Ms. Mandakini Ingle

Assistant Professor

Computer Science & Engineering

Subject :- SEPM

Subject Code :- BTCS504

Class :- CS 3 Year

SYLLABUS

Unit III: Software Design

The Software Design Process, Design Concepts and Principles, Software Modeling and UML, Architectural Design, Architectural Views and Styles, User Interface Design, Function oriented Design, SA/SD Component Based Design, Design Metrics.

Software Design

- Software Design is also a process to plan or convert the software requirements into a step that are needed to be carried out to develop a software system.

- There are several principles that are used to organize and arrange the structural components of Software design.

- Software Designs in which these principles are applied affect the content and the working process of the software from the beginning.

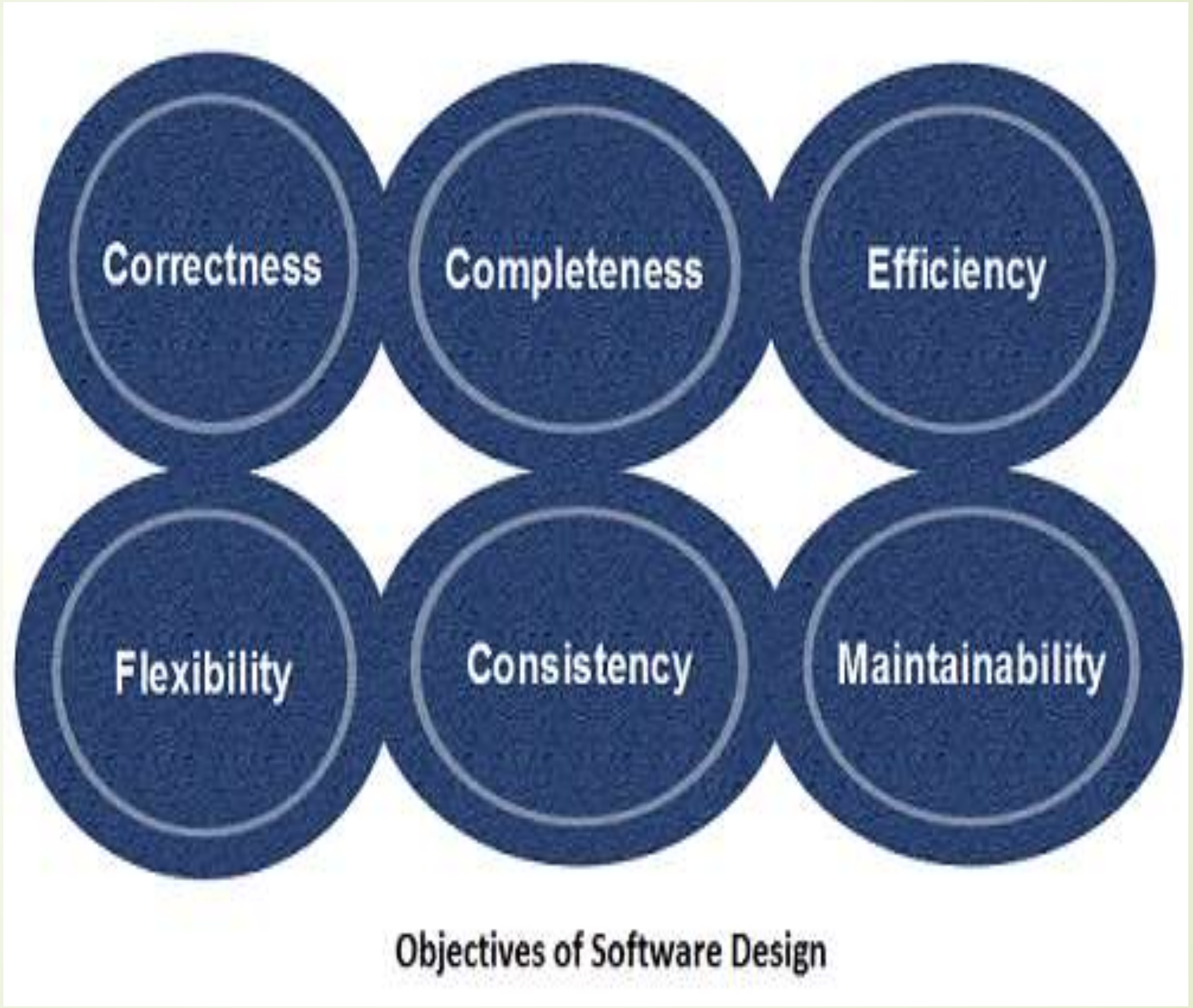
Software Design

- 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.

Software Design

- Software design is the process of envisioning and defining software solutions to one or more sets of problems.
- One of the main components of software design is the software requirements analysis (SRA).
- SRA is a part of the software development process that lists specifications used in software engineering
- Software Design is also a process to plan or convert the software requirements into a step that are needed to be carried out to develop a software system.

Objectives of Software Design



Objectives of Software Design

Objectives of Software Design

- The design must be implement All the explicit requirement contained in analysis model & must use allthe implicir requirement desired by customer

- The design must be readable & understandable to the programmer and developer.

- Design should provide complete picture of software.

Objectives of Software Design

- **Correctness:**Software design should be correct as per requirement.
- **Completeness:**The design should have all components like data structures, modules, and external interfaces, etc.
- **Efficiency:**Resources should be used efficiently by the program.
- **Flexibility:**Able to modify on changing needs.
- **Consistency:**There should not be any inconsistency in the design.
- **Maintainability:** The design should be so simple so that it can be easily maintainable by other designers.

Principles of Software Design

There are several principles that are used to organize and arrange the structural components of Software design.

Software Designs in which these principles are applied **affect the content and the working process of the software** from the beginning.

1. 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.

2. Traceable to analysis model –

The design process should be traceable to the analysis model which means it should satisfy all the requirements that software requires to develop a high-quality product.

3. 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.

4. 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.

5.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.

6. 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.

7. Degrade gently –[Robust]

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.

8. 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.

9. Review to discover errors –

The design should be reviewed which means that the overall evaluation should be done to check if there is any error present or if it can be minimized.

10. 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.

Software Design Fundamental concepts

The design concepts provide the software designer with a foundation from which more sophisticated methods can be applied. A set of fundamental design concepts has evolved. They are as follows:

1. Abstraction - Abstraction is the process or result of generalization by reducing the information content of a concept or an observable phenomenon, typically in order to retain only information which is relevant for a particular purpose. It is an act of Representing essential features without including the background details or explanations.[Data & Procedure]

Design concepts.............

2. Refinement - It is the process of elaboration. A hierarchy is developed by decomposing a macroscopic statement of function in a step-wise fashion until programming language statements are reached. In each step, one or several instructions of a given program are decomposed into more detailed instructions. Abstraction and Refinement are complementary concepts.

3. Modularity - Software architecture is divided into components called modules

Design concepts..............

4. Software Architecture - It refers to the overall structure of the software and the ways in which that structure provides conceptual integrity for a system. Good software architecture will yield a good return on investment with respect to the desired outcome of the project, e.g. in terms of performance, quality, schedule and cost.

5. Control Hierarchy - A program structure that represents the organization of a program component and implies a hierarchy of control.

Design concepts..............

5. Structural Partitioning - The program structure can be divided into both horizontally and vertically. Horizontal partitions define separate branches of modular hierarchy for each major program function. Vertical partitioning suggests that control and work should be distributed top down in the program structure.

6. Data Structure - It is a representation of the logical relationship among individual elements of data.

Design concepts..............

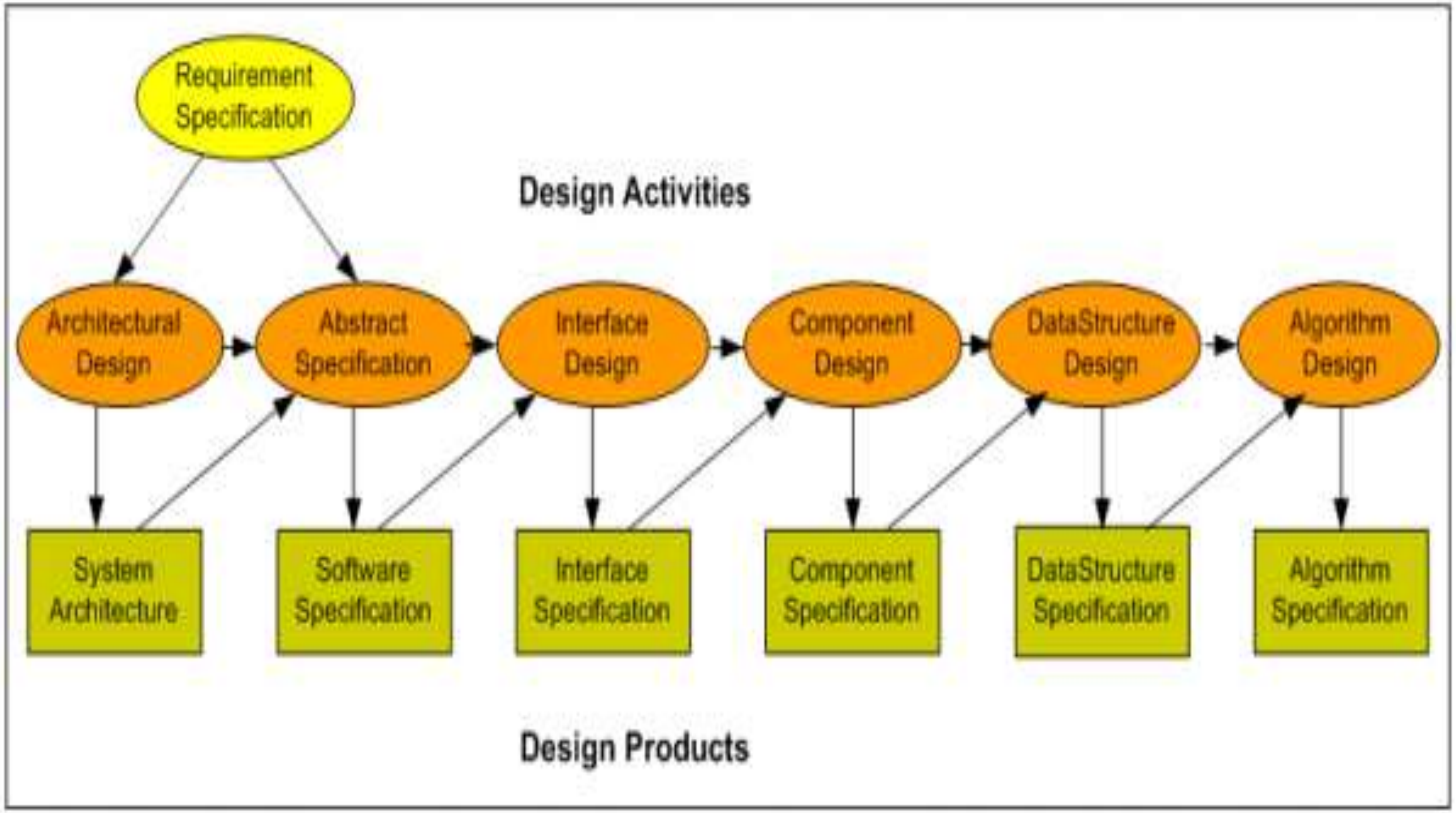
7. Software Procedure - It focuses on the processing of each module individually.

8. Information Hiding - Modules should be specified and designed so that information contained within a module is inaccessible to other modules that have no need for such information.

The Software Design Process

- The software design process is a sequence of steps carried through which the requirements are translated into software model.
- Several different high levels of abstraction.
- Software design is an iterative process through which requirements are translated into a ―blueprintǁ for constructing the software.
- Various phases of design process shown in figure...

A general model of design process



The Software Design Process

- ❖ In Architectural Design the subsystem components can be identified.
- ❖ Abstract specification is used to specify the subsystems.
- ❖ The interface between the sub-systems are design which is called Interface Design
- ❖ In Component Design of subsystem components is done
- ❖ Data structure design to hold the data in a particular framework
- ❖ For performing the require functionality algorithm used

Software Process Design MODELS

During the design process the software specifications are transformed into design models

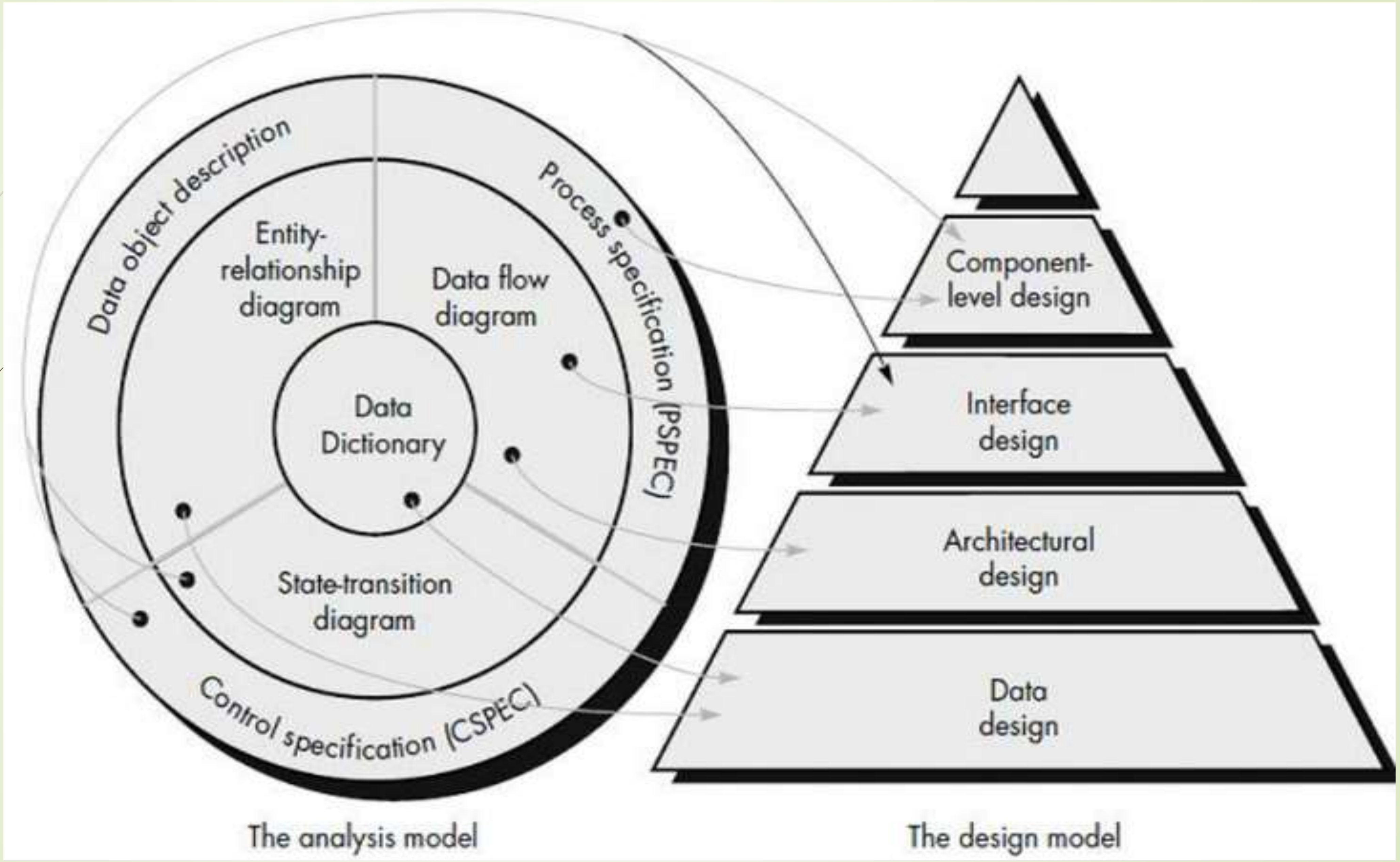
- Models describe the details of the data structures, system architecture, interface, and components.
- Each design product is reviewed for quality before moving to the next phase of software development.
- At the end of the design process a design model and specification document is produced.
- This document is composed of the design models that describe the data, architecture, interfaces and components

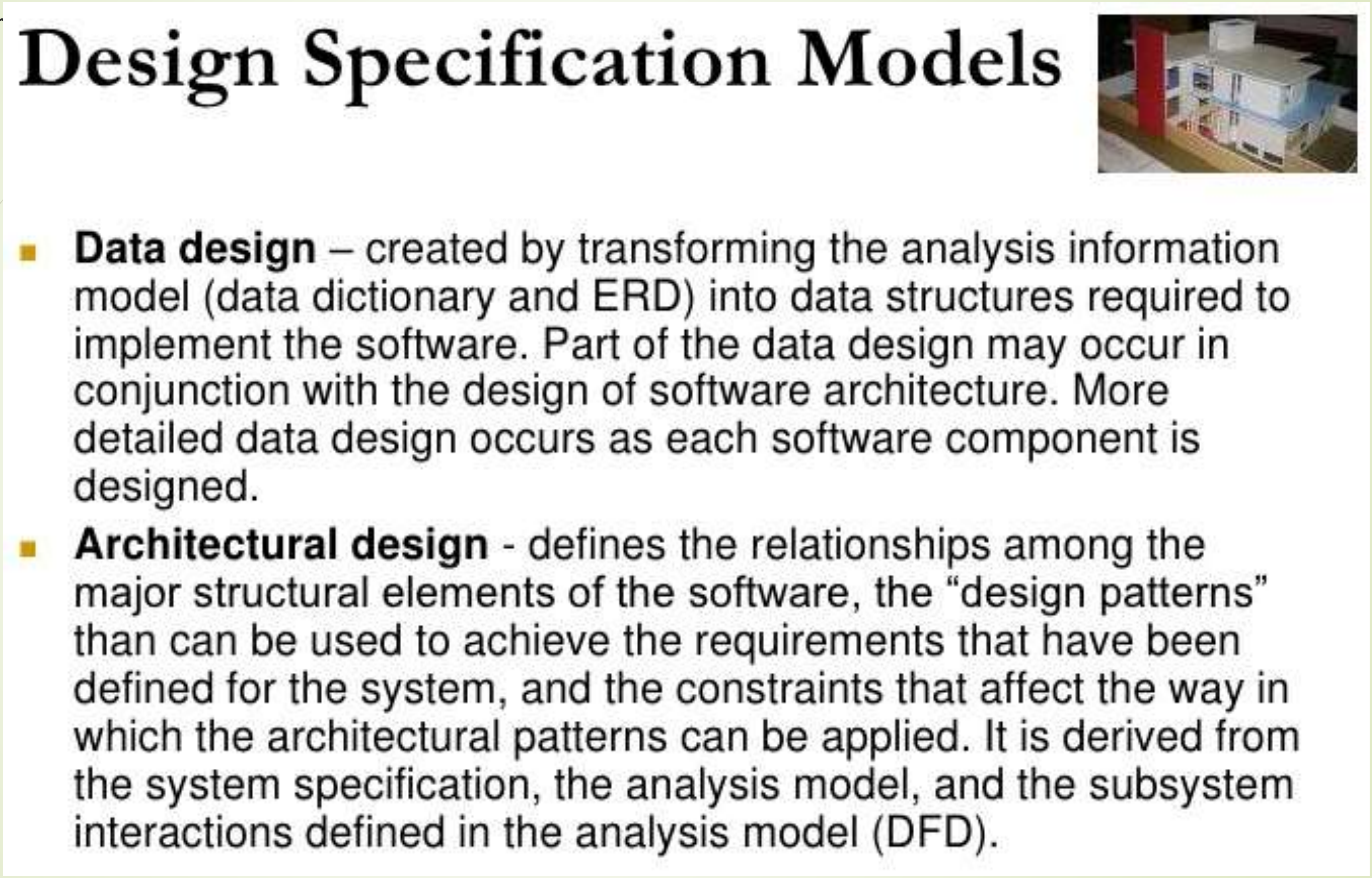
The Software Design model

The software design model can be divided into the following main Four levels of phases of design:

1. Data Design
2. Architectural Design
3. Interface Design
4. Component Level/Procedural Design

Elements of software Analysis model and Design Model





Design Specification Models

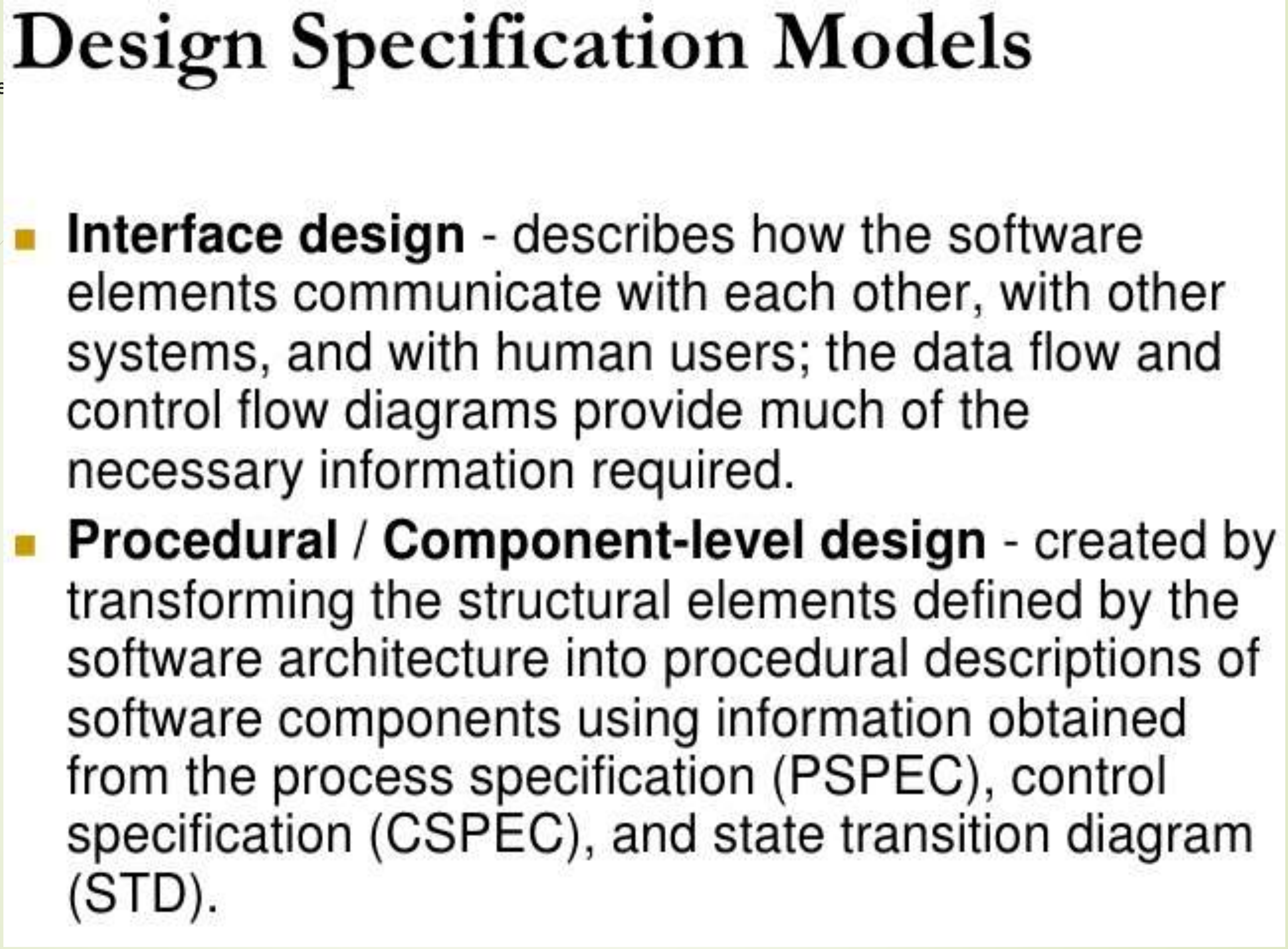
- **Data design** – created by transforming the analysis information model (data dictionary and ERD) into data structures required to implement the software. Part of the data design may occur in conjunction with the design of software architecture. More detailed data design occurs as each software component is designed.
- **Architectural design** - defines the relationships among the major structural elements of the software, the "design patterns" than can be used to achieve the requirements that have been defined for the system, and the constraints that affect the way in which the architectural patterns can be applied. It is derived from the system specification, the analysis model, and the subsystem interactions defined in the analysis model (DFD).

Issues in architectural design includes:

1. Gross decomposition of the systems into major components.
2. Allocation of functional responsibilities to components.
3. Component Interfaces
4. Component scaling and performance properties, resource consumption properties, reliability properties, and so forth.
5. Communication and interaction between components.

Interface design should include the following details:

1. Precise description of events in the environment, or messages from agents to which the system must respond.
2. Precise description of the events or messages that the system must produce.
3. Specification on the data, and the formats of the data coming into and going out of the system.
4. Specification of the ordering and timing relationships between incoming events or messages, and outgoing events or outputs.



Design Specification Models

- **Interface design** - describes how the software elements communicate with each other, with other systems, and with human users; the data flow and control flow diagrams provide much of the necessary information required.
- **Procedural / Component-level design** - created by transforming the structural elements defined by the software architecture into procedural descriptions of software components using information obtained from the process specification (PSPEC), control specification (CSPEC), and state transition diagram (STD).

The detailed design may include:

1. Decomposition of major system components into program units.
2. Allocation of functional responsibilities to units.
3. User interfaces
4. Unit states and state changes
5. Data and control interaction between units
6. Data packaging and implementation, including issues of scope and visibility of program elements
7. Algorithms and data structures

Architectural Design

1. The software needs the architectural design to represents the design of software & identify the subsystem of system.
2. 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.
3. Establish the overall structure of software.
4. Architectural design represents link between design specification and actual design process.
5. The software that is built for computer-based systems can exhibit one of these many architectural styles.

Architectural Design

Common activities of design precess

1. System structuring
2. Control modeling
3. Modular Decomposition

Architectural Style

1. The architectural style is a very specific solution to a particular software, which typically focuses on **how to organize the code created for the software**.
2. It focuses on creating the layers and modules of the software and allowing an appropriate interaction between the various modules for giving the right results upon implementation.
3. Apattern of creating system architecture or given problem.

Architectural Style

System categories define the architecture style..

1. Components
2. Connectors
3. Constraints
4. Semantic Model

Architectural styles Objectives

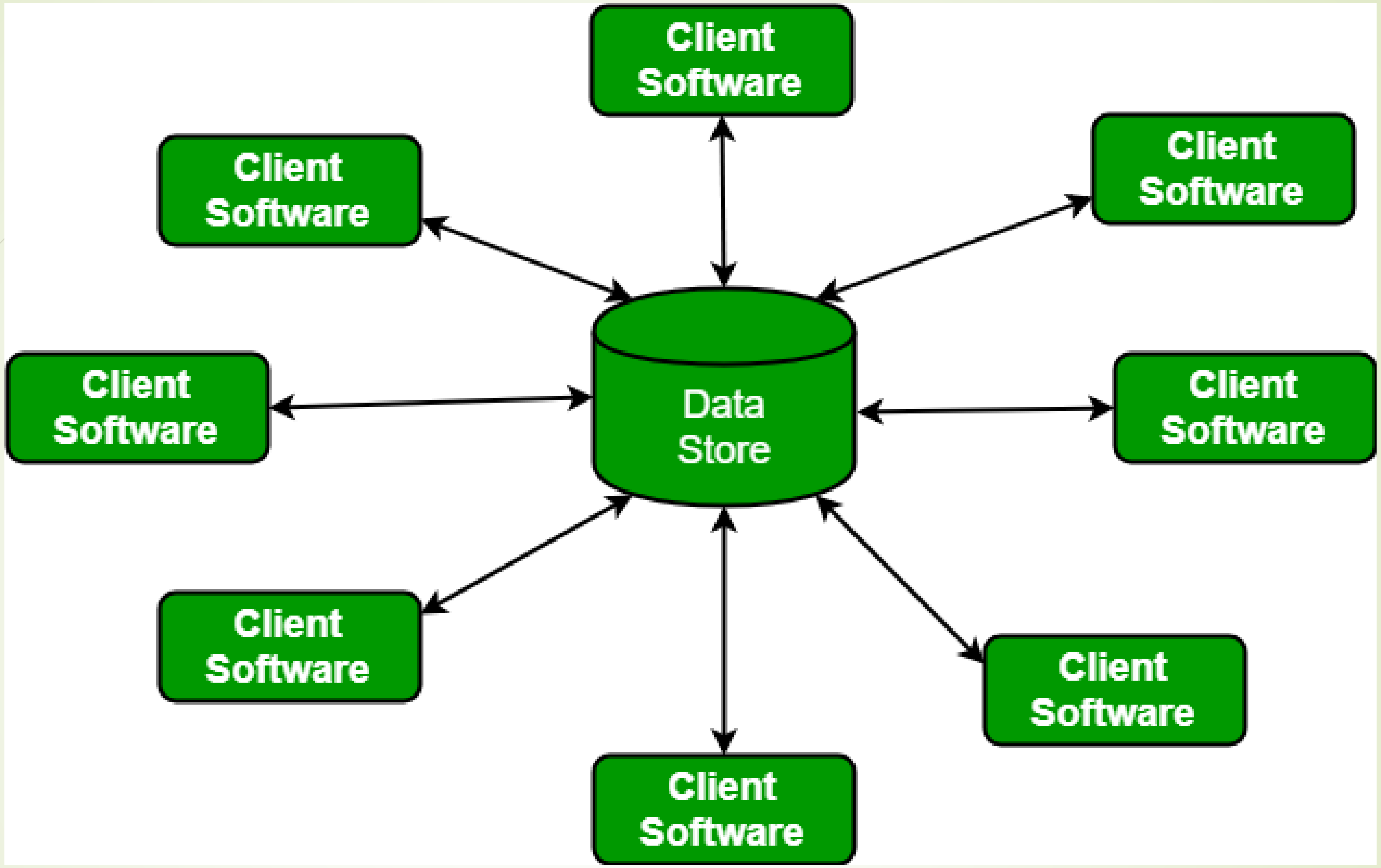
Each style will desecribe a system category that consists of :

1. A set of components(eg: a database, computational modules) that will perform a function required by the system.
2. The set of connectors will help in coordination, communication, and cooperation between the components.
3. Conditions that how components can be integrated to form the system.
4. Semantic models that help the designer to understand the overall properties of the system.
5. The use of architectural styles is to establish a structure for all the components of the system.

Types of Architectural styles:

1. Data centred architectures:

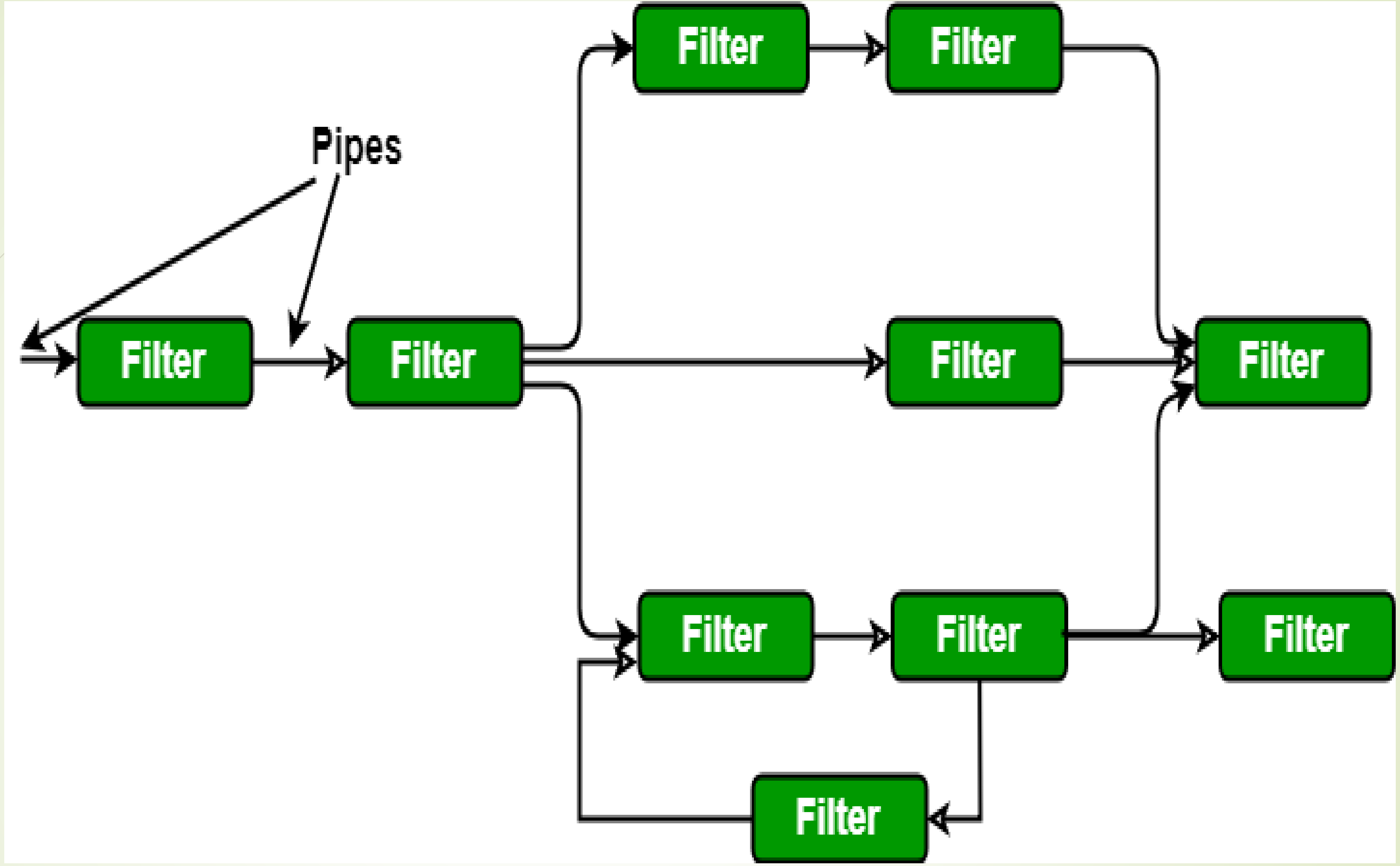
1. A data store will reside at the center of this architecture and is accessed frequently by the other components that update, add, delete or modify the data present within the store.
2. The figure illustrates a typical data centered style. The client software access a central repository. Variation of this approach are used to transform the repository into a blackboard when data related to client or data of interest for the client change the notifications to client software.
3. This data-centered architecture will promote integrability. This means that the existing components can be changed and new client components can be added to the architecture without the permission or concern of other clients.
4. Data can be passed among clients using blackboard mechanism.



Data centred architectures

2. Data flow architectures

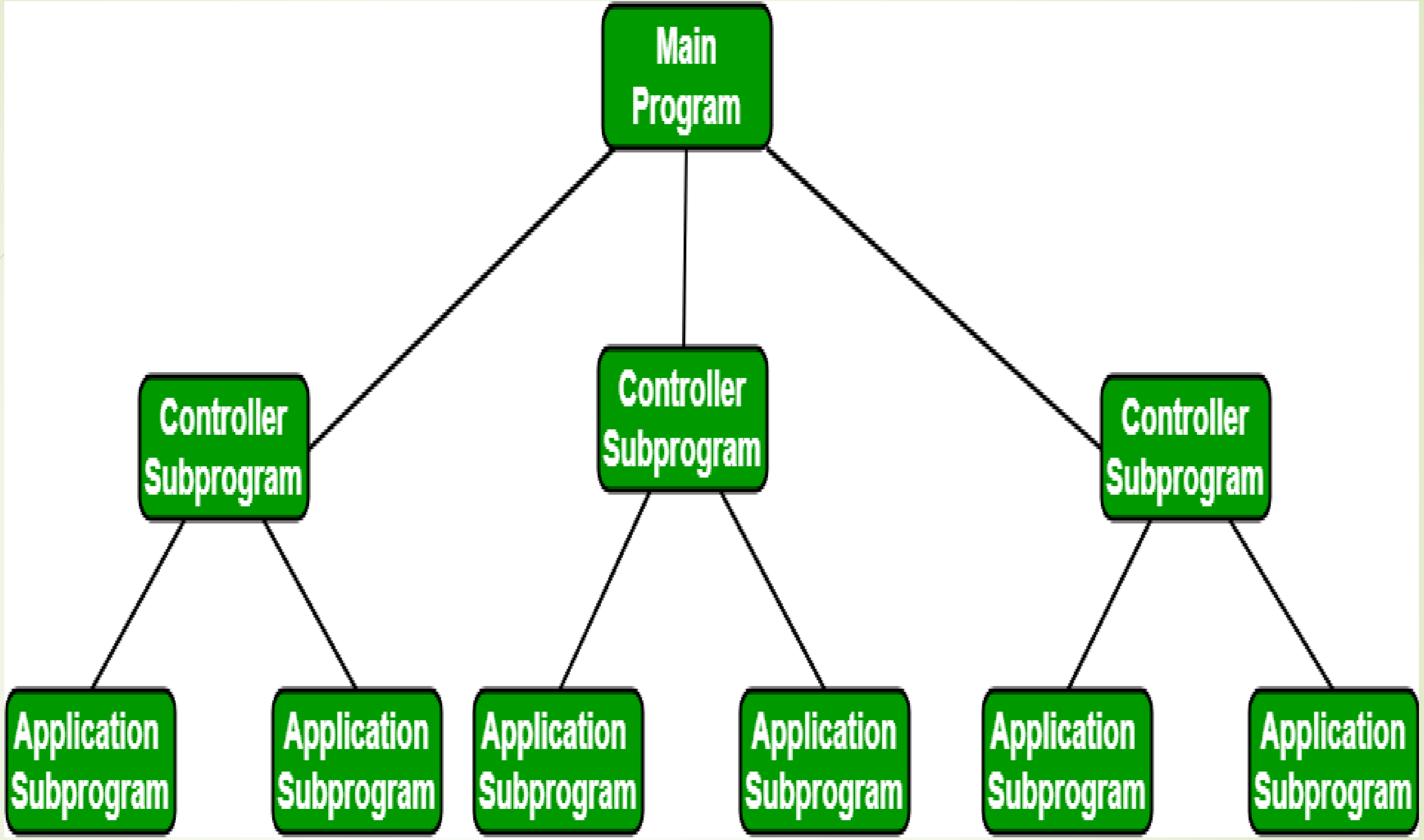
1. This kind of architecture is used when input data to be transformed into output data through a series of computational manipulative components.
2. The figure represents pipe-and-filter architecture since it uses both pipe and filter and it has a set of components called filters connected by pipes.
3. Pipes are used to transmit data from one component to the next.
4. Each filter will work independently and is designed to take data input of a certain form and produces data output to the next filter of a specified form. The filters don't require any knowledge of the working of neighboring filters.
5. If the data flow degenerates into a single line of transforms, then it is termed as batch sequential. This structure accepts the batch of data and then applies a series of sequential components to transform it.



Data flow architectures

3. Call and Return architectures

1. It is used to create a program that is easy to scale and modify. Many sub-styles exist within this category. Two of them are explained below.
2. Remote procedure call architecture: This components is used to present in a main program or sub program architecture distributed among multiple computers on a network.
3. Main program or Subprogram architectures: The main program structure decomposes into number of subprograms or function into a control hierarchy. Main program contains number of subprograms that can invoke other components



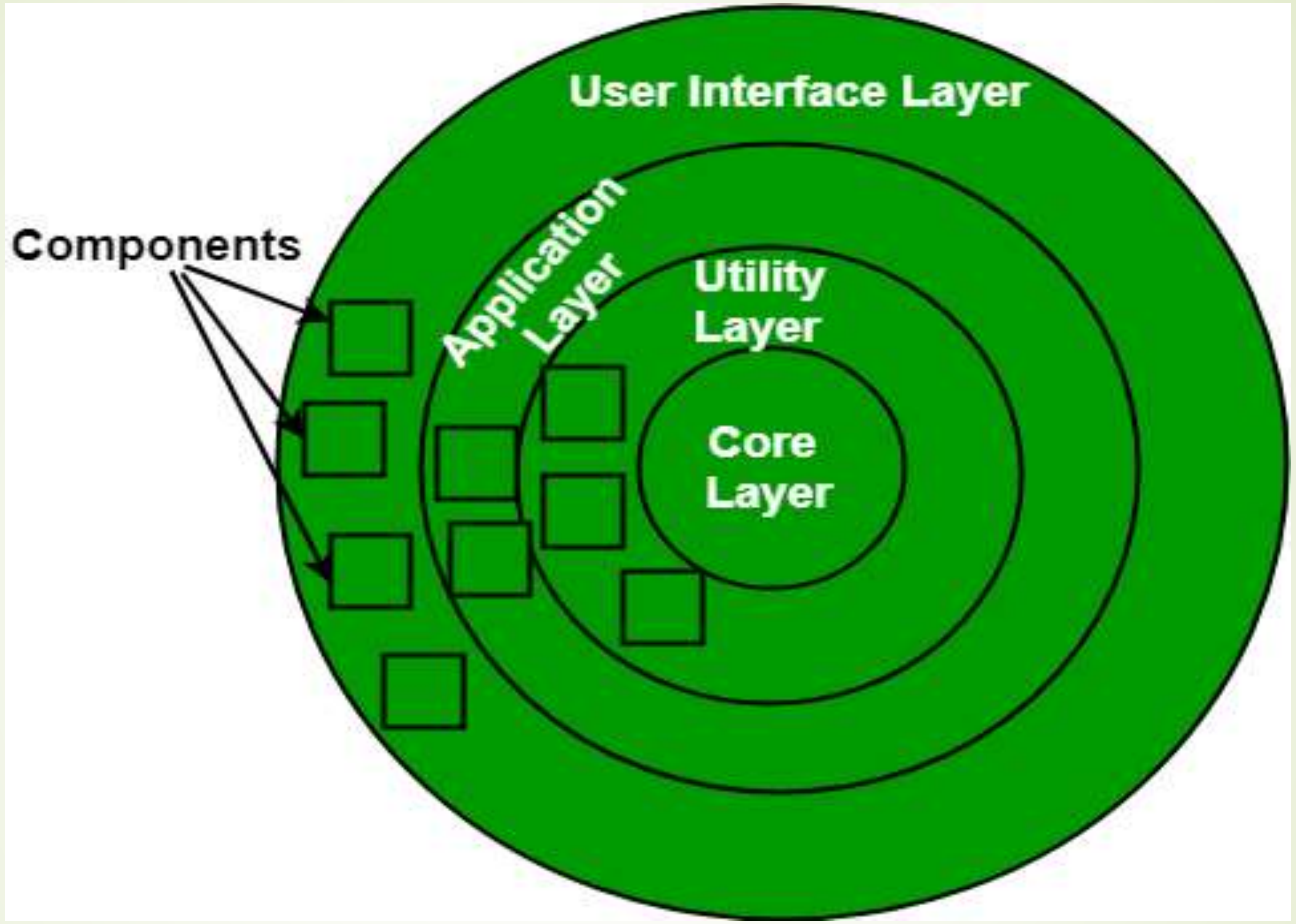
Call and Return architectures

4. Object Oriented architecture:

1. The components of a system encapsulate data and the operations that must be applied to manipulate the data. The coordination and communication between the components are established via the message passing.

5. Layered architecture:

1. A number of different layers are defined with each layer performing a well-defined set of operations. Each layer will do some operations that becomes closer to machine instruction set progressively.
2. At the outer layer, components will receive the user interface operations and at the inner layers, components will perform the operating system interfacing(communication and coordination with OS)
3. Intermediate layers to utility services and application software functions



Layered architecture:

Architectural Views

1. The latest thinking in architecture descriptions recommends the concept of architectural views.

2. Philippe Kruchten [Kruchten 95] describes an architecture for software intensive systems called **"the 4+1 Architectural View Model"**. It is based on the use of multiple, concurrent views. The views are used to describe the system from the viewpoint of different stakeholders, such as end-users, developers and project managers.

Different Types of View

- ❖ Logical view
- ❖ Development View
- ❖ Physical View
- ❖ Process View

Architectural Views

It is an architecture verification method for studying and documenting software architecture design and covers all the aspects of software architecture for all stakeholders. It provides four essential views –

1. **Logical view:** the services that the system provides to end-users depicted with UML model types: Class diagram, Communication diagram, Sequence diagram .

2. **Development view:** The system from a programmer's perspective, 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

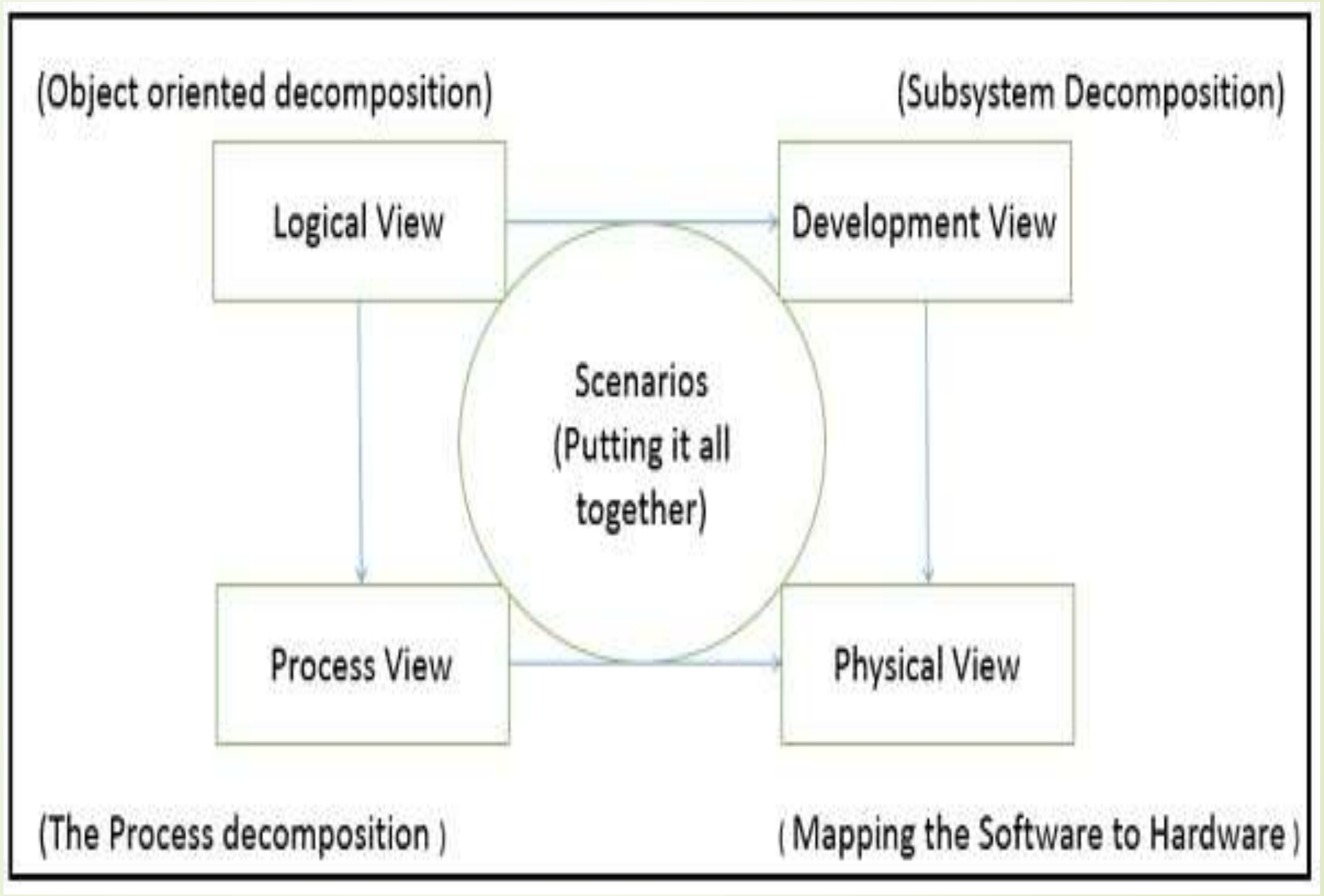
3. **Process view :** The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the runtime behavior of the system. The process view addresses concurrency, distribution, integrators, performance, and scalability, etc. UML Diagrams to represent process view include the Activity diagram.

4. **Physical view:** The physical view depicts the system from a system engineer's point-of-view. It is concerned with the topology of software components on the physical layer, as well as the physical connections between these components. This view is also known as the deployment view. UML Diagrams used to represent physical view include the Deployment diagram.

Architectural Views........

5. **Scenarios:** The description of an 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. UML Diagram(s) used to represent the scenario view include the Use case diagram.

Architectural Views Models


(Object oriented decomposition)
(Subsystem Decomposition)
Logical View
Development View
Scenarios (Putting it all together)
Process View
Physical View
(The Process decomposition)
(Mapping the Software to Hardware)

User interface design

- User interface design is an iterative process, where each iteration elaborate and refines the information developed in the preceding step
- General steps for user interface design

1) Using information developed during user interface analysis, define user interface objects and actions (operations)

2) Define events (user actions) that will cause the state of the user interface to change; model this behavior

3) Depict each interface state as it will actually look to the end user

4) Indicate how the user interprets the state of the system from information provided through the interface

During all of these steps, the designer must

- Always follow the three golden rules of user interfaces
- Model how the interface will be implemented
- Consider the computing environment (e.g., display technology, operating system, development tools) that will be used 2

User interface design

Interface design focuses on the following

- Designing effective interfaces for software systems
- The design of interfaces between software components
- The design of interfaces between the software and other nonhuman producers and consumers of information
- The design of the interface between a human and the computer
- Graphical user interfaces (GUIs) have helped to eliminate many of the most horrific interface problems
- However, some are still difficult to learn, hard to use, confusing, counterintuitive, unforgiving, and frustrating
- User interface analysis and design has to do with the study of people and how they relate to technology

Objectives

- To suggest some general design principles for user interface design
- To explain different interaction styles
- To introduce styles of information presentation
- To describe the user support which should be built-in to user interfaces
- To introduce usability attributes and system approaches to system evaluation

The user interface

- System users often judge a system by its interface rather than its functionality
- A poorly designed interface can cause a user to make catastrophic errors
- Poor user interface design is the reason why so many software systems are never used

Types of user interface

1. Command Line Interfaces

- With a command language, the user types commands to give instructions to the system
- May be implemented using cheap terminals
- Easy to process using compiler techniques
- Commands of arbitrary complexity can be created by command combination
- Concise interfaces requiring minimal typing can be created

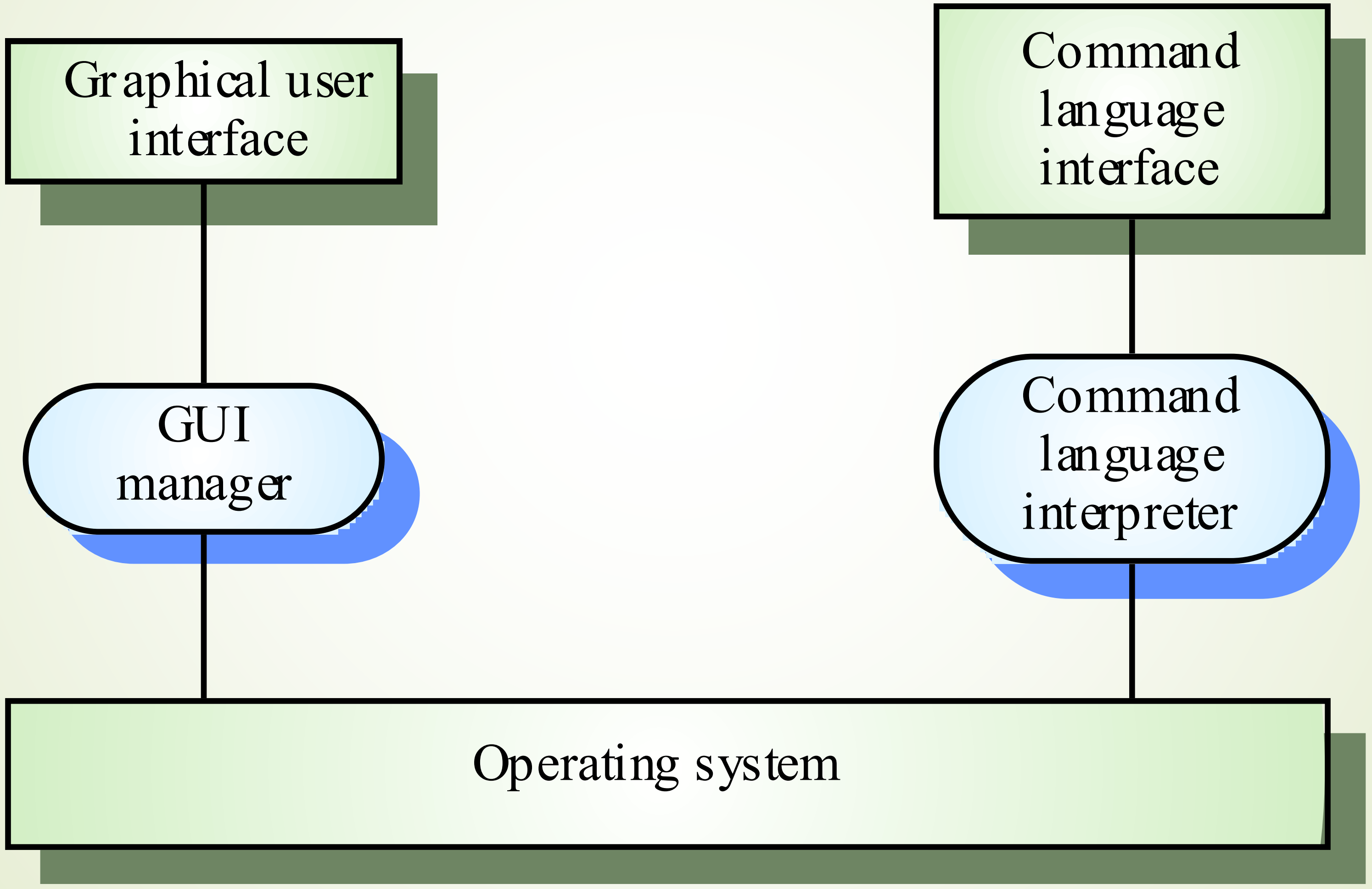
Command Line Interfaces Advantages

- Allow experienced users to interact quickly with the system
- Commands can be scripted

(!) **Problems**

- Users have to learn and remember a command language
- Not suitable for occasional or inexperienced users
- An error detection and recovery system is required
- Typing ability is required (!)

Multiple user interfaces



2. Graphical user interfaces

- Most users of business systems interact with these systems through graphical interfaces although, in some cases, legacy text-based interfaces are still used.
- Easy to use
- An inexperience user can use system easily.
- User can switch from one task to another task very easy
- Interact with many applications simultaneously.

GUI characteristics

Characteristic	**Description**
Windows	Multiple windows allow different information to be displayed simultaneously on the user's screen.
Icons	Icons different types of information. On some systems, icons represent files; on others, icons represent processes.
Menus	Commands are selected from a menu rather than typed in a command language.
Pointing	A pointing device such as a mouse is used for selecting choices from a menu or indicating items of interest in a window.
Graphics	Graphical elements can be mixed with text on the same display.

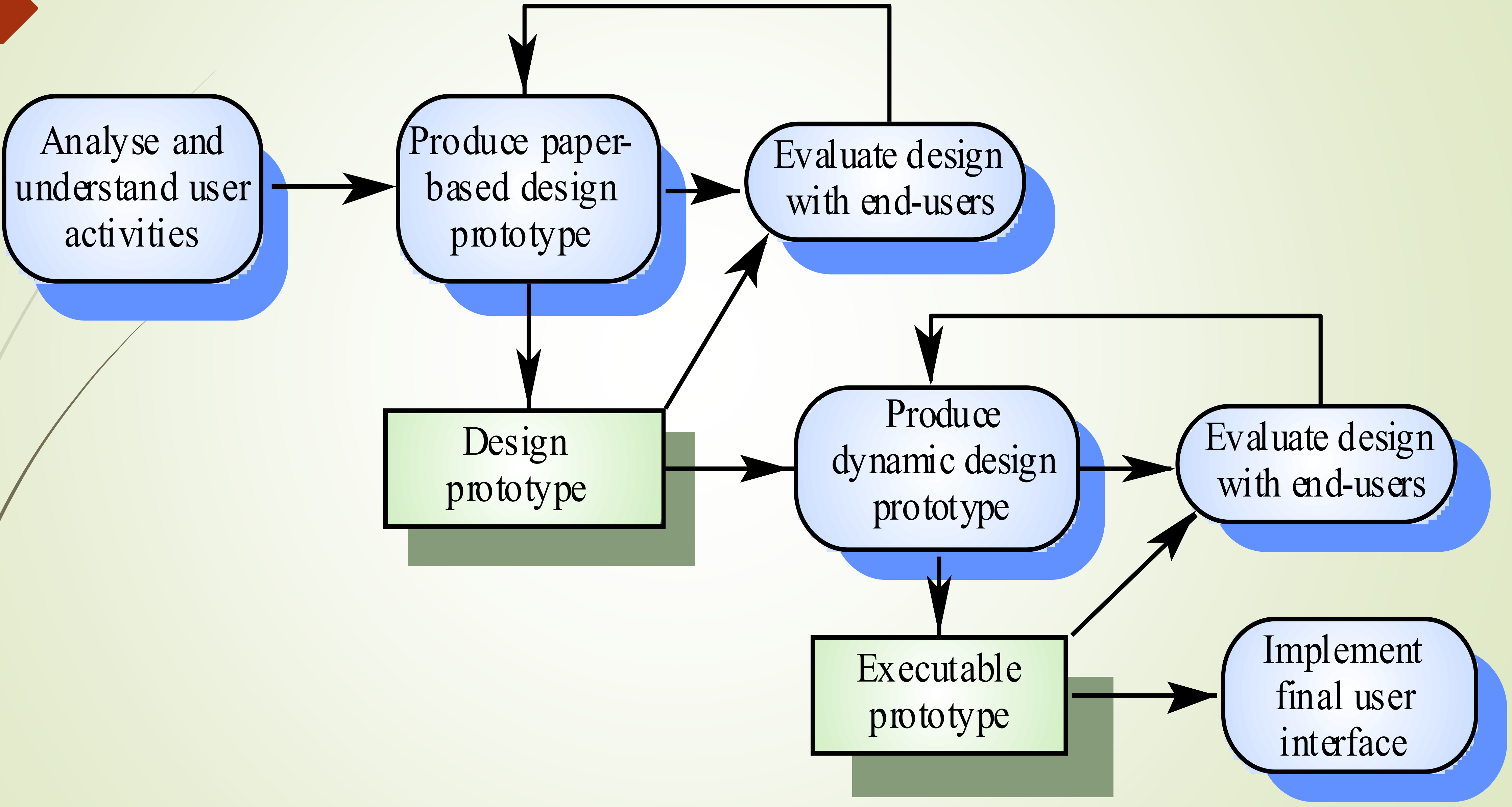
GUI advantages

- They are easy to learn and use.
  - Users without experience can learn to use the system quickly.
- The user may switch quickly from one task to another and can interact with several different applications.
  - Information remains visible in its own window when attention is switched.
- Fast, full-screen interaction is possible with immediate access to anywhere on the screen

User-centred design

- The aim of this chapter is to sensitise software engineers to key issues underlying the design rather than the implementation of user interfaces
- User-centred design is an approach to UI design where the needs of the user are paramount and where the user is involved in the design process
- UI design always involves the development of prototype interfaces

User interface design process


Analyse and understand user activities
Produce paper-based design prototype
Evaluate design with end-users
Design prototype
Produce dynamic design prototype
Evaluate design with end-users
Executable prototype
Implement final user interface

UI design principles

- UI design must take account of the needs, experience and capabilities of the system users
- Designers should be aware of people's physical and mental limitations (e.g. limited short-term memory) and should recognise that people make mistakes
- UI design principles underlie interface designs although not all principles are applicable to all designs

User interface design principles

Principle	Description
User familiarity	The interface should use terms and concepts which are drawn from the experience of the people who will make most use of the system.
Consistency	The interface should be consistent in that, wherever possible, comparable operations should be activated in the same way.
Minimal surprise	Users should never be surprised by the behaviour of a system.
Recoverability	The interface should include mechanisms to allow users to recover from errors.
User guidance	The interface should provide meaningful feedback when errors occur and provide context-sensitive user help facilities.
User diversity	The interface should provide appropriate interaction facilities for different types of system user.

User interface design principles

- **User familiarity**
  - The interface should be based on user-oriented terms and concepts rather than computer concepts. For example, an office system should use concepts such as letters, documents, folders etc. rather than directories, file identifiers, etc.
- **Consistency**
  - The system should display an appropriate level of consistency. Commands and menus should have the same format, command punctuation should be similar, etc.
- **Minimal surprise**
  - If a command operates in a known way, the user should be able to predict the operation of comparable commands

User interface design principles

- **Recoverability**
  - The system should provide some resilience to user errors and allow the user to recover from errors. This might include an undo facility, confirmation of destructive actions, 'soft' deletes, etc.
- **User guidance**
  - Some user guidance such as help systems, on-line manuals, etc. should be supplied
- **User diversity**
  - Interaction facilities for different types of user should be supported. For example, some users have seeing difficulties and so larger text should be available

User interface design principles

- **Simplicity:-** Design will be simple and common.
- **Visibility:-** Every tasks are visible to user.
- **Tolerance**:- Design will be flexible and tolerant.
  Because reducing cost of mistakes.
- **Reuse:-** Programmer can reuse internal and external components of system.
- **Structure principle:-** Good and clear structure
- **Feedback:-** Provide the user with visual and auditory feedback, maintaining two-way communication.

User interface design principles

- **Memory load Reduce:-** the amount of information that must be remembered between actions. Minimize the memory load.
- **Efficiency:-** Seek efficiency in dialogue, motion and thought. Minimize keystrokes and mouse movements.
- **Recoverability:-** Allow users to recover from their errors. Include undo facilities, confirmation of destructive actions, 'soft' deletes, etc.
- **User guidance:-** Incorporate some form of context-sensitive user guidance and assistance.

User-system interaction

- Two problems must be addressed in interactive systems design
  - How should information from the user be provided to the computer system?
  - How should information from the computer system be presented to the user?
- User interaction and information presentation may be integrated through a coherent framework such as a user interface metaphor

Interaction styles

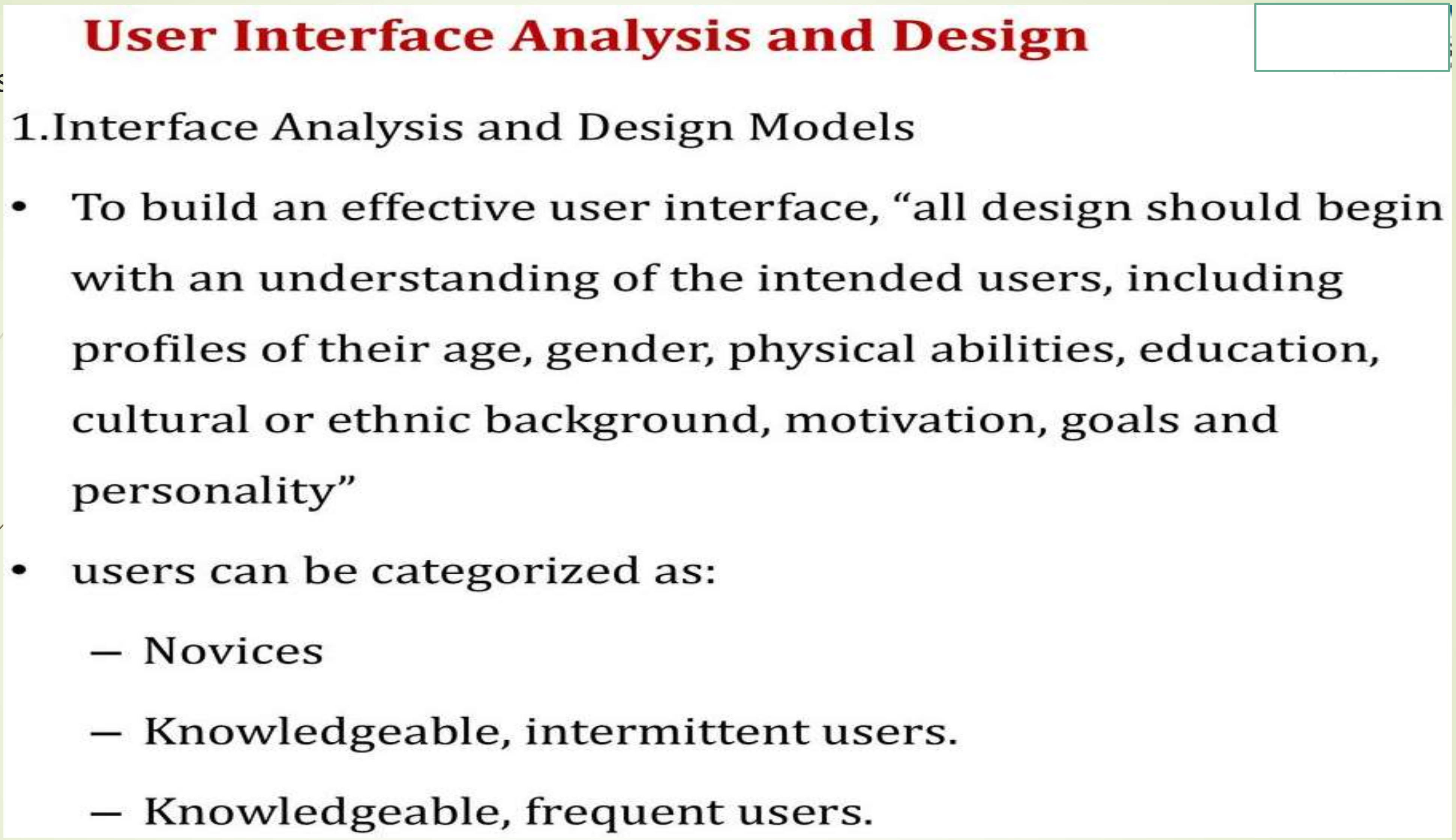
- Direct manipulation
- Menu selection
- Form fill-in
- Command language
- Natural language

Advantages and disadvantages

Interaction style	Main advantages	Main disadvantages	Application examples
Direct manipulation	Fast and intuitive interaction Easy to learn	May be hard to implement Only suitable where there is a visual metaphor for tasks and objects	Video games CAD systems
Menu selection	Avoids user error Little typing required	Slow for experienced users Can become complex if many menu options	Most general-purpose systems
Form fill-in	Simple data entry Easy to learn	Takes up a lot of screen space	Stock control, Personal loan processing
Command language	Powerful and flexible	Hard to learn Poor error management	Operating systems, Library information retrieval systems
Natural language	Accessible to casual users Easily extended	Requires more typing Natural language understanding systems are unreliable	Timetable systems WWW information retrieval systems

User interface (UI) analysis and design

User interface (UI) design analysis is **the study of how users use a particular product or system to achieve goals by performing tasks**. User analysis studies users, who they are and what tasks they might need to perform. Task analysis studies how the users actually use the product to perform their tasks.



User Interface Analysis and Design

1.Interface Analysis and Design Models

- To build an effective user interface, "all design should begin with an understanding of the intended users, including profiles of their age, gender, physical abilities, education, cultural or ethnic background, motivation, goals and personality"
- users can be categorized as:
  - Novices
  - Knowledgeable, intermittent users.
  - Knowledgeable, frequent users.

1. **Interface Design Models**
2. **Interface Design Process**

1. Interface Design Models

Four different models come into play when a user interface is analyzed and designed

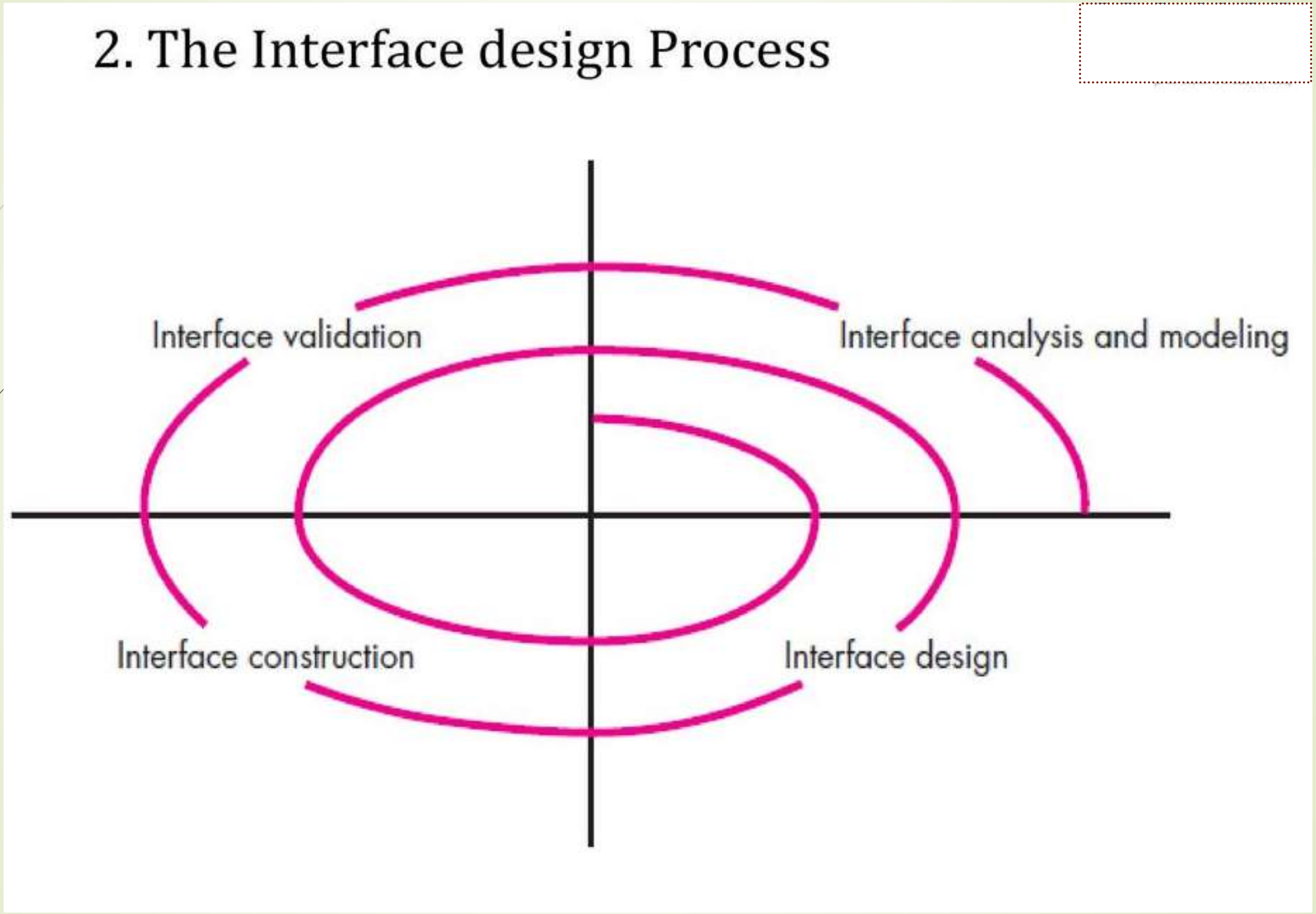
- **User profile model** – Established by a human engineer or software engineer
- **Design model** – Created by a software engineer
- **Implementation model** – Created by the software implementers
- **User's mental model** – Developed by the user when interacting with the application
- The role of the interface designer is to reconcile these differences and derive a consistent representation of the interface

2. Interface Design Process

The analysis and design process for user interfaces is iterative and can be represented using a spiral model. • Four distinct framework activities :

- **Interface analysis** focuses on the profile of the users who will interact with the system. – A more detailed task analysis is conducted. – Analysis of the user environment focuses on the physical work environment. Eg. Where will the interface be located physically? Will the user be sitting, standing, or performing other tasks unrelated to the interface?
- • **Interface design** defines a set of interface objects and actions (and their screen representations) to perform all defined tasks to meet every usability goal defined for the system. • Interface construction evaluates usage scenarios by creating prototype. As the iterative design process continues, a user interface tool kit may be used to
- • **Interface construction** evaluates usage scenarios by creating prototype. As the iterative design process continues, a user interface tool kit may be used to complete the construction of the interface.
- • **Interface validation** focuses on – User task correctness – task variations – Cover user requirement – Ease of use – Easy to learn – the users' acceptance

2. Interface Design Process…………


2. The Interface design Process

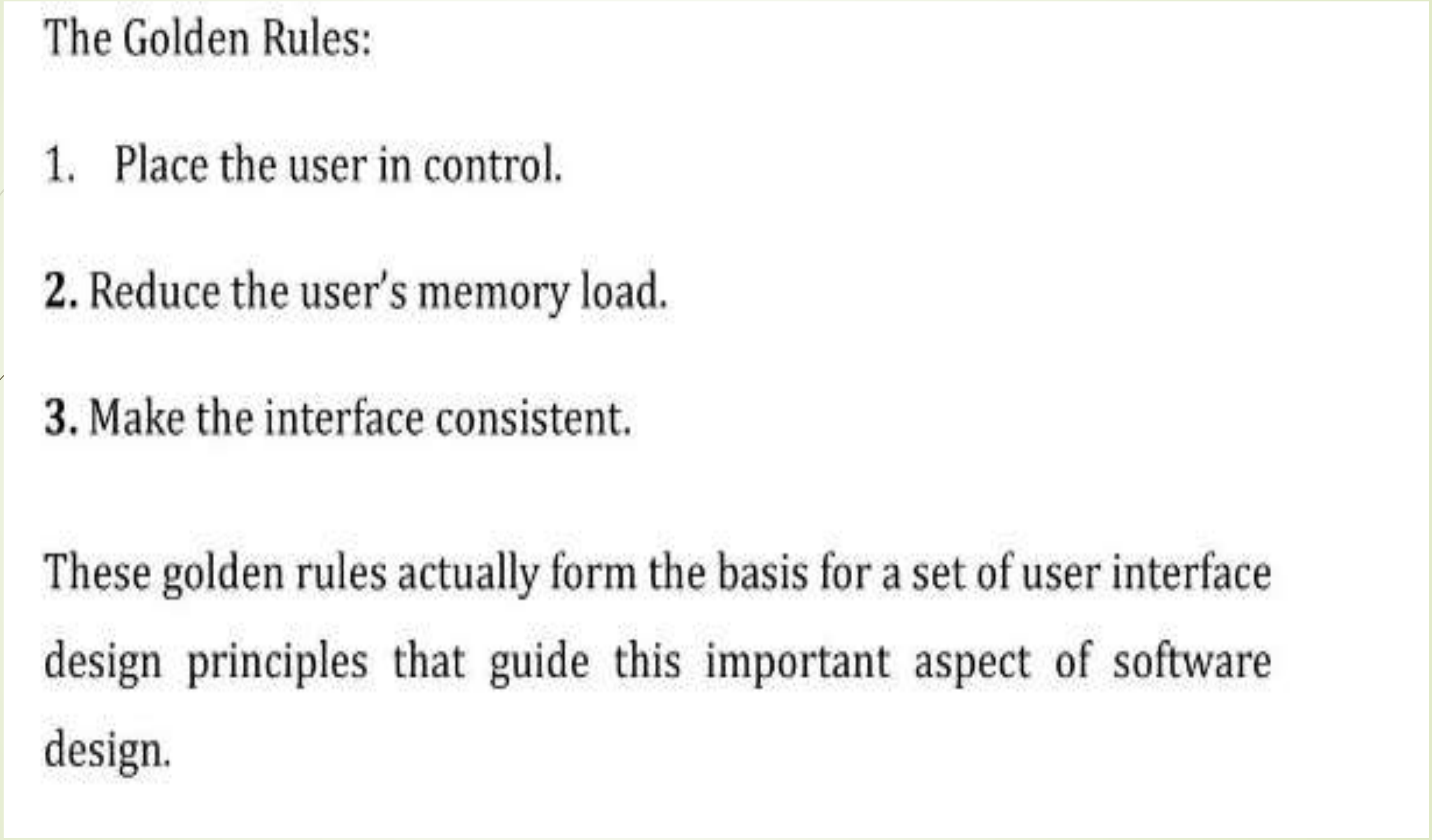
UI Design issues

1. System Response /time: -System Length and variability
2. User Help facilities:- online help , user manual or assistant form
3. Menu and command labelling:- handled by Keyboard and mouse
4. Error information handling:- Error or warning cause
5. Application accessibility
6. Internationalization

UI Design

- The definition of interface objects and the actions is an important step in interface design. Interface design, is an iterative process.
- Target, source, and application objects are identified.
- A source object (e.g., a report icon) is dragged and dropped onto a target object (e.g., a printer icon). The implication of this action is to create a hard-copy report.
- An application object represents application-specific data that are not directly manipulated as part of screen interaction.
- When all important objects and actions have been defined (for one design iteration), screen layout is performed.
- Screen layout is an interactive process in which graphical design and placement of icons, definition of descriptive screen text, specification and titling for windows, and definition of major and minor menu items are conducted.

Interface Design golden Rules



The Golden Rules:

1. Place the user in control.

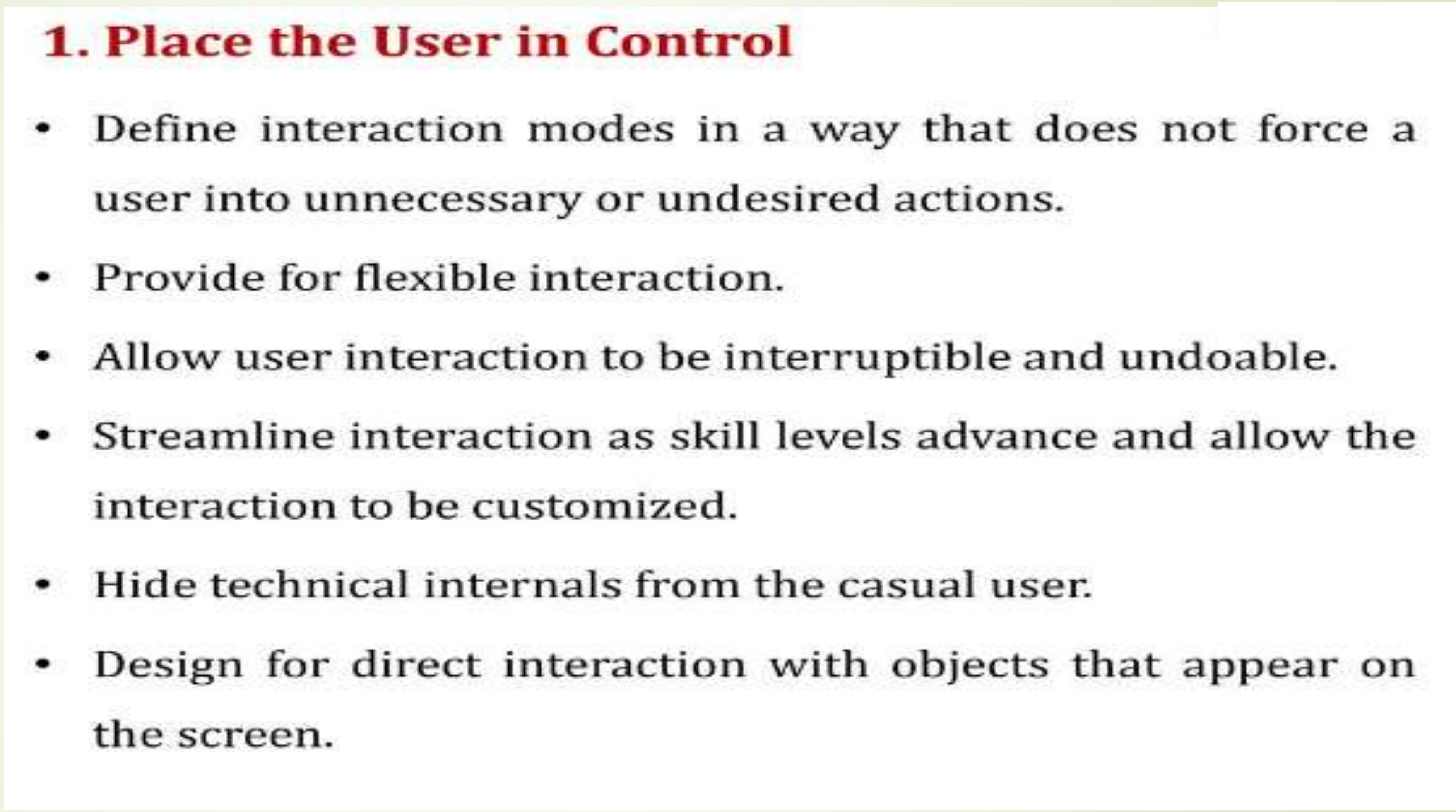
2. Reduce the user's memory load.

3. Make the interface consistent.

These golden rules actually form the basis for a set of user interface design principles that guide this important aspect of software design.

Interface Design golden Rules

1. **Modeless:**
2. **Flexible:**
3. **Interruptible:**
4. **Helpful preferences**
5. **Accessible and navigable**
6. **Interactive & facilitative**



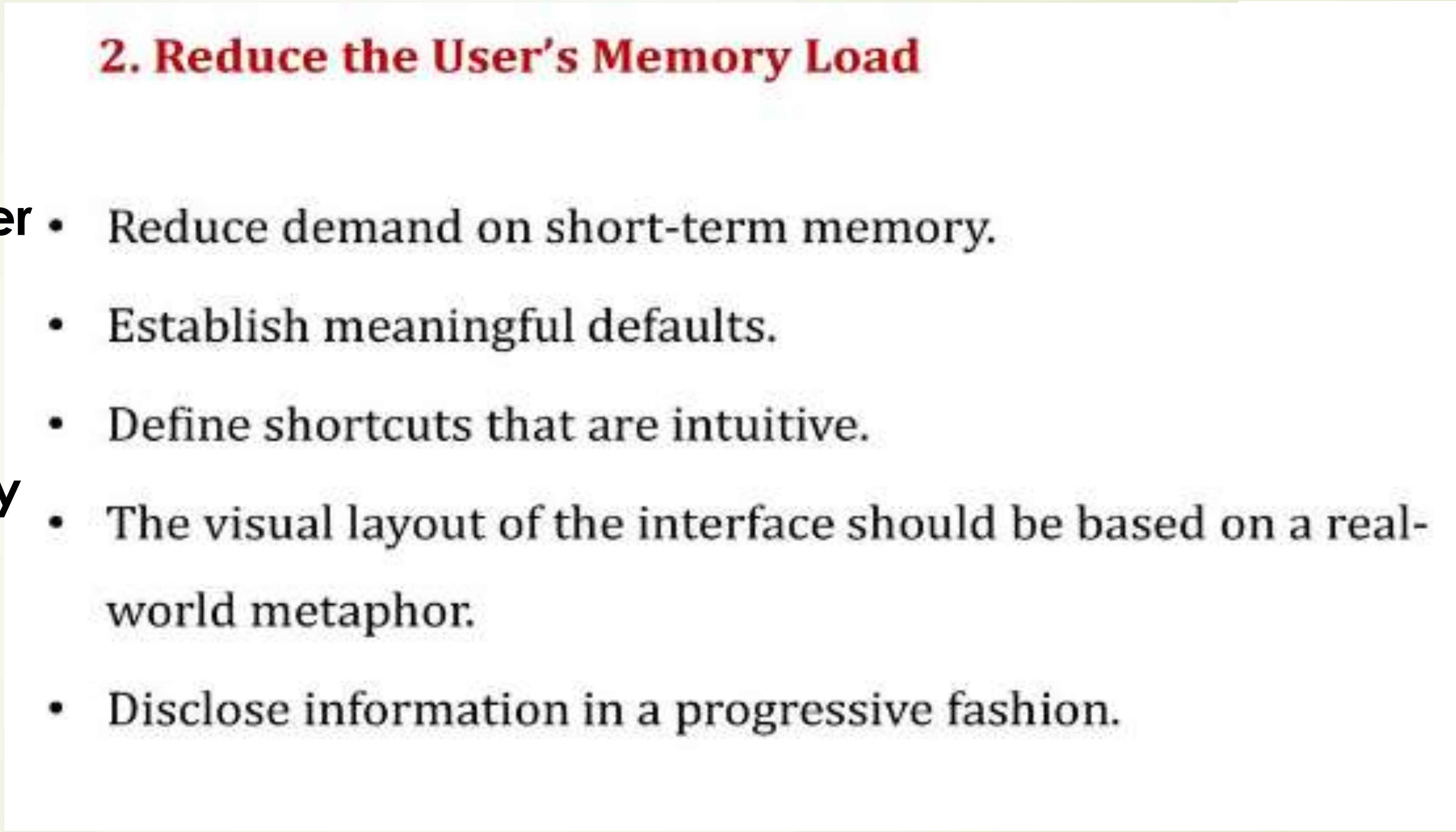
1. Place the User in Control

- Define interaction modes in a way that does not force a user into unnecessary or undesired actions.
- Provide for flexible interaction.
- Allow user interaction to be interruptible and undoable.
- Streamline interaction as skill levels advance and allow the interaction to be customized.
- Hide technical internals from the casual user.
- Design for direct interaction with objects that appear on the screen.

Interface Design golden Rules

1. Remember
2. Forgiving
3. Inform
4. Frequency
5. Context

   Organize

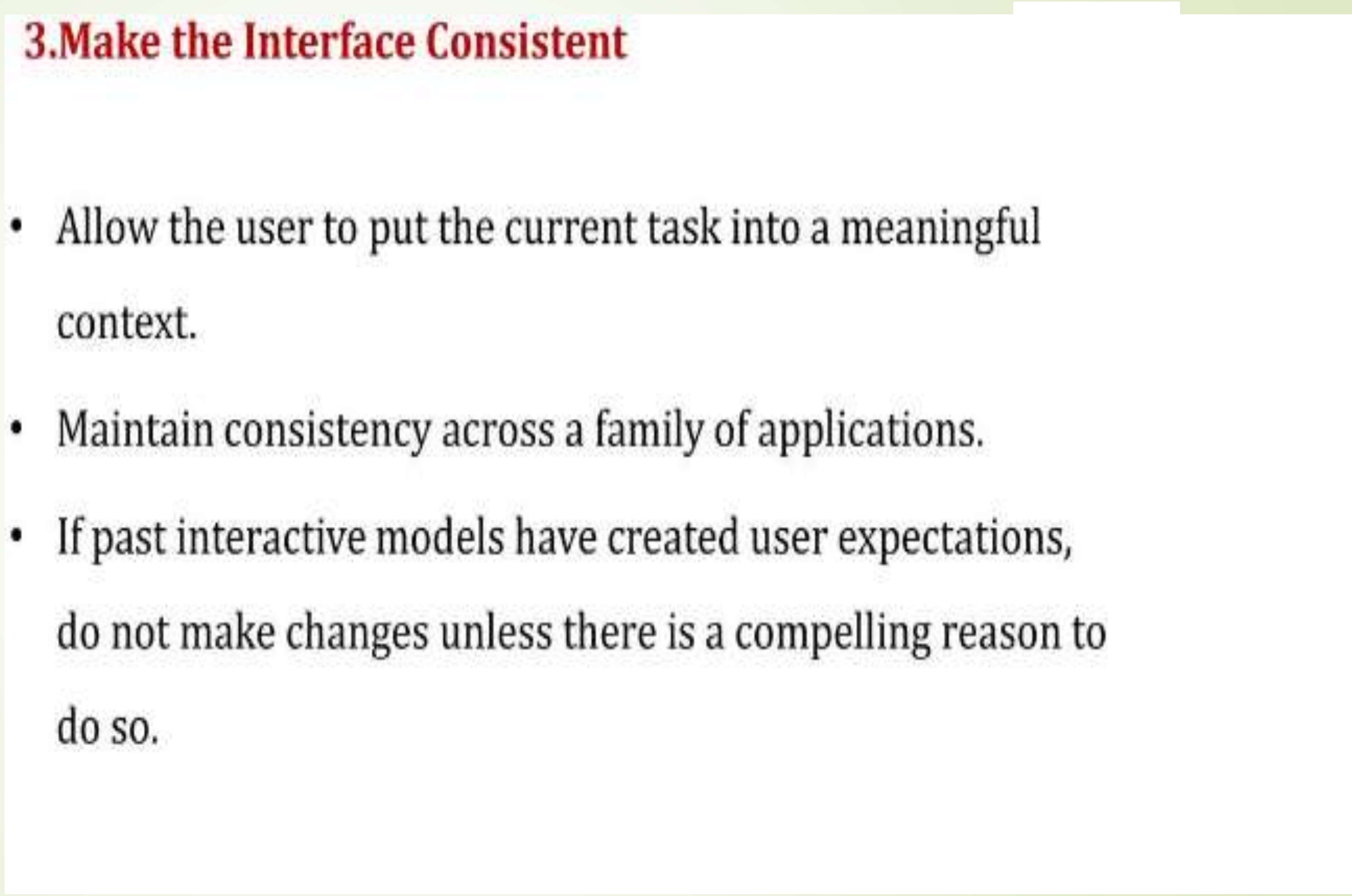


2. Reduce the User's Memory Load

- Reduce demand on short-term memory.
- Establish meaningful defaults.
- Define shortcuts that are intuitive.
- The visual layout of the interface should be based on a real-world metaphor.
- Disclose information in a progressive fashion.

Interface Design golden Rules

1. Continuity
2. Consistency
3. Predictable expectative attitude



3.Make the Interface Consistent

- Allow the user to put the current task into a meaningful context.
- Maintain consistency across a family of applications.
- If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so.

FUNCTION-ORIENTED DESIGN:

- In function-oriented design, the system is comprised of many smaller sub-systems known as functions.
- These functions are capable of performing 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.
- This design mechanism divides the whole system into smaller functions, which provides means of abstraction by concealing the information and their operation.
- It can share information among themselves by means of information passing and using information available globally.

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

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 expressed using a PDL.

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 concrete management specifications and documentation.

It focuses on the solidity, pliability, and maintainability of the system. Basically, the approach of SA/SD is based on the **Data Flow Diagram**. It is easy to understand SA/SD but it focuses on well-defined system boundary.

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

1.System
2.Process
3.Technology

SA/SD COMPONENT BASED DESIGN………..

- Structured analysis is a set of techniques and graphical tools that allow the analyst to develop a new kind of system specification that are easily understandable to the user.

Goals of SASD

- 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 involves 2 phases:

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

1. **Data Flow Diagram:**
2. 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. For example, if we have to choose between two paths of a process we can add an operator or and if two data flows are necessary for a process we can add an operator. The input of the process "check-order" needs the credit information and order information whereas the output of the process would be a cash-order or a good-credit-order.

SA/SD involves 2 phases:

2. Data Dictionary:

The content that is not described in the DFD is described in the data dictionary. It defines the data store and relevant meaning. A physical data dictionary for data elements that flow between processes, between entities, and between processes and entities may be included. This would also include descriptions of data elements that flow external to the data stores. 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 involves 2 phases:

3. State Transition Diagram:

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

4. 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 involves 2 phases:……………..

2. Design Phase:

Design Phase involves structure chart and pseudocode.

1.Structure Chart:

It is created by the data flow diagram. Structure Chart specifies how DFS's processes are grouped into tasks and allocate to the CPU. 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. Similar to other SASD tools, it is time and cost-independent and there is no error-checking technique associated with this tool. The modules of a structured chart are arranged arbitrarily and any process from a DFD can be chosen as the central transform depending on the analysts' own perception.

2.Pseudo Code:

It is the actual implementation of the system. It is an informal way of

Component Based Design:

- Component-based architecture focuses on the decomposition of the design into individual functional or logical
- components that represent well-defined communication interfaces containing methods, events, and
- properties. It provides a higher level of abstraction and divides the problem into sub-problems, each associated
- with component partitions.
- The primary objective of component-based architecture is to ensure component reusability. A component
- encapsulates functionality and behaviors of a software element into a reusable and self-deployable binary unit.

Component Based Design:

- Component-oriented software design has many advantages over the traditional object-oriented approaches
- suĐh as –
- Reduced time in market and the development cost by reusing existing components.
- Increased reliability with the reuse of the existing components.

Component Based Design:

- Component-oriented software design has many advantages over the traditional object-oriented approaches
- suĐh as –
- Reduced time in market and the development cost by reusing existing components.
- Increased reliability with the reuse of the existing components.

Principle Component Based Design:

- ❖ component-level design can be represented by using some intermediary representation (e.g. graphical, tabular, or text-based) that can be translated into source code.
- ❖ The design of data structures, interfaces, and algorithms should conform to well-established guidelines to help us avoid the introduction of errors. It has following salient features....

1. The software system is decomposed into reusable, cohesive, and encapsulated component units.
2. Each component has its own interface that specifies required ports and provided ports; each component hides its detailed implementation.

Principle Component Based Design:

3. A component should be extended without the need to make internal code or design modifications to the existing parts of the component.
4. Depend on abstractions component do not depend on other concrete components, which increase Connectors connected components, specifying and ruling the difficulty in expendability.
5. interaction among components.
6. The interaction type is specified by the interfaces of the components.
7. Components interaction can take the form of method invocations, asynchronous invocations, broadcasting, message driven interactions, data stream communications, and other protocol specific.

Principle Component Based Design:

7. Components interaction can take the form of method invocations, asynchronous invocations, broadcasting, message driven interactions, data stream communications, and other protocol specific.
8. For a server class, specialized interfaces should be created interactions. to serve major categories of clients. Only those operations that are relevant to a particular category of clients should be specified in the interface.
9. A component can extend to other components and still offer its own extension points. It is the concept of plug-in based architecture. This allows a plug-in to offer another plug-in API.

Component-Level Design Guidelines

It creates naming conventions for components that are specified as part of the architectural model and then refines or elaborates as part of the component-level model.....

- ❖ Attains architectural component names from the problem domain and ensures that they have meaning to all stakeholders who view the architectural model.
- ❖ Extracts the business process entities that can exist independently without any associated dependency on other entities.

Component-Level Design Guidelines....

- Recognizes and discover these independent entities as new components.
- Uses infrastructure component names that reflect their implementation-specific meaning.
- Models any dependencies from left to right and inheritance from top (base class) to bottom (derived classes).
- Model any component dependencies as interfaces rather than repres enting them as a direct component-to-component dependency.

DESIGN METRICS

In software development, a metric is the measurement of a particular characteristic of a program's performance or efficiency. Design metric measure the efficiency of design aspect of the software. Design model considering three aspects:

1. □ Architectural design
2. □ Object oriented design
3. □ User interface design

1. Architectural Design Metrics:

- ❖ Architectural design metrics focus on characteristics of the program architecture with an emphasis on the architectural structure and the effectiveness of modules.

- ❖ These metrics are black box in the sense that they do not require any knowledge of the inner workings of a particular software component.

2. Object Oriented Design Metrics:

There are nine measurable characteristics of object-oriented design and those are:

1. **Size:** It can be measured using following factors:

❖ Population: means total number of classes and operations.

❖ Volume: means total number of classes or operation that is collected dynamically.

❖ Length: means total number of interconnected design elements.

❖ Functionality: is a measure of output delivered to the customer.

2. Object Oriented Design Metrics:………..

2. **Complexity:** It is measured representing the characteristics that how the classes are interrelated with each other.
3. **Coupling: It** is a measure stating the collaboration between classes or number of messages that can be passed between the objects.
4. **Completeness:** It is a measure representing all the requirements of the design component.
5. **Cohesion:** It is a degree by which we can identified the set of properties that are working together to solve particular problem.

2. Object Oriented Design Metrics:……………………..

6. **Sufficiency:** It is a measure representing the necessary requirements of the design component.
7. **Primitiveness:** The degree by which the operations are simple, i.e. number of operations independent from other.
8. **Similarity**:The degree by which we measure that two or more classes are similar with respect to their functionality and behavior.
9. **Volatility:**Is the measure that represents the probability of changes that will occur.

3. User Interface Design Metrics:

Although there is significant literature on the design of human/computer interfaces, relatively little information has been published on metrics that would provide insight into the quality and usability of the interface.

1. **Layout appropriateness (LA)** is a worthwhile design metric for human/computer interfaces. A typical GUI uses layout entities—graphic icons, text, menus, windows, and the like—to assist the user in completing tasks. To accomplish a given task using a GUI, the user must move from one layout entity to the next
2. **Cohesion** metrics can be defined as the relative connection of on screen content to other screen contents. UI cohesion for screen is high.

UML:- Class Diagram

Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of object oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.
Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram.

Purpose of Class Diagrams

The purpose of class diagram is to model the static view of an application. Class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction.

The purpose of the class diagram can be summarized as −

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.
- Forward and reverse engineering.

Class Diagram…..

The following points should be remembered while drawing a class diagram −

•The name of the class diagram should be meaningful to describe the aspect of the system.

•Each element and their relationships should be identified in advance.

•Responsibility (attributes and methods) of each class should be clearly identified

•For each class, minimum number of properties should be specified, as unnecessary properties will make the diagram complicated.

•Use notes whenever required to describe some aspect of the diagram. At the end of the drawing it should be understandable to the developer/coder.

•Finally, before making the final version, the diagram should be drawn on plain paper and reworked as many times as possible to make it correct.

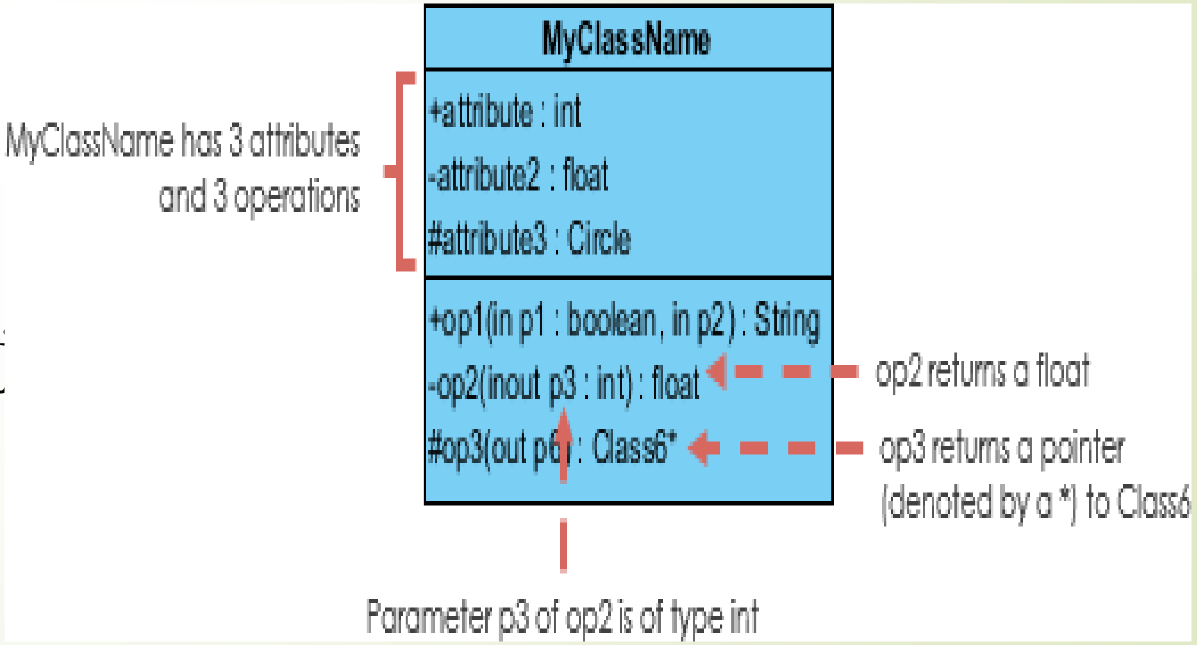
Class Diagram…..

UML Class Notation

A class represent a concept which encapsulates state (**attributes**) and behavior (**operations**). Each attribute has a type.

Each **operation** has a **signature**. *The class name is the **only** **mandatory information***.

- classes,
- their attributes,
- operations (or methods),
- and the relationships among obj



MyClassName
+attribute : int
-attribute2 : float
#attribute3 : Circle
+op1(in p1 : boolean, in p2) : String
-op2(inout p3 : int) : float
#op3(out p6) : Class6*
MyClassName has 3 attributes and 3 operations
op2 returns a float
op3 returns a pointer (denoted by a *) to Class6
Parameter p3 of op2 is of type int

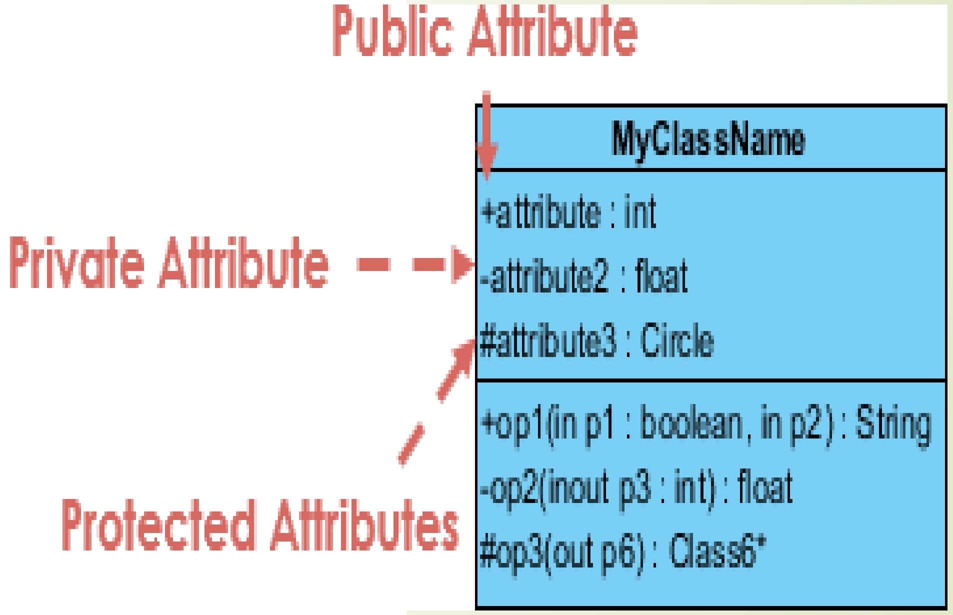
Class Diagram.....

Class Visibility

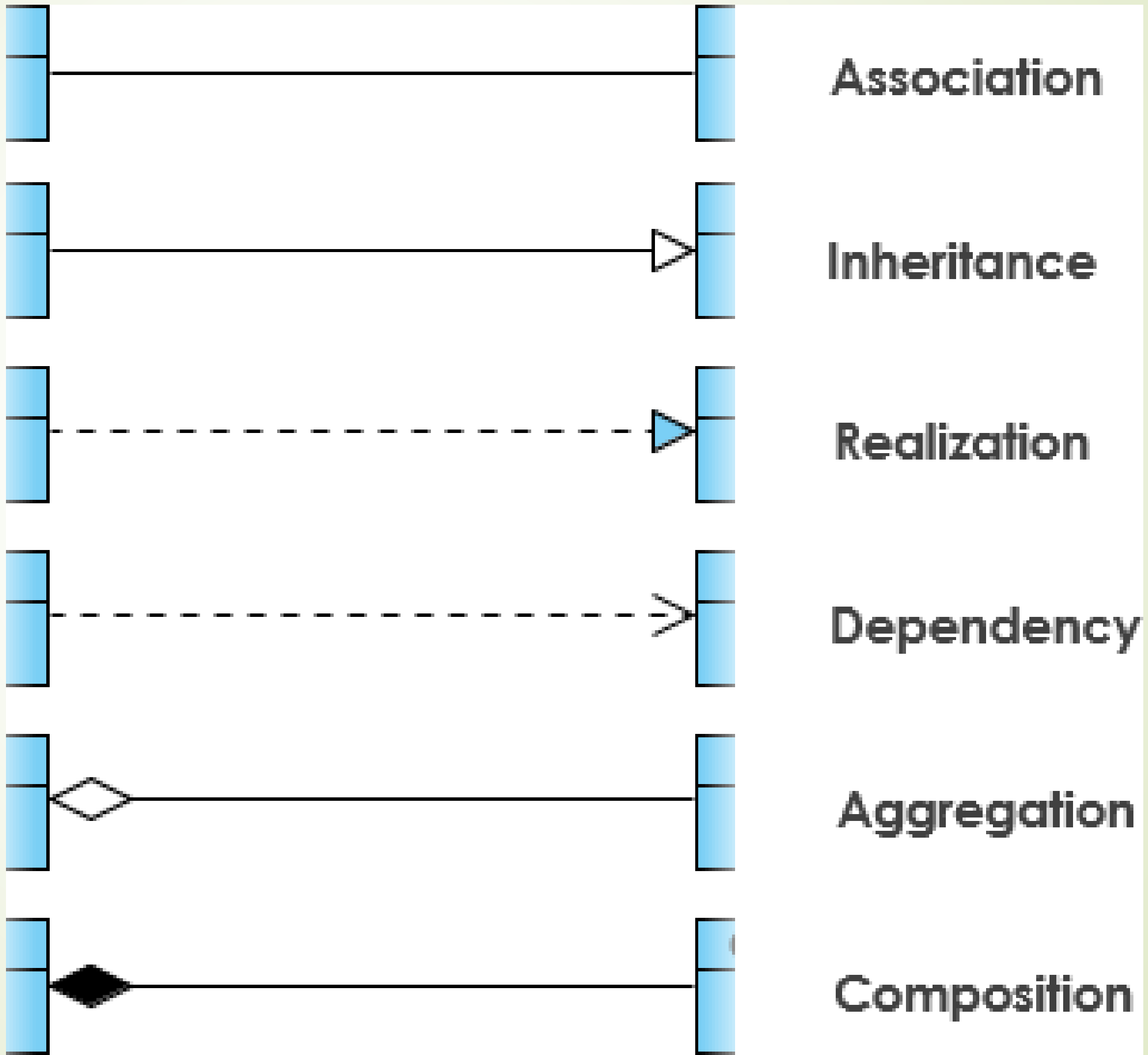
The +, - and # symbols before an attribute and operation name in a class denote the visibility of the attribute and

operation

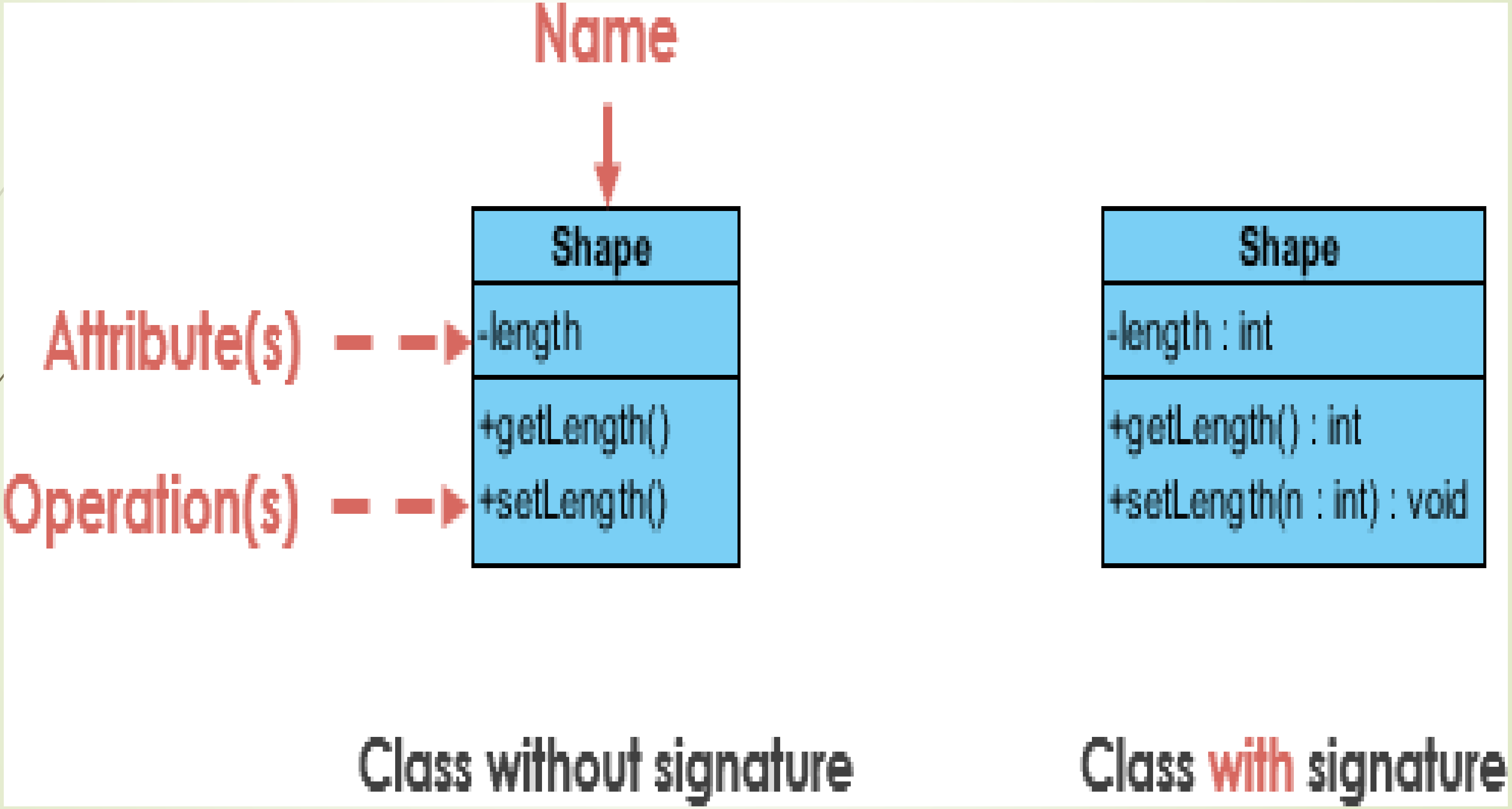
- + denotes public attributes or operations
- - denotes private attributes or operations
- # denotes protected attributes or operations



Class Diagram relationship



Class Diagram…..



Class Diagram.....

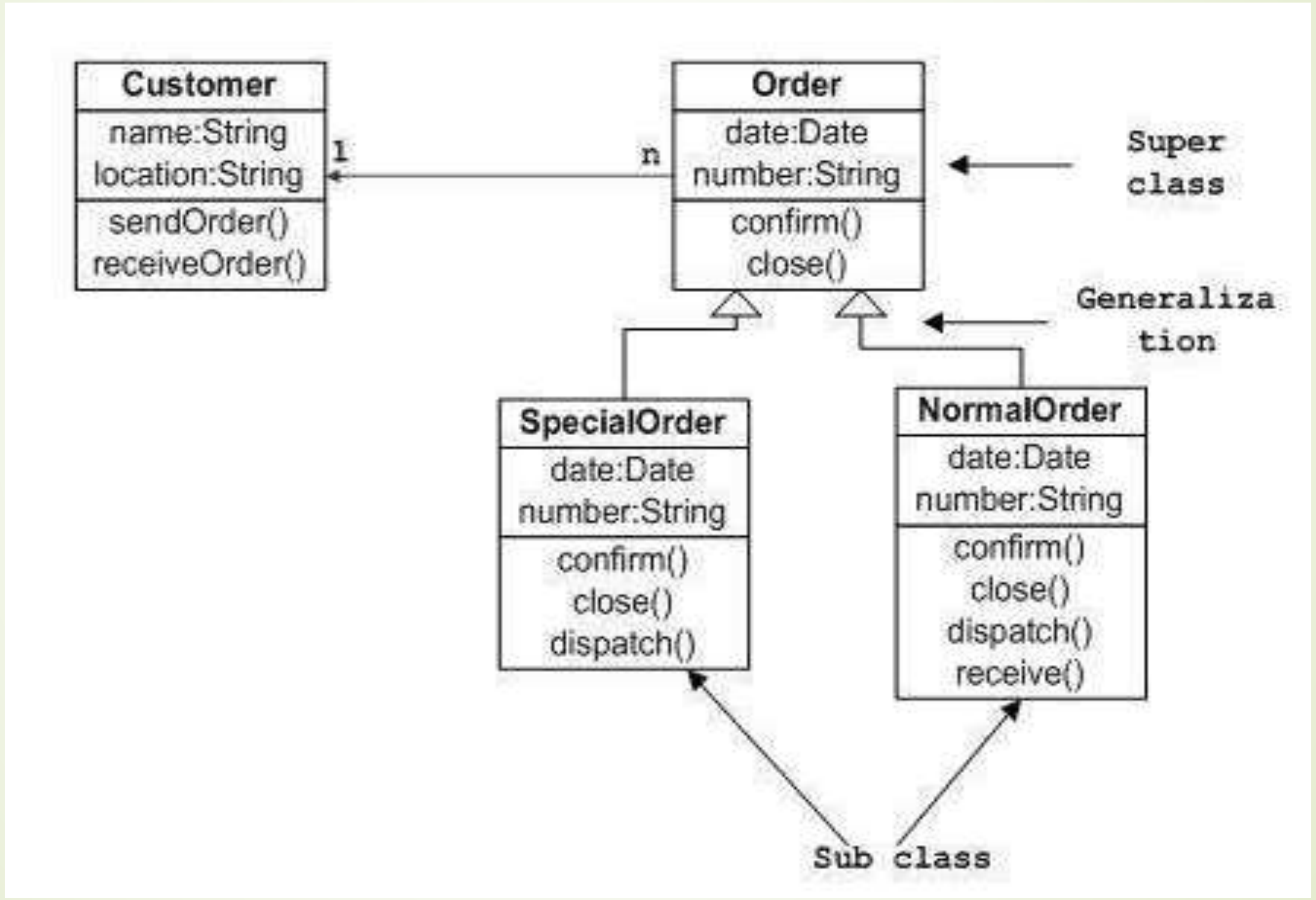
The following diagram is an example of an Order System of an application. It describes a particular aspect of the entire application.

•First of all, Order and Customer are identified as the two elements of the system. They have a one-to-many relationship because a customer can have multiple orders.

•Order class is an abstract class and it has two concrete classes (inheritance relationship) SpecialOrder and NormalOrder.

•The two inherited classes have all the properties as the Order class. In addition, they have additional functions like dispatch () and receive ().

Class Diagram for Online ordering System



State chart Diagrams

The name of the diagram itself clarifies the purpose of the diagram and other details. It describes different states of a component in a system. The states are specific to a component/object of a system.

A Statechart diagram describes a state machine. State machine can be defined as a machine which defines different states of an object and these states are controlled by external or internal events.

Activity diagram explained in the next chapter, is a special kind of a State chart diagram. As State chart diagram defines the states, it is used to model the lifetime of an object.

State chart Diagrams

Following are the main purposes of using Statechart diagrams –

•To model the dynamic aspect of a system.

•To model the life time of a reactive system.

•To describe different states of an object during its life time.

•Define a state machine to model the states of an object.

State chart Diagrams

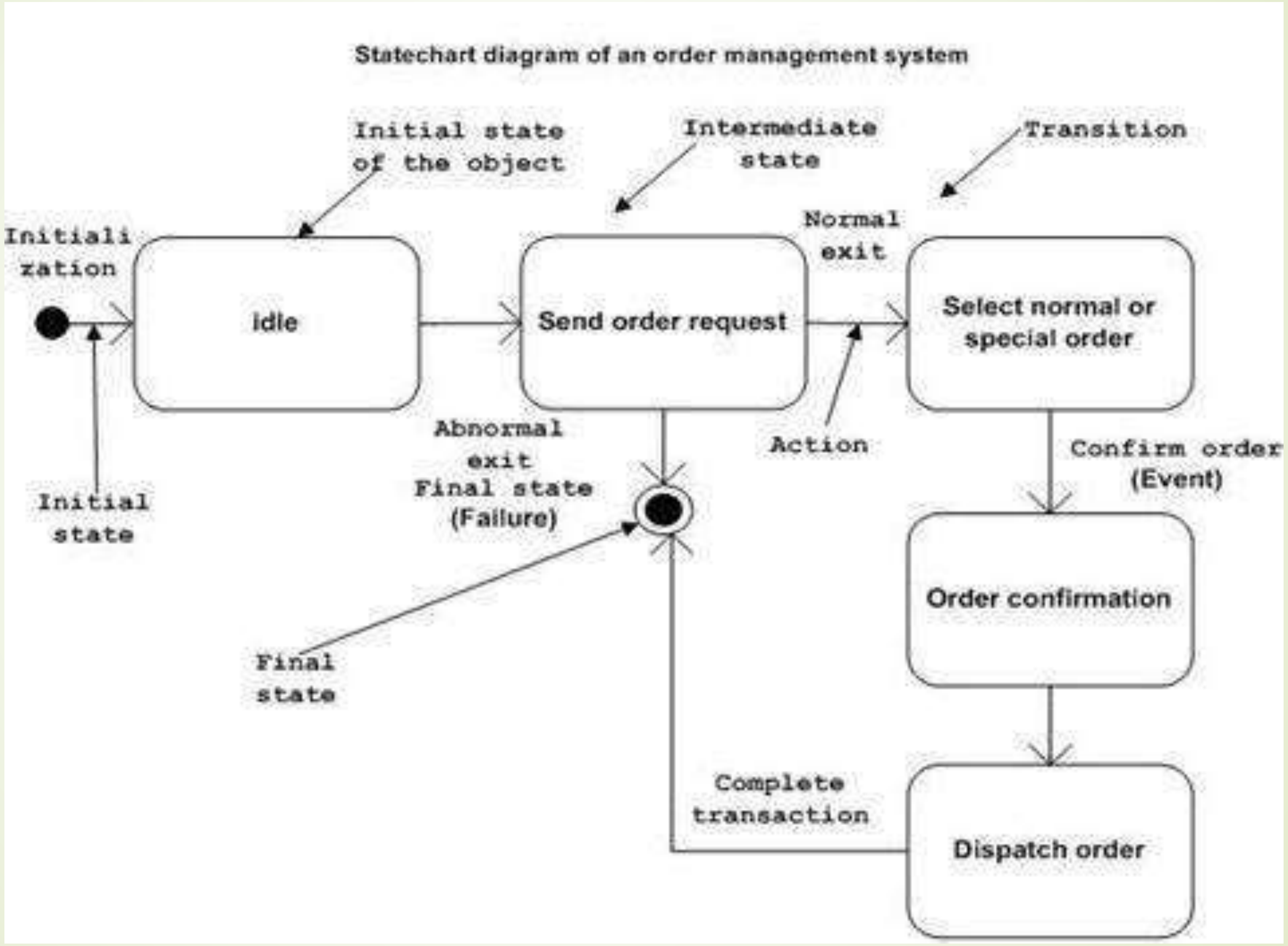
Before drawing a Statechart diagram we should clarify the following points –

•Identify the important objects to be analyzed.

•Identify the states.

•Identify the events.

State chart Diagram for Online ordering System



ANY QUERY