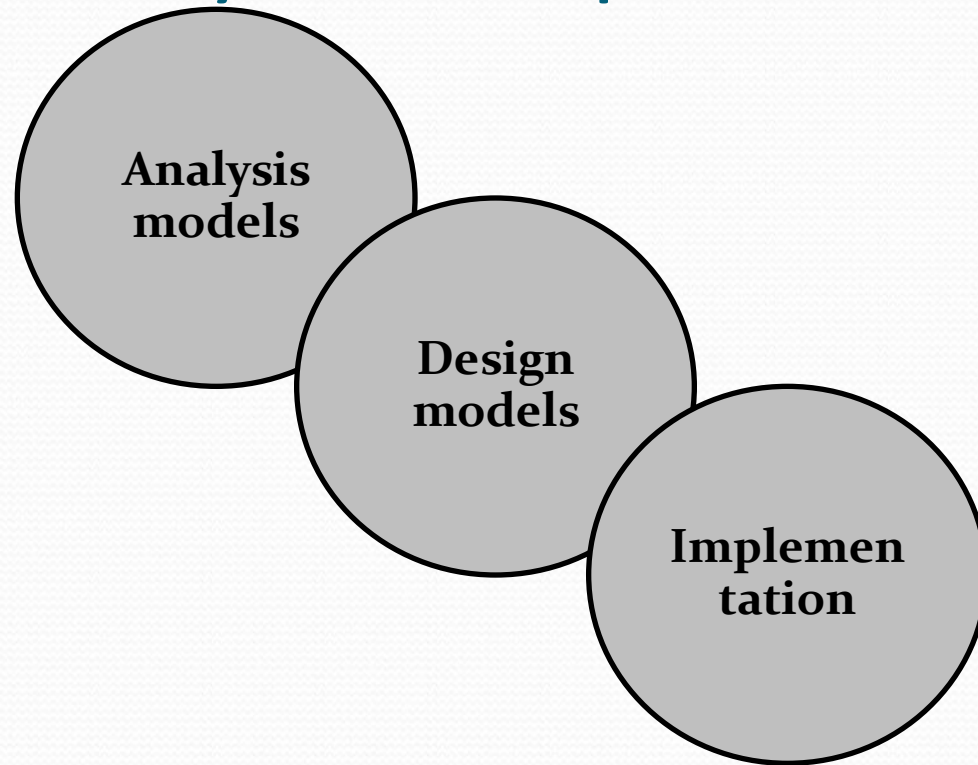


Design Models

Based on

- Chapter 13 – Design Concepts and Principles from the book Software Engineering: A Practitioner's Approach, R.S.Press Man
- I Sommerville Software Engineering
- R. Monahan' materials and Lethbridge/Laganière Architecting and Designing

Design Models form a Bridge between Analysis & Implementation

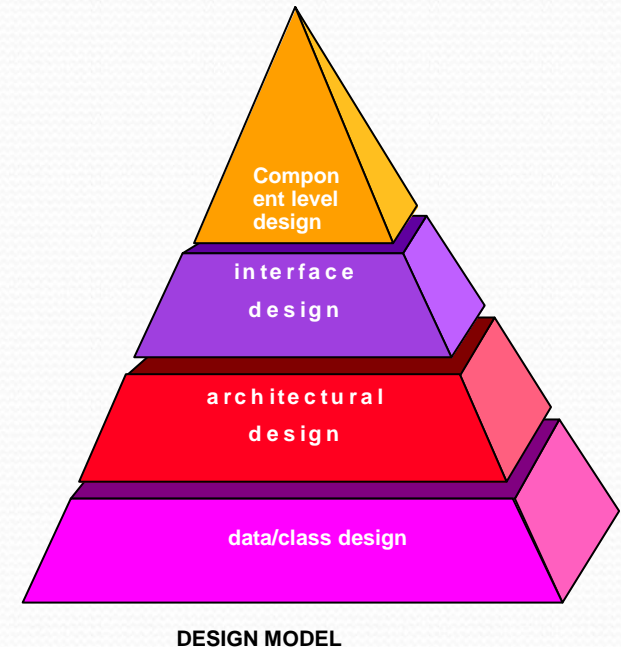


Design and Quality

- 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.

Design models

- Software requirements, manifested by the data, functional, and behavioural models, feed the design task
- The design task produces a data design, an architectural design, an interface design, and a component design



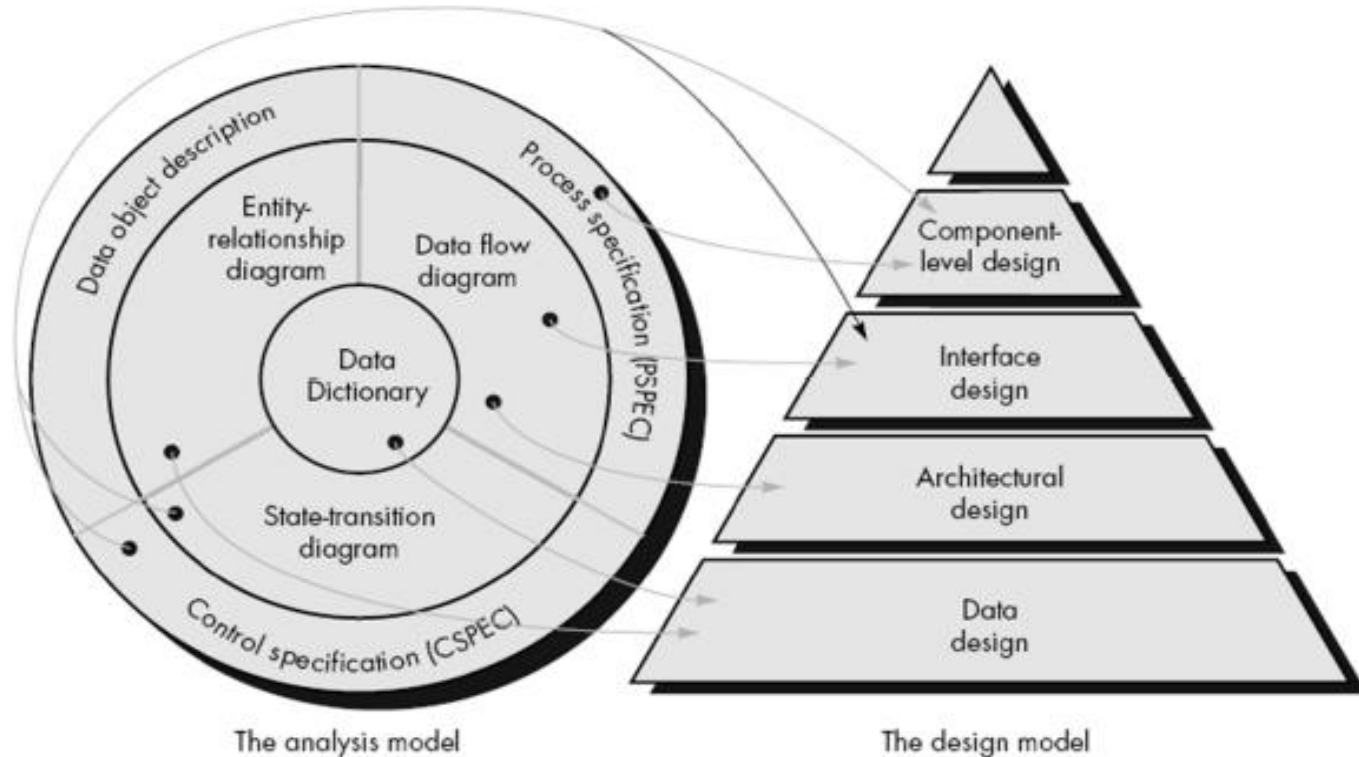
Design models

- The data design transforms the information domain model into the data structures that will be required to implement the software
- The architectural design defines the relationship between major structural elements of the software, the architectural styles and design patterns that can be used to achieve the requirements defined for the system, and the constraints that affect the way in which the architectural design can be implemented.

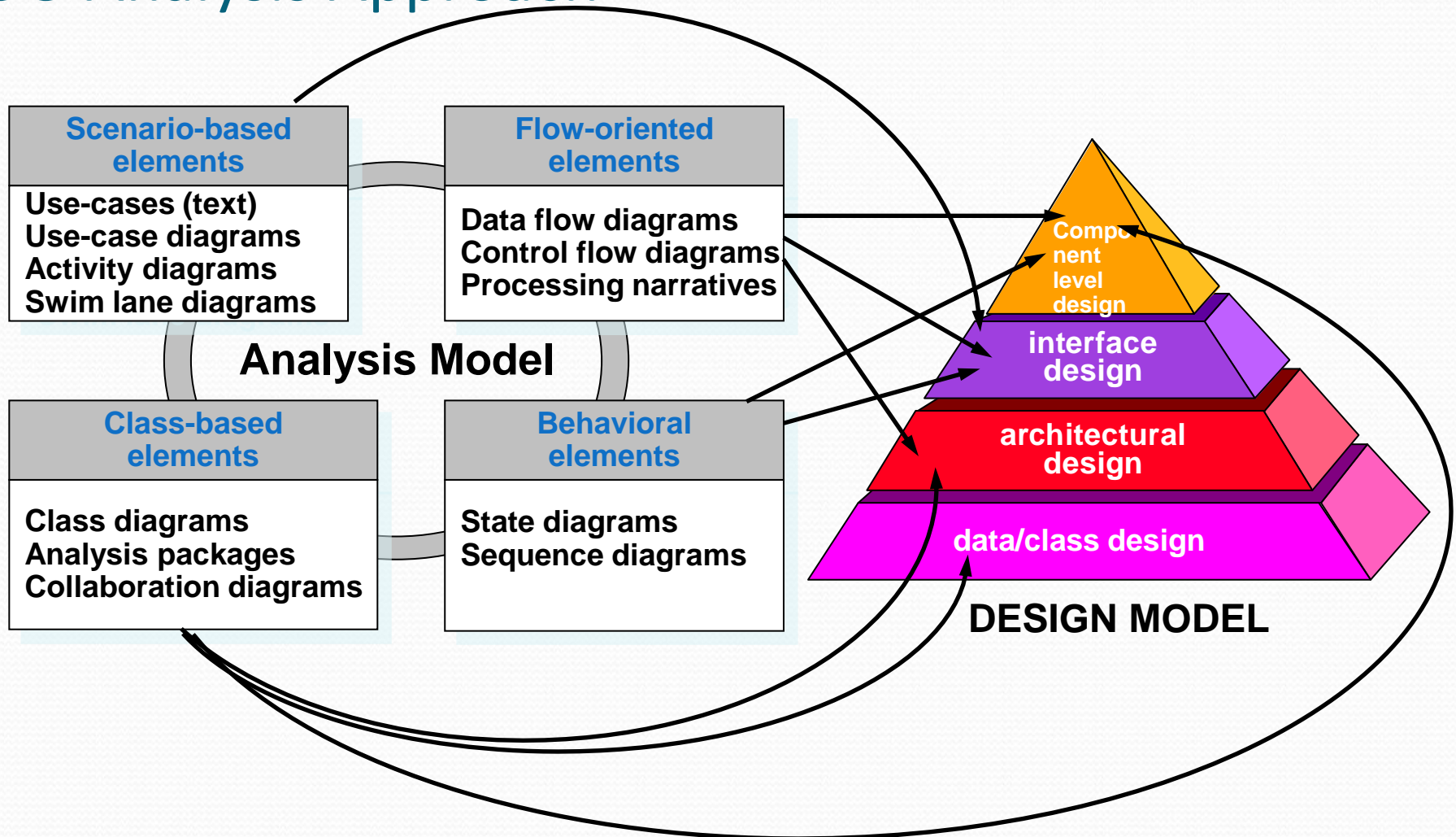
Design models

- The interface design describes how the software communicates with systems that interpolate with it, and with humans who use it.
- The component-level design transforms structural elements of the software architecture into a procedural description of software components (detail design of each component).

Design within the Context of SE-Structured Analysis Approach



Design within the Context of SE – OO Analysis Approach



Fundamental Concepts in System Design

- **abstraction**—data, procedure, control
- **refinement**—elaboration of detail for all abstractions
- **modularity**—compartmentalization of data and function
- **software architecture**—the overall structure of the software
- **information hiding**—controlled interfaces
- **functional independence**—single-minded function and low coupling
- **refactoring**—a reorganization technique that simplifies the design

Abstraction

- Different levels of abstraction:
 - Highest level: a solution is stated in broad terms using the language of the problem environment
 - Lower level: Problem-oriented terminology is coupled with implementation-oriented terminology in an effort to state a solution
 - Lowest abstraction level: the solution is stated in a manner that can be directly implemented
- Each step in the software process is a refinement in the level of abstraction of the software solution.

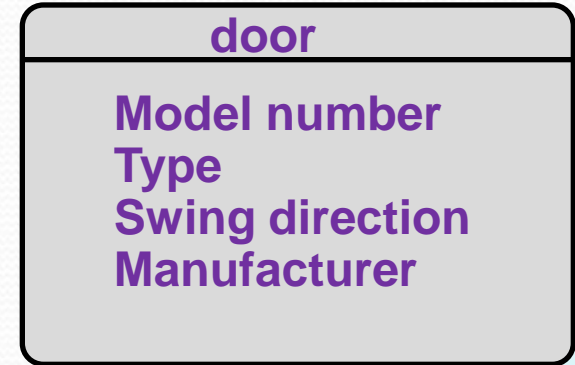
Abstraction

- Abstraction — data, procedure

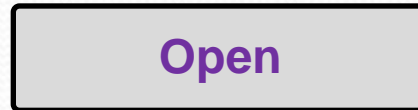
data abstraction



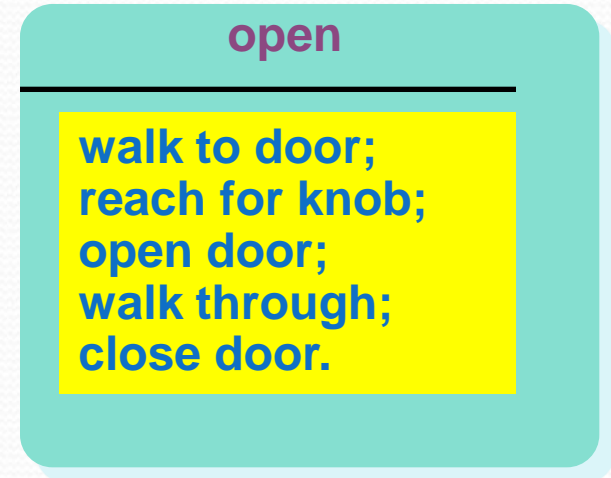
A *data abstraction* is a named collection of data that describes a data object



procedural abstraction

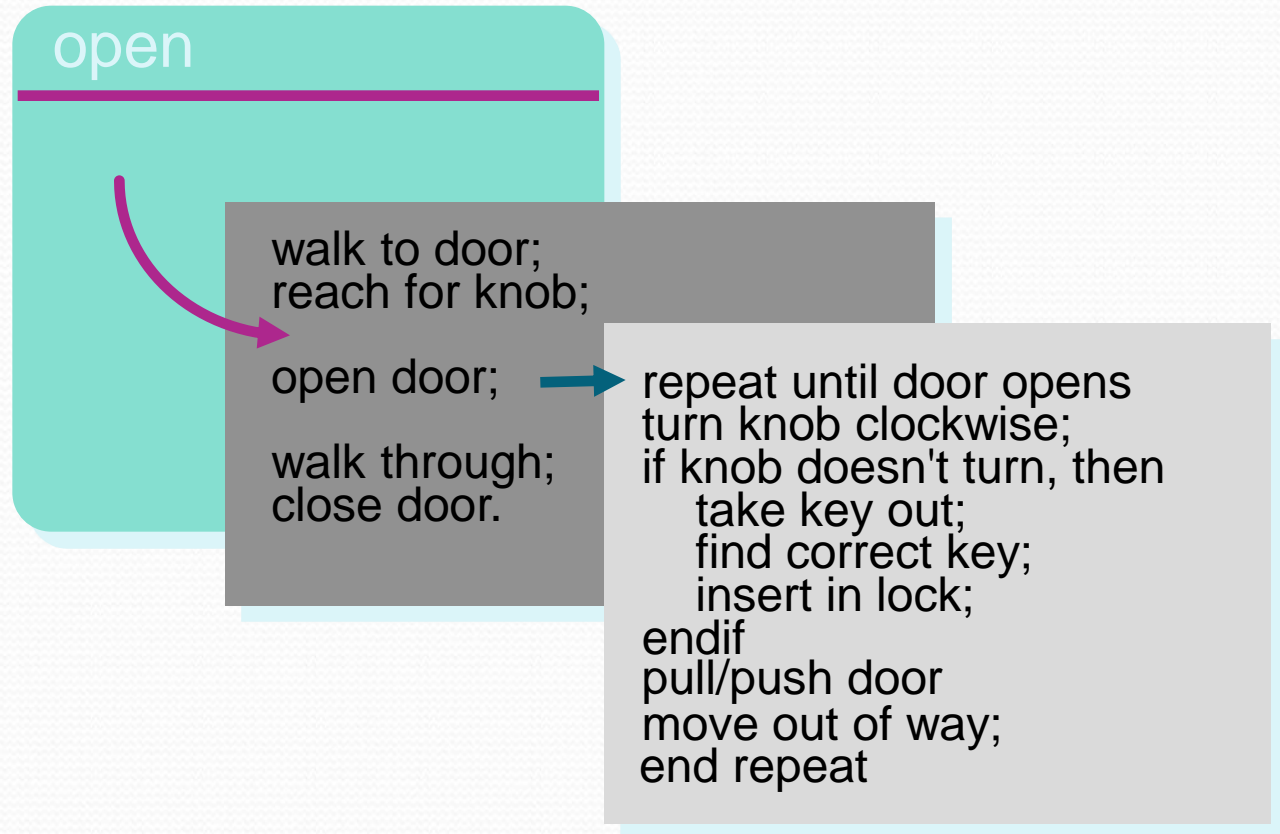


A *procedural abstraction* refers to a sequence of instructions that have a specific and limited function



Refinement

- Refinement — elaboration of detail for all abstraction, reveals low-level details. Top-down design strategy



Modularity

- Software architecture embodies modularity (modules, components) that are integrated to satisfy problem requirements
- “Modularity is the single attribute of software that allows a program to be intellectually manageable”

Component

- Any piece of software or hardware that has a clear role.
 - A component can be isolated, allowing you to replace it with a different component that has equivalent functionality.
 - Many components are designed to be reusable.
 - Conversely, others perform special-purpose functions.

Module

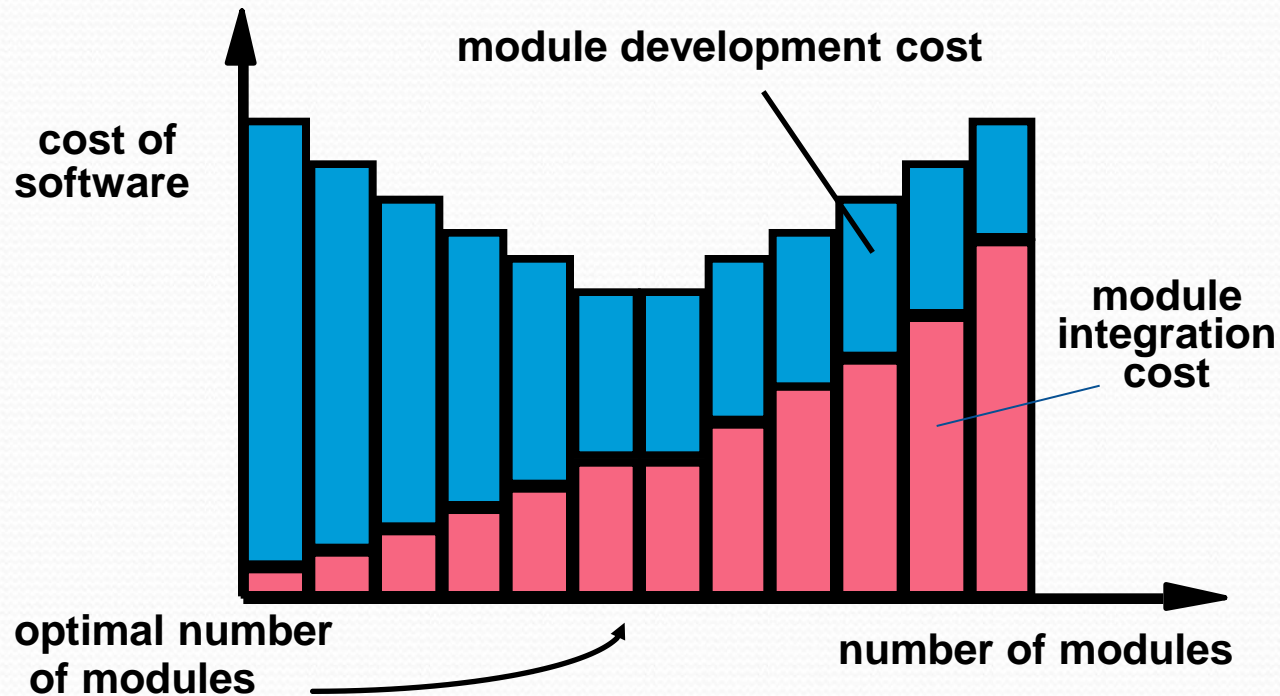
- A component that is defined at the programming language level
 - For example, methods, classes and packages are modules in Java.

Modularity and cost

- Given the same set of requirements, more modules means smaller individual size. The effort (cost) to develop an individual software module does decrease as the total number of modules increases
- As the number of modules grows, the effort (cost) associated with integrating the modules also grows

Modularity: Trade-offs

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



Modularity - Principles

- No answer of the right number, but the principles are
 - **Modular composability:** an effective design should enable to assemble existing components (reuse) into a new system (do not reinvent the wheel)
 - **Modular understandability:** if a module can be understood as a standalone unit (without reference to other modules), it will be easier to build and easier to change.

Modularity - Principles

- **Modular continuity:** If small changes to the system requirements result in changes to individual modules, rather than system wide changes, the impact of change-induced side effects will be minimized.
- **Modular protection:** If an abnormal condition occurs within a module and its effects are constrained within that module, the impact of error-induced side effects will be minimized

Architecture

- Architecture describes the overall structure of the software and the ways in which that structure provides conceptual integrity for a system
- Can be represented by one or more of following models
 - **Structural models:** represent architecture as an organized collection of program *components*
 - **Functional models:** represent the *functional hierarchy* of the system

Information Hiding

- Modules should be specified and designed so that information (algorithms and data) contained within a module is inaccessible to other modules that have no need for such information
- Hiding 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

Information Hiding

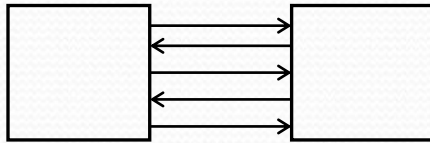
- It is about controlled interfaces
 - limits the global impact of local design decisions
 - emphasizes communication through controlled interfaces
 - leads to encapsulation—an attribute of high quality design
 - results in higher quality software

Functional Independence

- Achieved by developing modules with “single-minded” function and an “aversion” to excessive interaction with other modules
- Based on 2 criteria
 - Cohesion
 - The degree to which a module performs one and only one function (relative strength of the module)
 - Coupling
 - The degree to which a module is connected to other modules in the system (interdependence among modules)

Coupling

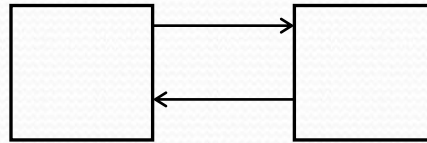
Great deal of dependence



☹ Highly coupled



Independent



Loosely coupled



Uncoupled ☺

Goal: as *loose* as possible = as *independent* as possible

Coupling and Cohesion-

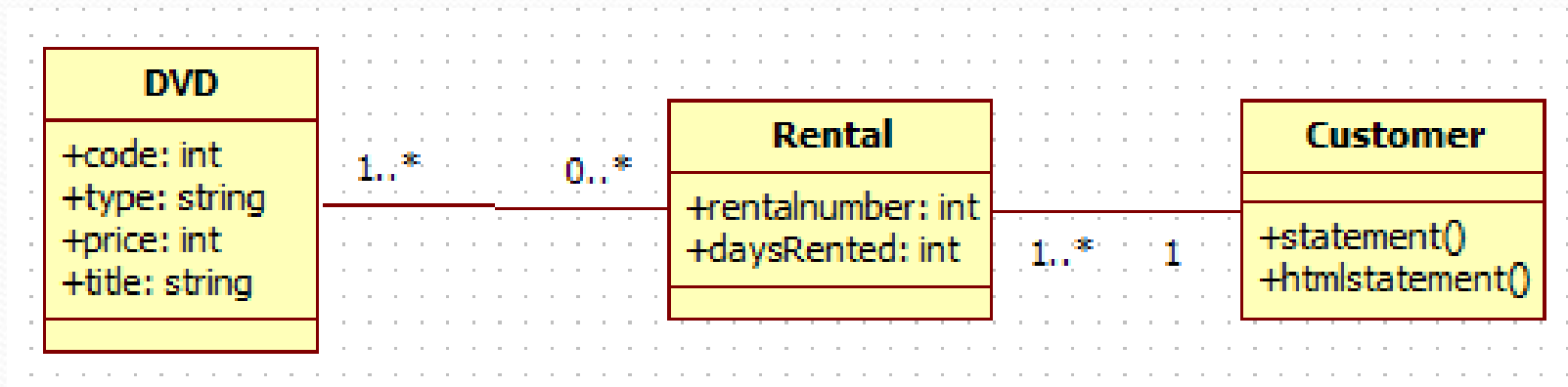
Example

- What is the degree of cohesion of the following modules ?
 - i. Module “InventorySearchByID” searches the records in inventory to see if any match the specified range of ID numbers. A data structure is returned containing any matching records.
 - ii. Module “ProcessPurchase” removes the purchased product from inventory, prints a receipt for the customer and updates the log.
 - iii. Module “FindSet” processes the user's request, determines the set of items from inventory that match the request, and formats the items into a list that can be shown to the customer

Refactoring

- A reorganization technique that simplifies the design of a component without changing its function and behavior.
- 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.

Refactoring example



- We need to produce a statement/report of rentals for each customer to be print out and in html format for displaying on the web.
- The statement of each customer should contains all rentals, DVD title and the amount of each rental.

Refactoring – Original design

- Statement()

For each customer

While a Rental found

pay_one_Rental = 0

DVD_list=""

For each DVD of the Rental

Find DVD title

Find DVD price

Add DVD title to DVD_list


pay_one_Rental = pay_one_Rental + price * daysRented

End for

Display Rental number, DVD list, pay_one_Rental

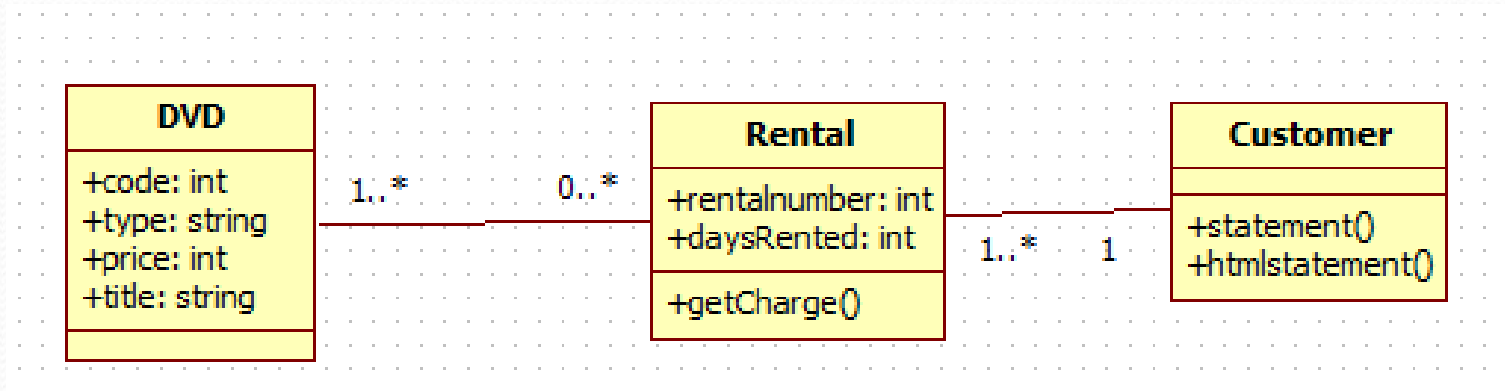
end

This code is about
Calculate charge of each
Rental



After refactoring

- Problem:
 - Code redundancy: rewrite the same code for `htmlStatement`
 - Not correct location : should be in Rental class
- Refactoring



Design after refactoring

- Statement()

For each customer

While a Rental found

//pay_one_Rental = o


//DVD_list=""

Call getCharge(Rental)

Display Rental number, DVD list, amount

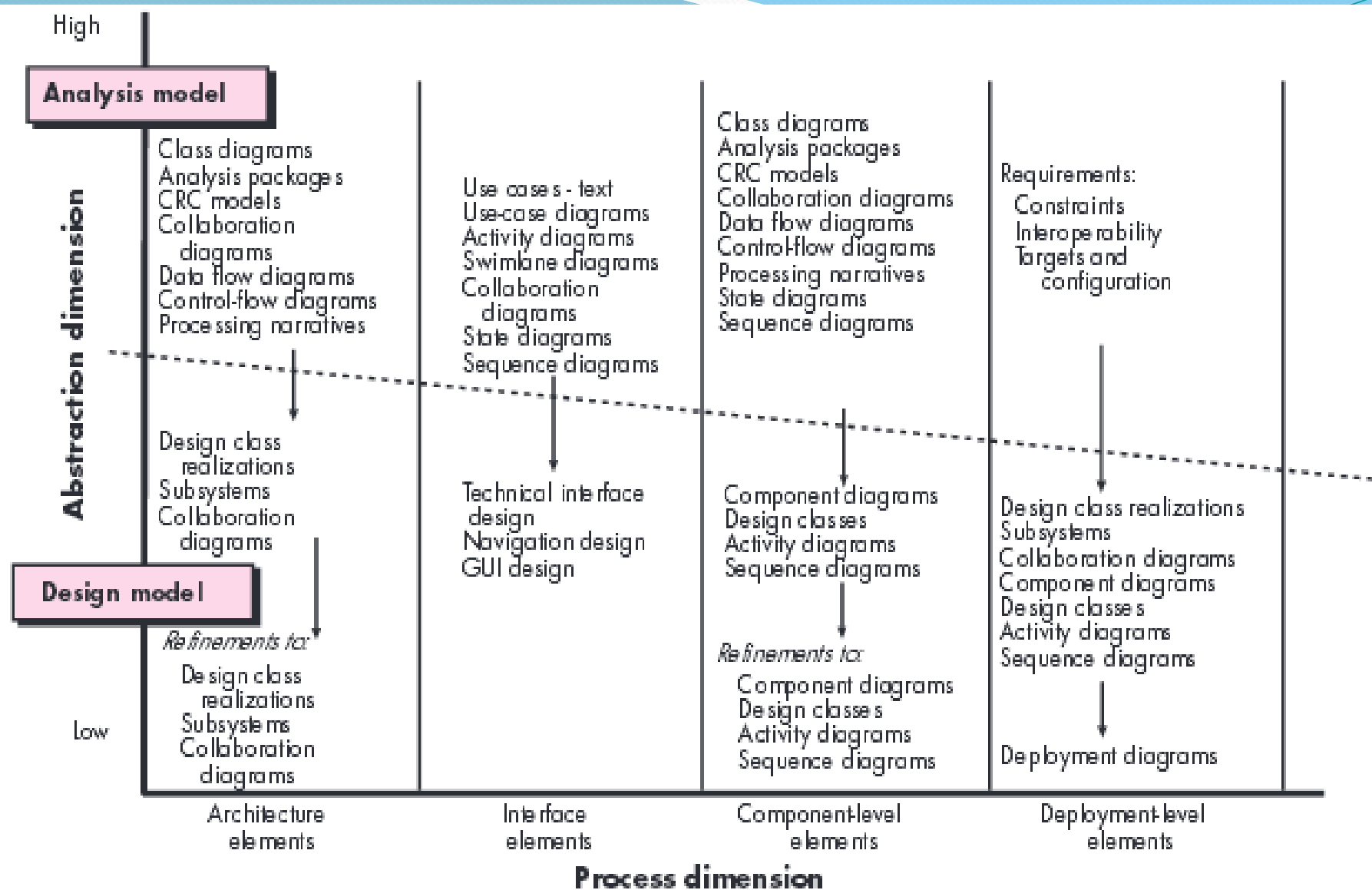
end

This code is about
Calculate charge of each
Rental

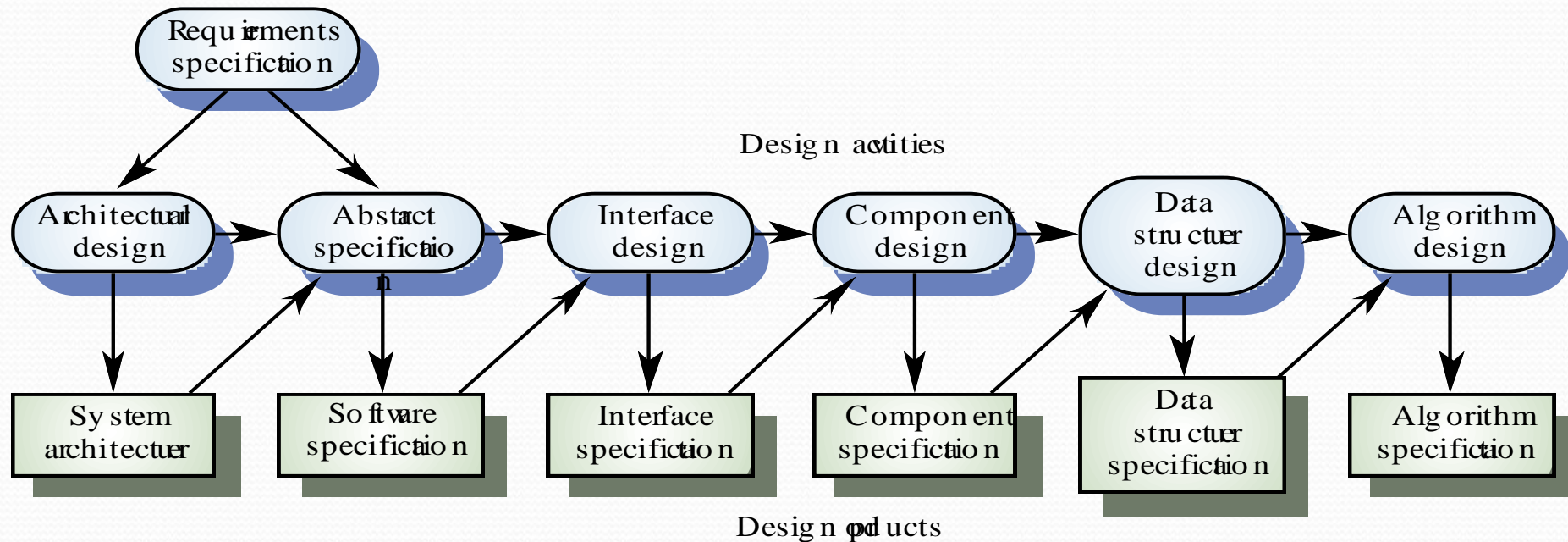


The Design model (image next slide)

- The *process dimension* indicates the evolution of the design model as design tasks are executed as part of the software process. The *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 elements of the design model use many of the same UML diagrams that were used in the analysis model. The difference is that these diagrams are refined and elaborated as part of design; more implementation-specific detail is provided



Phases in the Design Process



Design Phases

- ***Architectural design:*** Identify sub-systems/Component
- ***Abstract specification:*** Specify sub-systems
- ***Interface design:*** Describe sub-system interfaces
- ***Component design:*** Decompose sub-systems into components/Refinement
- ***Data structure design:*** Design data structures to hold problem data.
- ***Algorithm design:*** Design algorithms for problem functions.

Summary

- The intent of software design is to apply a set of principles, concepts, and practices that lead to the development of a high-quality system or product
- The design process moves from a “big picture” view of software to a more narrow view that defines the detail required to implement a system.

Summary

- The process begins by focusing on architecture. Subsystems are defined; communication mechanisms among subsystems are established; components are identified, and a detailed description of each component is developed including data structure, algorithm and graphical user interfaces of each component.