Shri Vaishnav Vidyapeeth Vishwavidyalaya, Indore (M.P.)

Shri Vaishnav Institute of Information Technology

Department of Computer Science and Engineering

Lecture

on

"Architectural Design"

.

Architectural Design

- The software needs the architectural design to represents the design of software.
- 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 software that is built for computer-based systems can exhibit many architectural styles.

Architectural Design

- **Objectives:**

- ✓ To introduce architectural design and to discuss its importance
- ✓ To explain why multiple models are required to document software architecture to describe types of architectural model that may be used.
- ✓ Allow evaluation of the thing's properties before it is built
- ✓ Provides well understood tools and techniques for constructing the thing from its blueprint.

Architectural Design

- *The objective of using architectural styles is to establish a structure for all the components present in a system.*
- The software that is built for computer-based systems also exhibits one of many architectural styles.

Architectural Style

- A set of components (e.g: a database, computational modules) that will perform a function required by the system.
- The set of connectors will help in coordination, communication, and cooperation between the components.
- Conditions that how components can be integrated to form the system.
- Semantic models that help the designer to understand the overall properties of the system

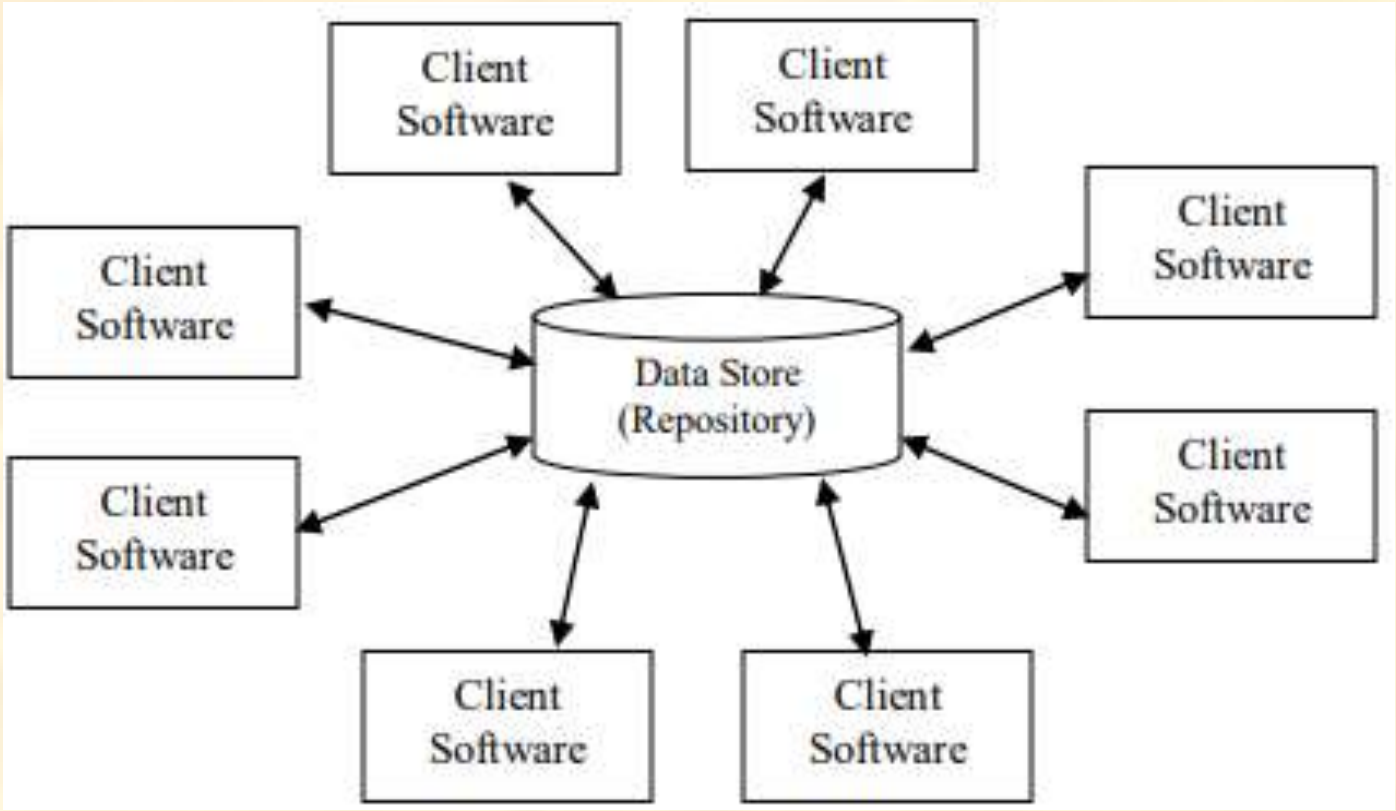
Architectural Style

- The commonly used architectural styles are:
- *Data-centered Architectures:* A data store (e.g., a file or database) resides at the center of this architecture and is accessed frequently by other components that update, add, delete, or otherwise modify data within the store.
- A typical Data-centered style.
- Clients of this are accesses a central repository.

Architectural Style

- Data-centered architectures promote integrity.
- That is, existing components can be changed and new client components can be added to the architecture without concern about other clients (because the client components operate independently).
- Client components independently execute processes.

Architectural Style



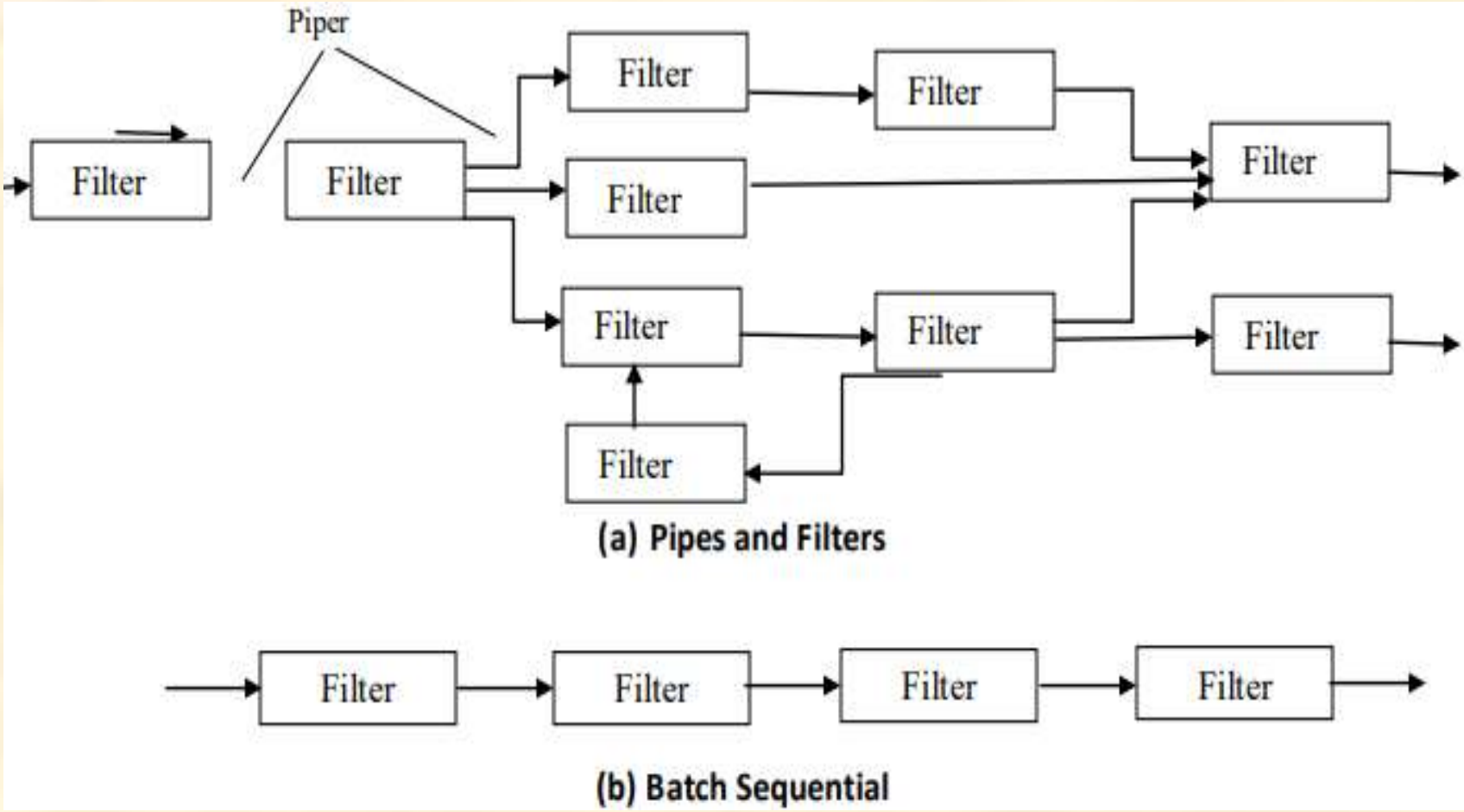
Architectural Style

- *Data-flow Architectures:* This architecture is applied when input data are to be transformed through a series of computational or manipulative components into output data.
- A pipe and filter pattern has a set of components, called filters, connected by pipes that transmit data from one component to the next.
- Components, called filters, connected by pipes that transmit data from one component to the next.

Architectural Style

- Each filter works independently of those components upstream and downstream, is designed to expect data input of a certain form, and produces data output of a specified form.
- However, the filter does not require knowledge of the working of its neighboring filters.
- If the data flow degenerates into a single line of transforms, it is termed batch sequential. This pattern accepts a batch of data and then applies a series of sequential components (filters) to transform it.

Architectural Style



(a) Pipes and Filters

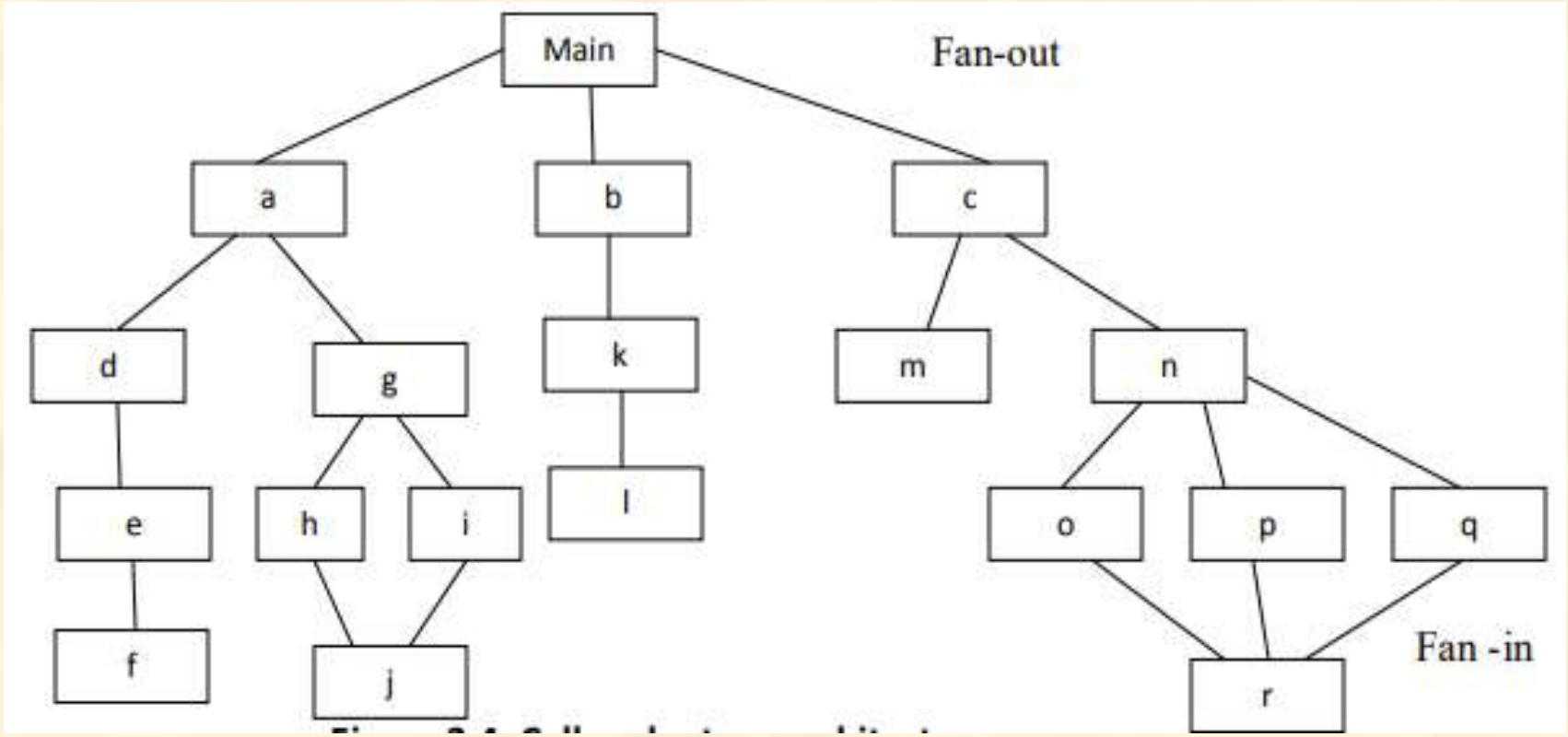
(b) Batch Sequential

Architectural Style

- *Call and return Architectures:*

- ✓ The program structure can be easily modified or scaled.
- ✓ The program structure is organized into modules within the program.
- ✓ In this architecture how modules call each other.
- ✓ The program structure decomposes the function into control hierarchy where a main program invokes number of program components.

Architectural Style



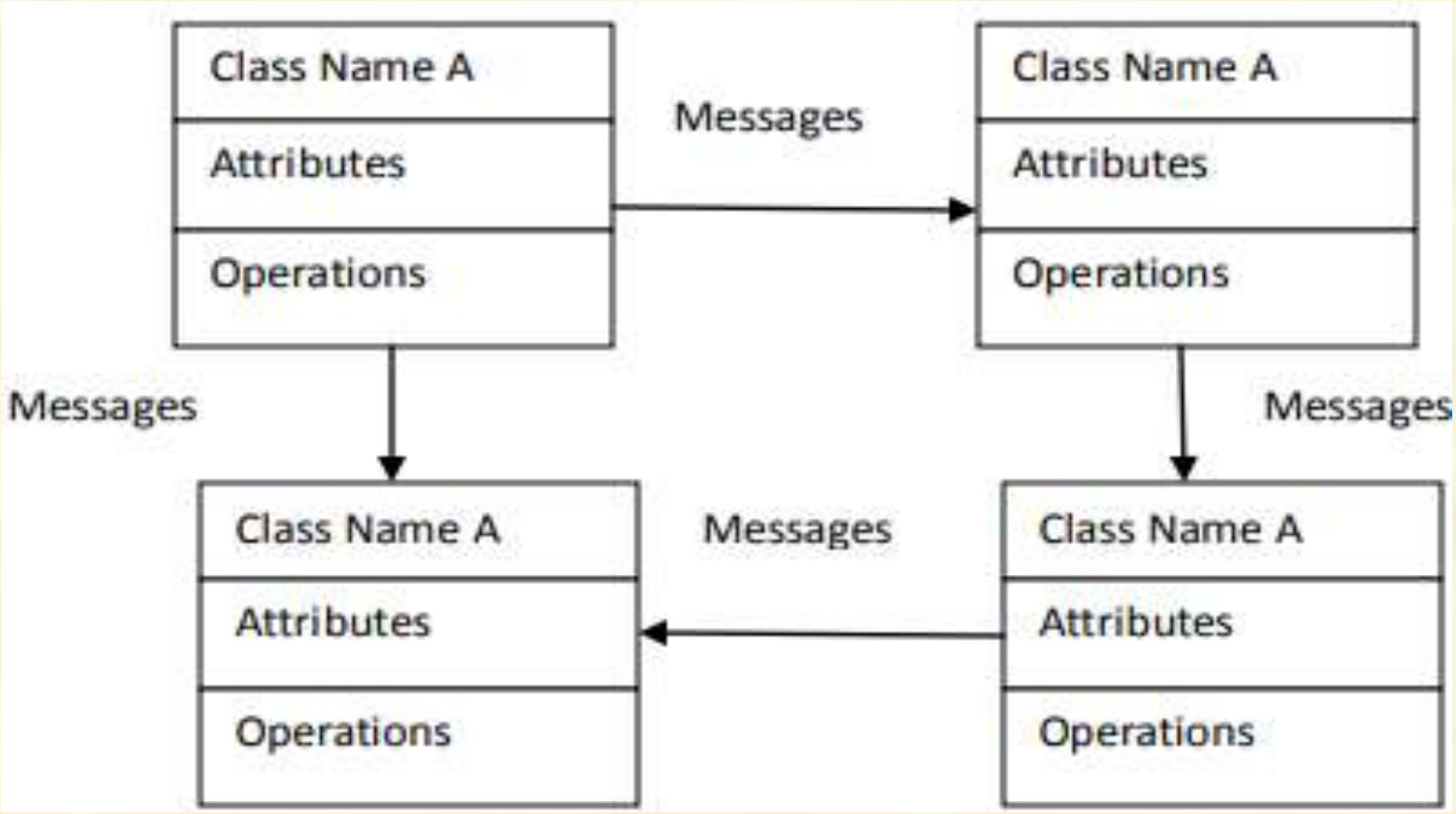
Architectural Style

- *Object-oriented Architecture:*

✓ The components of a system encapsulate data and the operations that must be applied to manipulate the data.

✓ Communication and coordination between components is accomplished via message passing

Architectural Style



Thank -You

Shri Vaishnav Vidyapeeth Vishwavidyalaya, Indore (M.P.)

Shri Vaishnav Institute of Information Technology

Department of Computer Science and Engineering

Lecture

on

"Architectural Views"

.

Architectural Views

- Architecture views are representations of the overall architecture that are meaningful to one or more stakeholders in the system.
- The architect chooses and develops a set of views that will enable the architecture to be communicated to, and understood by, all the stakeholders, and enable them to verify that the system will address their concerns

Architectural Views

- Each architectural model only shows one view or perspective of the system.
- It might show how a system is decomposed into modules, how the run-time processes interact or the different ways in which system components are distributed across a network.
- The views are used to describe the system how the viewpoint of different stakeholders, such as end-users, developers and project managers.

Architectural Views

- The four views of the model are logical, development, process and physical view
- In addition, selected use cases or scenarios are used to illustrate the architecture serving as the 'plus one' view. Hence the model contains **4+1** views:

Architectural Views

- **Logical View:**

- ✓ The logical view is concerned with the functionality that the system provides to end-users.
- ✓ UML diagrams used to represent the logical view include, class diagrams, and state diagrams.

Architectural Views

- **Development View:**

- ✓ The development view illustrates a system from a programmer's perspective and is concerned with software management.
- ✓ This view is also known as the implementation view.
- ✓ It uses the UML Component diagram to describe system components.
- ✓ UML Diagrams used to represent the development view include the Package diagram

Architectural Views

- **Physical View:**

- ✓ The physical view depicts the system from a system engineer's point of view.
- ✓ It shows the system hardware and how software components are distributed across the processors in the system.
- ✓ This view is also known as the deployment view.
- ✓ UML diagrams used to represent the physical view include the deployment diagram.

Architectural Views

- **Process View:**

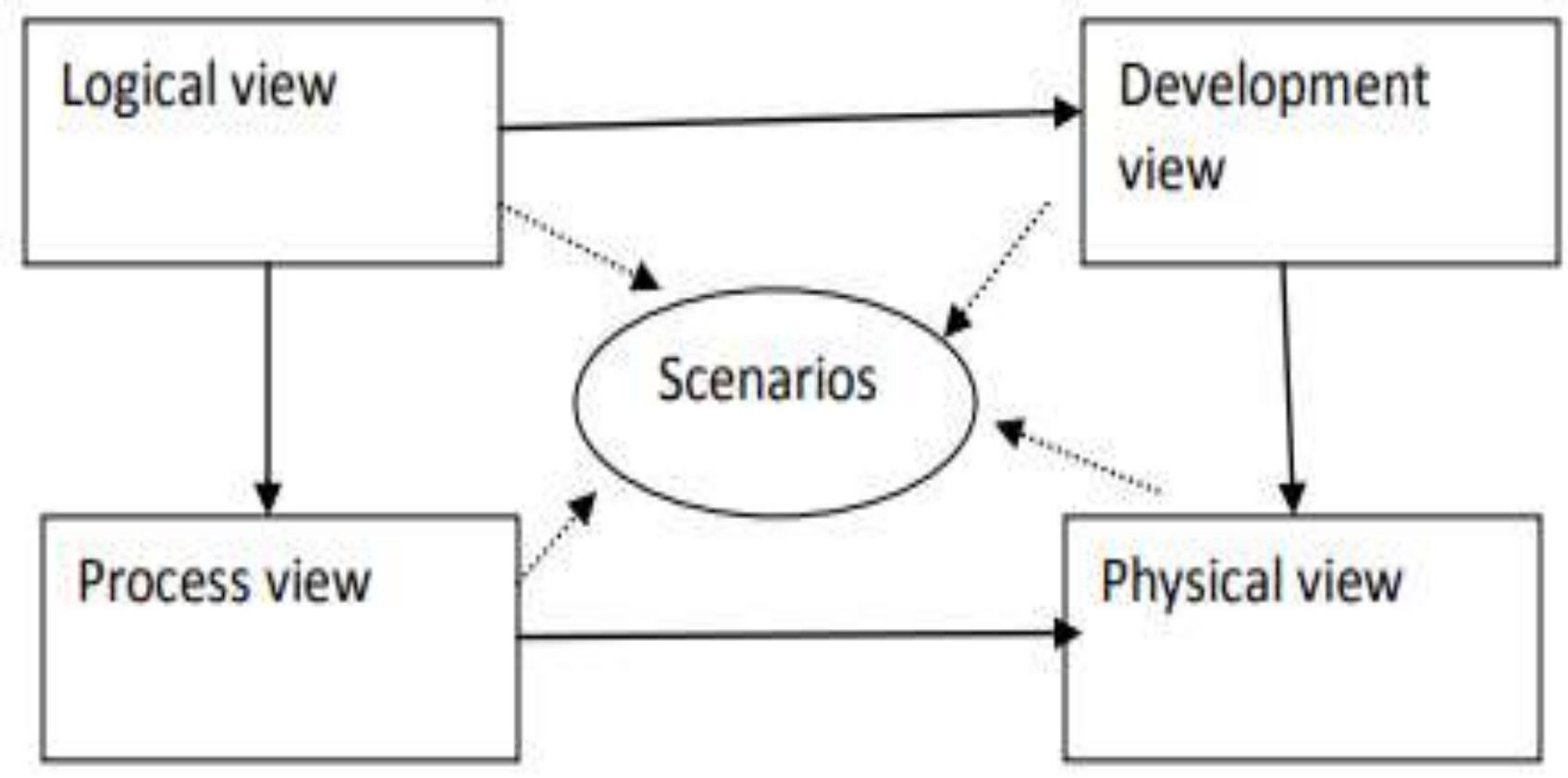
- ✓ The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the runtime behavior of the system.
- ✓ The process view addresses concurrency, distribution, integrators, performance, and scalability, etc.
- ✓ UML diagrams to represent process view include the activity diagram

Architectural Views

- **Scenarios:**

- ✓ The description of architecture is illustrated using a small set of use cases, or scenarios, which become a fifth view.
- ✓ The scenarios describe sequences of interactions between objects and between processes.
- ✓ They are used to identify architectural elements and to illustrate and validate the architecture design.
- ✓ They also serve as a starting point for tests of an architecture prototype.
- ✓ This view is also known as the use case view

Architectural Views



Thank -You

Shri Vaishnav Vidyapeeth Vishwavidyalaya, Indore (M.P.)

Shri Vaishnav Institute of Information Technology

Department of Computer Science and Engineering

Lecture

on

"Design Metrics"

.

Design Metrics

- A software metric is a measure of software characteristics which are measurable or countable.
- Software metrics are valuable for many reasons, including measuring software performance, planning work items, measuring productivity, and many other uses.

Design Metrics

- Software metrics can be classified into three categories −

1) *Product metrics* − Describes the characteristics of the product such as size, complexity, design features, performance, and quality level.
2) *Process metrics* − These characteristics can be used to improve the development and maintenance activities of the software.

Design Metrics

3) *Project metrics* – This metrics describe the project characteristics and execution.

- ✓ Note that as the project proceeds, the project manager will check its progress from time-to-time and will compare the effort, cost, and time with the original effort, cost and time.
- ✓ Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity.

Process Metrics

- To measure the efficiency and effectiveness of the software process, a set of metrics is formulated based on the outcomes derived from the process. These outcomes are listed below

- ✓ Number of errors found before the software release
- ✓ Defect detected and reported by the user after delivery of the software
- ✓ Time spent in fixing errors
- ✓ Work products delivered
- ✓ Human effort used
- ✓ Estimated cost compared to actual cost.

Product Metrics

- ✓ In software development process, a working product is developed at the end of each successful phase.
- ✓ Each product can be measured at any stage of its development.
- ✓ Metrics are developed for these products so that they can indicate whether a product is developed according to the user requirements.
- ✓ If a product does not meet user requirements, then the necessary actions are taken in the respective phase.

Design Metrics

- In addition, product metrics assess the internal product attributes in order to know the efficiency of the following.

✓ Analysis, design, and code model

✓ Potency of test cases

✓ Overall quality of the software under development.

Project Metrics

- Project metrics enable the project managers to assess current projects, track potential risks, identify problem areas, adjust workflow, and evaluate the project team's ability to control the quality of work products.
- Project metrics serve two purposes. One, they help to minimize the development schedule by making necessary adjustments in order to avoid delays and alleviate potential risks and problems.
- Two, these metrics are used to assess the product quality on a regular basis-and modify the technical issues if required

Cyclomatic Complexity

- Cyclomatic complexity is a software metric used to measure the complexity of a program.
- Thomas J. McCabe developed this metric in 1976.
- McCabe interprets a computer program as a set of a strongly connected directed graph.
- Nodes represent parts of the source code having no branches and arcs represent possible control flow transfers during program execution

Cyclomatic Complexity

- It is a software metric used to indicate the complexity of a program.
- It is computed using the Control Flow Graph of the programCyclomatic
- Finally, Complexity may be defined as-

✓ It is a software metric that measures the logical complexity of the program code.

✓ It counts the number of decisions in the given program code.

✓ It measures the number of linearly independent paths through the program code.

Cyclomatic Complexity

Cyclomatic Complexity	Meaning
1 – 10	• Structured and Well Written Code • High Testability • Less Cost and Effort
10 – 20	• Complex Code • Medium Testability • Medium Cost and Effort
20 – 40	• Very Complex Code • Low Testability • High Cost and Effort
> 40	• Highly Complex Code • Not at all Testable • Very High Cost and Effort

Cyclomatic Complexity

- Calculating Cyclomatic Complexity-

- ✓ Cyclomatic complexity is calculated using the control flow representation of the program code.
- ✓ In control flow representation of the program code,
- ✓ Nodes represent parts of the code having no branches.
- ✓ Edges represent possible control flow transfers during program execution

Cyclomatic Complexity

- There are three methods of computing Cyclomatic complexities.
- **Method 1:** Total number of regions in the flow graph is a Cyclomatic complexity.
- **Method 2:** The Cyclomatic complexity, V (G) for a flow graph G can be defined as ***V (G) = E - N + 2***

E is total number of edges in the flow graph.,
N is the total number of nodes in the flow graph.

- **Method 3:** The Cyclomatic complexity V (G) for a flow graph G can be defined as ***V (G) = P + 1***

P is the total number of predicate nodes contained in the flow G

Cyclomatic Complexity

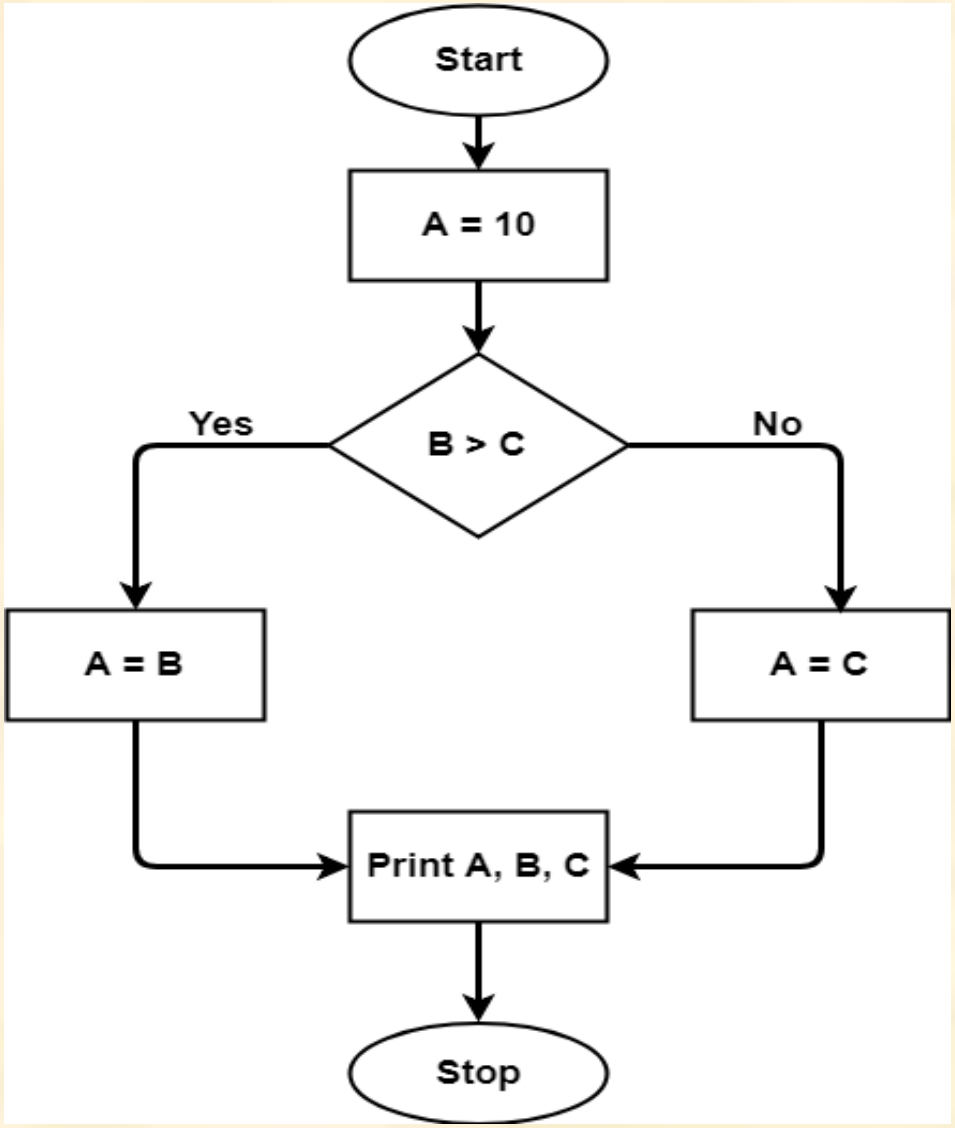
- Example:

```
{
1. If (a<b)
2. F1 () ;
else{
3. If (a<c)
4. F2 () ;
else
5. F3 () ;
}
}
```

Cyclomatic Complexity

```
A = 10
IF B > C THEN
A = B
ELSE
A = C
ENDIF
Print A
Print B
Print C
```

Cyclomatic Complexity



Thank -You

Shri Vaishnav Vidyapeeth Vishwavidyalaya, Indore (M.P.)

Shri Vaishnav Institute of Information Technology

Department of Computer Science and Engineering

Lecture

on

"Function Oriented Design"

.

Function Oriented Design

- In function-oriented design, the system is comprised of many smaller sub-systems known as functions.
- These functions can perform significant task in the system.
- The system is considered as top view of all functions.
- Function oriented design inherits some properties of structured design where divide and conquer methodology is used.

Function Oriented Design

- This design mechanism divides the whole system into smaller functions, which provides means of abstraction by hiding the information and their operation.
- These functional modules can share information among themselves by means of information passing and using information available globally.

Function Oriented Design

- Another characteristic of functions is that when a program calls a function, the function changes the state of the program.
- Function oriented design works well where the system state does not matter and program/functions work on input rather than on a state.

Function Oriented Design

- ***Functional design process***:

- ✓ **data-flow design:** Model the data processing in the system using data-flow diagrams.
- ✓ **Structural decomposition:** Model how functions are decomposed to sub-functions using graphical structure charts.
- ✓ **Detailed design:** The entities in the design and their interfaces are described in detail. These may be recorded in a data dictionary and the design.

Thank -You

Shri Vaishnav Vidyapeeth Vishwavidyalaya, Indore (M.P.)

Shri Vaishnav Institute of Information Technology

Department of Computer Science and Engineering

Lecture

on

"SA/SD Component Based Design"

.

SA/SD Component Based Design,

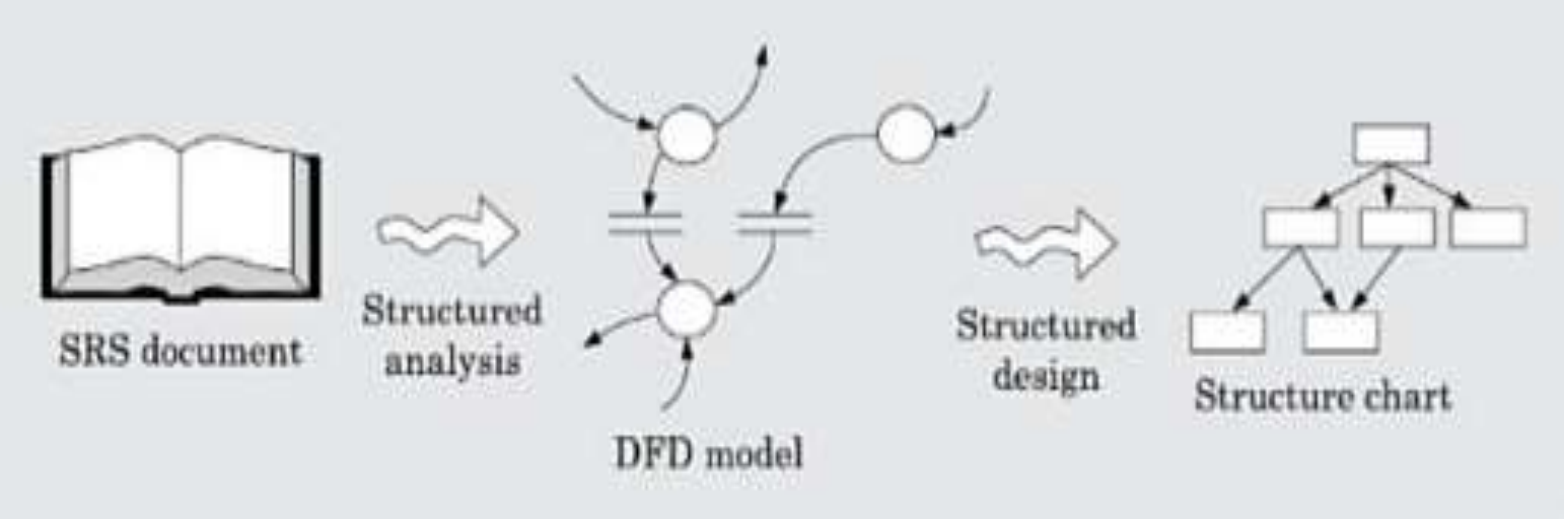
- As the name itself implies, SA/SD methodology involves carrying out two distinct activities:

✓ *Structured analysis (SA)*

✓ *Structured design (SD).*

- The roles of structured analysis (SA) and structured design (SD) have been shown schematically in below figure.

SA/SD Component Based Design,



SA/SD Component Based Design,

- Structured Analysis and Structured Design (SA/SD) is a diagrammatic notation that is designed to help people understand the system.
- The basic goal of SA/SD is to improve quality and reduce the risk of system failure.
- It establishes real management specifications and documentation.
- It focuses on the hardness, flexibility, and maintainability of the system.

SA/SD Component Based Design,

- The structured analysis activity transforms the SRS document into a graphic model called the DFD model.
- During structured analysis, functional decomposition of the system is achieved.
- The purpose of structured analysis is to capture the detailed structure of the system as supposed by the user

SA/SD Component Based Design,

- During structured design, all functions identified during structured analysis are mapped to a module structure.
- This module structure is also called the high level design or the software architecture for the given problem.
- This is represented using a structure chart.
- The purpose of structured design is to define the structure of the solution that is suitable for implementation

SA/SD Component Based Design,

- ***Goals of SA/SD***

✓ Improve Quality and reduce the risk of system failure

✓ Establish concrete requirements specifications and complete requirements documentation

✓ Focus on Reliability, Flexibility, and Maintainability of system

SA/SD Component Based Design,

- SA/SD is combined known as SAD and it mainly focuses on the following 3 points:

✓ System

✓ Process

✓ Technology

SA/SD Component Based Design

- SA/SD involves 2 phases:

✓ **Analysis Phase:** It uses Data Flow Diagram, Data Dictionary, State Transition diagram and ER diagram.

✓ **Design Phase:** It uses Structure Chart and Pseudo Code.

SA/SD Component Based Design

- Structure analysis/ structure design recursively divide complex processes into sub diagrams until many small processes are left that are easy to implement.
- When the resulting processes are simple enough, the decomposition stops, and a process specification is written for each lowest level process
- Process specification may be expressed with decision tables, pseudo code or other techniques.

SA/SD Component Based Design

- ***Analysis Phase:***

✓ Analysis Phase involves data flow diagram, data dictionary, state transition diagram, and entity-relationship diagram

- ***Design Phase:***

✓ Design Phase involves structure chart and pseudocode.

SA/SD Component Based Design

- *Data Flow Diagram:*

- ✓ A DFD model only represents the data flow aspects and does not show the sequence of execution of the different functions and the conditions based on which a function may or may not be executed.
- ✓ In the data flow diagram, the model describes how the data flows through the system.
- ✓ We can incorporate the Boolean operators and & or link data flow when more than one data flow may be input or output from a process.

SA/SD Component Based Design

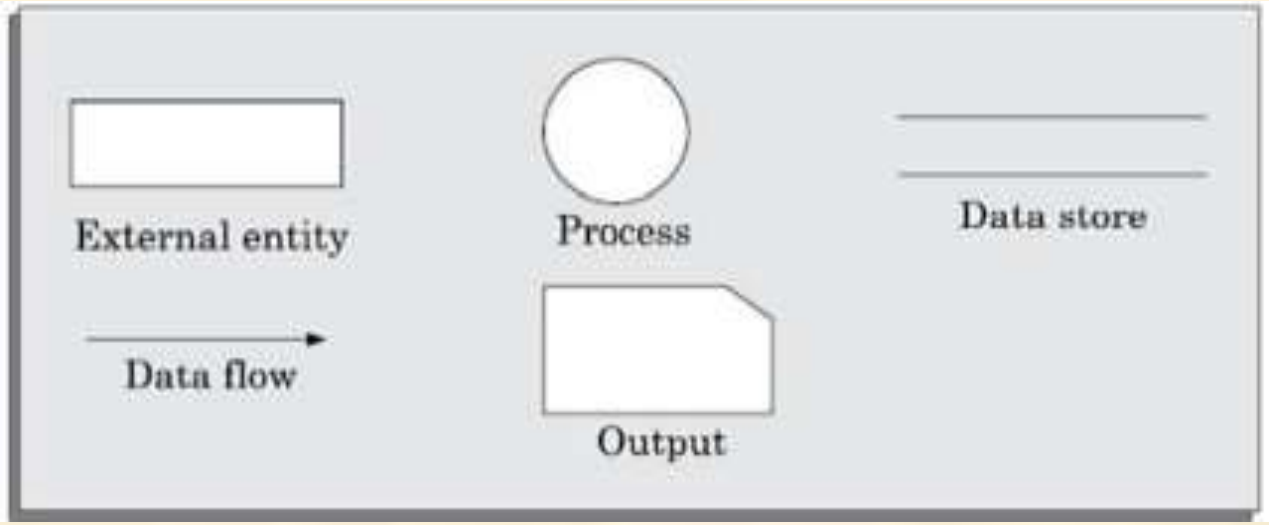
- *Data Flow Diagram:*

✓ In the DFD terminology, each function is called a process or a bubble.

✓ Each function as a processing station that consumes some input data and produces some output data

SA/SD Component Based Design

- *Primitive symbols used for constructing DFDs*



SA/SD Component Based Design

- **Function symbol:** A function is represented using a circle. This symbol is called a process or a bubble. Bubbles are annotated with the names of the corresponding functions
- **External entity symbol:** represented by a rectangle. The external entities are essentially those physical entities external to the software system which interact with the system by inputting data to the system or by consuming the data produced by the system.

SA/SD Component Based Design

- **Data flow symbol:** A directed arc (or an arrow) is used as a data flow symbol.
- A data flow symbol represents the data flow occurring between two processes or between an external entity and a process in the direction of the data flow arrow.
- **Output symbol:** The output symbol is used when a hard copy is produced.

SA/SD Component Based Design

- **Data store symbol:** A data store is represented using two parallel lines.
- ✓ It represents a logical file. That is, a data store symbol can represent either a data structure or a physical file on disk.
- ✓ Connected to a process by means of a data flow symbol.
- ✓ The direction of the data flow arrow shows whether data is being read from or written into a data store.

SA/SD Component Based Design

- **Data Dictionary:**
- The content that is not described in the DFD is described in the data dictionary.
- Every DFD model of a system must be accompanied by a data dictionary. A data dictionary lists all data items that appear in a DFD model
- It defines the data store and relevant meaning. A logical data dictionary may also be included for each such data element. All system names, whether they are names of entities, types, relations, attributes, or services, should be entered in the dictionary.

SA/SD Component Based Design

- **State Transition Diagram:**

- ✓ State transition diagram is like the dynamic model.
- ✓ It specifies how much time the function will take to execute, and data access triggered by events.
- ✓ It also describes all the states that an object can have, the events under which an object changes state and the activities were undertaken during the life of an object

SA/SD Component Based Design

- **ER Diagram:**

- ✓ ER diagram specifies the relationship between data store.
- ✓ It is basically used in database design.
- ✓ It basically describes the relationship between different entities.

SA/SD Component Based Design

- *Design Phase:*
- **Structure Chart:**

✓ The aim of structured design is to transform the results of the structured analysis into a structure chart.

✓ A structure chart represents the software architecture.

✓ The structured chart does not show the working and internal structure of the processes or modules and does not show the relationship between data or data-flows.

SA/SD Component Based Design

- **Pseudo Code:**

✓ It is the actual implementation of the system.

✓ It is an informal way of programming that doesn't require any specific programming language or technology.

Thank -You

Shri Vaishnav Vidyapeeth Vishwavidyalaya, Indore (M.P.)

Shri Vaishnav Institute of Information Technology

Department of Computer Science and Engineering

Lecture

on

"Software Design and UML"

.

Software Modeling

- Software models are ways of expressing a software design. Usually some sort of abstract language or pictures are used to express the software design.
- For object-oriented software, an object modeling language such as UML is used to develop and express the software design.

Unified Modeling Language (UML)

- Over the past decade, Grady Booch, James Rumbaugh, and Ivar Jacobson have collaborated to combine the best features of their individual object-oriented analysis and design methods into a unified method.
- The result, called the Unified Modeling Language (UML), has become widely used throughout the industry.

Software Modeling

- UML allows a software engineer to express an analysis model using a modeling notation that is governed by a set of syntactic, semantic, and programmatic rules. In UML, a system is represented using five different “views” describe the system from distinctly different perspectives.
- Each view is defined by a set of diagrams.

Software Modeling

- The following views are present in UML:
- ***User model view.*** This view represents (product) from the user's (called actors in UML) perspective.
- The use-case is the modeling approach of choice for the user model view.
- This important analysis representation describes a usage scenario from the end-user's perspective.

Software Modeling

- ***Structural model view***. Data and functionality are viewed from inside the system.
- ✓ That is, static structure (classes, objects, and relationships) is modeled.
- ***Behavioral model view.*** This part of the analysis model represents the dynamic or behavioral aspects of the system.
- ✓ It also depicts the inter actions or collaborations between various structural elements described in the user model and structural model views.

Software Modeling

- ***Implementation model view.*** The structural and behavioral aspects of the system are represented as they are to be built.
- ***Environment model view.*** The structural and behavioral aspects of the environment in which the system is to be implemented are represented.

UML Diagram Types

- ❑ User model view represent through:
- ✓ **Use-case Diagram:** Shows actors, use-cases, and the relationships between them.
- ❑ Structural model view represents through
- ✓ **Class Diagram:** Shows relationships between classes and pertinent information about classes themselves.
- ✓ **Object Diagram:** Shows a configuration of objects at an instant in time

Software Modeling

- Behavioral model view represents through

- ✓ *Interaction Diagrams:* Show an interaction between groups of collaborating objects. Two types: Collaboration diagram and sequence diagram
- ✓ *Package Diagrams:* Shows system structure at the library/package level.
- ✓ *State Diagram:* Describes behavior of instances of a class in terms of states, stimuli, and transitions.
- ✓ *Activity Diagram:* Very similar to a flowchart—shows actions and decision points, but with the ability to accommodate concurrency.

Software Modeling

- Environment model view represent through

✓ *Deployment Diagram:* Shows configuration of hardware and software in a distributed system.

- Implementation model view represent through

✓ *Component Diagram:* It shows code modules of a system. This code module includes application program, ActiveX control, Java beans and back end databases. It representing interfaces and dependencies among software architect.

Thank -You

Shri Vaishnav Vidyapeeth Vishwavidyalaya, Indore (M.P.)

Shri Vaishnav Institute of Information Technology

Department of Computer Science and Engineering

Lecture

on

"The Software Design Process: Design Concepts and Principles"

.

Software Design Process

- Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.
- The design phase of software development deals with transforming the customer requirements as described in the SRS documents into a form implementable using a programming language.

Software Design Process

- Software design is the third step in SDLC, which moves the concentration from problem domain to solution domain.
- It tries to specify how to fulfill the requirements mentioned in SRS.

Software Design Process

- Software design is an iterative process through which requirements are translated into a “blueprint” for constructing the software.
- Initially, the blueprint depicts a holistic view of software.
- That is, the design is represented at a high level of abstraction at level that can be directly traced to the specific system objective and more detailed data, functional, and behavioral requirements

Software Design Levels

- The software design process can be divided into the following three levels of phases of design:

1. Interface Design
2. Architectural Design
3. Detailed Design

Software Design Level

1. Interface Design

- Interface design is the specification of the interaction between a system and its environment.
- This phase proceeds at a high level of abstraction with respect to the inner workings of the system i.e., during interface design, the internal of the systems are completely ignored and the system is treated as a black box.

Software Design Level

2. Architectural Design

- Architectural design is the specification of the major components of a system, their responsibilities, properties, interfaces, and the relationships and interactions between them.
- In architectural design, the overall structure of the system is chosen, but the internal details of major components are ignored

Software Design Level

3. Detailed Design

- *Detailed Design* is the specification of the internal elements of all major system components, their properties, relationships, processing, and often their algorithms and the data structures.
- *Detailed design* deals with the implementation part of what is seen as a system and its sub-systems in the previous two designs.
- It is more detailed towards modules and their implementations. It defines logical structure of each module and their interfaces to communicate with other modules

Software Design Level

- The detailed design may include:
- Decomposition of major system components into program units.
- Allocation of functional responsibilities to units.
- Unit states and state changes
- Data and control interaction between units
- Algorithms and data structures

Design Principles

- Software design is both a process and a model. The design process is a sequence of steps that enable the designer to describe all aspects of the software to be built.
- The design model is equivalent of an architect's plans for a house.
- It begin by representing the totality of the thing to be built and slowly refines the thing to provide guidance for constructing each detail

Design Principles

- Similarly, the design model that is created for software provides a variety of different views of the computer software.
- Basic design principles enable the software engineer to navigate the design process.

Design Concepts and Principles

- *Should not suffer from "Tunnel Vision" –*
- While designing the process, it should not suffer from "tunnel vision" which means that is should not only focus on completing or achieving the aim but on other effects also.

Design Concepts and Principles

- *Should not "Reinvent The Wheel" –*
- The design process should not reinvent the wheel that means it should not waste time or effort in creating things that already exist. Due to this, the overall development will get increased.

Design Concepts and Principles

- *Minimize Intellectual distance –*
- The design process should reduce the gap between real-world problems and software solutions for that problem meaning it should simply minimize intellectual distance.

Design Concepts and Principles

- *Exhibit uniformity and integration –*
- The design should display uniformity which means it should be uniform throughout the process without any change. Integration means it should mix or combine all parts of software i.e. subsystems into one system.

Design Concepts and Principles

- *Accommodate change –*
- The software should be designed in such a way that it accommodates the change implying that the software should adjust to the change that is required to be done as per the user's need.

Design Concepts and Principles

- *Degrade gently –*
- The software should be designed in such a way that it degrades gracefully which means it should work properly even if an error occurs during the execution.

Design Concepts and Principles

- *Assessed or quality –*
- The design should be assessed or evaluated for the quality meaning that during the evaluation, the quality of the design needs to be checked and focused on.

Design Concepts and Principles

- *Design is not coding and coding is not design –*
- Design means describing the logic of the program to solve any problem and coding is a type of language that is used for the implementation of a design.

Design Concepts

- The following brief overview of important software design concepts that span both traditional and object-oriented software development.
- ***Abstraction*** is the act of representing essential features without including the background details or explanations. The abstraction is used to reduce complexity and allow efficient design and implementation of complex software systems.

Design Concepts

- Many levels of abstraction can be posed.
- *At the highest* level of abstraction, a solution is stated in broad terms using the language of the problem environment.
- *At lower levels* of abstraction, a more detailed description of the solution is provided.

Design Concepts

- Here, there are two common abstraction mechanisms
  - ✓ Functional Abstraction
  - ✓ Data Abstraction
- *Functional Abstraction*
  - ✓ A module is specified by the method it performs.
  - ✓ The details of the algorithm to accomplish the functions are not visible to the user of the function.
- *Data Abstraction*
  - ✓ Details of the data elements are not visible to the users of data. Data Abstraction forms the basis for **Object Oriented design approaches**.

Design Concepts

❑ Modularity

- ✓ Modularity specifies to the division of software into separate modules which are differently named and addressed and are integrated later on in to obtain the completely functional software.
- ✓ It is the only property that allows a program to be intellectually manageable.
- ✓ Single large programs are difficult to understand and read due to a large number of reference variables, control paths, global variables, etc.

Design Concepts

- *Refinement*

✓ Refinement is a top-down design strategy originally proposed by Niklaus Wirth.

✓ Refinement is actually a process of elaboration.

✓ You begin with a statement of function that is defined at a high level of abstraction.

Design Concepts

- *Separation of Concerns*

- ✓ *Separation of concerns* is a design concept that suggests that any complex problem can be more easily handled if it is subdivided into pieces that can each be solved and/or optimized independently.
- ✓ A concern is a feature or behavior that is specified as part of the requirements model for the software.

Thank -You

Shri Vaishnav Vidyapeeth Vishwavidyalaya, Indore (M.P.)

Shri Vaishnav Institute of Information Technology

Department of Computer Science and Engineering

Lecture

on

"User Interface Design"

.

User Interface Design

- User interface is the front-end application view to which user interacts in order to use the software.
- User can manipulate and control the software as well as hardware by means of user interface.
- Today, user interface is found at almost every place where digital technology exists, right from computers, mobile phones, cars, music players, airplanes, etc.

User Interface Design

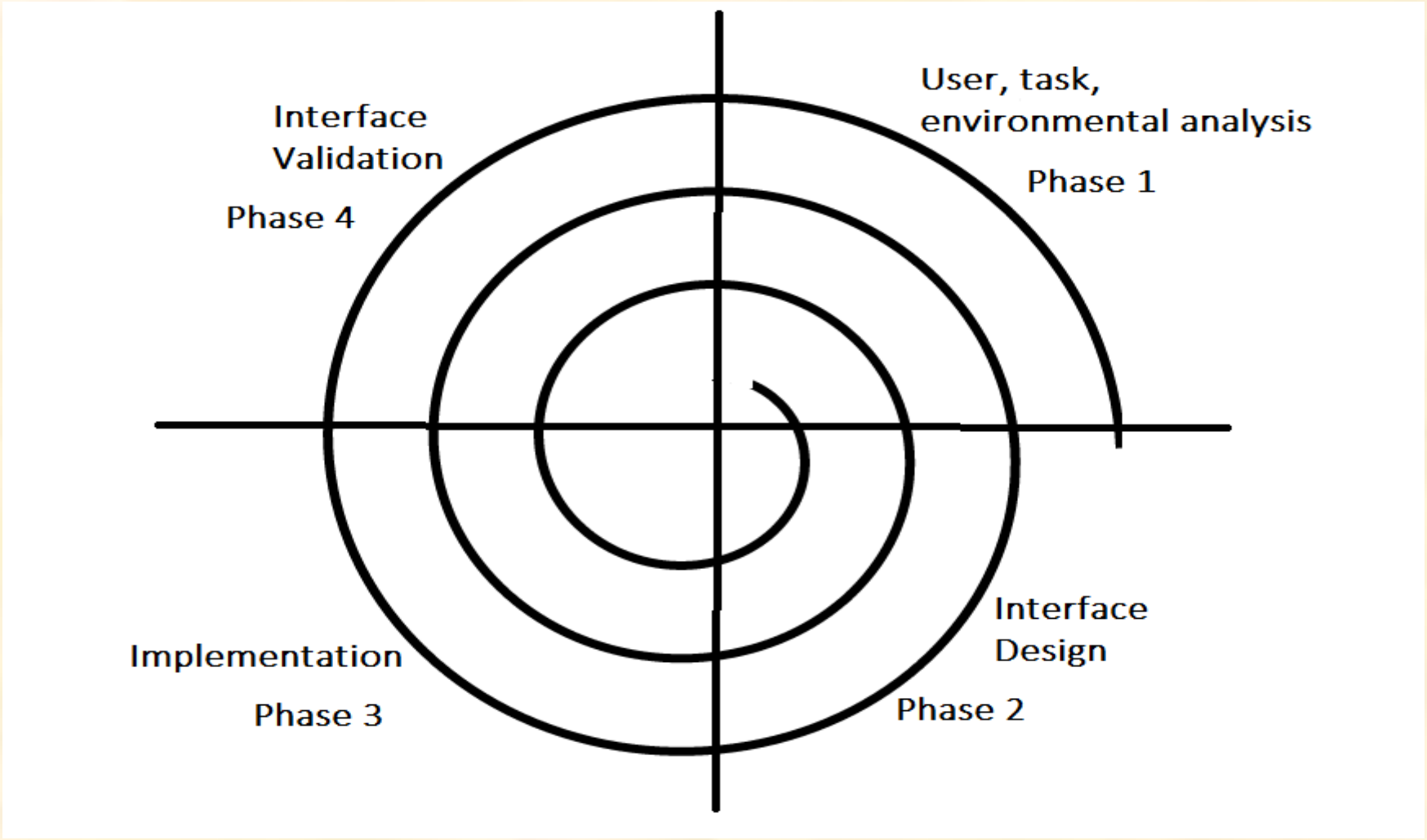
- The software becomes more popular if its user interface is:

- ✓ Attractive
- ✓ Simple to use
- ✓ Responsive in short time
- ✓ Clear to understand
- ✓ Consistent on all interface screens

User Interface Design

- There are two types of User Interface:
- **Command Line Interface:** Command Line Interface provides a command prompt, where the user types the command and feeds to the system. The user needs to remember the syntax of the command and its use.
- **Graphical User Interface:** Graphical User Interface provides the simple interactive interface to interact with the system. GUI can be a combination of both hardware and software. Using GUI, user interprets the software.

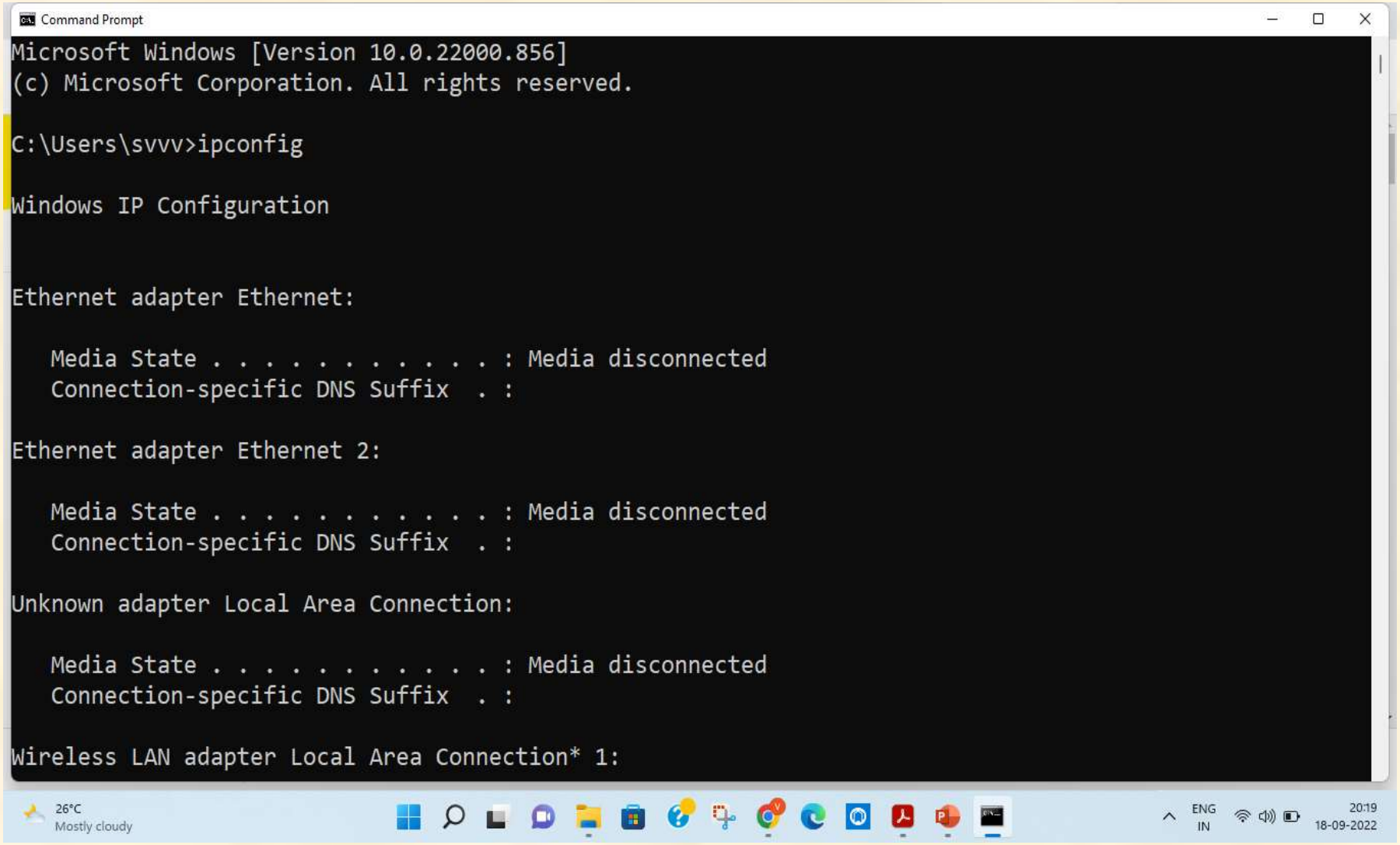
User Interface Design Process



User Interface Design

- A command is a text-based reference to set of instructions, which are expected to be executed by the system.
- There are methods like macros, scripts that make it easy for the user to operate.
- CLI uses less amount of computer resource as compared to GUI.

Command Interface Example



User Interface Design

- Typically, GUI is more resource consuming than that of CLI.
- With advancing technology, the programmers and designers create complex GUI designs that work with more efficiency, accuracy and speed.

User Interface Design

- **GUI Elements**
- GUI provides a set of components to interact with software or hardware.
- Every graphical component provides a way to work with the system. A GUI system has following elements such as:

1. ***Window*** - An area where contents of application are displayed. Contents in a window can be displayed in the form of icons or lists. Windows can be minimized, resized or maximized to the size of screen. They can be moved anywhere on the screen

User Interface Design

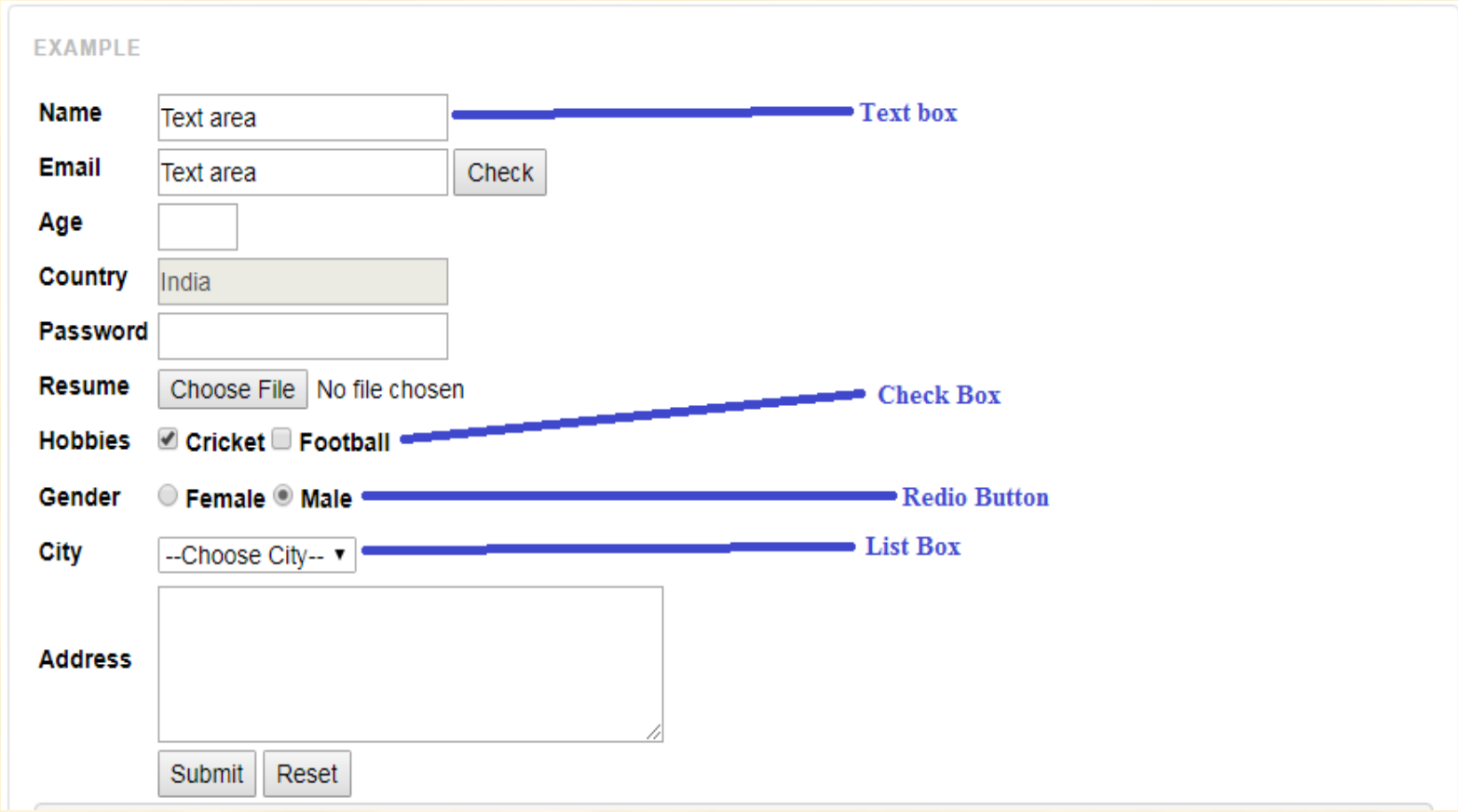
2. Tabs- If an application allows executing multiple instances of itself, they appear on the screen as separate windows. This interface also helps in viewing preference panel in application. All modern web-browsers use this feature.
3. Menu- Menu is an array of standard commands, grouped together and placed at a visible place (usually top) inside the application window. The menu can be programmed to appear or hide on mouse clicks.

User Interface Design

4. Icon- An icon is small picture representing an associated application. When these icons are clicked or double clicked, the application window is opened.
5. Cursor- Interacting devices such as mouse, touch pad, digital pen are represented in GUI as cursors. On screen cursor follows the instructions from hardware in almost real-time. They are used to select menus, windows and other application features.

User Interface Design

Application specific GUI components



User Interface Design

- Features of GUIs include:
- ✓ They are much easier to use for beginners.
- ✓ They enable you to easily exchange information between software using cut and paste or 'drag and drop’.
- ✓ They use a lot of memory and processing power. It can be slower to use than a command-line interface if you are an expert user.
- ✓ They can be irritating to experienced users when simple tasks require a number of operations.

User Interface Design Principles

- **Structure:**

- ✓ Design should organize the user interface purposefully, in the meaningful and usual based on precise, consistent models that are apparent and recognizable to users, putting related things together and separating unrelated things, differentiating dissimilar things and making similar things resemble one another.
- ✓ The structure principle is concerned with overall user interface architecture.

User Interface Design Principles

- **Simplicity:**

✓ The design should make the simple, common task easy, communicating clearly and directly in the user's language, and providing good shortcuts that are meaningfully related to longer procedures.

- **Visibility:**

✓ The design should make all required options and materials for a given function visible without distracting the user with unnecessary or redundant data.

User Interface Design Principles

- **Feedback:**

✓ The design should keep users informed of actions or interpretation, changes of state or condition, and bugs or exceptions that are relevant and of interest to the user through clear, brief, and unambiguous language familiar to users

Thank -You