CSE1005: Software Engineering

Module No. 2: Requirements Engineering and Design

Dr. K Srinivasa Reddy
SCOPE, VIT-AP University

Module No. 2: Requirements Engineering and Design

Requirements: Requirements Engineering, UML Model, Developing Use Cases, Building the Requirements Model, Negotiating Requirements, Validating Requirements.

Design: Design within the Context of Software Engineering, Design Process, Design Concepts, Design Model.

Text Book:

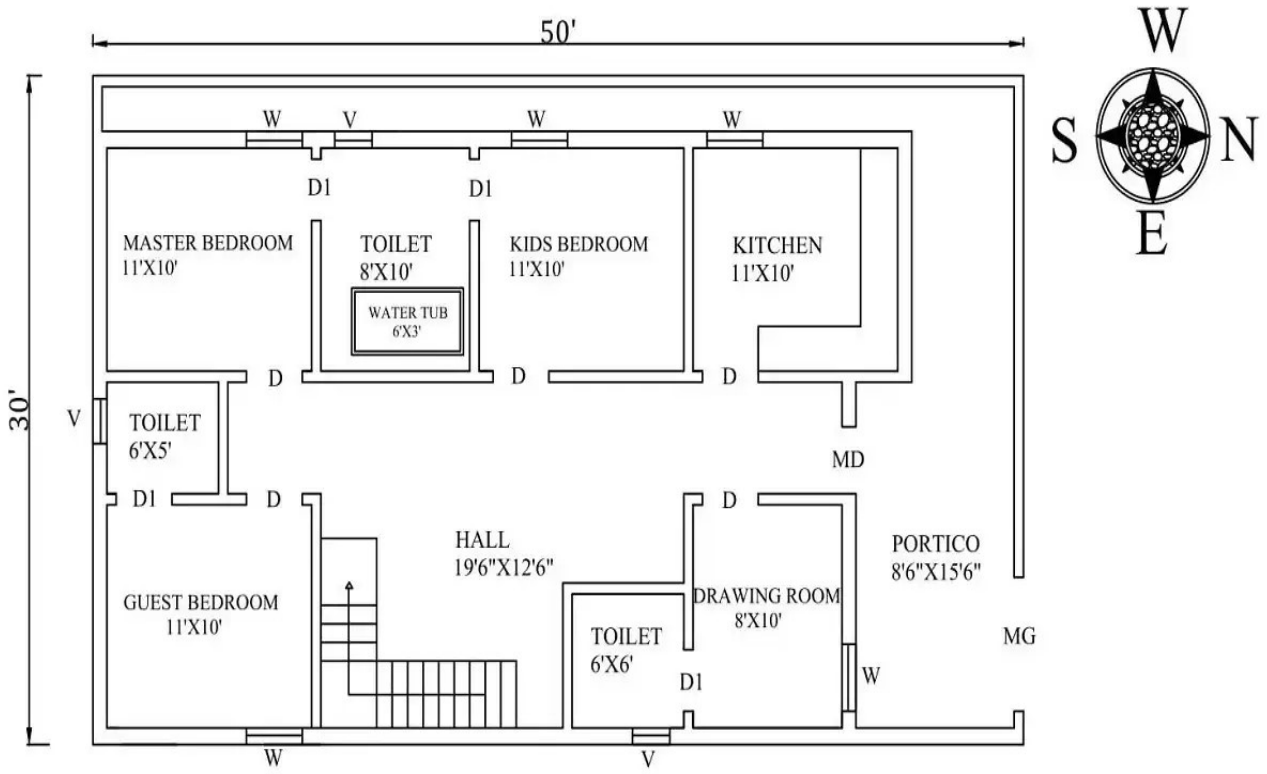
1. Roger Pressman, "Software Engineering: A Practitioner's Approach", McGraw-Hill, 7th Edition, 2016.

Expected Outcome: Analyse the requirements and design the structure, behavior of the software system using UML design techniques (CO2)

House Construction: Requirements, Analysis

- Land purchase
- Planning and execution
- Do you want to hire a contractor or do it yourself?
- What materials will you need?
- How big your house should be?
- Start making detailed plans and getting the permits and plan approvals.
- After Permits & Plans – Implementation
- Begin excavation, start removing the topsoil and digging down to the foundation.
- Once the foundation is in place - start putting up the walls and roof - framing.
- Once the framing is done - start installing the finishes, such as Interior Design, doors, siding, roofing, etc.
- The final stage is landscaping - the lawn, plants, and trees.

House Construction



In previous stages, focused on obtaining

- Requirements
- An analysis model
- Now, set the stage for construction called as **Design**

What is Design?

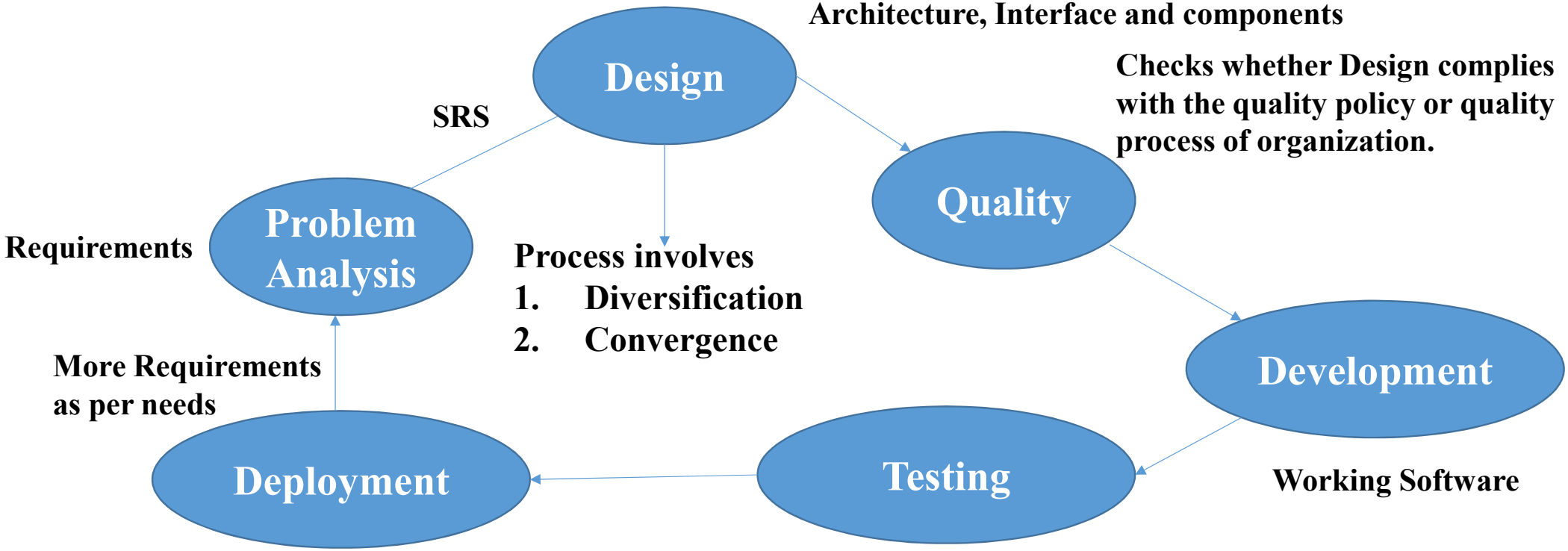
- A **meaningful representation** of something **to be built.**
- It's a **middle stage** between analysis and actual software being developed.
- It's a **process** by which **requirements** are **translated** into a **blueprint** for constructing a software. **(How we get?)**
- **Blueprint** gives the holistic view of a software i.e., **how software is going to be look like.**

Problem domain

Who does the design process?
S/W Engineers, design engineers, expertise in design process etc..

Solution domain

Software Design Process

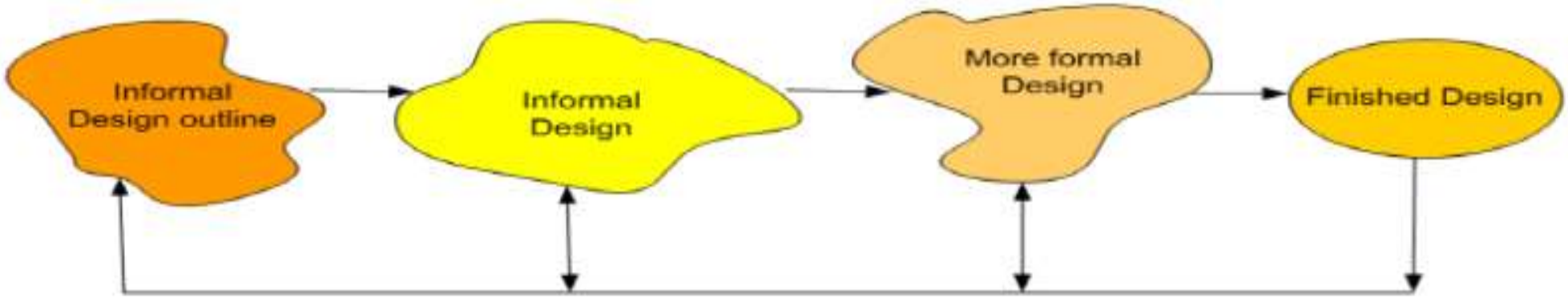


Software Design Process is an iterative process

- **Diversification : Many** types of designs are being proposed for a particular problem. [Don't restrict to one design for a problem, there is no unique way to design a software]
- **Convergence:** Out of those many designs, choose the best design for a particular problem through brainstorming sessions where many people participate and debate to adopt as a solution.

Software Design Process

- Software design is an **iterative process** through which requirements are translated into a **"blueprint"** for constructing the software.
- The **outcome** of a software delivery depends upon the design.



Design Process

- **Quality of design** should be assessed with technical reviews with **feedback** during the iterative process
- The output of the design phase is **Software Design Document (SDD).**

- Design is what almost **every engineer wants to do**.
- It's a **creative activity – out of the box thinking**
- Software design encompasses **the set of principles, concepts,** and **practices** that **lead to the development** of a **high-quality** system or product.
- **Most critical activity** during system development
  – If **design goes bad**, entire software **deliverable goes bad**
- Design creates a **representation** or **model** of the **software**
- The design model provides in detail about
  - Data structures
  - Software architecture,
  - Interfaces, and
  - Components that are necessary to implement the system.
- Design is the place where **software quality is established**.
- Has **great impact** on testing and maintenance

From Analysis Model to Design

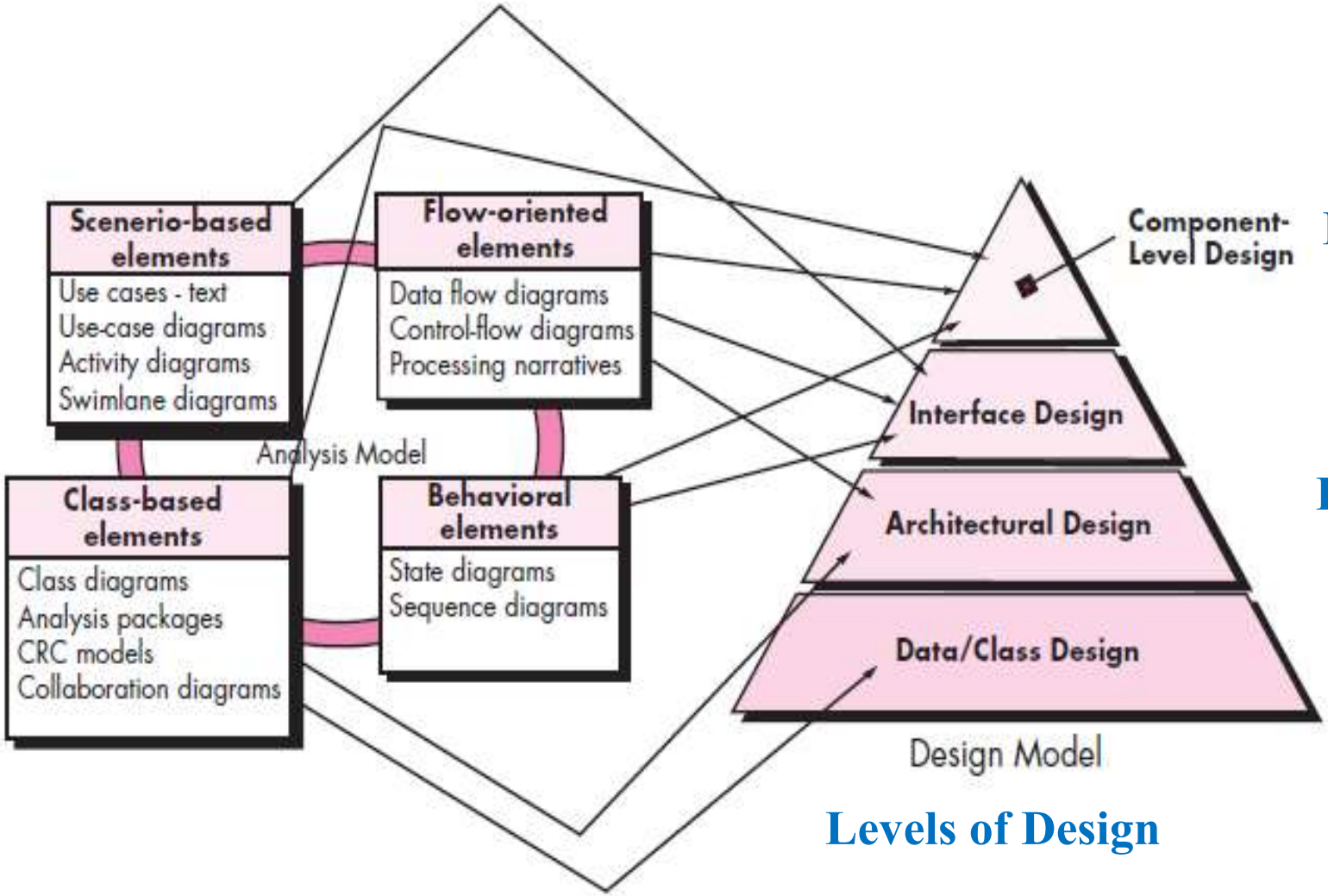
- The **goal of design** is to produce a **model or representation** that exhibits:
  - **Firmness**: A program **should not have any bugs** that inhibit its function.
  - **Commodity**: A program should be **suitable for the purposes** for **which it was intended.**
  - **Delight**: The experience of using the program should be a **pleasurable one**.

Design within The Context of Software Engineering

- 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).**

- Software design should **always begin** with
  - A **consideration of data** - the foundation for all other elements of the design.
  - After the foundation is laid, the **architecture** must be derived.
  - Only then perform other design tasks.

Translating the Requirements Model into the Design Model



Levels of Design

Outcome of Design

Procedural description of software components

Human Computer Interaction

Relationships among structural elements

Data Structures

Translating the Requirements Model into the Design Model

Data / Class Design:

- Transforms **class models** into **design class realizations** and the **requisite data structures** required to implement the software.
  - Part of class design may occur in conjunction with the design of software architecture.

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 architecture can be implemented.

Translating the Requirements Model into the Design Model

The Interface Design:

- Describes **how** the **software communicates with systems that interoperate with it,** and **with humans** who use it.
  - An interface implies a **flow of information** (e.g., data and/or control) and a specific **type of behavior**.
  - **Usage scenarios** and **behavioral models** provide much of the information required for interface design.

The Component-Level Design:

- Transforms **structural elements** of the software architecture into a **procedural description** of software components.
  - Information obtained from the **class-based models, flow models, and behavioral models** serve as the basis for component design.

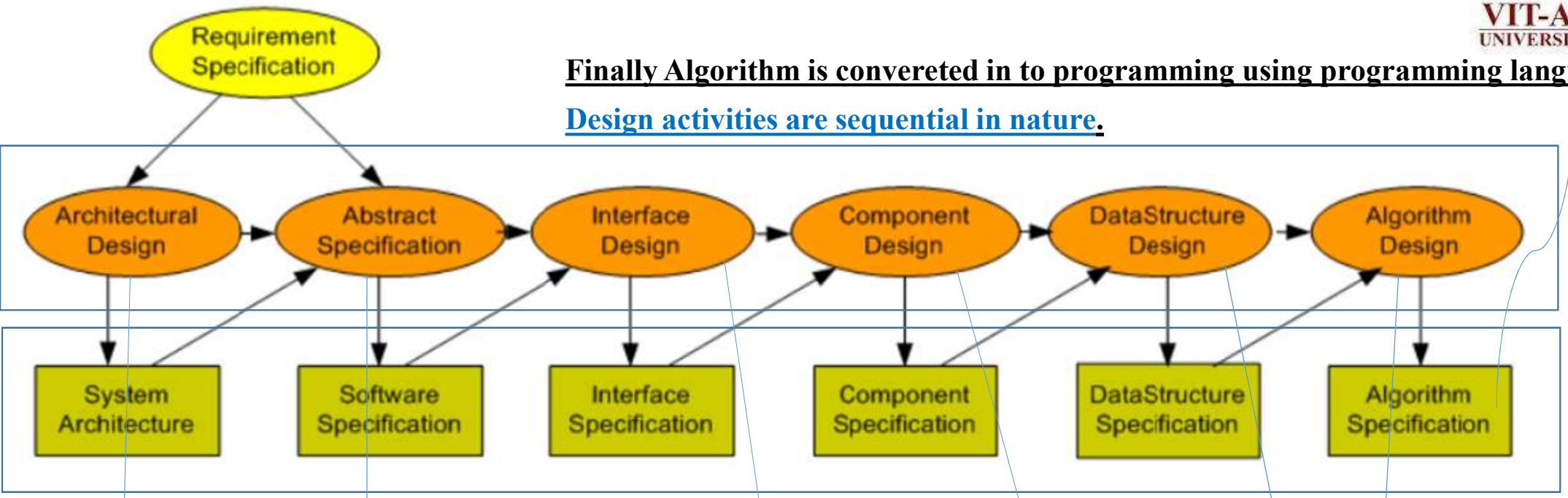
Software Design Process

Basic **input** is **requirement specification** to perform **design activities**.

VIT-AP UNIVERSITY

Finally Algorithm is convereted in to programming using programming language.

Design activities are sequential in nature.



Outcome of Design activities are Design Outcomes.

Will Identify the subsystems, Output is System architecture

Will specify the **constraints** of the subsystems and **define** what subsystem is supposed to do. Output is software specification. System architecture is input to abstract specification

Will specify the **interfaces** for the subsystems and how they are interrelated. Output is Interface specification.

Will specify, what **services** each subsystem will perform. Output is Component specification.

Will specify, **databases, data structures** etc. Output is Data Structure specification.

Will specify, **algorithm design** for services. Output is Algorithm specification

The Design and Quality

Quality means customer **satisfaction.**

Design quality depends on **three characteristics**

1. The design **must implement** all functionalities of the system correctly

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

3. The design should **provide** a **complete picture of the software, addressing the data, functional, and behavioral domains** from an **implementation** perspective also should be **easily amendable** to change.

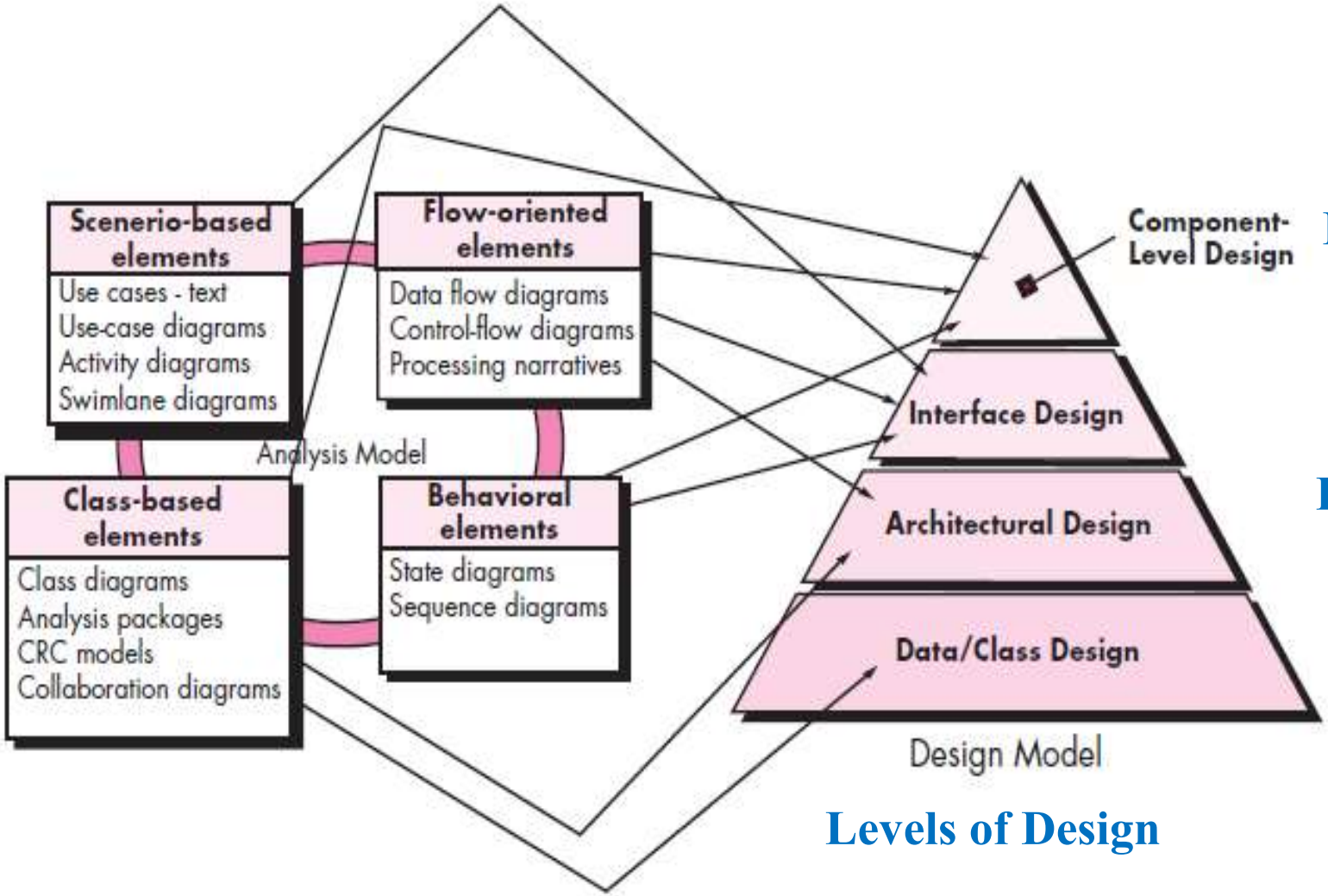
Design Quality Guidelines

Technical criteria for good design: In order to evaluate the quality of a design representation, members of the software team must establish technical criteria for good design.

A design should

1. **Exhibit** an architecture that
   - is created using recognizable architectural styles or patterns
   - is composed of components that exhibit good design characteristics
   - can be implemented in an evolutionary fashion
2. **Be modular**, i.e., The software should be logically partitioned into elements or subsystems
3. **Contain distinct representations** of data, architecture, interfaces, and components.
4. **Lead to data structures** that are appropriate for the classes to be implemented and are drawn from recognizable data patterns.
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** using a notation that effectively communicates its meaning.

Translating the Requirements Model into the Design Model



Levels of Design

Outcome of Design

Procedural description of software components

Human Computer Interaction

Relationships among structural elements

Data Structures

The Design and Quality

Quality means customer **satisfaction.**

Design quality depends on **three characteristics**

1. Must **implement** all functionalities of the system correctly

2. Must be a **readable, understandable** guide for those **who generate code, who test** and subsequently **support** the software.

3. Should **provide** a **complete picture of the software, addressing the data, functional, and behavioral domains** from an **implementation** perspective also should be **easily amendable** to change.

Design Quality Attributes : FURPS

Elevator

1. Functionality:

- Tells us why the customer wants us to create the product, how the customer is going to use it, and how the customer will decide that the product satisfies his or her needs.
- It is **assessed by evaluating** the **feature set** and **capability** of the program, **generality** of the functions that are delivered, and the overall security of the system.

1. **What if the elevator does not move between the floors?**

   It is utterly useless.

   If the product does not solve customer problems or does not satisfy customer needs and expectations, then the whole work of the team goes waste and it is waste of time and resources

Design Quality Attributes : FURPS

Elevator

2. Elevator moves between floors, but all (e.g., somebody put the control buttons on the ceiling) **or some** (e.g., there are no Braille markings for blind people)**, cannot operate it.**
It is remains useless.

2. Usability:

- It helps to describe the **ability of the users**, the **operating conditions** and then distinguish them from the developers and the office environment during the development and the early stages of the testing.
- **Operating conditions include:**
  - Physical, mental (what they can), educational (what they knows), social (what they used to do) characteristics of the user;
  - **Environmental conditions** (such as the **position** of the user, light, available space, noise, whether operating the product is the primary function of the user, what else is the user doing when running the product, etc.);
  - **Process conditions** (such as operation velocity and tempo, distracting factors, price of error).

Design Quality Attributes : FURPS

Elevator

3. The elevator moves and our users can operate it, but it fails to do the job in 1 of 10 cases

It is functioning, usable and considered as a minimal viable product.

At least 9 of 10 customers now ride the elevator instead of crawling the stairs.

3. Reliability: (Durability)

- The interval between failures or the maximum length of the interval between scheduled services..
- **The ability** of the product to withstand rough conditions, such as **unexpected or intentionally malicious user behavior**
- It is **evaluated by** accuracy (Frequency/Severity of Error) of output, Availability (Failure Frequency (Robustness/Durability/Resilience), Failure Extent & Time-Length(Recoverability/Survivability)), Predictability (Stability).
- It is about **how fast** and using which efforts the system must be able to **recover after the crash.**

Design Quality Attributes : FURPS

Elevator

4. The elevator moves, the users can operate it, and it works whenever it is needed.

Still, it **moves slowly or consumes too much electricity.**

It typically satisfies the end-users but has an unacceptable economic performance.

4. Performance:

- **Measured** by speed, response time, resource consumption (processor time, power, RAM, cache, network traffic etc.), throughput, and efficiency.

Design Quality Attributes : FURPS

Elevator

5. Supportability:

- How the operation team on the customer side will build, distribute, install, configure, monitor, and update the product.
- It is also about how our or the other team will be able to modify or extend the product in the future.

5. The elevator moves, the users can operate it, and it works whenever it is needed, giving good performance.

But it would be a nightmare for the service team.

This part of the requirements does not affect the end-users at all.

It is also impossible to discover these requirements by the analysis of the day-by-day usage of the system.

Design Quality Attributes : The FURPS Quality Attributes

VIT-AP UNIVERSITY

1. **Functionality** is **assessed by evaluating** the feature set and capability of the program, generality of the functions that are delivered, and the overall security of the system.
2. **Usability** is **assessed by** considering **human factors, overall aesthetics, consistency and documentation.**
3. **Reliability** is **evaluated by** accuracy (Frequency/Severity of Error) of output, Availability (Failure Frequency (Robustness/Durability/Resilience), Failure Extent & Time-Length (Recoverability/Survivability)), Predictability (Stability).
4. **Performance** is **measured by** speed, response time, resource consumption (power, ram, cache, etc.), throughput, and efficiency
5. **Supportability**
   - It combines the ability to extend the program (extensibility), adaptability, serviceability - these three attributes represent maintainability
   - Testability, compatibility, configurability, are the terms using which a system can be easily installed and found the problem easily.
   - Supportability also consists of more attributes such as compatibility, extensibility, fault tolerance, modularity, reusability, robustness, security, portability, scalability.

Design Concepts

VIT-AP UNIVERSITY

1. Abstraction
2. Architecture
3. Patterns
4. Separation of concerns
5. Modularity
6. Information hiding
7. Functional independence
8. Refinement
9. Refactoring
10. Object Oriented Design Concepts
11. Design Classes

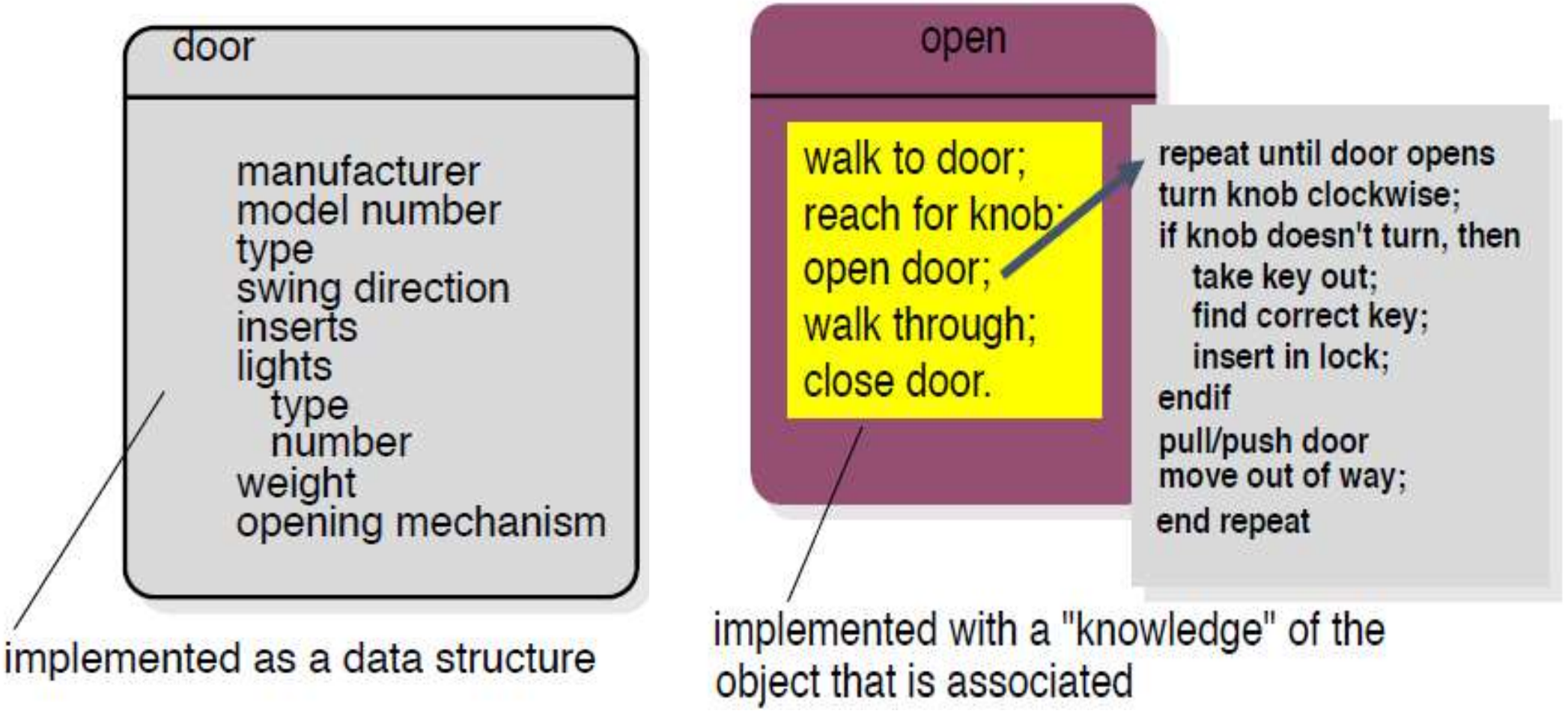
Design Concepts – Abstraction - Hide Irrelevant data :

- **Hide the details to reduce complexity** and **increases efficiency or quality**.
- A solution is stated in large terms using the language of the problem environment at the **highest level abstraction**.
- The **lower level of abstraction** provides a more detail description of the solution.
- **Procedural abstraction:** A sequence of instructions that have a specific and limited function.

  Open the Door
  - Open the Door implies a **long sequence of procedural steps** (e.g., walk to the door, reach out and grasp knob, turn knob and pull door, step away from moving door, etc.)
- **Data abstraction:** A collection of data that describes a data object
  - **Door** is a data abstraction in the Procedural abstraction **Open the door**
  - A set of attributes that describe the door will be door type, swing direction, opening mechanism, weight, dimensions.
- Procedural abstraction **make use of** information contained in the attributes of the data abstraction.

Design Concepts - Abstraction:

“open the door”



Design Concepts - Architecture:

- **The complete structure** of the software is known as **software architecture.**
- It is the **development work product** that gives the **highest return on investment** with respect to quality, schedule, and cost.
- Structure provides **conceptual integrity** for a system in a number of ways.
- **Properties** that should be specified as part of an **architectural design**:
  - **Structural properties** – defines **components** of a system (e.g., modules, objects, filters), and their interaction.
  - **Extra-functional properties** – **characteristics** like performance, capacity, reliability, security, adaptability, etc.
  - **Families of related systems - repeatable patterns** that are commonly encountered in the design of families of similar systems, the design should have the **ability to reuse** architectural building blocks.

Design Concepts - Architecture:

VIT-AP UNIVERSITY

Representation of the architectural design: Different models based on properties

- **Structural models** – An organized **collection of program components**
- **Framework models** – Attempts to identify **repeatable architectural design patterns** encountered in similar types of application, which leads to an increase in the level of abstraction.
- **Dynamic models** - Specifies the **behavioral aspect** of the software architecture and indicates **how the structure or system configuration changes as the function changes due to change in the external environment**
- **Process models - Focuses on the design of the business or technical process**, which must be implemented in the system that has inputs, outputs, events and information that are involved in the process.
- **Functional models -** Represent the **functional hierarchy of a system**.

Design Concepts – Patterns – Repeated form:

- A design pattern **describes a design structure** that structure solves a particular design problem within a specific context and within a set of forces.
- (A **design pattern** is a general repeatable solution to a commonly occurring problem in software design.)

- The **intent of each design pattern** is to provide a description that enables a designer to determine
    1. Whether the pattern is **applicable** to the current work,
    2. Whether the pattern can be **reused** (saving design time), and
    3. Whether the pattern can **serve as a guide** for developing a similar, but functionally or structurally different pattern

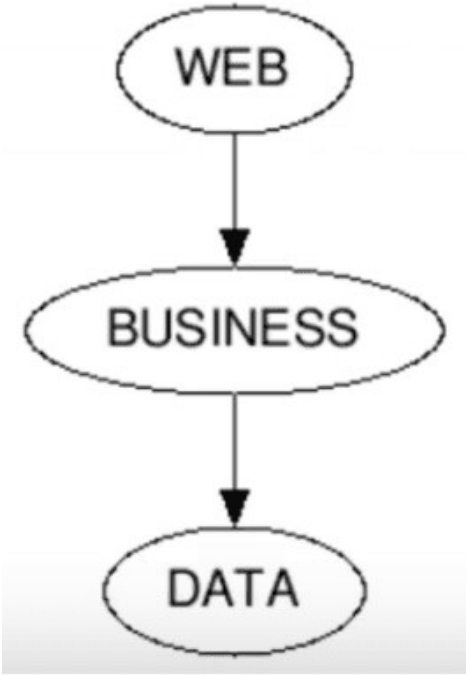
Design Concepts - Separation of Concerns:

- An important design principle for **separating a computer program into distinct sections**.
- Each section **addresses** a separate concern, a set of information that affects the code of a computer program.
- 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.
- Lead to **software modularity, functional independence, and refinement**
- Every element of a software application - a component, a layer, a package, a class or a method should have one concern and implement it well.
  - All **methods** with similar concerns will grouped into a **class**.
  - All **classes** with similar concerns will be grouped into **packages**.
  - So on and so forth.

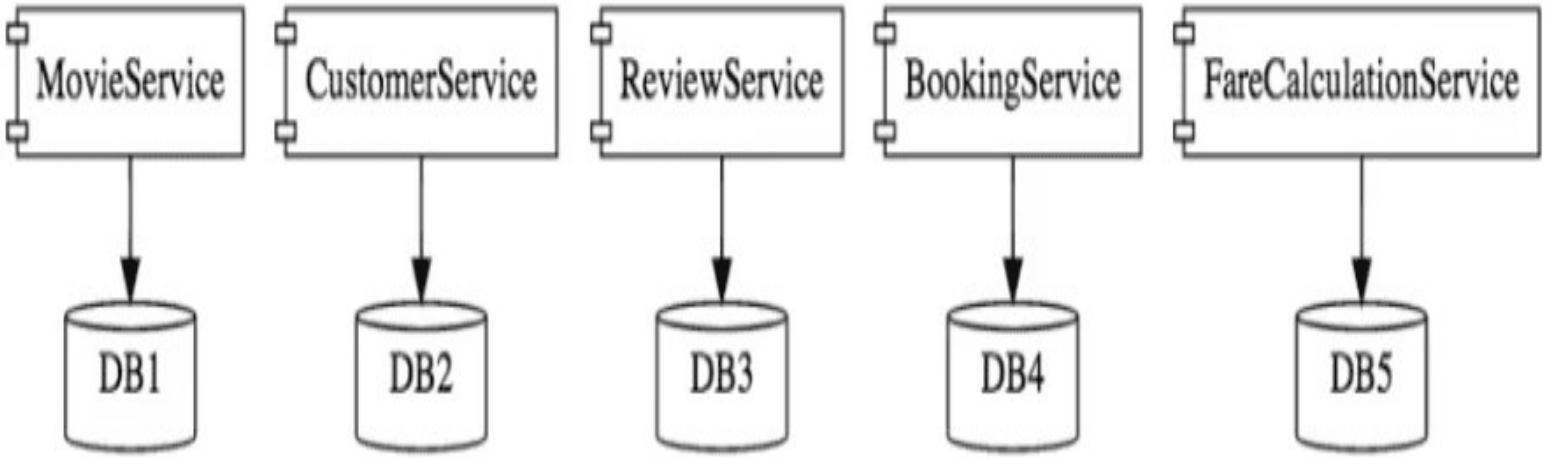
Design Concepts - Separation of Concerns:

VIT-AP UNIVERSITY

- Organizing an application into separate layers is one way of separating out concerns.
- Here are some of the important responsibilities of each of these layers
  - The **Web layer** – focuses on presenting data to its users
  - The **Business layer -** focuses on implementing the core application logic, correctly and efficiently
  - The **Data layer -** responsibility of talking to the data store.
- Each layer has a concern and implements it well.



Design Concepts - Separation of Concerns:



All the micro-services shown are part of a movie application. But each one of them has its own independent concern.

Design Concepts - Modularity:

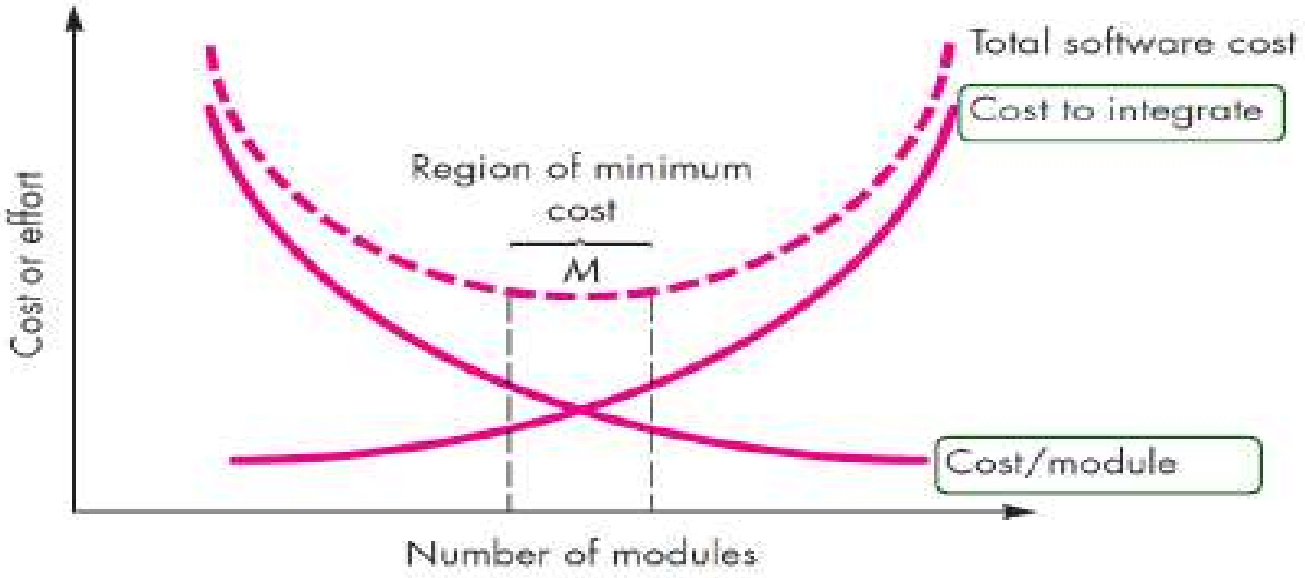
- Modularity **dividing the software into uniquely named and addressable components** (called as **modules**.)
- Each module can be developed **independent of other modules**.
- Based on **divide and conquer** principle
- After developing the modules, they are **integrated together** to meet the software requirements.
- **Modularizing a design helps**
  - To plan the development in a more effective manner,
  - Reduces the design complexity
  - Accommodate changes easily,
  - Easier and faster implementation by allowing parallel development of various parts of a system.
  - Conduct testing and debugging effectively and efficiently, and
  - Conduct maintenance work without adversely affecting the functioning of the software

Design Concepts - Modularity:

- ***What is the "right" number and size of the modules?***
- As the number of modules grows, the effort associated with integration also grows.

A number, *M,* of modules that would result in minimum development cost but care should be taken to stay in the vicinity of *M.*

Under_modularity or over_modularity should be avoided.



Relationship between Number of modules and cost of effort

Design Concepts - Information Hiding:

VIT-AP UNIVERSITY

- Modules should be specified and designed in such a way that the **information (algorithms and data)** contained **within a module** are **not accessible to other modules.**
- Modules **pass only the required information to each other**, to accomplish the software functions.
- The way of **hiding unnecessary details** is referred to as **information hiding.**
  - **Abstraction** - helps to **define** the procedural (or informational) entities that make up the software.
  - **Hiding** - defines and enforces access **constraints** to **both procedural detail within a module** and any **local data structure** used by the module
- Data and procedural details are hidden from other modules, leads to **encapsulation** - an attribute of high quality design

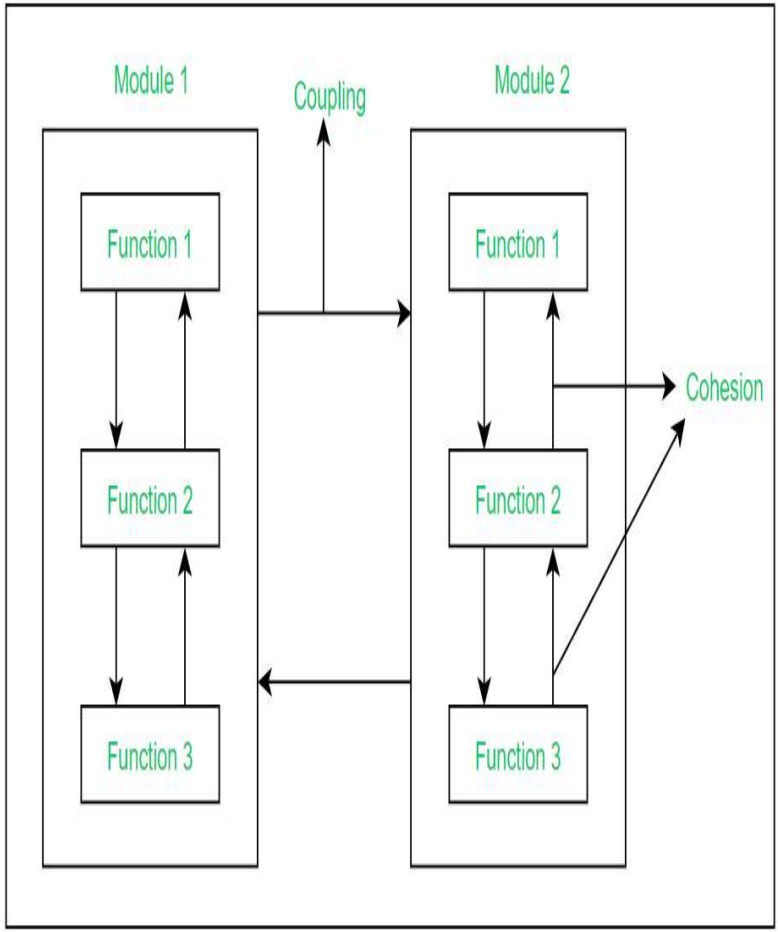
Design Concepts - Information Hiding:

VIT-AP UNIVERSITY

- Information hiding is of immense use when modifications are required during the testing and maintenance phase.
- **Advantages:**
  - Emphasizes communication through controlled interfaces
  - Decreases the probability of adverse effects
  - Restricts the effects of changes in one component on others
  - Results in higher quality software.
  - Leads to low coupling

Design Concepts - Functional Independence:

- The **functional independence** is measured using two criteria:
- **Cohesion**
  - An indication of the **relative functional strength** of a module
  - Cohesion is an **extension** of the information hiding concept.
  - A cohesive module performs a single task and it requires a small interaction with the other components in other parts of the program.
- **Coupling**
  - An **indication of interconnection between modules** in a structure of software (measures the relative interdependence among modules).



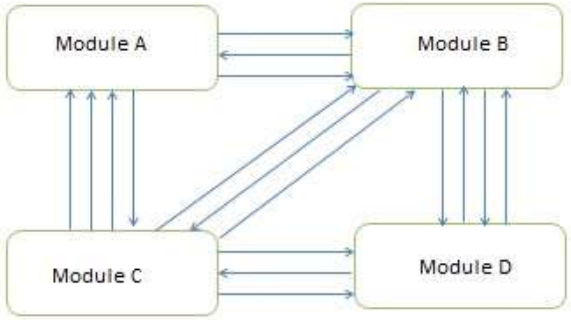
A good software design requires **high cohesion** and **low coupling**.

Design Concepts - Functional Independence:

VIT-AP UNIVERSITY

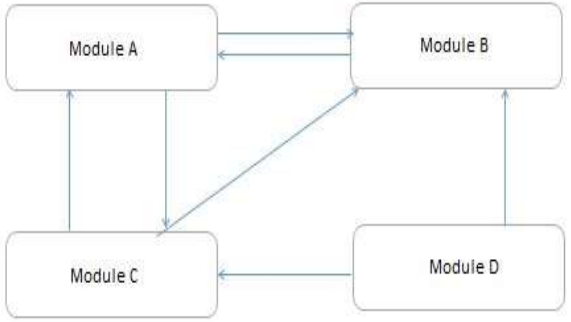
High Coupling

- These type of systems have interconnections with program units **dependent** on each other.
- Changes to one subsystem **leads to high impact on the other subsystem.**



Low Coupling

- These type of systems are made up of components which are **independent or almost independent.**
- A change in one subsystem **does not affect** any other subsystem.



Design Concepts - Functional Independence:

- A function having **high cohesion and low coupling** is called **functionally independent** of other modules.
- The functional independence is the **concept of separation** and related to the concept of **modularity, abstraction and information hiding**.
- Functional independence is achieved by developing functions that **perform only one kind of task** and **do not excessively interact with other modules**.
- Functional independence is a **good design feature** which ensures **software quality.**
- **Advantages of functional Independence:**
  - Better understandability
  - Complexity of design is reduced
  - Easier to maintain, test, and
  - Reduces error propagation and
  - Reuse of module is possible

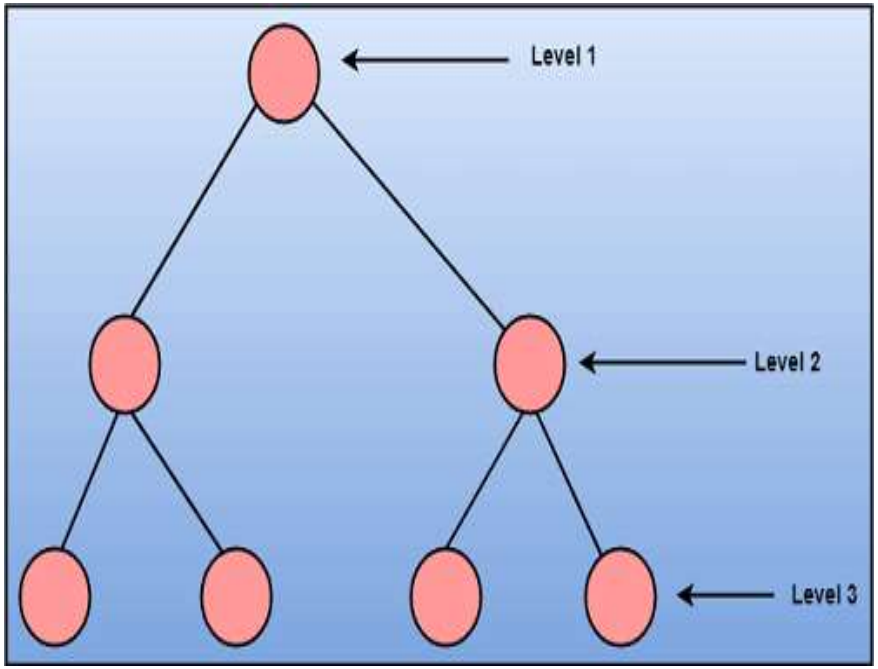
Design Concepts

VIT-AP UNIVERSITY

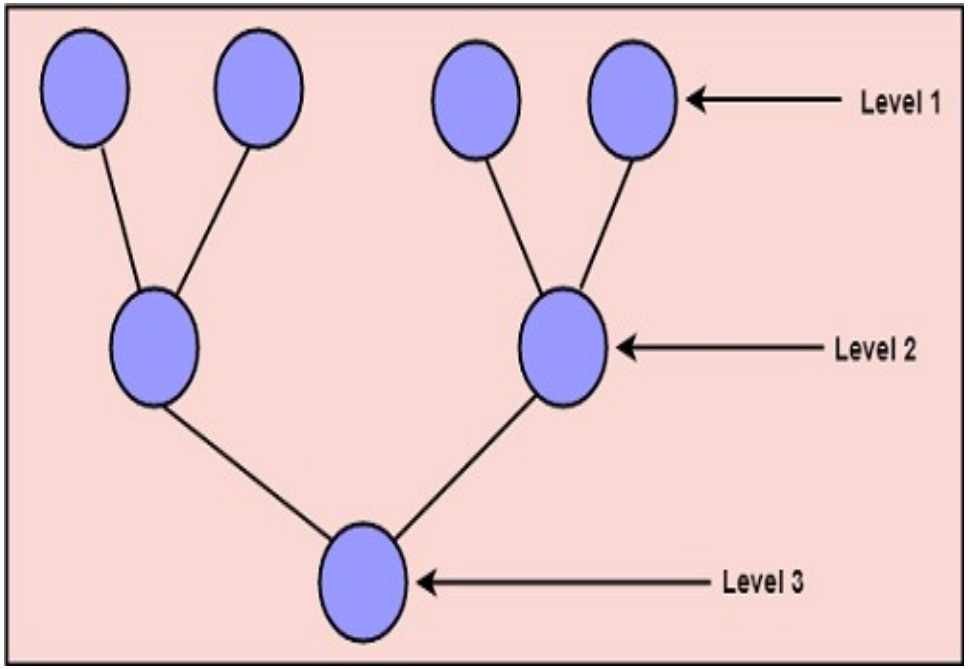
1. Abstraction
2. Architecture
3. Patterns
4. Separation of concerns
5. Modularity
6. Information hiding
7. Functional independence
8. Refinement
9. Refactoring
10. Object Oriented Design Concepts
11. Design Classes

Design Concepts - Refinement:

- **Top-down design approach:** starts with the **identification of the main components and then decomposing them into their more detailed sub-components.**



- **Bottom-up Approach:** A bottom-up approach **begins with the lower details and moves towards up the hierarchy.**



Design Concepts - Refinement:

- Refinement is a **top-down design approach**.
- A **hierarchy is established** by decomposing a statement of function in a stepwise (detailed) manner till the programming language statement are reached.
- It is a process of **elaboration.**
- Refinement is very necessary to find out any error if present and then to reduce it.
- **Abstraction** and **refinement** are **complementary** concepts**.**
  - Abstraction **enables to** specify procedure and data internally but **suppress** the need for "outsiders" to have knowledge of low-level details.
  - Refinement **helps to reveal low-level details** as **design progresses.**
  - Both concepts allows to create a complete design model as the design evolves.

Design Concepts - Refactoring:

```
double basePrice = anOrder.basePrice();
return (basePrice > 100)
```

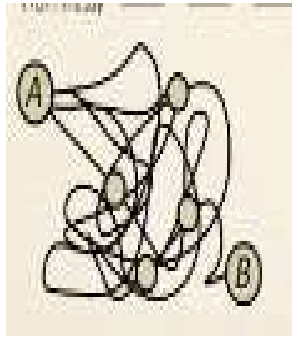
```
return (anOrder.basePrice() > 100)
```

```
double basePrice = _quantity * _itemPrice;
if (basePrice > 1000)
        return basePrice * 0.95;
else
        return basePrice * 0.98;
```

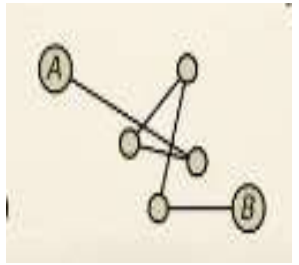
```
if (basePrice() > 1000)
        return basePrice() * 0.95;
else
        return basePrice() * 0.98;

double basePrice(){
        return _quantity * _itemPrice;
}
```

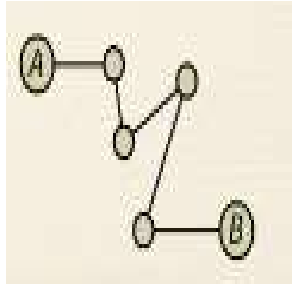
Design Concepts - Refactoring:

```
String foundPerson(String[] people){
for (int i = 0; i < people.length; i++)    {
        if (people[i]. equals (“KSR”) ) { return “KSR”; }
        if (people[i]. equals (“CSE”) ) { return “CSE”; }
        }
}
```




```
String foundPerson(String[] people){
        List candidates = Array.asList(new String[] {“KSR”, “CSE”})
        for (int i = 0; i < people.length; i++)
                if (candidates[i]. contains (people[i]))
                        return people[i];
}
```





To replace complicated algorithms with clearer or simple ones

Design Concepts - Refactoring

VIT-AP UNIVERSITY

- A **reorganization technique** which simplifies or reduces the design of components **without changing its function behavior.**

- The **process of changing the software system** in a way that it **does not change the external behavior** of the code still **improves its internal structure**.

- **During the refactoring process,** the existing design is checked for any type of **flaws like redundancy, poorly constructed algorithms and data structures, etc., in order to improve the design.**

Design Concepts - Object-Oriented Design Concepts

VIT-AP UNIVERSITY

- The Object-Oriented (OO) paradigm is widely used in modern software engineering.
- OO design concepts such as such as classes, objects, polymorphism, encapsulation, inheritance, dynamic binding, information hiding, interface, constructor, destructor. among others are used in the design methods.
- Most of the languages like C++, Java, .net are use object oriented design concept.
- The main advantage of object oriented design is that improving the software development and maintainability, faster and low cost development, and creates a high quality software.
- The disadvantage of the object-oriented design is that larger program size and it is not suitable for all types of program.

Design Concepts – Design Classes:

The model of software is defined as a **set of design classes.**

- Every class describes the **elements of problem domain** and that **focus on features of the problem which are user visible.**
- **Five different types** of design classes and **each type represents the layer of the design architecture**, which are as follows:

1**. User interface classes**
    - Designed for Human Computer Interaction(HCI).
    - Define all abstraction which is required for HCI.

2. **Business domain classes**
    - **Refinements** of the **analysis classes**.
    - **Identify attributes and methods** which are required to implement the elements of the business domain.

Design Concepts – Design Classes:

VIT-AP UNIVERSITY

3. Process classes

- **Implements the lower level business abstractions** needed to **completely manage the business domain class.**

4. Persistence classes

- **Represent data stores** (data bases) that will **persist beyond** the execution of the software.

5. System Classes

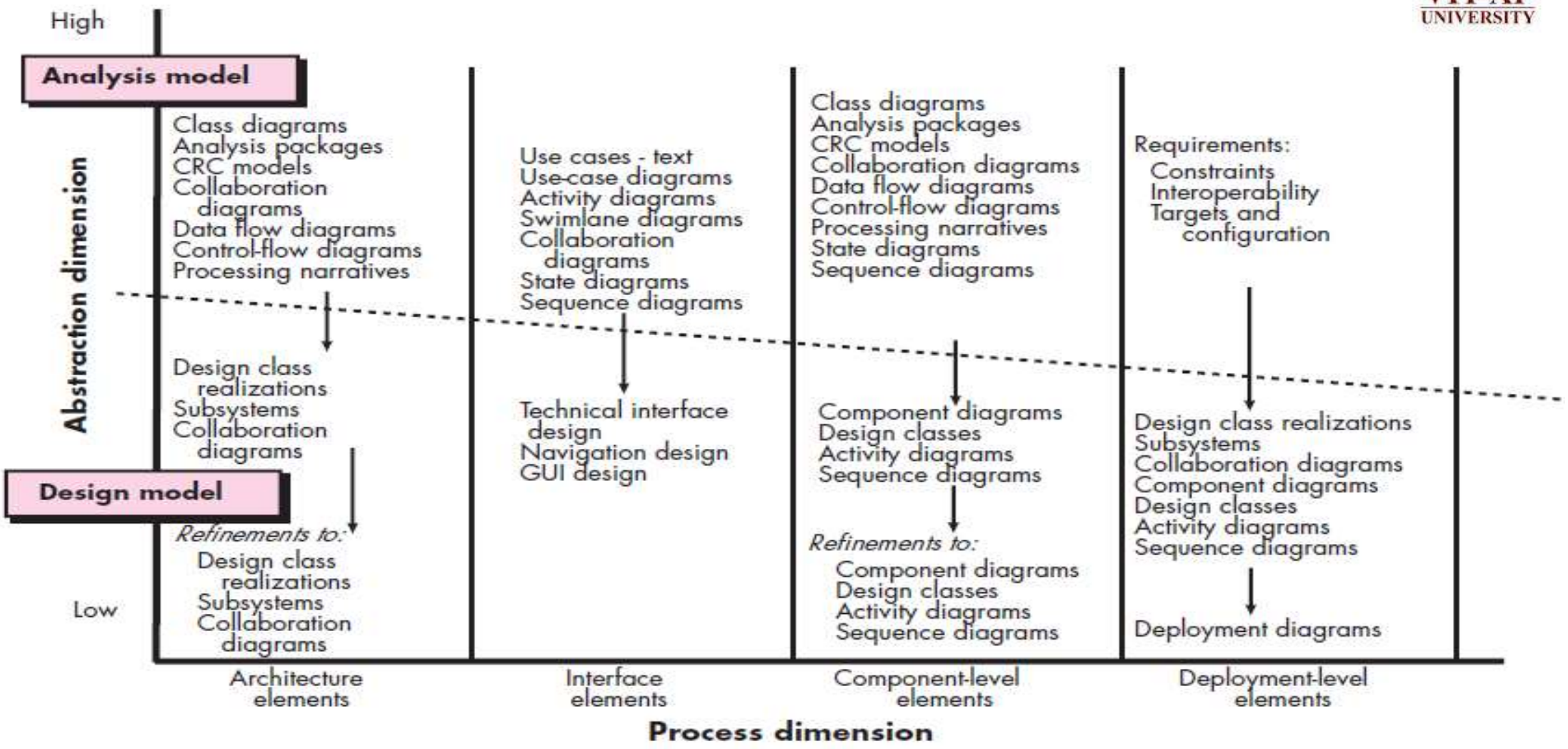
- **Implement software management and control functions** that **allow to operate and communicate** within its computing environment and the outside world.

Design Concepts – Design Classes - Characteristics:

VIT-AP UNIVERSITY

1. **Complete and sufficient**
   - A design class must include all attributes & methods, and sufficient which are required to exist for the class.
2. **Primitiveness :** Each class method focuses on providing one service
3. **High cohesion**
   - A **small and focused set of responsibilities**, for **implementing** these responsibilities the design classes are applied **single-mindedly to the methods and attribute.**
4. **Low-coupling**
   - The minimum acceptable of collaboration must be kept in the model (Called as **Law of Demeter** - a method should only send messages to methods in neighboring classes or **"Each unit should only talk to its friends; Don't talk to strangers"**)
   - If a design model is highly coupled then the system is difficult to implement, to test and to maintain over time.

Design Model: Dimensions of the design model



The Design Model:

Two different dimensions of Design Model:

- The ***process dimension : (Horizontally)***
  - It indicates the **evolution of the parts of the design model** as each design task is executed as part of the software process.

- The ***abstraction dimension : (Vertically)***
  - It represents the **level of detail as each element of the analysis model (requirements model)** is **transformed into the design model** and then iteratively refined.

The Design Model:

VIT-AP UNIVERSITY

- 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, and
  - Architectural structure and style, components that reside within the architecture, and
  - Interfaces between the components and with the outside world are all emphasized.

The design model has four major elements:

1. Data
2. Architecture,
3. Components, and
4. Interface

The Design Model - Data Design Elements

Data design (or data architecting)

- Creates a model of data and / or information that is represented at a high level of abstraction and their interrelationship.
- In many software applications, the **architecture of the data** will have a profound **influence on the architecture of the software** that must process it.

- **The structure of data can be viewed at three levels:**
  - **Program Component Level:** The **design of data structure & algorithms.**
  - **Application Level:** **Translate data model into a database**, so that the specific business objectives of a system could be achieved
  - **Business Level:** The collection of information stored in different **databases should be reorganized into data warehouse,** which enables **data mining** that has an influential impact on the business.

The Design Model - Architectural Design Elements

Describes the software's **top-level structure and identifies its components**
(Ex. Floor Plan of house, that depicts the overall layout of the rooms; their size, shape, and relationship to one another; and doors and windows that allow movement in and out of the rooms.)

- IEEE defines architectural design as **'the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.'**
- The architectural design **(system design)** element is represented as a **set of interconnected subsystem** that are derived from **analysis packages in the requirement model.**
- The architectural model is derived from **three sources:**
  1. The information about the application domain to built the software.
  2. Requirement model elements like data flow diagram or analysis classes, relationship and collaboration between them.
  3. The availability of architectural styles and patterns

The Design Model - Architectural Design Elements

- Functions performed by an architectural design:
  - It **defines an abstraction level** at which the designers can specify the **functional and performance behavior of the system.**
  - It acts as a **guideline for enhancing the system** (when ever required) by describing those features of the system that can be modified easily without affecting the system integrity.
  - It **evaluates** all top-level designs.
  - It **develops and documents** top-level design for the external and internal interfaces.
  - It **develops preliminary versions** of user documentation.
  - It **defines and documents** preliminary test requirements and the schedule for software integration.

The Design Model - Interface Design Elements

- The interface design for software is **analogous to a set of detailed drawings (and specifications)** for the doors, windows, and external utilities (e.g., water, electrical, gas, telephone) tells us **how things and information flow into and out of the house and within the rooms that are part of the floor plan.**

- The interface design elements for software **represents the information flow within it and out of the system and how they communicate between the components defined as part of architecture.**

The **three important elements** of the interface design:

1. The User Interface (also called usability design)
2. The external interface to the other systems, devices, networks etc.
3. The internal interface between various components.

The Design Model - Interface Design Elements

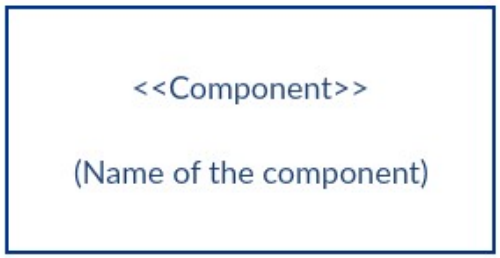
- **UI design (increasingly called usability design)** is a major software engineering action
- **Usability design incorporates**
  - Visual elements (e.g., layout, color, graphics, interaction mechanisms),
  - Ergonomic elements (e.g., information layout and placement, metaphors, UI navigation),
  - Technical elements (e.g., UI patterns, reusable components).

The Design Model – Component-Level Design Elements

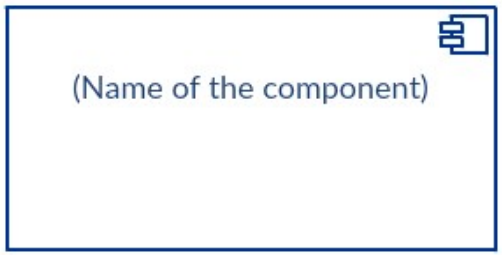
- **Describes** the **internal details of each software component** (detailed drawing and specifications of each room, wiring, place of switches…)
- The **processing of data structure** occurs in a **component** and **an interface** that **allows** all the component operations.
- **UML Component Diagram**
  - Shows the **relationship between different components** in a system.
  - Benefits of Component
    - Imagine the **system's physical structure**.
    - Pay attention to the system's components and **how they relate**.
    - Emphasize the **service behavior as it relates to the interface**.

The Design Model – Component-Level Design Elements

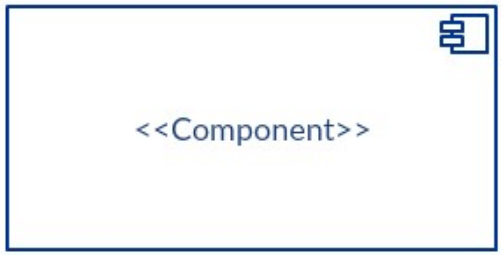
Component Symbol Notation: Three ways



Rectangle with the component stereotype (the text <<component>>)



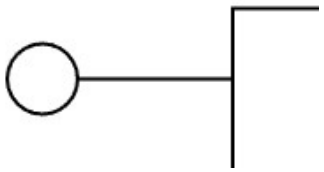
Rectangle with the component icon in the top right corner and the name of the component.



Rectangle with the component icon and the component stereotype

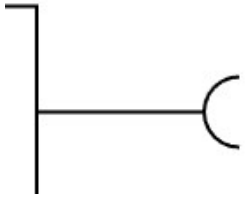
The Design Model – Component-Level Design Elements

- **Provided Interface and the Required Interface**



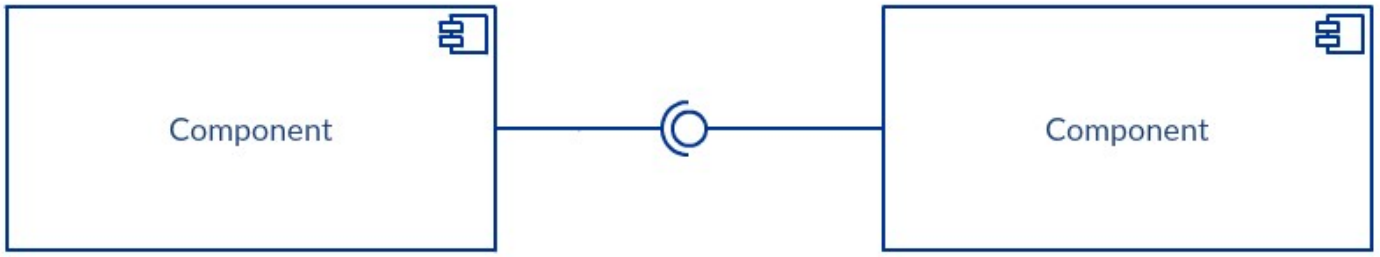
Provided interfaces:

- A straight line from the component box with an **attached circle**.
- These symbols represent the interfaces where a **component produces information used by the required interface of another component**.



Required interfaces:

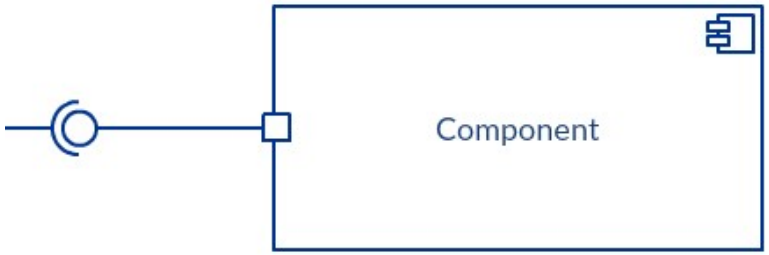
- A straight line from the component box with an **attached half circle (also represented as a dashed arrow with an open arrow).**
- These symbols represent the interfaces **where a component requires information in order to perform its proper function.**



one component is providing the service that the other is requiring

The Design Model – Component-Level Design Elements

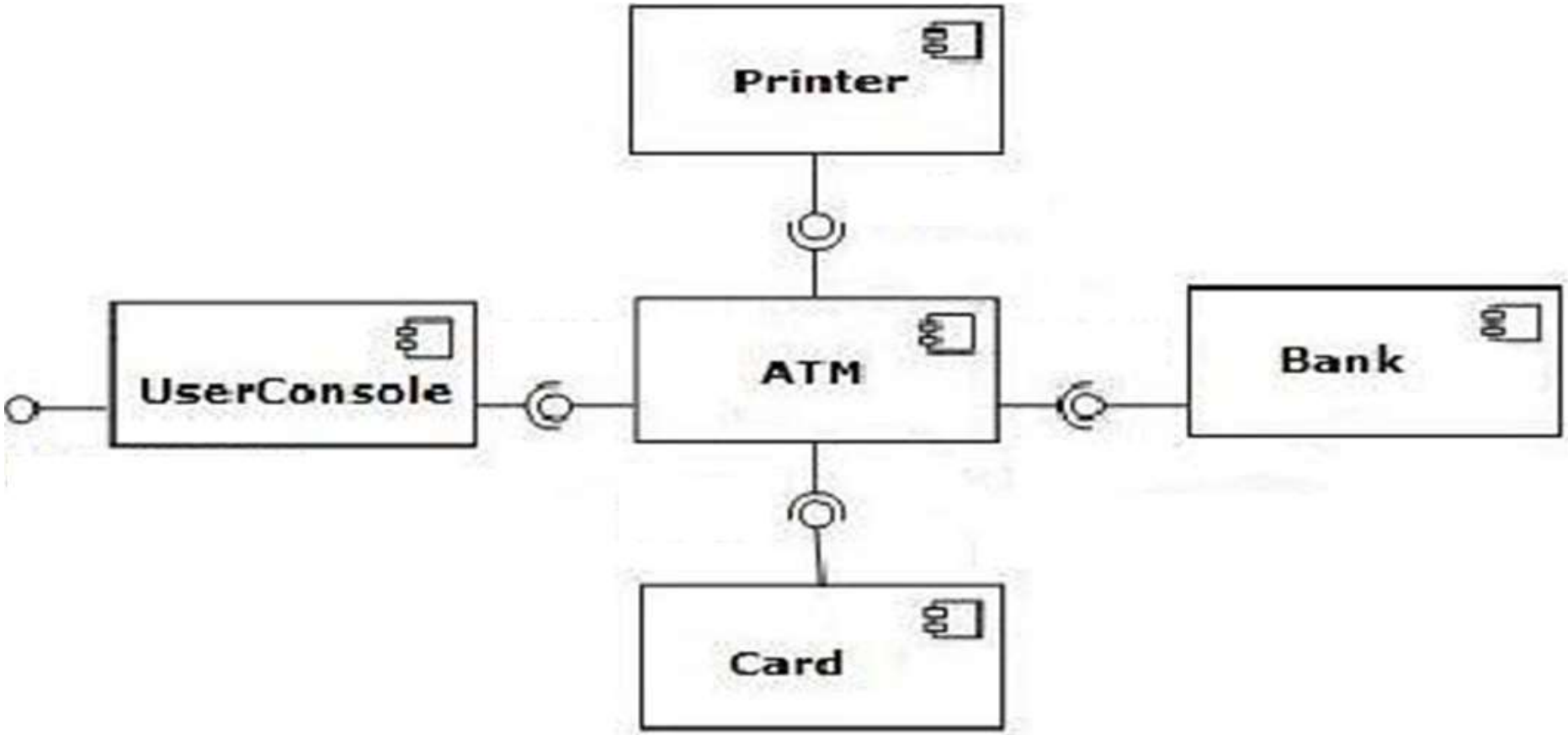
- **Port**



- Port (represented by the **small square at the end of a required interface or provided interface**) is used when the component **delegates the interfaces to an internal class**.
- **Dependencies**

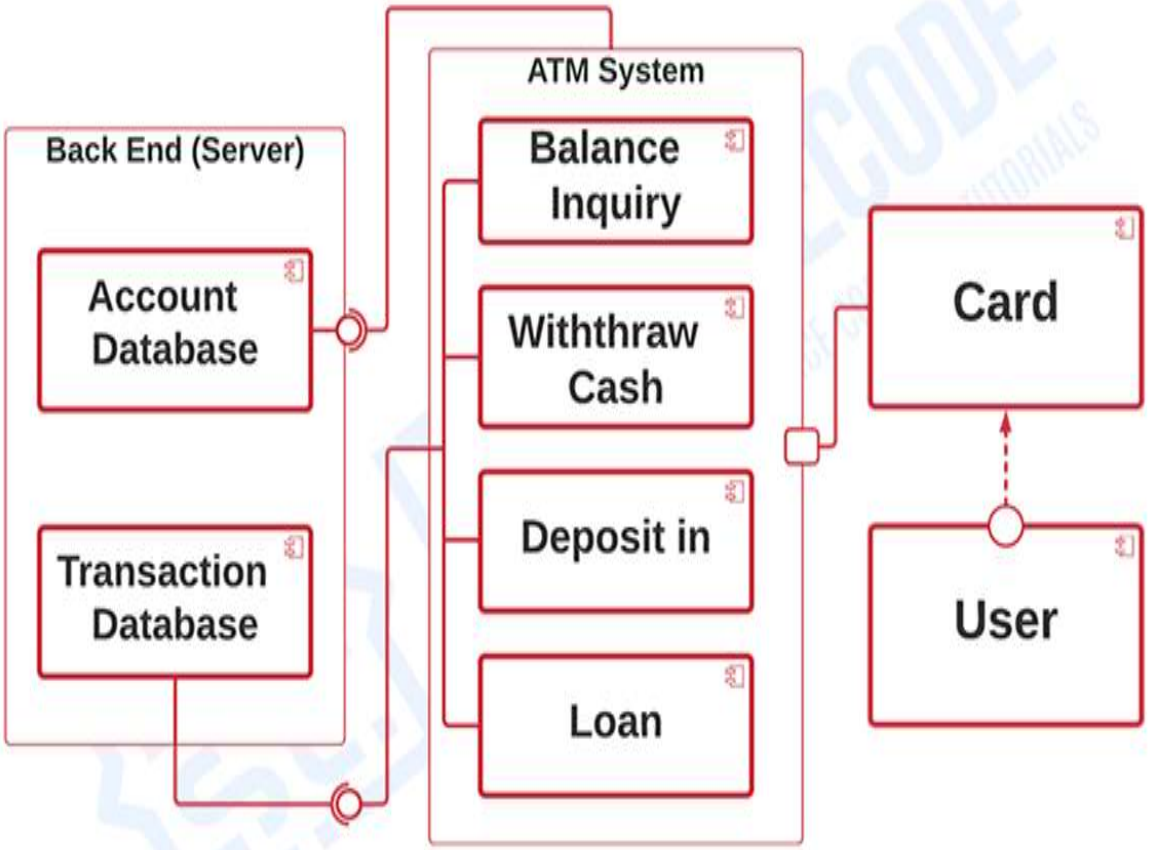
The Design Model – Component-Level Design Elements

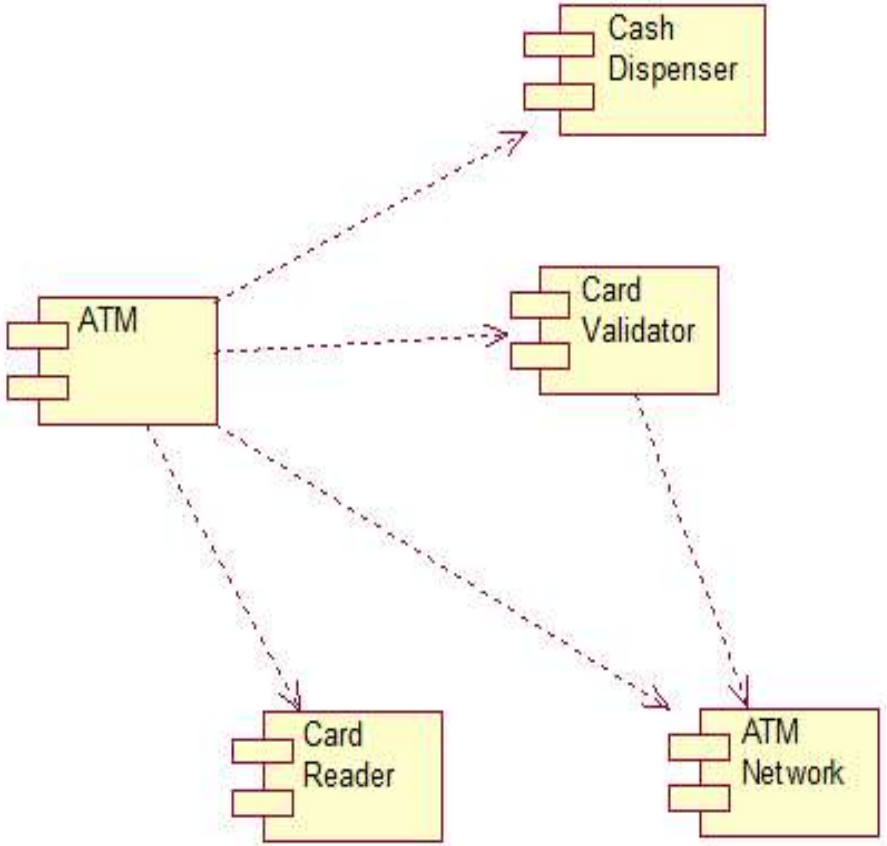
ATM – Component Diagram



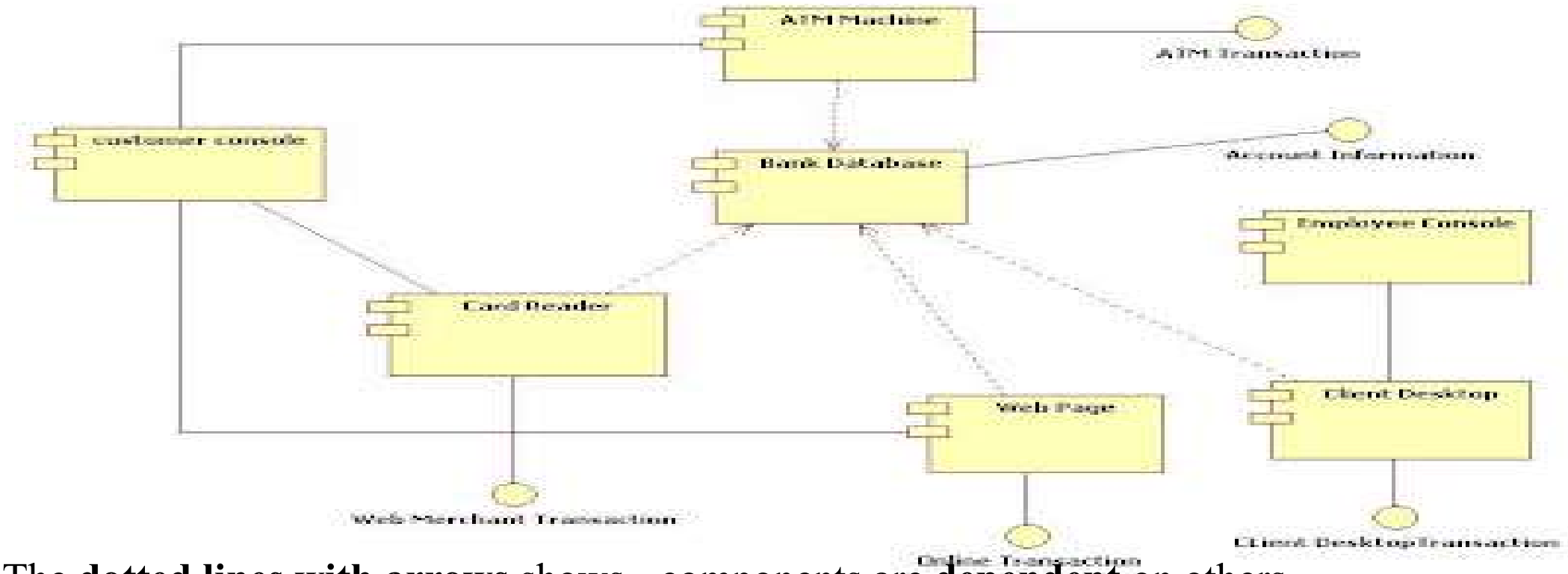
The Design Model – Component-Level Design Elements

ATM – Component Diagram



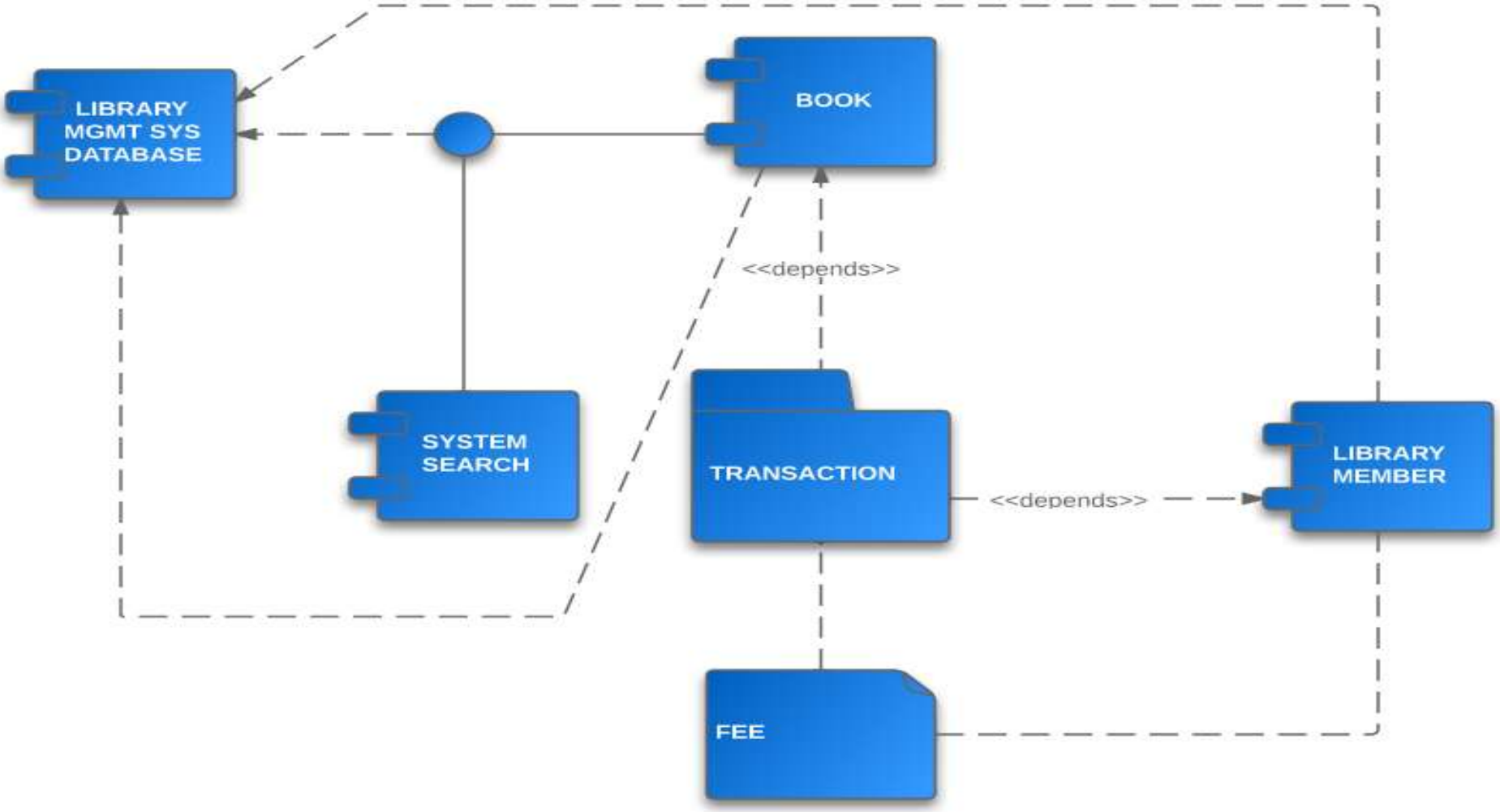


The Design Model – Component-Level Design Elements
ATM – Component Diagram



The **dotted lines with arrows** shows - components are **dependent** on others.
Example: card reader, web page, client desktop, and ATM system are all dependent on the bank database.
The **dotted lines with circles** at the end, known as **"lollipop" symbols**, indicate a **realization** relationship.

The Design Model – Component-Level Design Elements

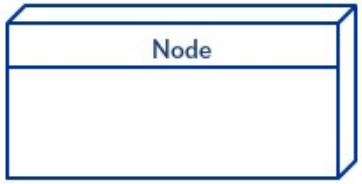


Component diagram for a library management system

The Design Model – Deployment level design elements

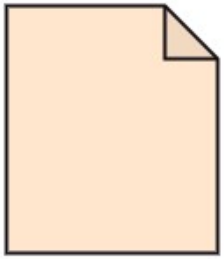
- The **deployment level design (Deployment Diagram)** element shows the **software functionality** and **subsystem allocated** in the **physical computing environment (hardware)** which supports the software.
- Using it one can understand **how the system will be physically deployed on the hardware.**
- It consists of **nodes** such as the software itself, the users' devices, and their connections.
- The deployment diagram shows **the computing environment** but does **not explicitly indicate configuration details.**

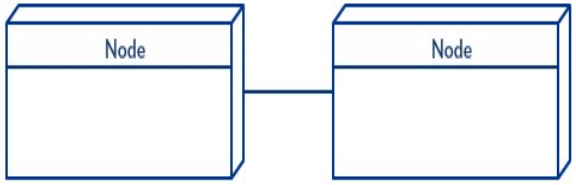
The Design Model – Deployment level design elements



Nodes represented as a cube, is a **physical entity that executes one or more components, subsystems or executables**.
A node could be a hardware or software element.

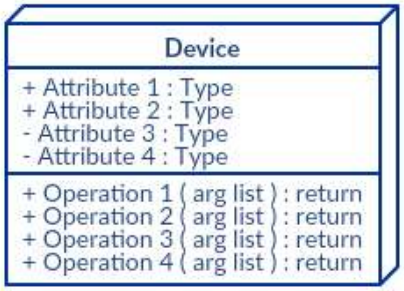
Artifacts are concrete elements that are **caused by a development process.**
Examples: Libraries, archives, configuration files, executable files etc.



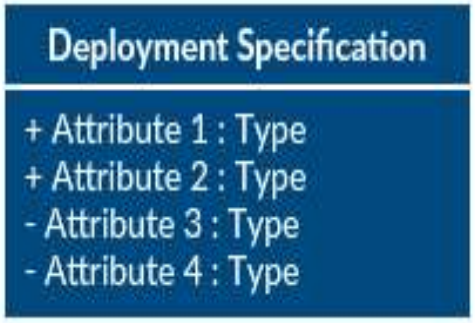


Path of communication between nodes represented by a **solid line** between two nodes.

The Design Model – Deployment level design elements

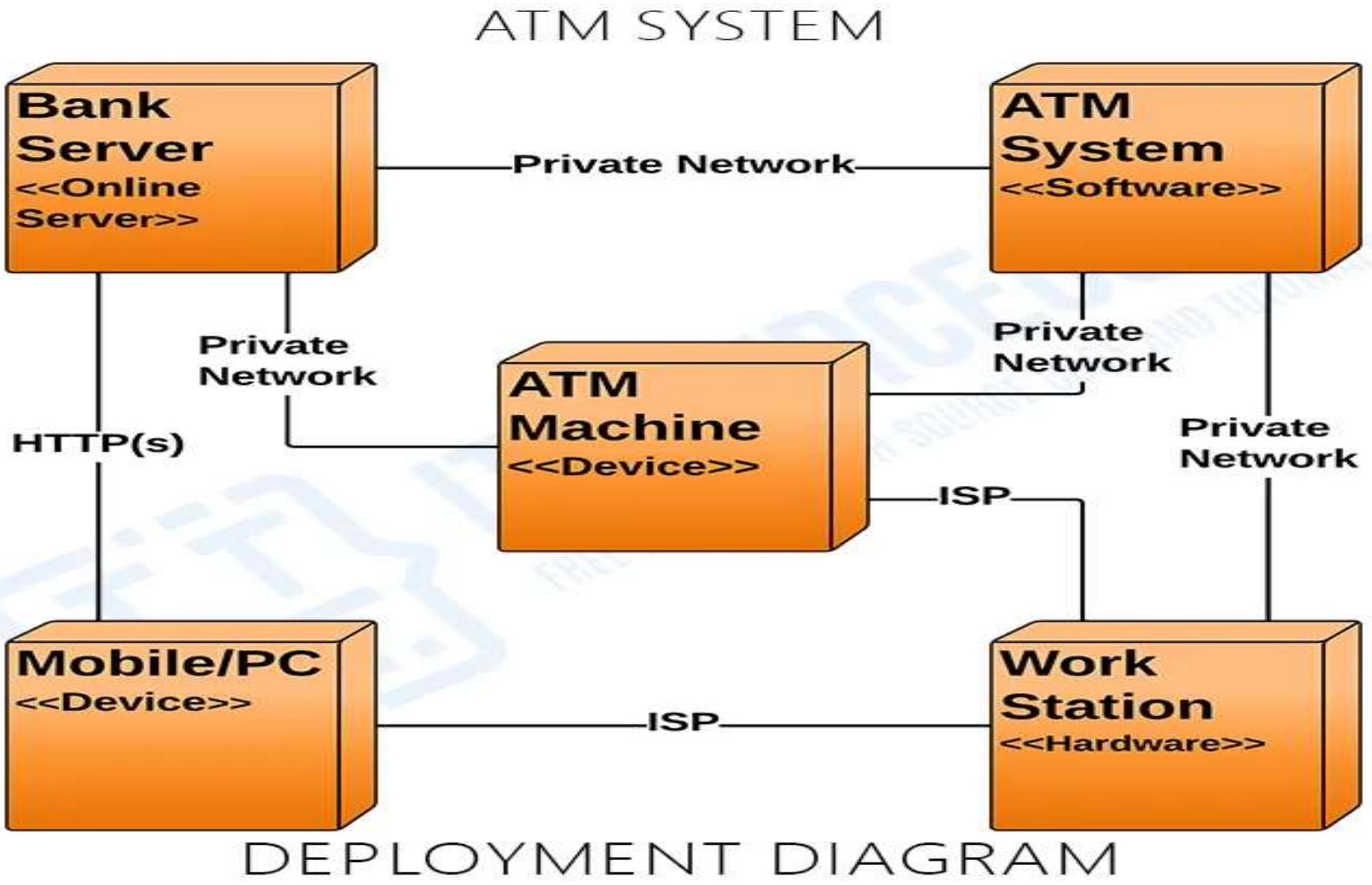


Devices: A device is a node that is used to represent a **physical computational resource in a system.**
Example: An application server.



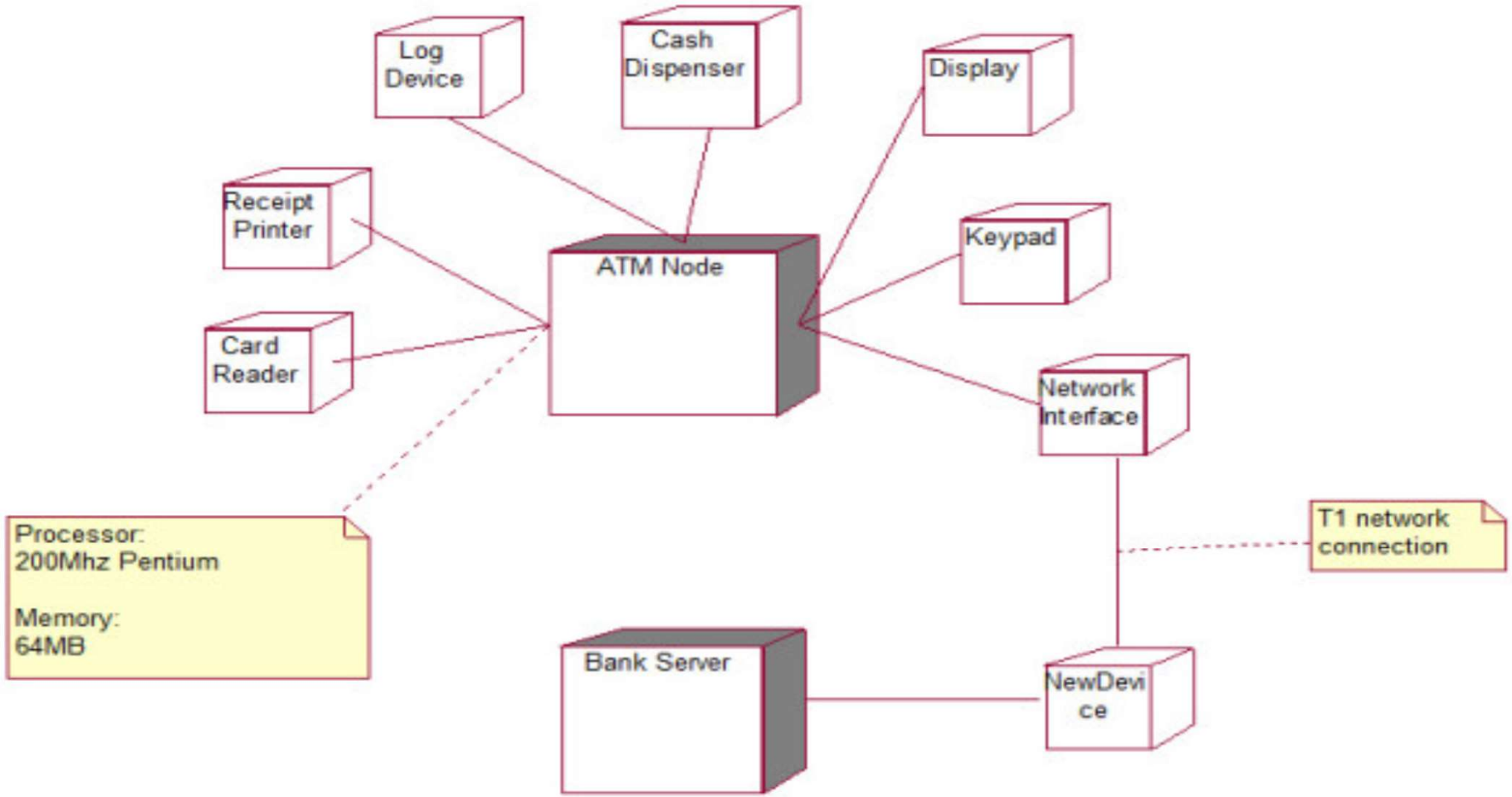
Deployment Specifications: A configuration file, such as a text file or an XML document.
It describes **how an artifact is deployed on a node.**

The Design Model – Deployment level design elements



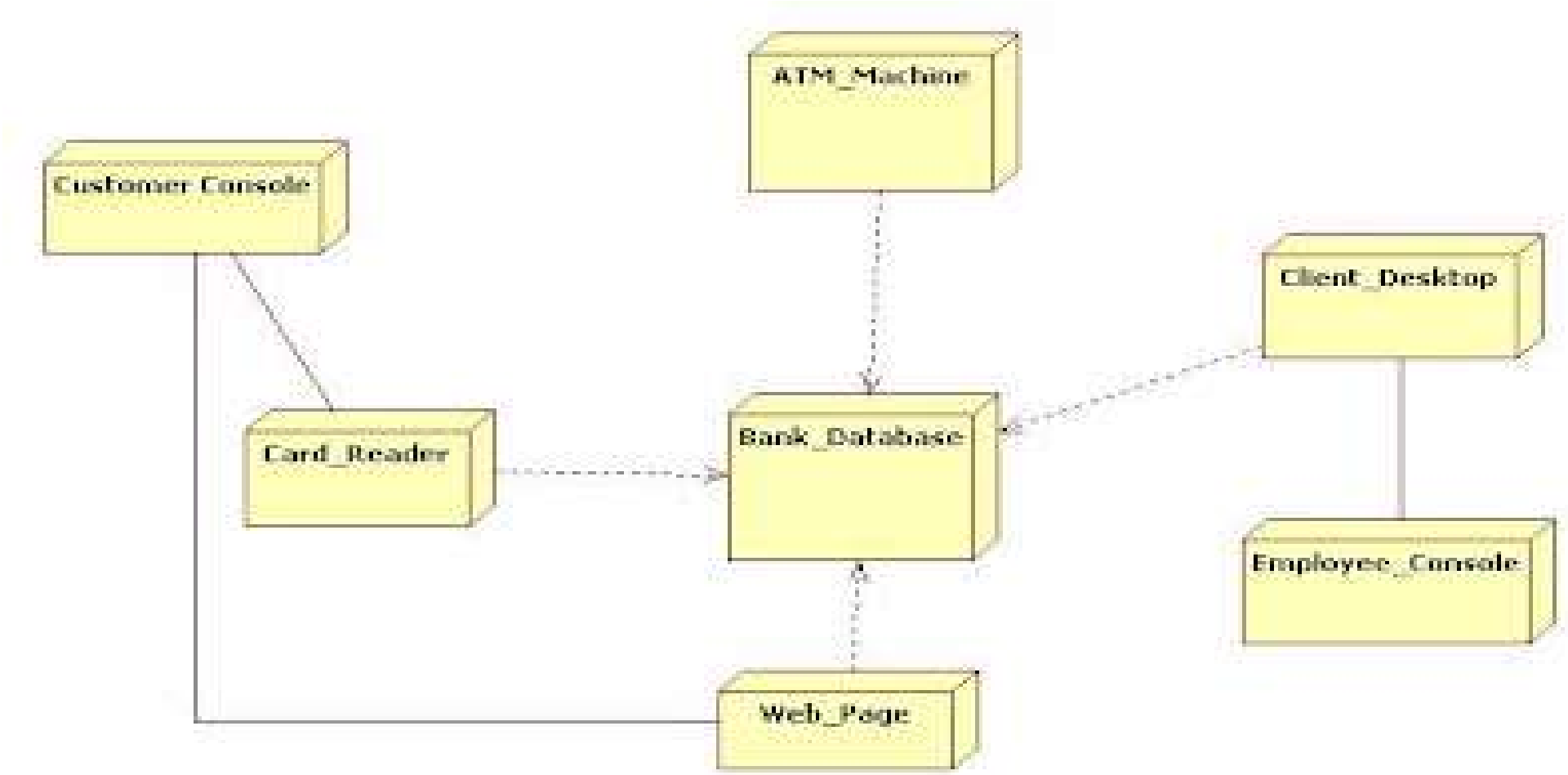
The Design Model – Deployment level design elements

ATM – Deployment Diagram

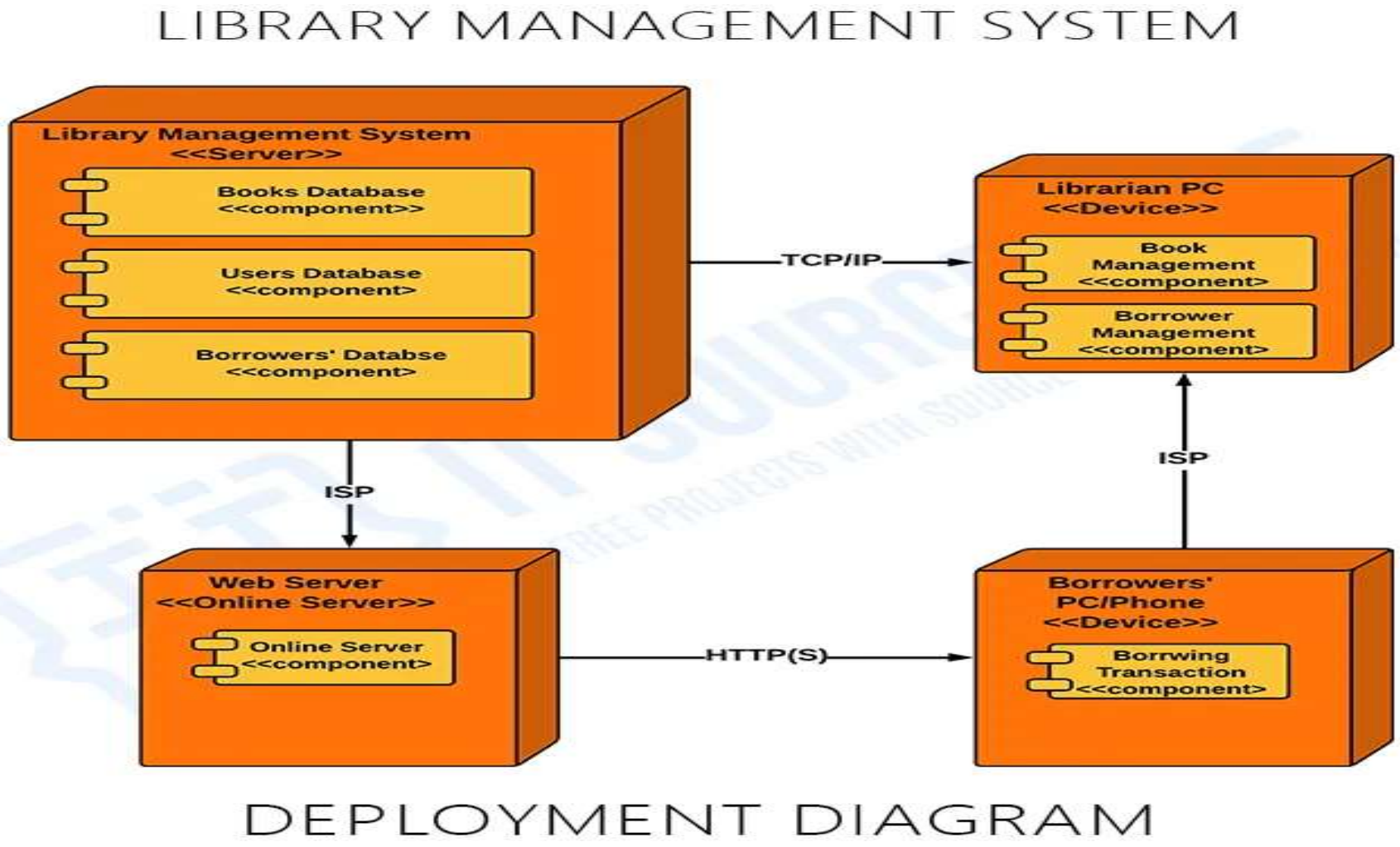


The Design Model – Deployment level design elements

ATM – Deployment Diagram



The Design Model – Deployment level design elements



The Design Model – Deployment level design elements

- **Artifact** : A rectangle with the name and term "artifact" enclosed by two arrows represents a software-created product.
- **Association** : A message or other sort of communication between nodes is indicated by a line.
- **Component** : A rectangle with two tabs that indicate a software part is called a component.
- **Dependency** : A dashed line that ends in an arrow denotes the dependency of one node or component on another.
- **Interface** : A contract relationship is indicated by a circle. Those items that realize the interface are required to fulfill some sort of task.
- **Node** : A three-dimensional box represents a hardware or software object.
- **Node as Container** : This is a node that has another node within it, such as the nodes that contain components.
- **Stereotype** : A device housed within the node, displayed at the top of the node and flanked by two arrows.

The Design Model – Deployment level design elements – Safe Home Project