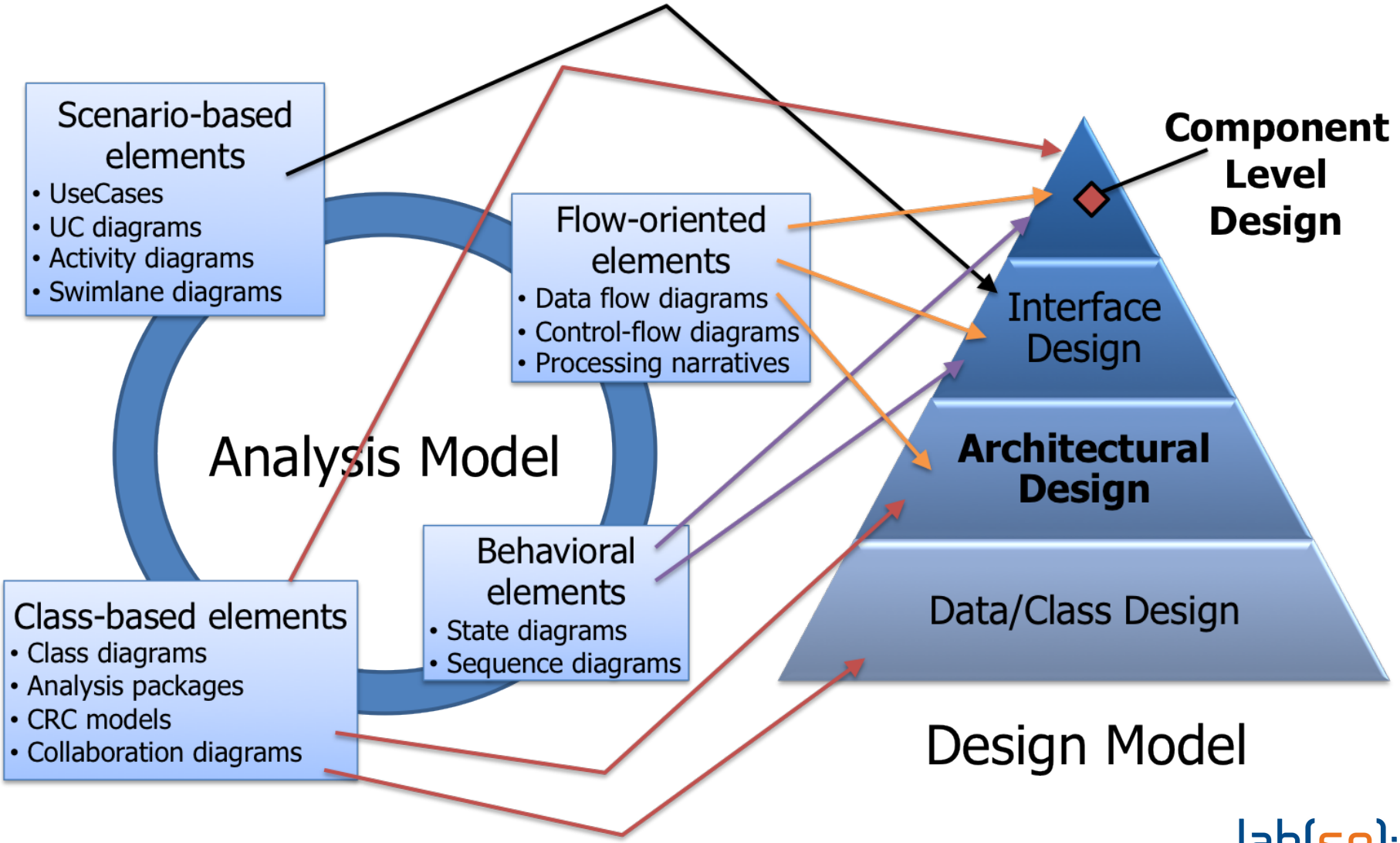
lab(se);

Software Engineering Design

- **Data/Class design**
  - transforms analysis classes into implementation classes and data structures
- **Architectural design**
  - defines relationships among the major software structural elements
- **Interface design**
  - defines how software elements, hardware elements, and end-users communicate
- **Component-level design**
  - transforms structural elements into procedural descriptions of software components

lab(se);

Analysis Model ⇒ Design Model



lab(se);

Design and Quality

- Design must implement all of the **explicit requirements** contained in the analysis model
  - must also accommodate all of the implicit requirements desired by the customer
- Design must be a readable and **understandable** guide
  - for those who generate code, test and subsequently support the software
- Design should provide a complete picture of the software
  - addressing the data, functional, and behavioral domains from an **implementation** perspective

lab(se);

Quality Guidelines

A design should :

1. 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. be **modular**
3. contain distinct representations of data, architecture, interfaces, and components
4. lead to data structures that are
   - appropriate for the classes to be implemented
   - drawn from recognizable data patterns

lab(se);

Quality Guidelines

A design should :

5. lead to components that exhibit independent functional characteristics
6. lead to interfaces that reduce the complexity of connections between components and with the external environment
7. be derived using a **repeatable** method that is driven by information obtained during software requirements analysis
8. be represented effectively for communicating its meaning

lab(se);

Quality Attributes - FURPS

[Gra87] developed by HP

- **F**unctionality
  - assessed by evaluating feature set and capabilities of the program and generality of the functions that are delivered
- **U**sability
  - assessed by considering **human factors**, overall aesthetics
- **R**eliability
  - evaluated by measuring failure rate, accuracy of output, failure recovery ability
- **P**erformance
  - measure by considering processing speed, response time, throughput, efficiency, resource consumption, etc
- **S**upportability
  - **maintainability** (extensibility, adaptability, serviceability)
  - compatibility, ease of configuration, ease of installation, etc

lab(se);

Fundamental Software Design Concepts

- Abstraction
  - data, procedure
- 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

lab(se);

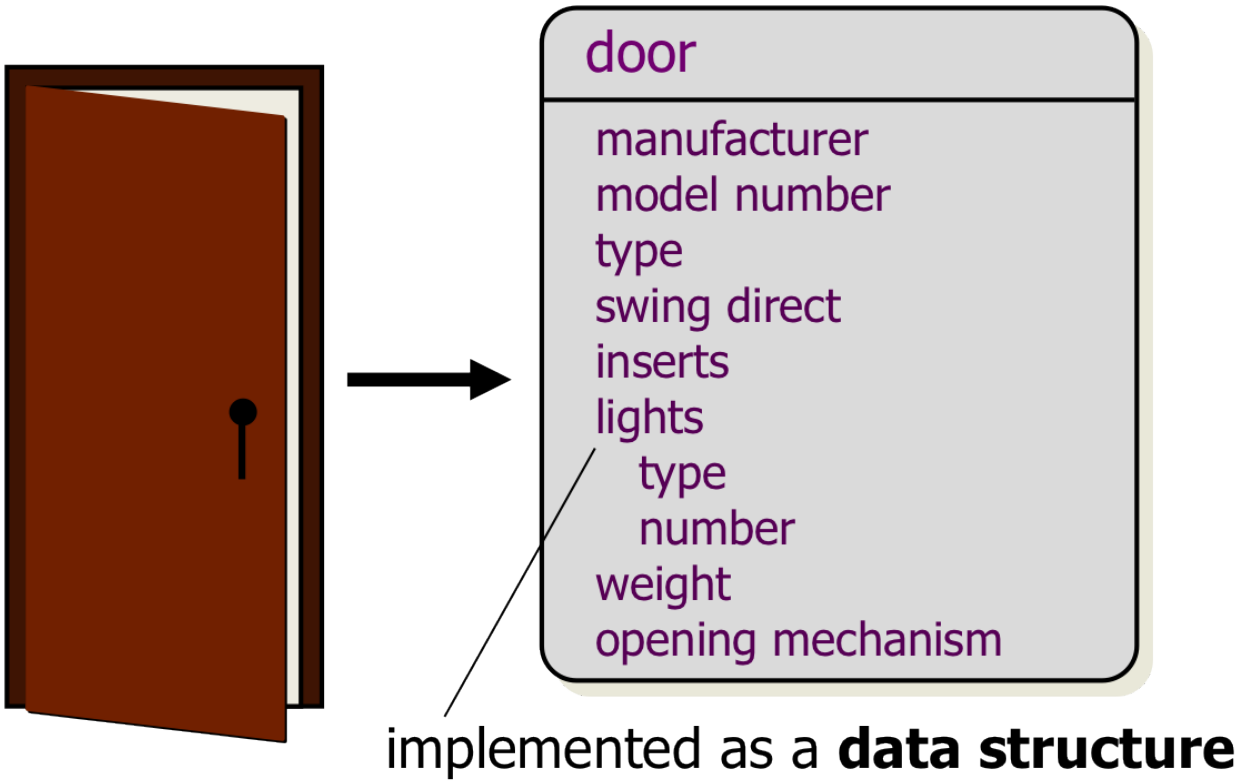
Fundamental Software Design Concepts

- Hiding
  - controlled interfaces
- Functional independence
  - **single-minded function(cohesion) 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

lab(se);

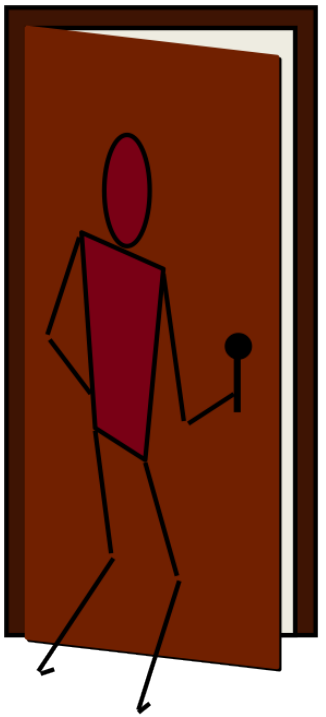
Data Abstraction

named collection of data that describes data objects



Procedural Abstraction

sequence of instructions that have a specific and limited function



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

Software Architecture Design

"The overall structure of the software and the ways the structure provides conceptual integrity for a system." - Shaw et. al.

- Structural properties
  - components of a system
  - manner the components are packaged and interact with one another.
- Extra-functional properties
  - how the design architecture achieves requirements for performance, capacity, reliability, security, adaptability, etc
- Families of related systems
  - repeatable patterns that are commonly encountered in the design of families of similar systems
  - ability to reuse architectural building blocks
- Representations
  - Structural, Framework, Dynamic, Process Functional model
  - Architectural Description Languages (ADLs)

lab(se);

Design Pattern

- The best designers in any field have an 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
- 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)

lab(se);

Design Pattern Template

- Pattern name
  - describes the essence of the pattern in a short/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 responsibilities participants collaborate to carry out their
- Related patterns
  - cross-references related design patterns

lab(se);

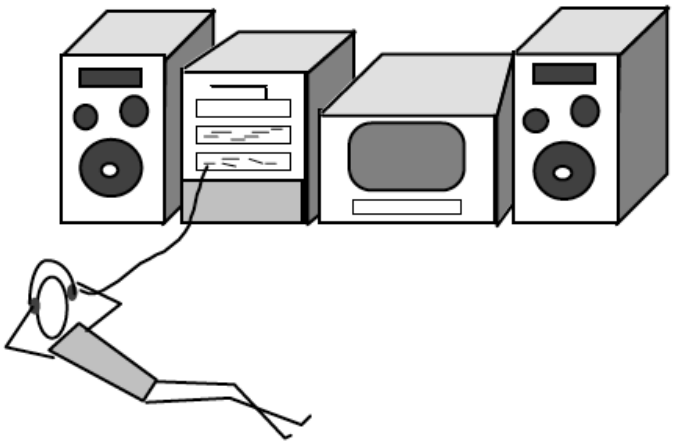
Separation of Concerns

- Any complex problem can be more easily handled if it is subdivided into pieces that can each be solved and/or optimized independently
- A **concern** is a feature or behavior that is specified as part of the requirements model for the software
- By separating concerns into smaller, and therefore more manageable pieces, a problem takes less effort and time to solve

Modular Design

- "Modularity is the single attribute of software allows a program to be intellectually manageable" - Myers, G.
- Monolithic software cannot be easily grasped
  - number of control paths and variables, span of reference, overall complexity make understanding close to impossible

→ In all instances, break the design into many modules, make understanding easier, and reduce software building cost

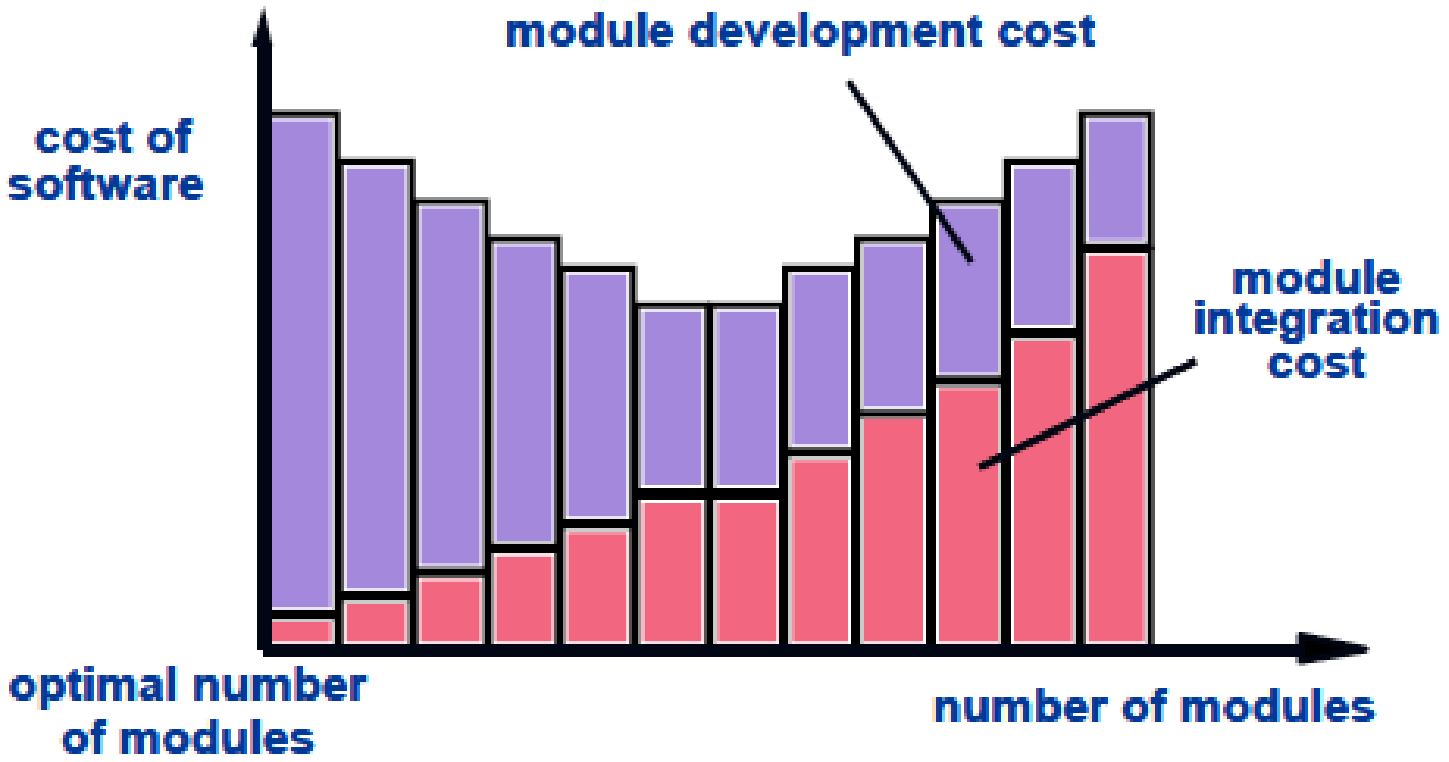


Easier to build, change, fix, ...

lab(se);

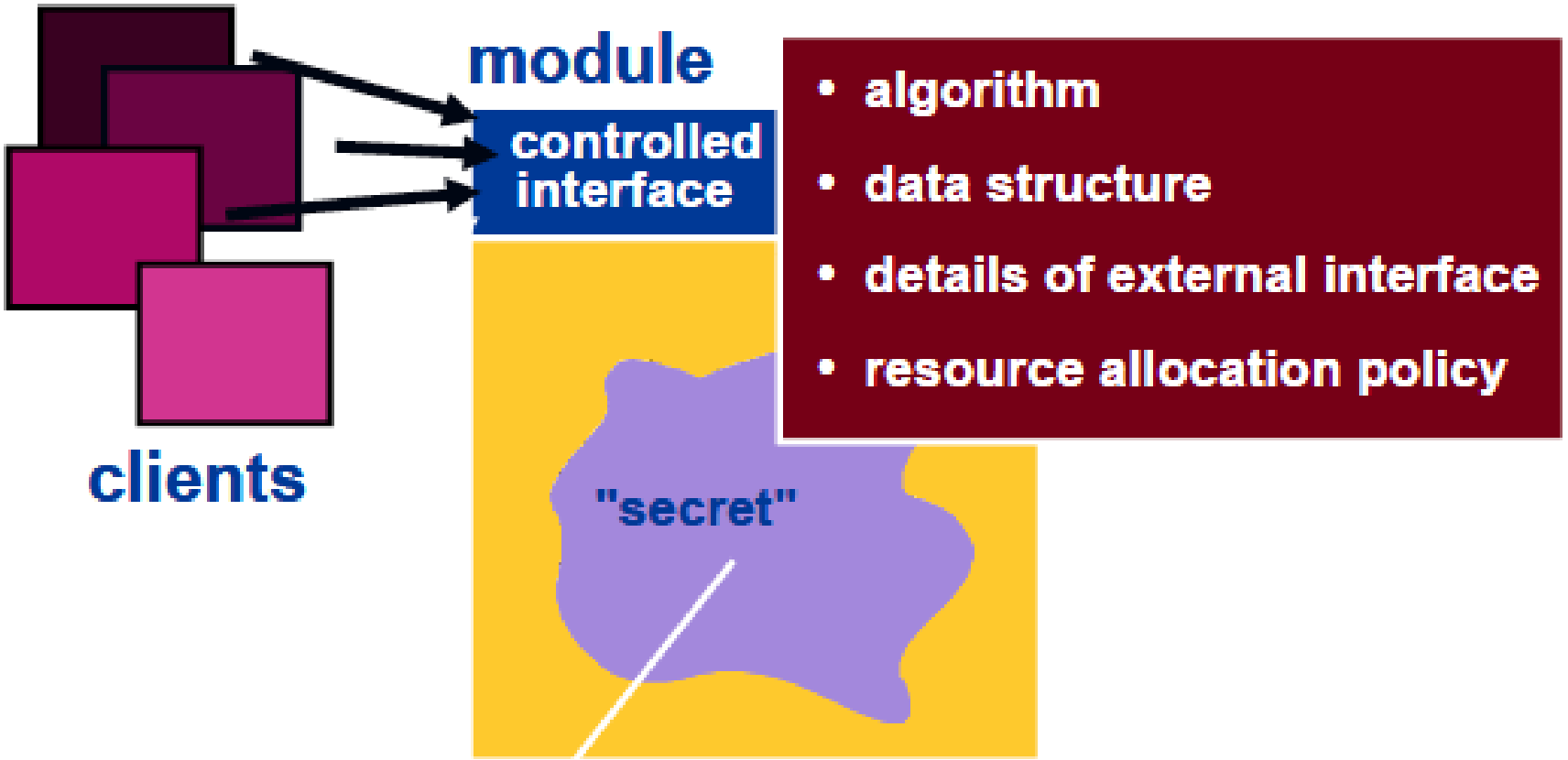
Modularity: Trade-offs

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



lab(se);

Information Hiding



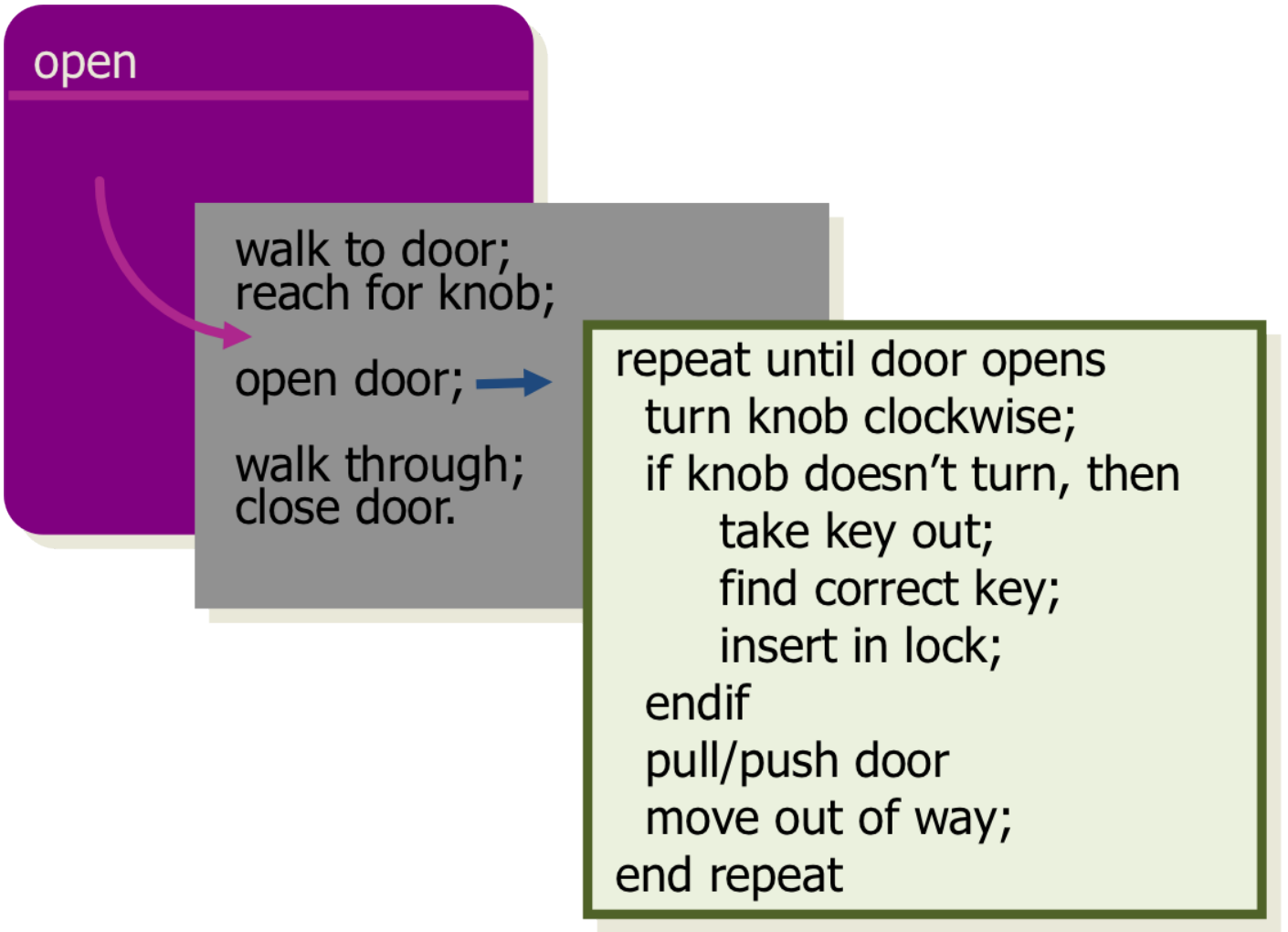
a specific design decision

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

lab(se);

Stepwise Refinement



lab(se);

Functional Independence

- achieved by developing modules with **single-minded** function and an **aversion** to excessive interaction with other modules

- **Cohesion**: indicaties relative functional strength of a module
  - a cohesive module performs a single task, requiring little interaction with other components in other parts of a program (should (ideally) do just one thing)

- **Coupling** : indicates relative interdependence among modules
  - Coupling depends on the interface complexity between modules, the point at which entry or reference is made to a module, and what data pass across the interface.

lab(se);

Functional Independence

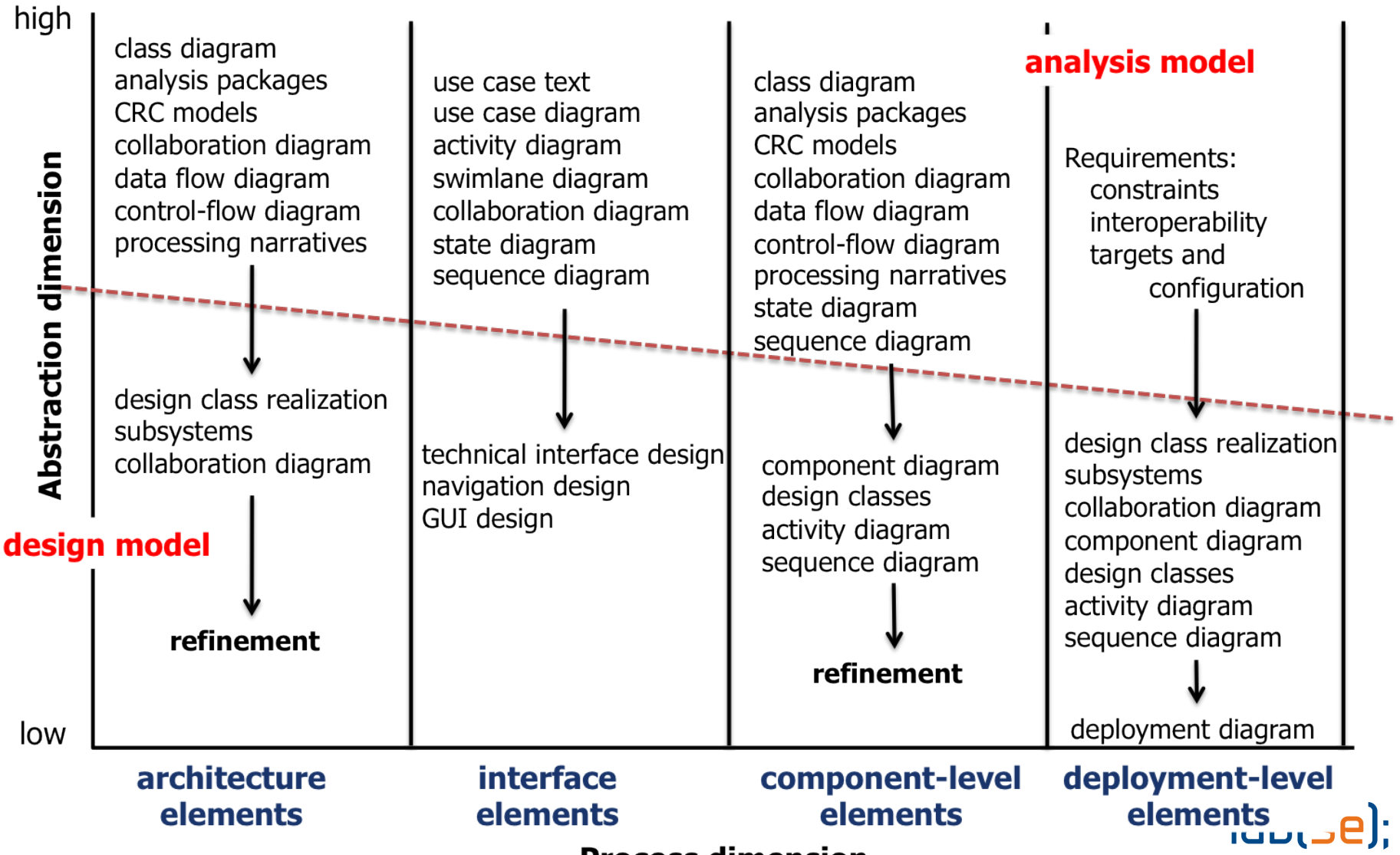
COHESION - the degree to which a module performs one and only one function.

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

Refactoring

- 1999, Martin Fowler:
  - "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, existing design is examined 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

The Design Model



Design Model Elements

- Data elements
  - data model $\rightarrow$ data structures
  - data model $\rightarrow$ database architecture
- Architectural elements
  - application domain
  - analysis classes, their relationships, collaborations and behaviors are transformed into design realizations
  - patterns and "styles" (Chapters 9 & 12)
- 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

lab(se);

Data Elements

- high level model depicting user's view of the data or information
- design of data structures and operators is essential to creation of high-quality applications
- translation of data model into database is critical to achieving system business objectives
- reorganizing databases into a data warehouse enables data mining or knowledge discovery that can impact success of business itself

Architectural Elements

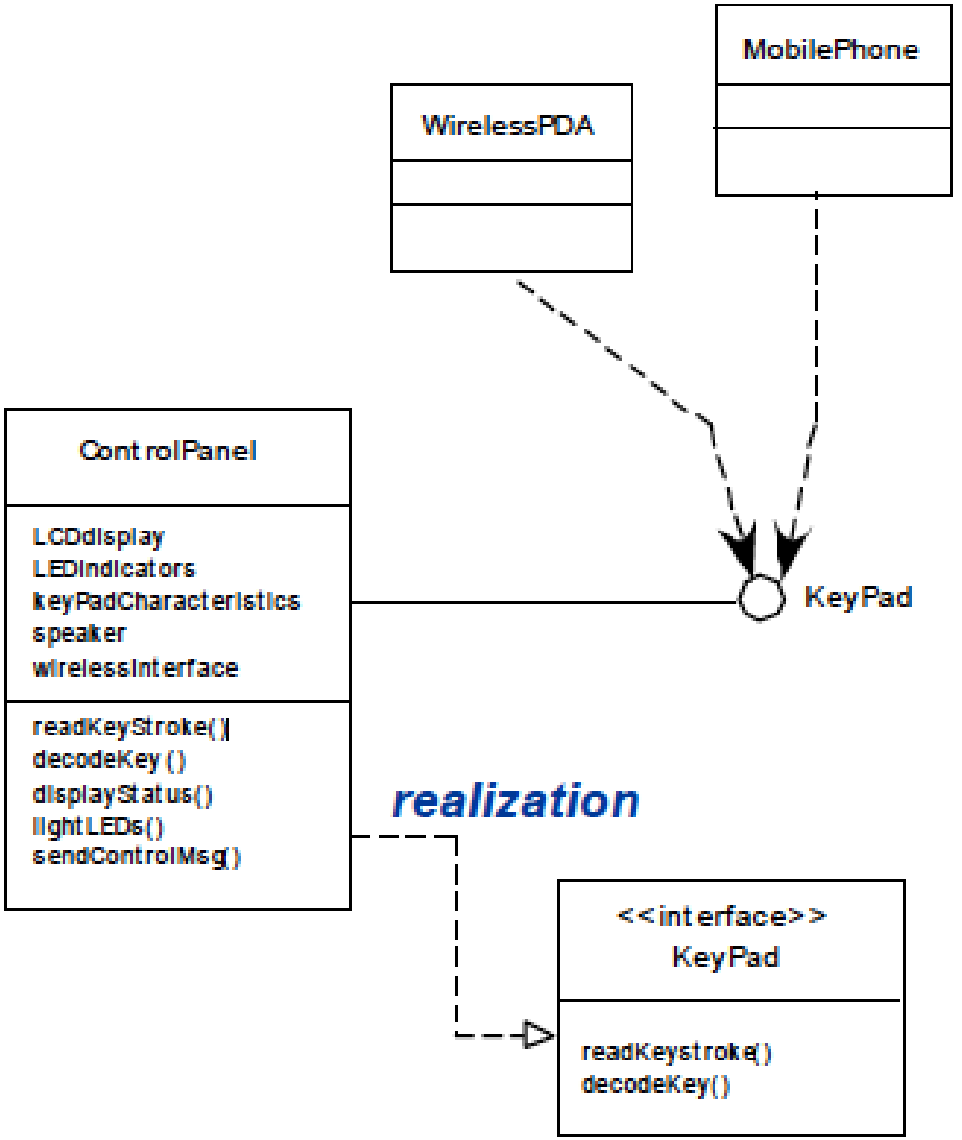
- provides an overall view of the software product
- derived from
  - information about the application domain relevant to software
  - relationships and collaborations among specific analysis model elements
  - availability of architectural patterns (Chapter 16) and styles (Chapter 13)
- usually depicted as a set of interconnected systems that are often derived from the analysis packages within the requirements model

Interface Elements

- interface : set of operations that describes the externally observable behavior of a class and provides access to its public operations
- important elements
  - User Interface (UI)
  - external interfaces to other systems, devices, networks or other producers of consumers of information
  - internal interface between various design components
- modeled using UML communication diagrams

lab(se);

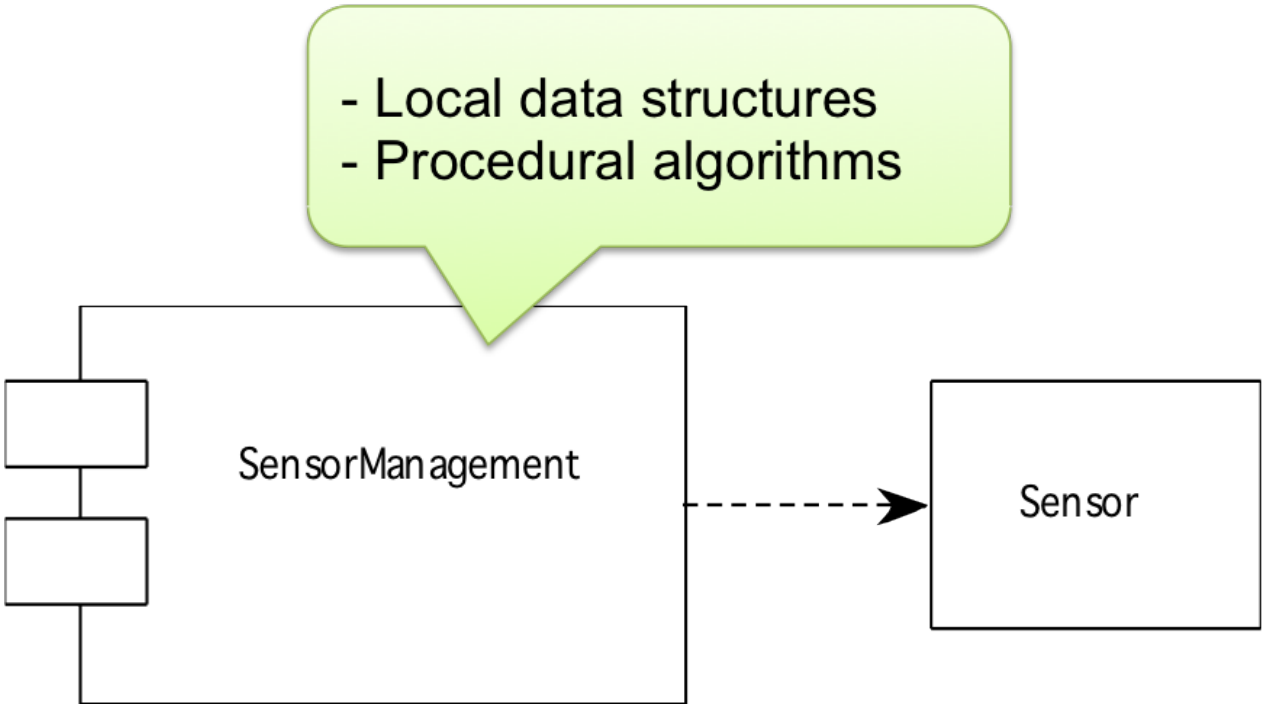
Interface Elements



Component Elements

- describes the internal detail of each software component
- defines
  - data structures for all local data objects
  - algorithmic detail for all component processing functions
  - interface that allows access to all component operations
- modeled using UML component diagrams, UML activity diagrams, pseudocode (PDL), and sometimes flowcharts

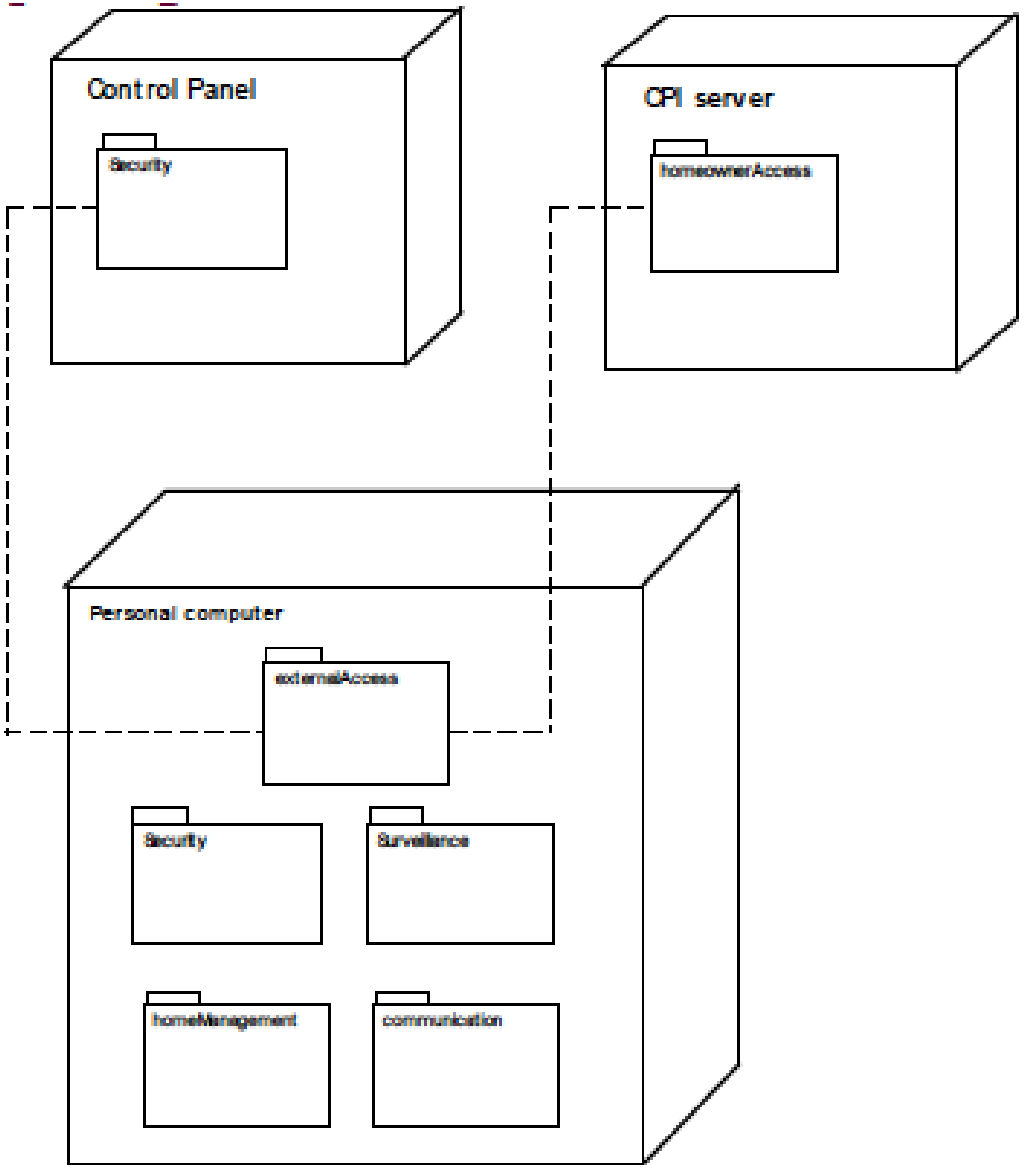
Component Elements



Deployment Elements

- indicate how software functionality and subsystems will be allocated within the physical computing environment
- modeled using UML deployment diagrams
- descriptor from deployment diagrams show the computing environment but does not indicate configuration details
- instance form deployment diagrams identifying specific named hardware configurations are developed during the latter stages of design

Deployment Elements



Design Principles

- The design process should not suffer from 'tunnel vision'
- The design should be **traceable** to the analysis model
- The design should exhibit **uniformity** and integration
- The design should be structured to accommodate **change**
- The design should be structured to degrade gently, even when aberrant data, events are encountered
- **Design is not coding, coding is not design**
- The design should be assessed for quality as it is being created, not after the fact
- The design should be reviewed to minimize conceptual (semantic) errors.

lab(se);

Object-Oriented Design Concepts

- Design classes
  - Entity classes
  - Boundary classes
  - Controller classes
- Inheritance
  - all responsibilities of a superclass is immediately inherited by all subclasses
- Messages
  - stimulate some behavior to occur in the receiving object
- Polymorphism
  - a characteristic that greatly reduces the effort required to extend the design

lab(se);

Object-Oriented Design Concepts

- Design options :
  - The class can be designed and built from scratch. That is, inheritance is not used
  - The class hierarchy can be searched to determine if a class higher in the hierarchy (a superclass)contains most of the required attributes and operations. The new class inherits from the superclass and additions may then be added, as required
  - The class hierarchy can be restructured so that the required attributes and operations can be inherited by the new class
  - Characteristics of an existing class can be overridden and different versions of attributes or operations are implemented for the new class

Polymorphism

Conventional approach ...

```
case of graphtype:
        if graphtype = linegraph then DrawLineGraph (data);
        if graphtype = piechart then DrawPieChart (data);
        if graphtype = histogram then DrawHisto (data);
        if graphtype = kiviat then DrawKiviat (data);
end case;
```

graphtype draw

All of the graphs become subclasses of a general class called graph. Using a concept called overloading, each subclass defines an operation called draw. An object can send a draw message to any one of the objects instantiated from any one of the subclasses. The object receiving the message will invoke its own draw operation to create the appropriate graph.

lab(se);

Design Classes

- Analysis classes are refined during design to become entity classes
- Boundary classes are developed during design to create the interface (e.g., interactive screen or printed reports) that the user sees and interacts with as the software is used
  - Boundary classes are designed with the responsibility of managing the way entity objects are represented to users
- Controller classes are designed to manage
  - the creation or update of entity objects
  - the instantiation of boundary objects as they obtain information from entity objects
  - complex communication between sets of objects
  - validation of data communicated between objects or between the user and the application

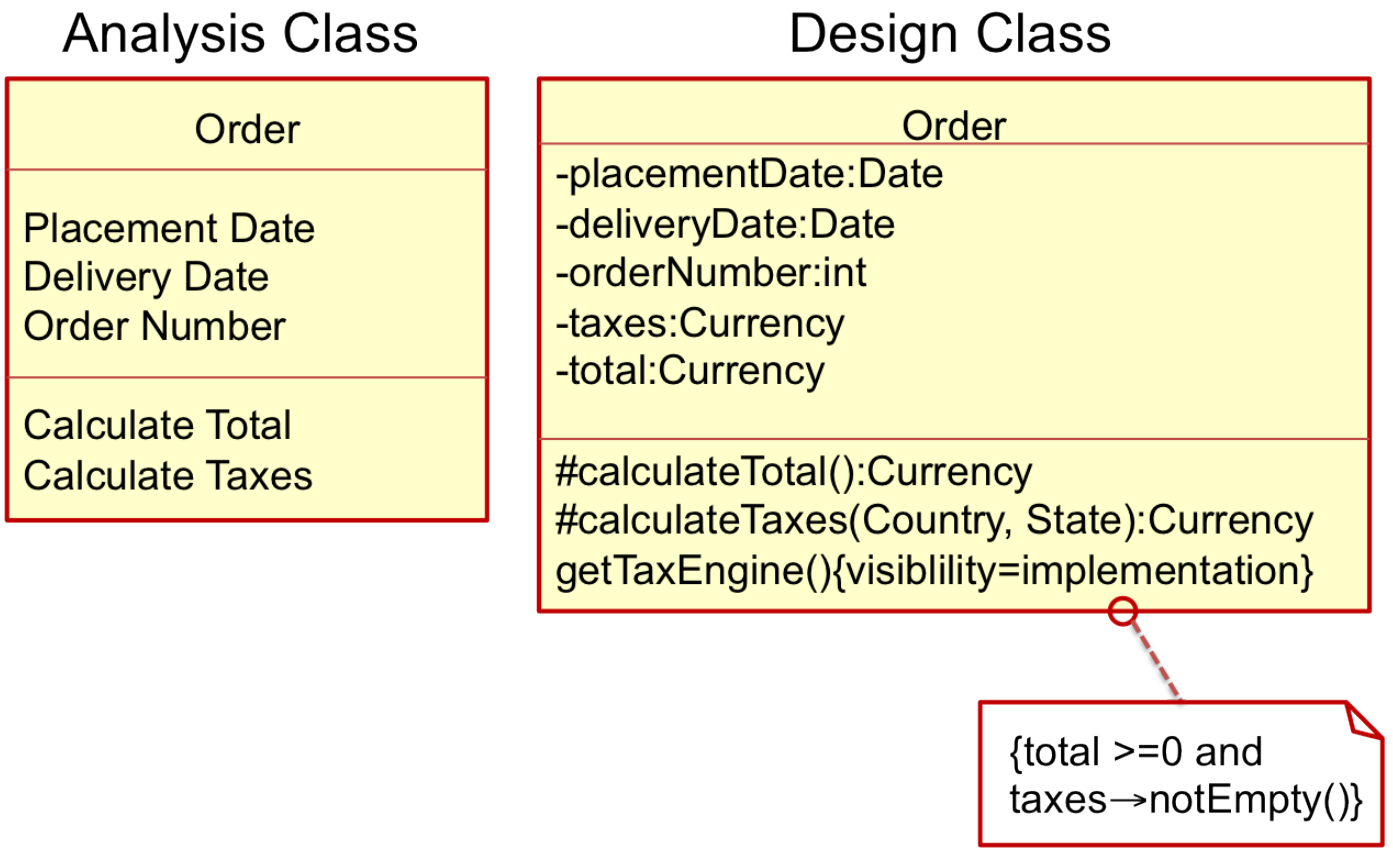
lab(se);

Design Classes Characteristics

- **Complete and Sufficient**
  - includes all necessary attributes and methods
  - contains only those methods needed to achieve class intent
- **Primitiveness**
  - each class method focuses on providing one service
- **High cohesion**
  - small, focused, single-minded classes
- **Low coupling**
  - class collaboration kept to minimum

lab(se);

Analysis and Design Versions of a Class



lab(se);