Design Concepts

Lecture # 16, 17
11, 12 March

Rubab Jaffar
rubab.jaffar@nu.edu.pk

Software Engineering
CS-303

TODAY'S OUTLINE

- Steps involved in design
- Phases of Design Process
- Design Model
- Translating analysis model into design model

WHERE DO WE STAND?

- We have learned how to work with our customers to understand what they want from us
- We have developed a requirement document capturing a good understanding of the customer's needs and how the proposed system should behave – the acceptable solution
- What do we have to do now?
- Design is the **HOW?**
  - How the system will be constructed?
  - How the solution will work?
  - How the modules will interact with one another?

"SOFTWARE DESIGN MANIFESTO"

- What is design?
- It's where you stand with a foot in two worlds—the world of technology and the world of people and human purposes—and you try to bring the two together. . . .

- Design is what almost every engineer wants to do. It is the place where creativity rules—where stakeholder requirements, business needs, and technical considerations all come together in the formulation of a product or system.

CHARACTERISTICS OF GOOD DESIGN

- Firmness: A program should not have any bugs that inhibit its function.
- Commodity: A program should be suitable for the purposes for which it was intended.
- Delight: The experience of using the program should be a pleasurable one.
- To accomplish this, you must practice diversification and then convergence.

DIVERSIFICATION& CONVERGENCE

- "Diversification is the acquisition of a alternatives, the raw material of design: components, component solutions, and knowledge.
- Once this diverse set of information is assembled, then choose elements that meet the requirements defined by requirements engineering and the analysis model .
- As this occurs, alternatives are considered and rejected and you converge on "one particular configuration of components, and thus the creation of the final product"

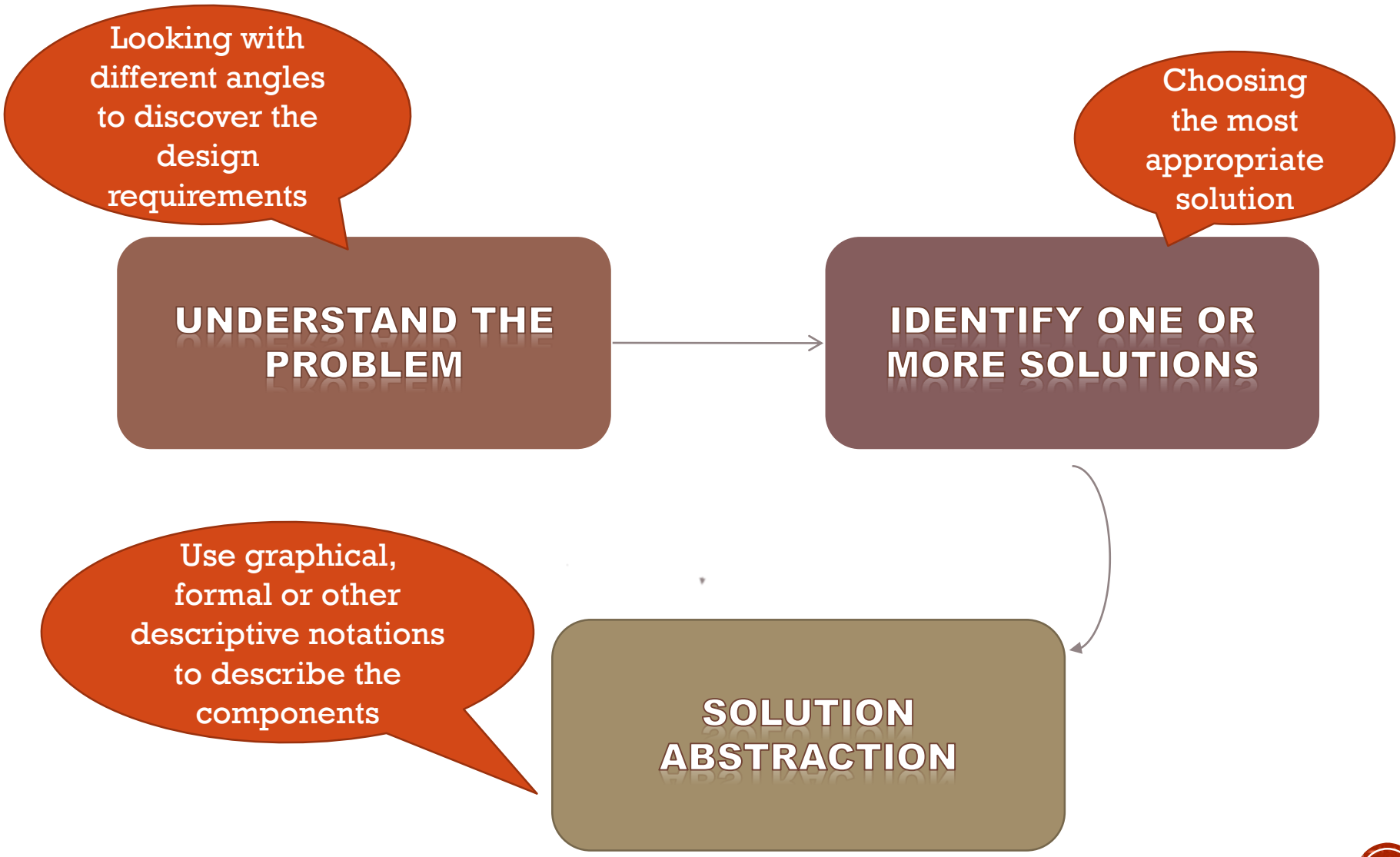
WHY DESIGN IS IMPORTANT?

- Design allows you to model the system or product that is to be built. This model can be assessed for quality and improved before code is generated, tests are conducted, and end users become involved in large numbers. Design is the place where software quality is established.

DESIGN PROCESS

- Software designing is an art.
- It is a creative process,
- Software design is an iterative process through which requirements are translated into a blueprint for constructing the software
  - Design begins at a high level of abstraction that can be directly traced back to the data, functional, and behavioral requirements
  - As design iteration occurs, subsequent refinement leads to design representations at much lower levels of abstraction
- It is the most difficult and challenging task
- Throughout the design process, the quality of the evolving design is assessed with a series of technical reviews

STAGES OF DESIGN



PROCESS OF DESIGN ENGINEERING

Steps followed:

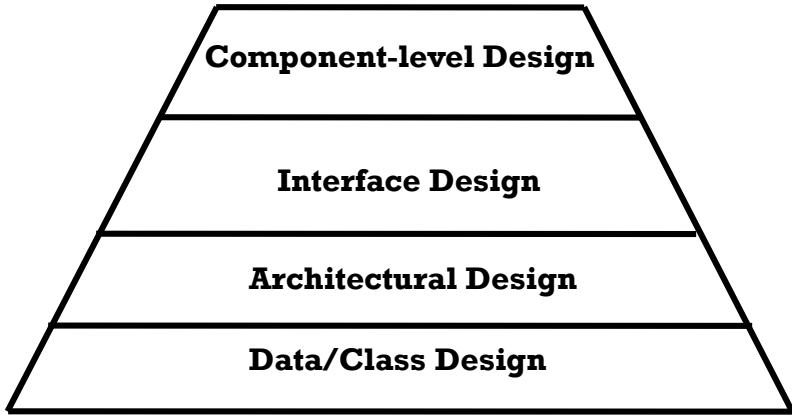
1. Software specifications are transformed into design models
2. Models describe the details of data structures, system architecture, interface and components
3. Design product is then reviewed for quality before moving to the next phase of software development
4. A design model and specification document is produced – containing models, architecture, interfaces and components

STEPS OF SOFTWARE DESIGN

- Design depicts the software in a number of different ways.
  - Architecture of the system or product must be represented.
  - Interfaces that connect the software to end users, to other systems and devices, and to its own constituent components are modeled.
  - Finally, the software components that are used to construct the system are designed.
- Each of these views represents a different design action, but all must conform to a set of basic design concepts that guide software design work.
- A design model that encompasses architectural, interface, component level, and deployment representations is the primary work product that is produced during software design.

DESIGN MODEL

- Design model elements are not always developed in a sequential fashion
  - Preliminary architectural design sets the stage
  - It is followed by interface design and component-level design, which often occur in parallel
- The design model has the following layered elements
  - Data/class design
  - Architectural design
  - Interface design
  - Component-level design
- A fifth element that follows all of the others is deployment-level design

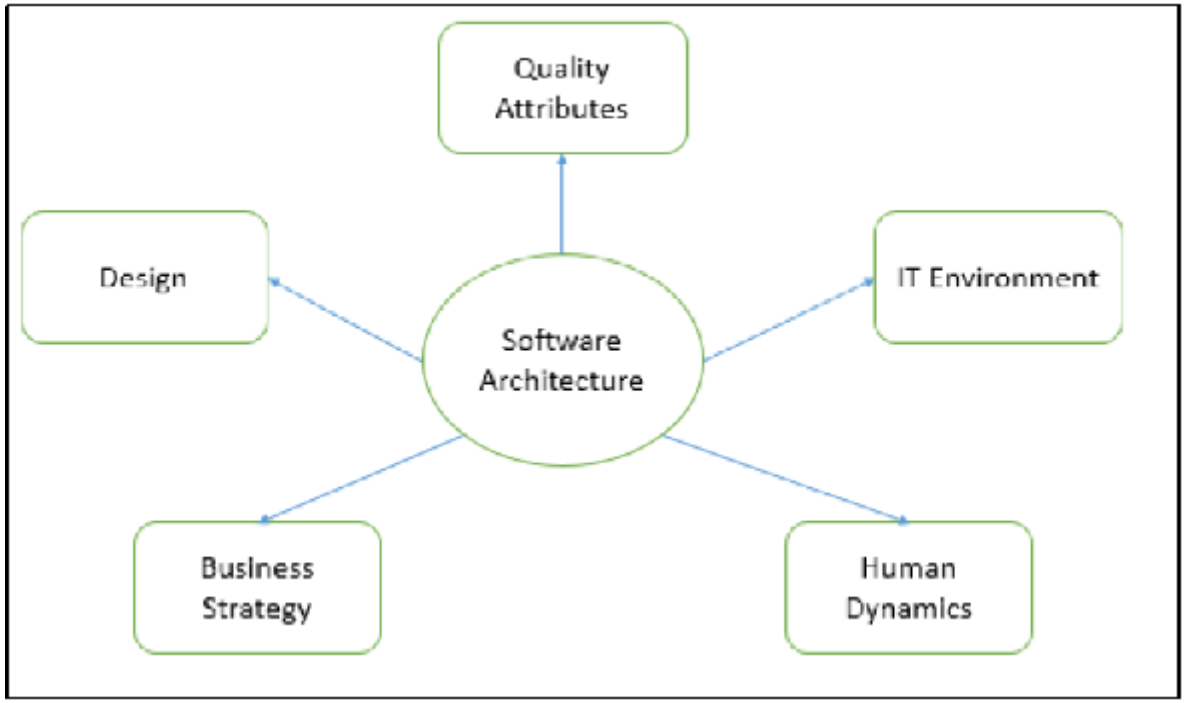


DESIGN ELEMENTS

- Data/class design
  - Creates a model of data and objects that is represented at a high level of abstraction
- Architectural design
  - Depicts the overall layout of the software
- Interface design
  - Tells how information flows into and out of the system and how it is communicated among the components defined as part of the architecture
  - Includes the user interface, external interfaces, and internal interfaces
- Component-level design elements
  - Describes the internal detail of each software component by way of data structure definitions, algorithms, and interface specifications
- Deployment-level design elements
  - Indicates how software functionality and subsystems will be allocated within the physical computing environment that will support the software

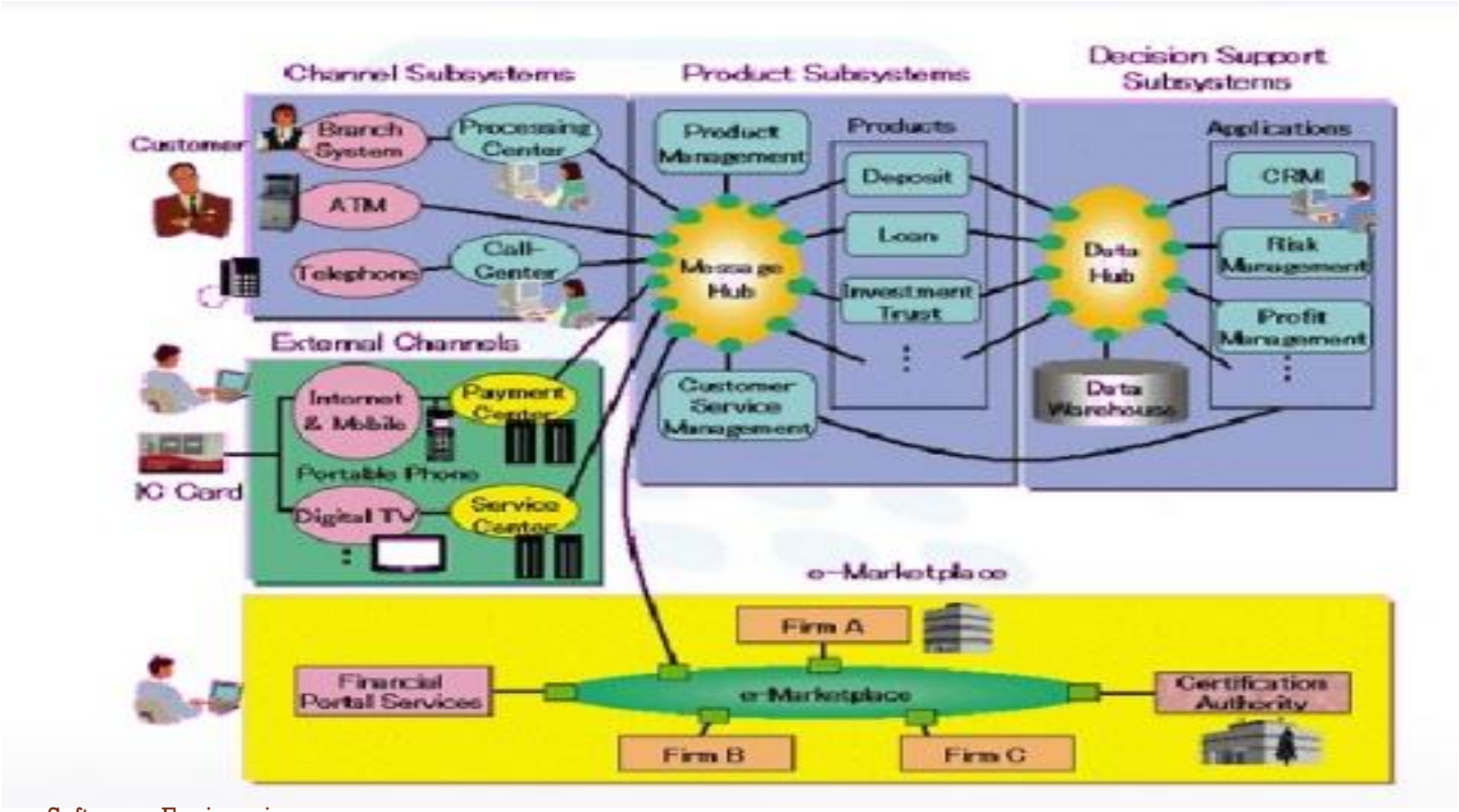
DESIGN PHASES/ACTIVITIES

- *Architectural design* gives you overall view of the software that you are going to



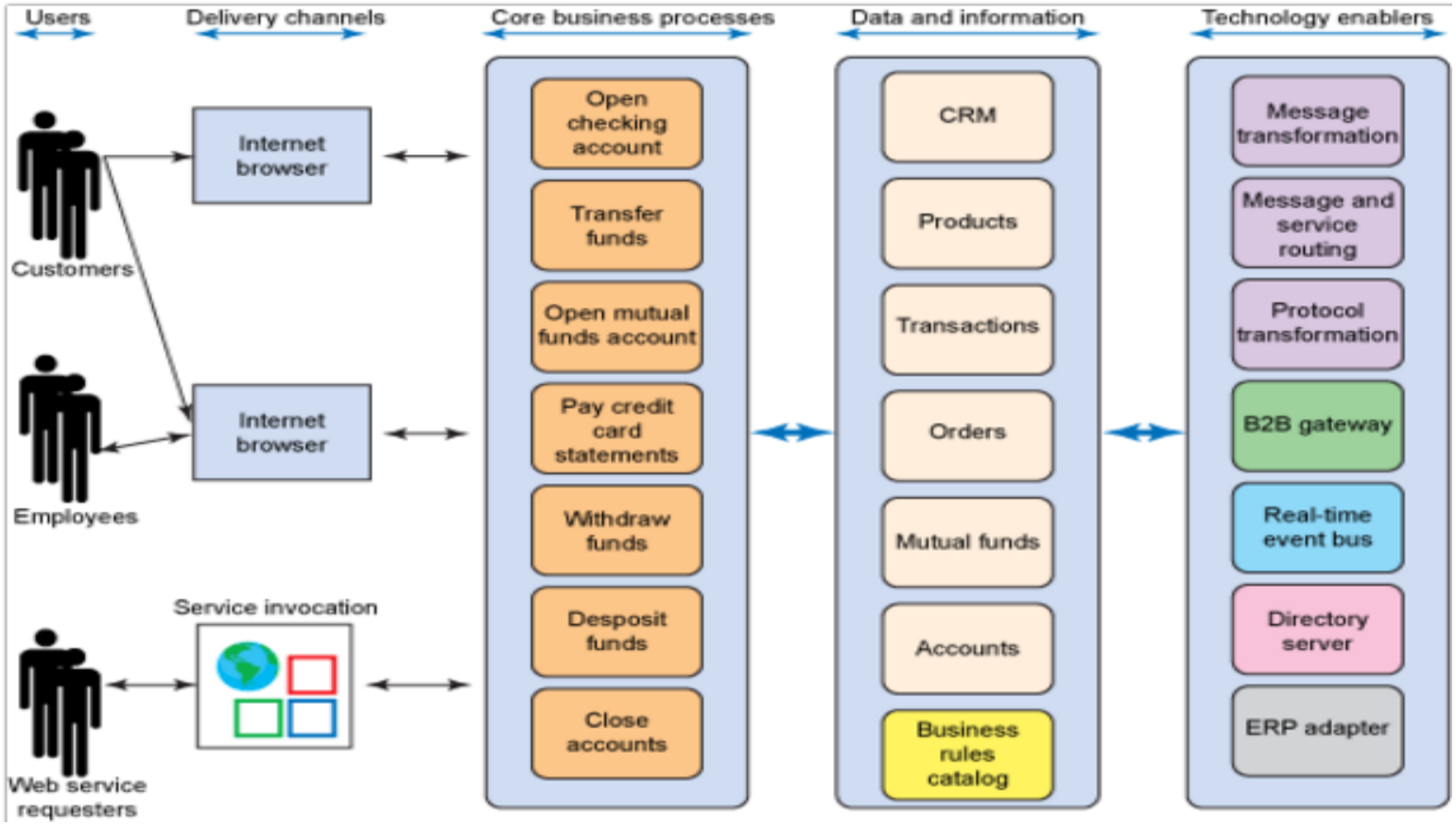
ARCHITECTURAL DESIGN - EXAMPLE

- the sub-systems making the system and their relationship are identified
- depicted as a set of interconnected subsystems



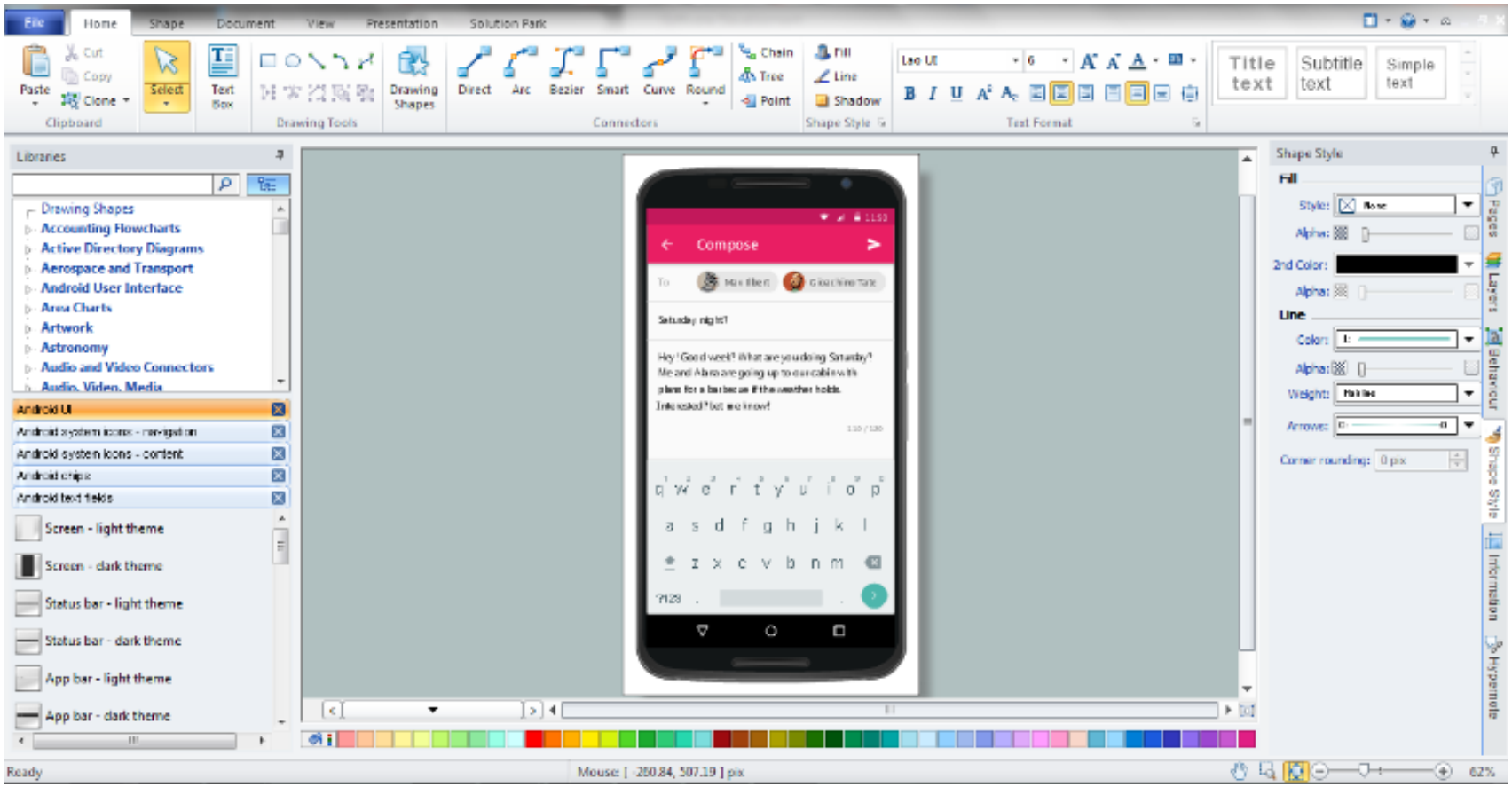
DESIGN PHASES/ACTIVITIES

- *Abstract specification* for each sub system, an abstract specification of its services and constraints are produced



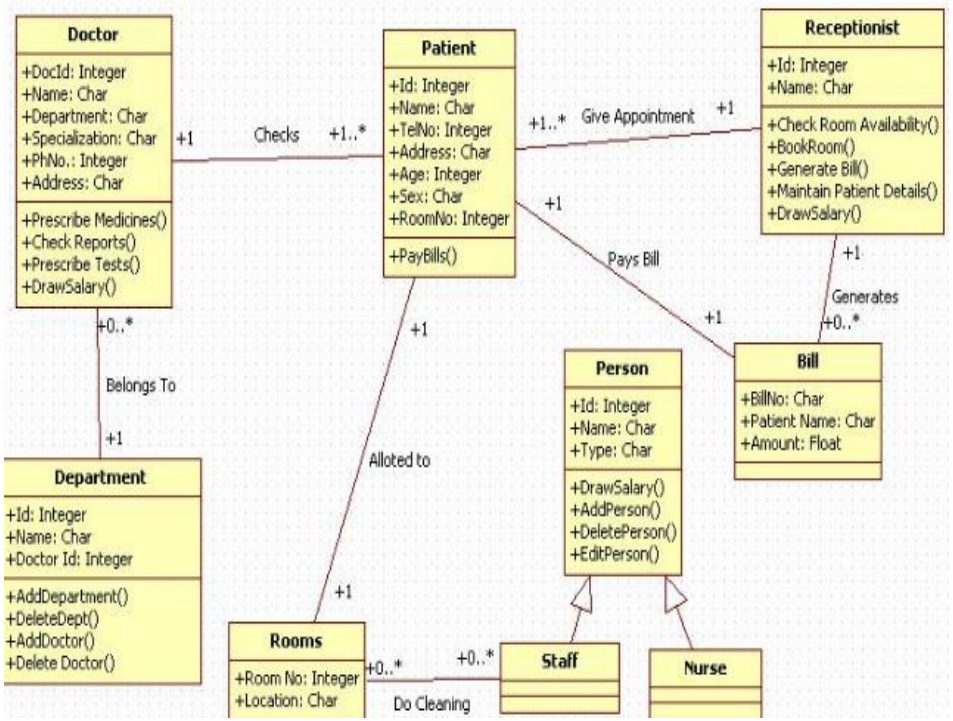
DESIGN PHASES/ACTIVITIES

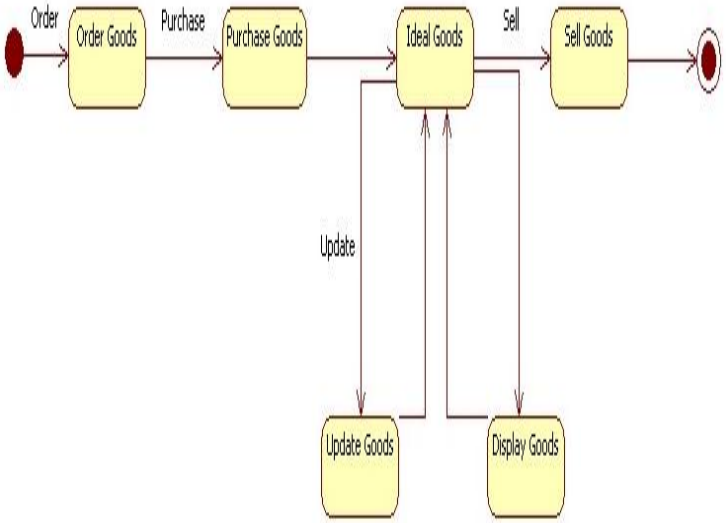
- *Interface design* for each sub system, its interfaces with others is designed



DESIGN PHASES/ACTIVITIES

- *Component design:* services are allocated to the component and the interfaces for the components are designed





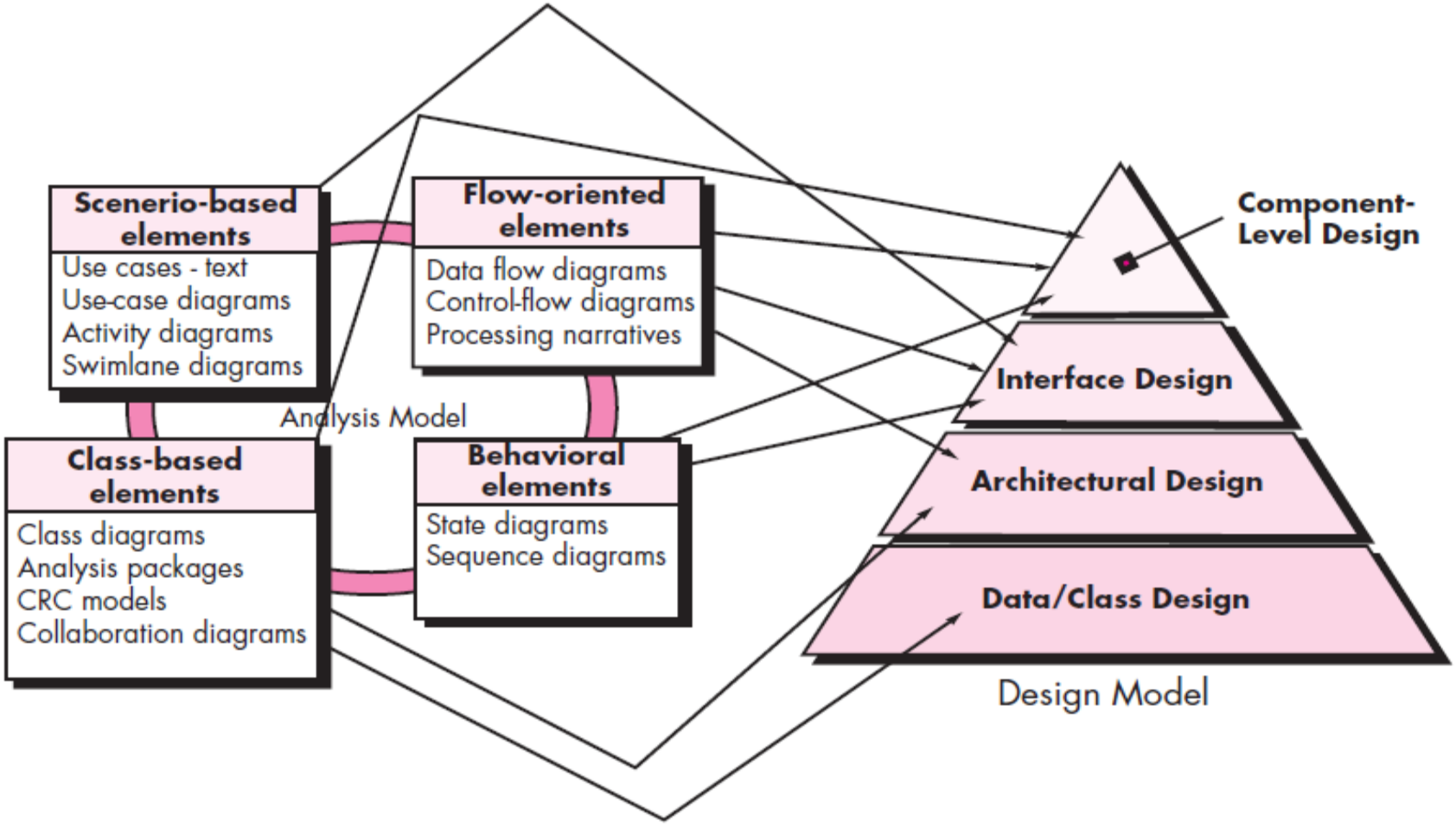
DESIGN PHASES/ACTIVITIES

- *Data structure design* the data structures used in this system are designed in detailed
- *Algorithm design* the algorithms used in this system are designed in detail

DESIGN IN CONTEXT OF SE

- Software design sits at the technical kernel of software engineering and is applied regardless of the software process model that is used. Beginning once software requirements have been analyzed and modeled, software design is the last software engineering action within the modeling activity and sets the stage for construction (code generation and testing).

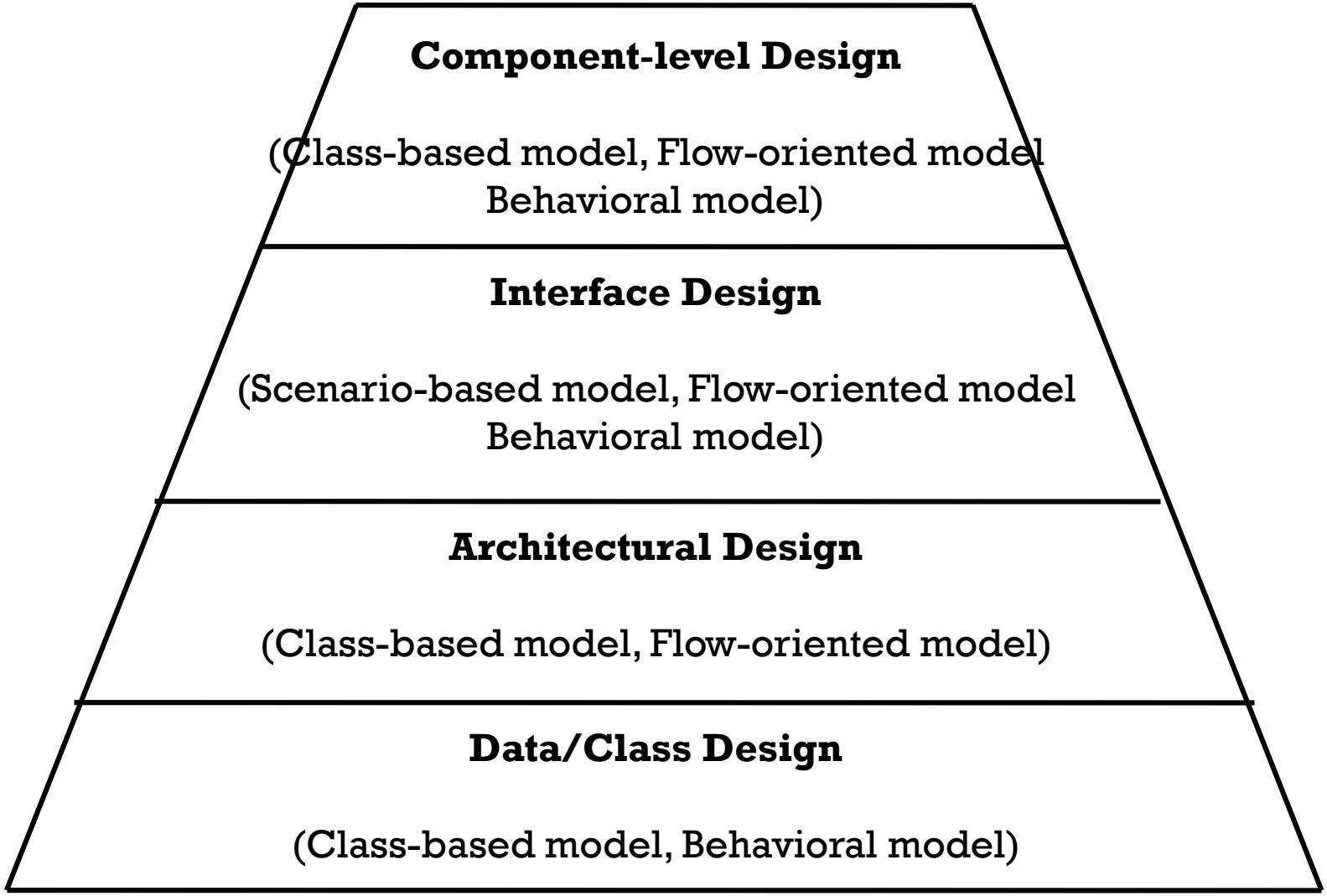
TRANSLATING THE ANALYSIS MODEL INTO THE DESIGN MODEL



FROM ANALYSIS MODEL TO DESIGN MODEL

- Each element of the analysis model provides information that is necessary to create the four design models
  - The **data/class design** transforms analysis classes into design classes along with the data structures required to implement the software
  - The **architectural design** defines the relationship between major structural elements of the software
  - The **interface design** describes how the software communicates with systems that interoperate with it and with humans that use it
  - The **component-level design** transforms structural elements of the software architecture into a procedural description of software components

From Analysis Model to Design Model (continued)



DESIGN AND QUALITY/GOALS OF A GOOD DESIGN

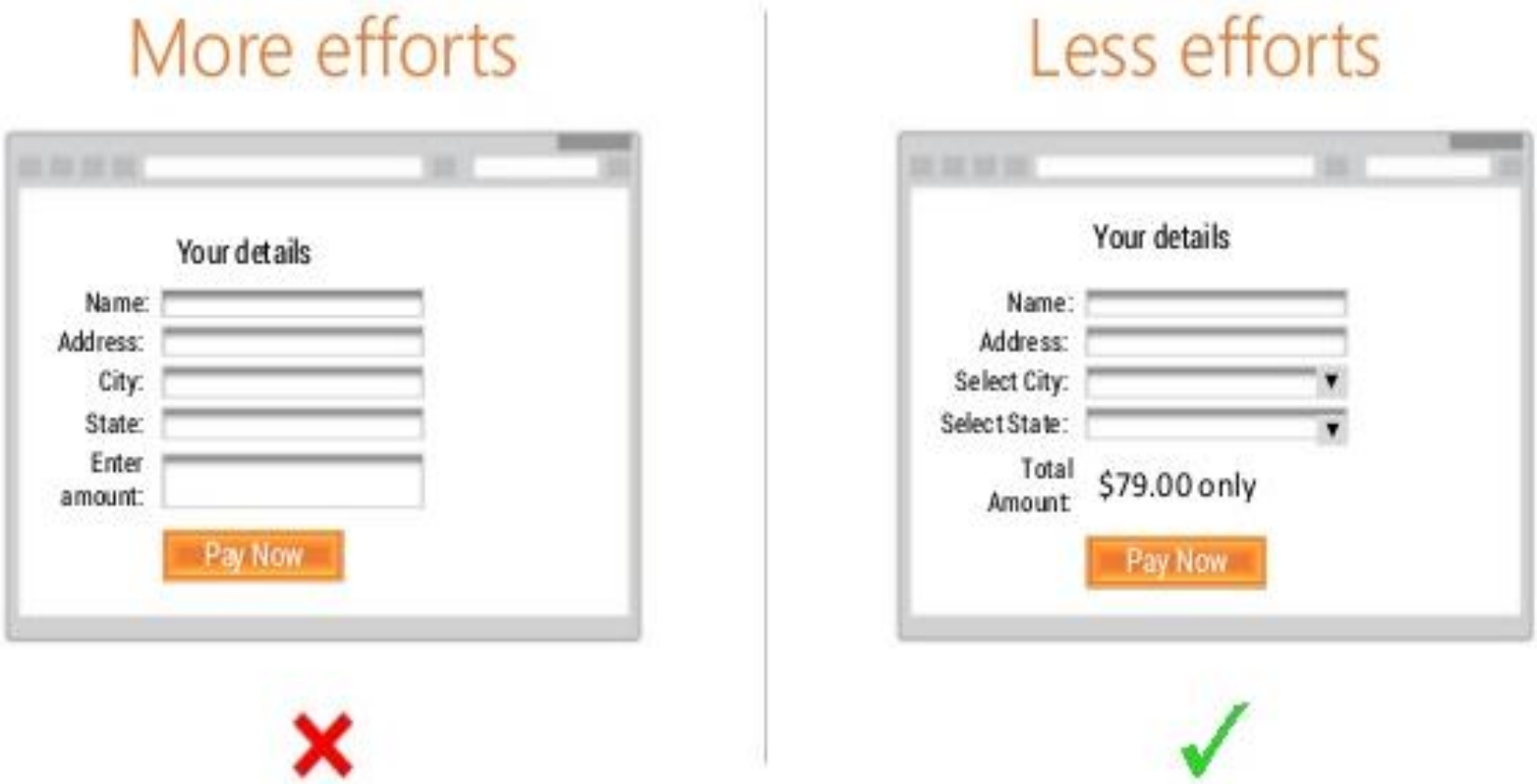
- The design must implement all of the explicit requirements contained in the analysis model, and it must accommodate all of the implicit requirements desired by the customer.
- The design must be a readable, understandable guide for those who generate code and for those who test and subsequently support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

GOOD DESIGN VS BAD DESIGN

Bad Design	Good Design
A design that is vague about underlying requirements	Good design is the one in which all the customer's requirements are incorporated
When you make a change in one piece of code and it breaks some other part which you never expected	Good designs are ones that are easily readable, easily maintainable, easily understood (good interface)
Bad user interface would add pain and complexity for user implicit requirements are unmentioned, for example ease of use, maintainability etc	A good software has the right amount of separation of concerns and modularity (the code encapsulated in the right logical tiers/modules/layers) making it maintainable.

"a product's quality is a function of how much it changes the world for the better." – meaning user satisfaction is more important than anything in determining software quality.

GOOD DESIGN VS BAD DESIGN



"Writing a clever piece of code that works is one thing; designing something that can support a long-lasting business is quite another."

QUALITY GUIDELINES

- A design should exhibit an architecture that (1) has been created using recognizable architectural styles or patterns, (2) is composed of components that exhibit good design characteristics and (3) can be implemented in an evolutionary fashion
  - For smaller systems, design can sometimes be developed linearly.
- A design should be modular; that is, 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.
- 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.

DESIGN PRINCIPLES

- The design process should not suffer from 'tunnel vision.'
- The design should be traceable to the analysis model.
- The design should not reinvent the wheel.
- The design should "minimize the intellectual distance" [DAV95] between the software and the problem as it exists in the real world.
- 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, or operating conditions 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.

From Davis [DAV95]

DESIGN CONCEPTS

1. Abstraction
2. Architecture
3. Design Patterns
4. Modularity
5. Information Hiding
6. Functional Independence
7. Refinement
8. Refactoring
9. OO design concepts
10. Design Classes—provide design detail that will enable analysis classes to be implemented

DESIGN CONCEPTS-ABSTRACTION

Abstraction: IEEE defines abstraction as

- **'A view of a problem that extracts the essential information relevant to a particular purpose and ignores the remainder of the information.'**
- **An abstraction of a component describes the external behavior of that component without bothering with the internal details that produce the behavior .It is essential for partitioning.**

TYPES OF ABSTRACTION

- **Data abstraction – a named collection of data that describes a data object.**
  - (e.g. door type, swing direction, weight, dimension)
- **Procedural abstraction – A technique in which a main function consists of a sequence of function calls and each function is implemented separately. All of the details of the implementation to a particular sub-problem is placed in a separate function. The main function becomes a more abstract outline of what the program does**
  - Eg: Word OPEN for a door
- **Control abstraction – It implies a program control mechanism without specifying internal details**
  - Example: door opening by sensing finger prints or eye scanning

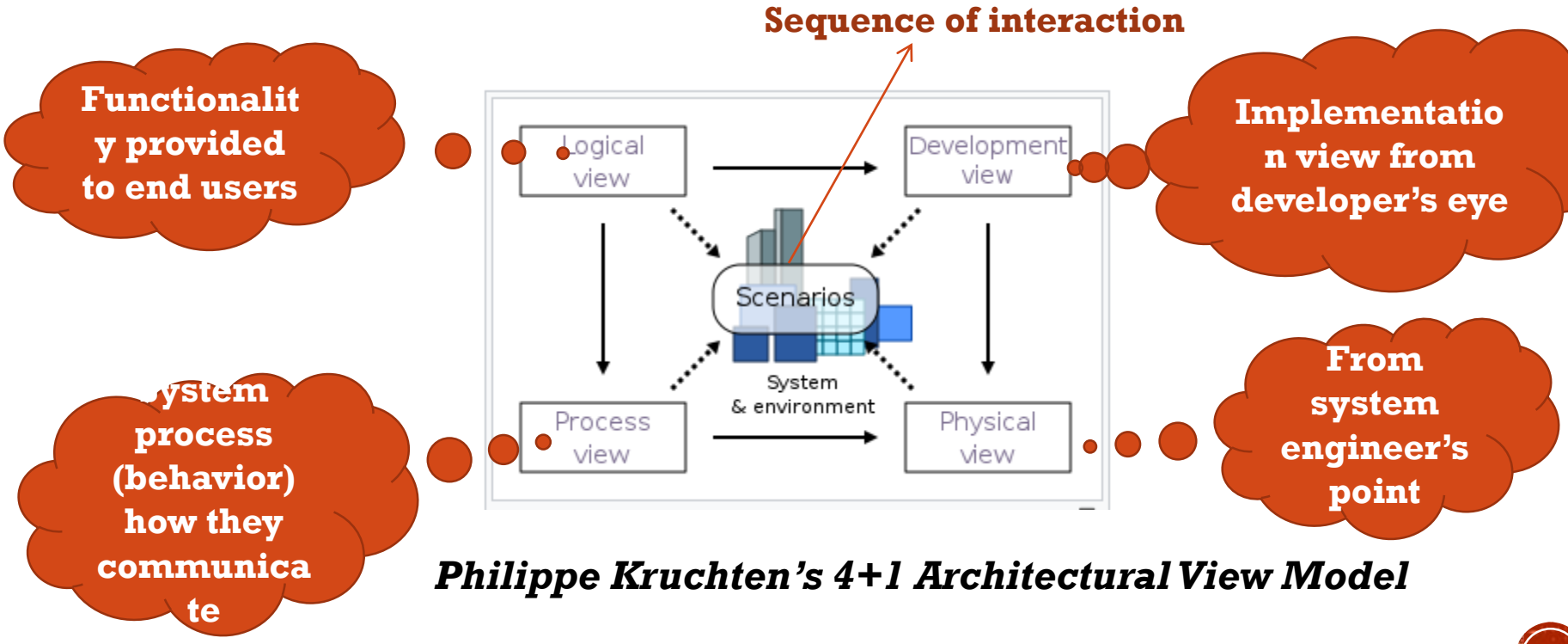
DESIGN CONCEPTS- ARCHITECTURE

- The overall structure of the software and the ways in which the structure provides conceptual integrity for a system

- A set of properties should be specified as part of an architectural design:
- Structural properties. This aspect of the architectural design representation defines the components of a system (e.g., modules, objects) and the manner in which those components are packaged and interact with one another.
- Extra-functional properties. The architectural design description should address how the design architecture achieves requirements for performance, reliability, security, adaptability, and other system characteristics.
- Families of related systems. the design should have the ability to reuse architectural building blocks.

CHARACTERISTICS OF ARCHITECTURE DESIGN

- ✓**multiple views of stakeholders**
- ✓In order to reduce the complexity **separate design model view** points for various stakeholders should be provided – ***separation of concern***



Philippe Kruchten's 4+1 Architectural View Model

DESIGN CONCEPTS-PATTERNS

- **A design structure that solves a particular design problem within a specific context**
- **A common solution to a common problem in a given context – a *template* acts as a pattern for designing.**
- **It provides a description that enables a designer to determine**
  - **whether the pattern is applicable,**
  - **whether the pattern can be reused, and**
  - **whether the pattern can serve as a guide for developing similar patterns**

PATTERN-BASED SOFTWARE DESIGN

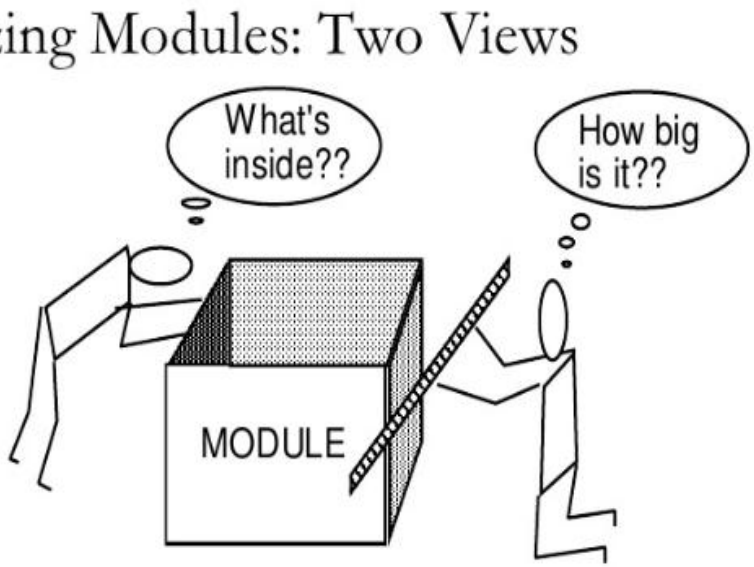
- Mature engineering disciplines make use of thousands of design patterns for such things as buildings, highways, electrical circuits, factories, weapon systems, vehicles, and computers
- Design patterns also serve a purpose in software engineering
- Architectural patterns
  - Define the overall structure of software
  - Indicate the relationships among subsystems and software components
  - Define the rules for specifying relationships among software elements
- Design patterns
  - Address a specific element of the design such as an aggregation of components or solve some design problem, relationships among components, or the mechanisms for effecting inter-component communication
  - Consist of creational, structural, and behavioral patterns
- Coding patterns
  - Describe language-specific patterns that implement an algorithmic or data structure element of a component, a specific interface protocol, or a mechanism for communication among components

DESIGN PATTERN TEMPLATE

- ***Pattern name***—describes the essence of the pattern in a short but expressive name
- ***Intent***—describes the pattern and what it does
- ***Also-known-as***—lists any synonyms for the pattern
- ***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
- ***Consequences***—describes the "design forces" that affect the pattern and the potential trade-offs that must be considered when the pattern is implemented
- ***Related patterns***—cross-references related design patterns

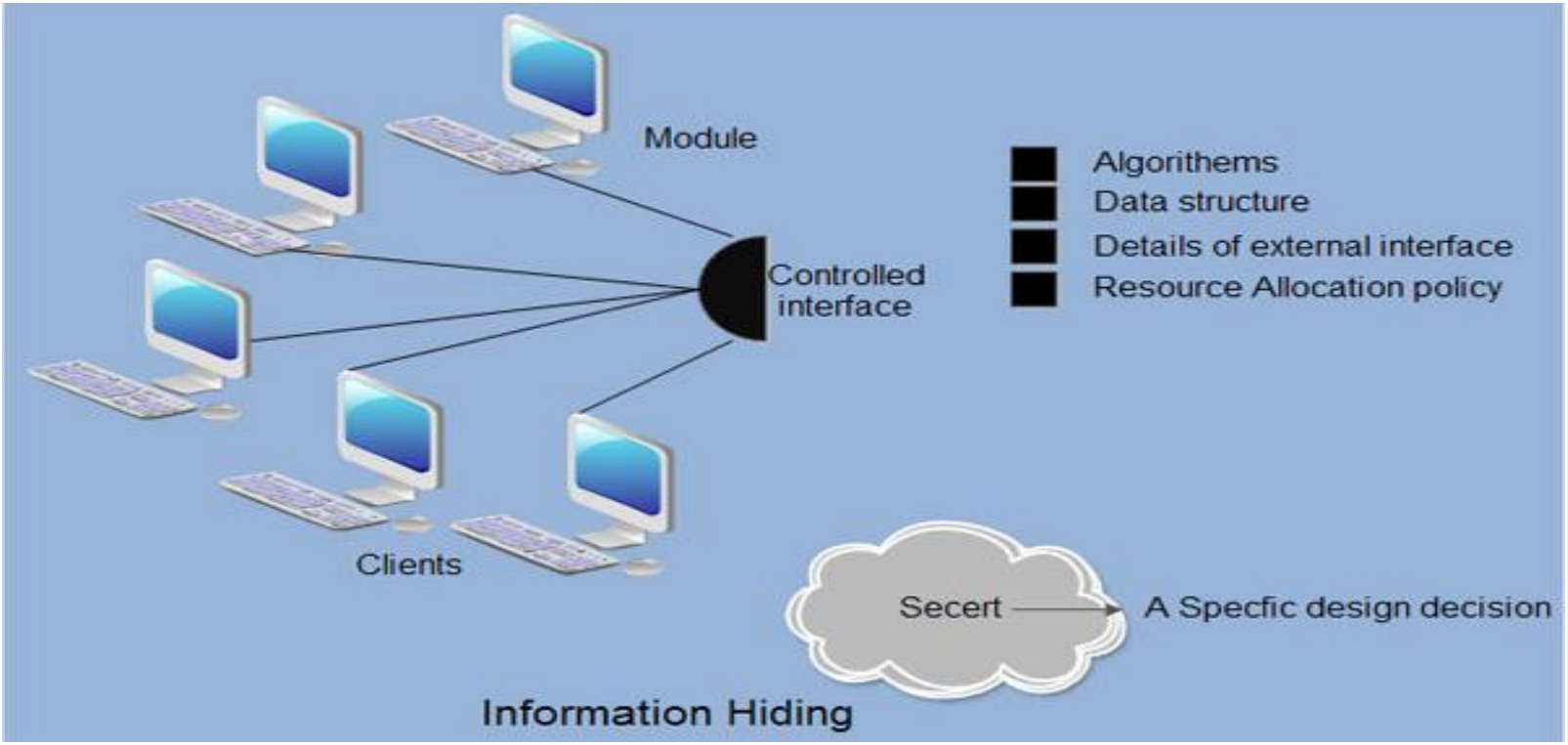
DESIGN CONCEPTS- MODULARITY

- Separately named and addressable components (i.e., modules) that are integrated to satisfy requirements (divide and conquer principle)
- Makes software intellectually manageable so as to grasp the control paths, span of reference, number of variables, and overall complexity
- helps to plan the development in a more effective manner, accommodate changes easily, conduct testing and debugging effectively and efficiently, and conduct maintenance



DESIGN CONCEPTS-INFORMATION HIDING

- The designing of modules so that the algorithms and local data contained within them are inaccessible to other modules
- This enforces access constraints to both procedural (i.e., implementation) detail and local data structures



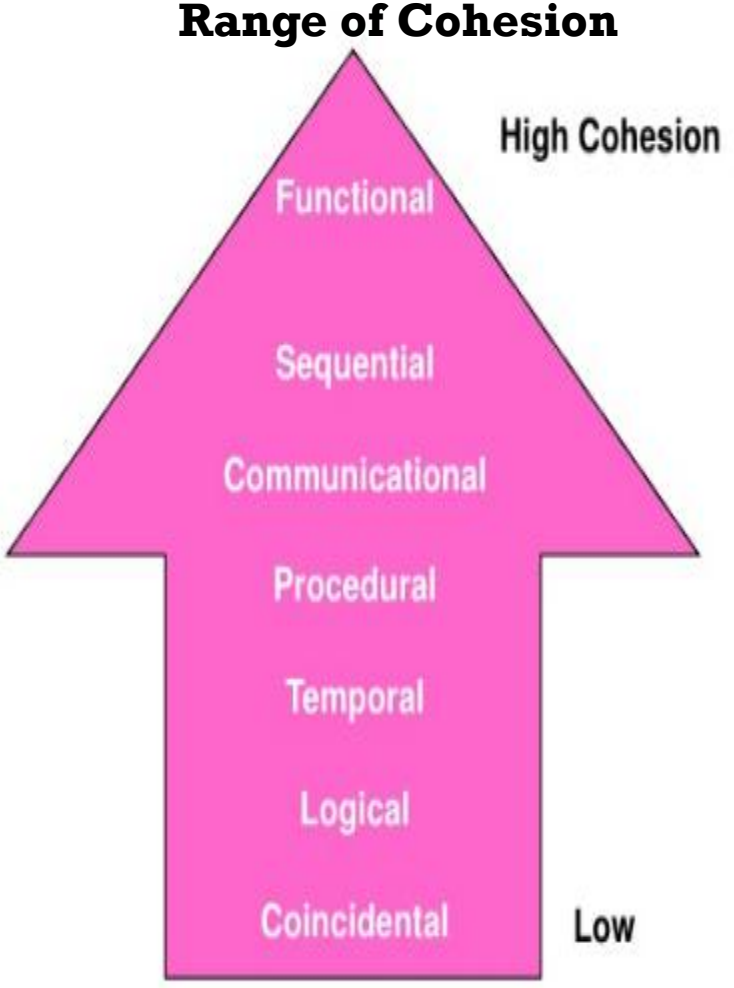
DESIGN CONCEPTS-FUNCTIONAL INDEPENDENCE

- Modules that have a "single-minded" function and an aversion to excessive interaction with other modules
- There are measures by which the quality of design of a modules and the interaction among them can be measured. These measures are called coupling and cohesion.

  - High cohesion – a module performs only a single task
  - Low coupling – a module has the lowest amount of connection needed with other modules

FUNCTIONAL INDEPENDENCE - COHESION

- "The measure of the strength functional relatedness of elements within a module". Cohesion refers to the dependence within and among a module's internal elements
- All elements of component are directed toward and essential for performing the same task
- The more cohesive a module, the more closely related its pieces are to each other.

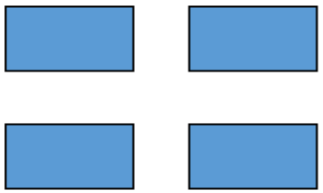


FUNCTIONAL INDEPENDENCE – COUPLING

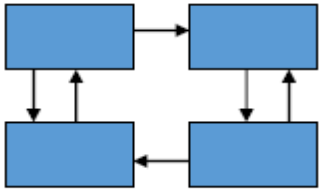
- Ways that modules can de dependent on each other:
  - *The reference made from one module to another*
    - *Module A invokes Module B*
    - *Module A depends on Module B for completion of its function*
  - *The amount of data passed from one module to another*
    - *Module A passes a parameter (contents of an array) to Module B*
  - *The amount of control that one module has over the other*
    - *Module A passes a control flag to Module B*
    - *Value of the flag tells module B the state of some resource or subsystem, which procedure to invoke or whether to invoke a procedure*

COUPLING

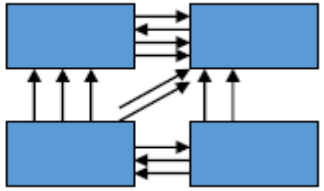
Coupling: Degree of Dependence Among Components



No dependencies



Loosely coupled-some dependencies



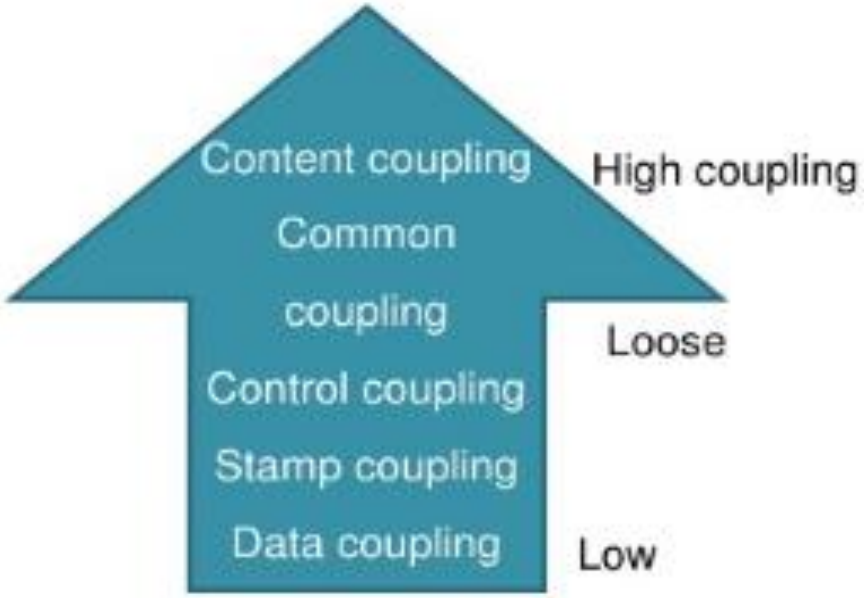
Highly coupled-many dependencies

High coupling makes modifying parts of the system difficult, e.g., modifying a component affects all the components to which the component is connected.

TYPES OF COUPLING

- Content Coupling
- Common Coupling
- Control Coupling
- Stamp Coupling
- Data Coupling

The goal is to keep degree of coupling as low as possible.



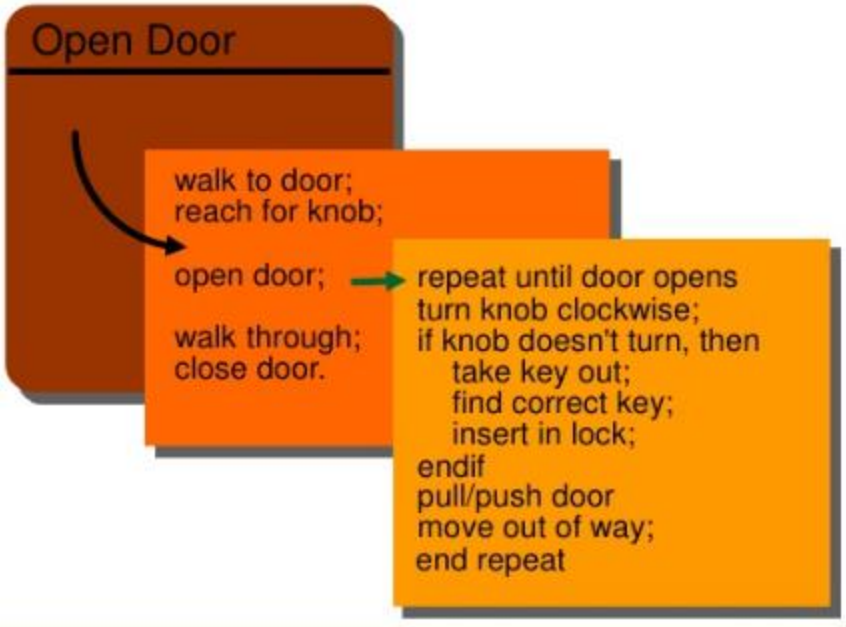
Range of coupling

DIFFERENCE BETWEEN COHESION AND COUPLING

Cohesion	Coupling
Cohesion is the indication of the relationship within **module**.	**Coupling** is the indication of the relationships between modules.
Cohesion shows the module's relative **functional** strength.	Coupling shows the relative **independence** among the modules.
Cohesion is a degree (quality) to which a component / module focuses on the **single** thing.	Coupling is a degree to which a component / module is connected to the **other** modules.
While designing you should strive for **high cohesion** i.e. a cohesive component/ module focus on a single task (i.e., **single-mindedness**) with little interaction with other modules of the system.	While designing you should strive for **low coupling** i.e. **dependency** between modules should be less.
Cohesion is the kind of natural extension of data hiding for example, **class** having all members visible with a package having default visibility.	Making private fields, private methods and non public classes provides loose coupling.
Cohesion is **Intra – Module** Concept.	Coupling is **Inter -Module** Concept.

DESIGN CONCEPTS- STEPWISE REFINEMENT

- A process of elaboration
- It is a top down design strategy
  - Development of a program by successively refining levels of procedure detail
  - Complements abstraction, which enables a designer to specify procedure and data and yet suppress low-level details



DESIGN CONCEPTS-REFACTORING

- A reorganization technique that simplifies the design (or internal code structure) of a component without changing its function or external behaviour.

- When software is re-factored, the existing design is examined for redundancy, unused design elements, inefficient or unnecessary algorithms, poorly constructed or inappropriate data structures, or any other design failures.

OO 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

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.

That is all