Software Engineering

[CACS253]

BCA 4th Sem

Er. Sital Prasad Mandal

Mechi Multiple Campus
Bhadrapur, Jhapa, Nepal

https://ctal-softwareeng.blogspot.com/

Unit 4

Software Design

1.Design Concept:

i. Abstraction
ii. Architecture
iii. Patterns
iv. Modularity
v. Cohesion & Coupling
vi. Information Hiding
vii. Functional Independence
viii.Refinement

2.Architectural Design:

i. Repository Model
ii. Client Server Model
iii. Layered Model

3.Modular Decomposition

4. Procedal Design
Using Structured Methods

5. User Interface Design:

Human-Computer Interaction

6. Information Presentation

i. Interface Evaluation
ii. Design Notation

Software Design

Software Design: is the process to transform the user requirements into some suitable form, which helps the programmer in software coding and implementation. During the software design phase, the design document is produced, based on the customer requirements as documented in the SRS document. Hence the aim of this phase is to transform the SRS document into the design document.
Software design is an iterative process through which requirements are translated into the blueprint for building the software.

The following items are designed and documented during the design phase:

- Different modules required.
- Control relationships among modules.
- Interface among different modules.
- Data structure among the different modules.
- Algorithms required to implement among the individual modules.

Software Quality Guidelines

- A design is generated using the recognizable architectural styles and compose a good design characteristic of components and it is implemented in evolutionary manner for testing.
- A design of the software must be modular i.e the software must be logically partitioned into elements.
- In design, the representation of data , architecture, interface and components should be distinct.
- A design must carry appropriate data structure and recognizable data patterns.
- Design components must show the independent functional characteristic.
- A design creates an interface that reduce the complexity of connections between the components.
- A design must be derived using the repeatable method.
- The notations should be use in design which can effectively communicates its meaning.

Quality attributes

The attributes of design name as 'FURPS' are as follows:

Functionality: It evaluates the feature set and capabilities of the program.

Usability: It is accessed by considering the factors such as human factor, overall aesthetics, consistency and documentation.

Reliability: It is evaluated by measuring parameters like frequency and security of failure, output result accuracy, the mean-time-to-failure(MTTF), recovery from failure and the program predictability.

Performance: It is measured by considering processing speed, response time, resource consumption, throughput and efficiency.

Supportability:

- It combines the ability to extend the program, adaptability, serviceability. These three term defines the maintainability.
- Testability, compatibility and configurability are the terms using which a system can be easily installed and found the problem easily.
- Supportability also consists of more attributes such as compatibility, extensibility, fault tolerance, modularity, reusability, robustness, security, portability, scalability.

Objectives of Software Design:

Correctness: A good design should be correct i.e. it should correctly implement all the functionalities of the system.

Efficiency: A good software design should address the resources, time, and cost optimization issues.

Understandability: A good design should be easily understandable, for which it should be modular and all the modules are arranged in layers.

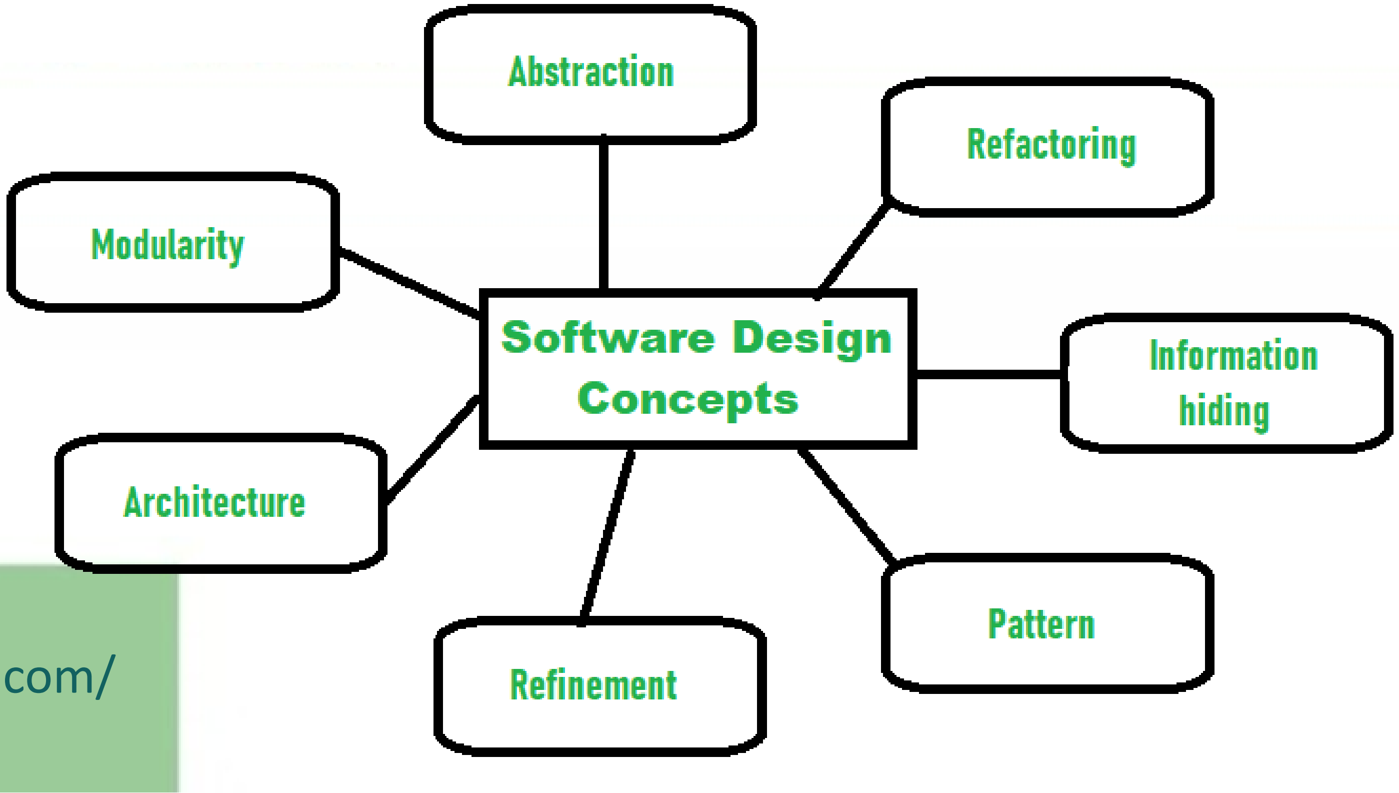
Completeness: The design should have all the components like data structures, modules, and external interfaces, etc.

Maintainability: A good software design should be easily amenable to change whenever a change request is made from the customer side.

[Mock Up] Design Concepts

- ❑ Concepts are defined as a principal idea or invention that comes into our mind or in thought to understand something.
- ❑ The **software design concept** simply means the idea or principle behind the design. It describes how you plan to solve the problem of designing software, the logic, or thinking behind how you will design software. It allows the software engineer to create the model of the system or software or product that is to be developed or built.
- ❑ The software design concept provides a supporting and essential structure or model for developing the right software.

The following points should be considered while designing Software:



Software Design Concepts

1. Abstraction - hide Irrelevant data

- ✓ A solution is stated in large terms using the language of the problem environment at the highest level abstraction.
- ✓ The lower level of abstraction provides a more detail description of the solution.
- ✓ A sequence of instruction that contain a specific and limited function refers in a procedural abstraction.
- ✓ A collection of data that describes a data object is a data abstraction.
- ✓ *Here, there are two common abstraction mechanisms*

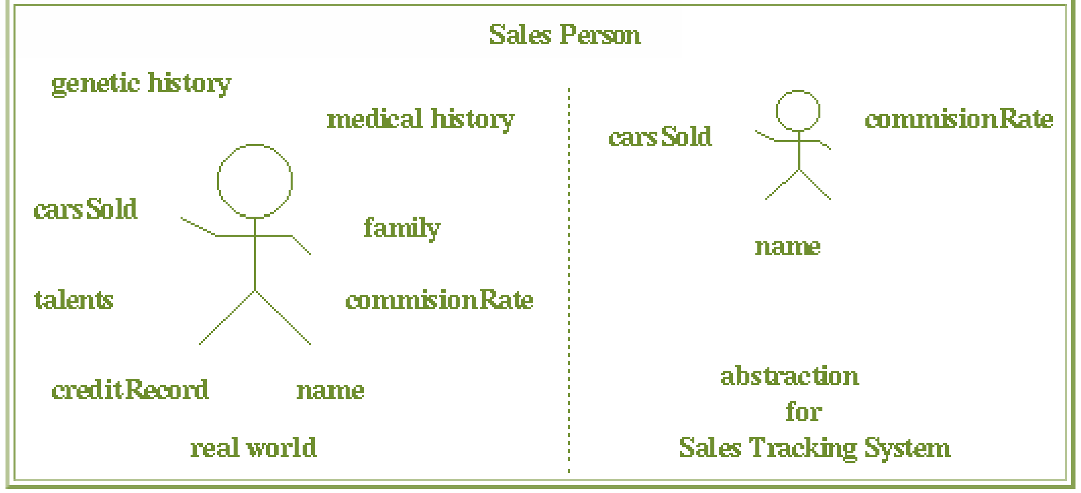
Functional Abstraction

- ✓ 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.
- ✓ Functional abstraction forms the basis for **Function oriented design approaches**.

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**.

Figure : Abstraction of a SalesPerson



Software Design Concepts

2. Architecture - design a structure of something

- ✓ The complete structure of the software is known as software architecture.
- ✓ Structure provides conceptual integrity for a system in a number of ways.
- ✓ The architecture is the structure of program modules where they interact with each other in a specialized way.
- ✓ The components use the structure of data.
- ✓ The aim of the software design is to obtain an architectural framework of a system.
- ✓ The more detailed design activities are conducted from the framework.

3. Patterns - a repeated form

A design pattern describes a design structure and that structure solves a particular design problem in a specified content.

4. Information hiding - hide the information

Modules must be specified and designed so that the information like algorithm and data presented in a module is not accessible for other modules not requiring that information.

Software Design Concepts

5. Modularity - subdivide the system

- ✓ A software is separately divided into name and addressable components. Sometime they are called as modules which integrate to satisfy the problem requirements.
- ✓ Modularity is the single attribute of a software that permits a program to be managed easily.

The desirable properties of a modular system are:

- ○ Each module is a well-defined system that can be used with other applications.
- ○ Each module has single specified objectives.
- ○ Modules can be separately compiled and saved in the library.
- ○ Modules should be easier to use than to build.
- ○ Modules are simpler from outside than inside.

Advantages of Modularity

- ✓ It allows large programs to be written by several or different people
- ✓ It encourages the creation of commonly used routines to be placed in the library and used by other programs.

DisAdvantages of Modularity

- ✓ Inter-module communication problems may be increased
- ✓ More linkage required, run-time may be longer

Software Design Concepts

6. Functional independence:

- ✓ The functional independence is the concept of separation and related to the concept of modularity, abstraction and information hiding.
- ✓ The functional independence is accessed using two criteria i.e Cohesion and coupling.
- ✓ ***It is measured using two criteria:***

Cohesion

- ✓ Cohesion is an extension of the information hiding concept.
- ✓ A cohesive module performs a single task and it requires a small interaction with the other components in other parts of the program.

Coupling

Coupling is an indication of interconnection between modules in a structure of software.

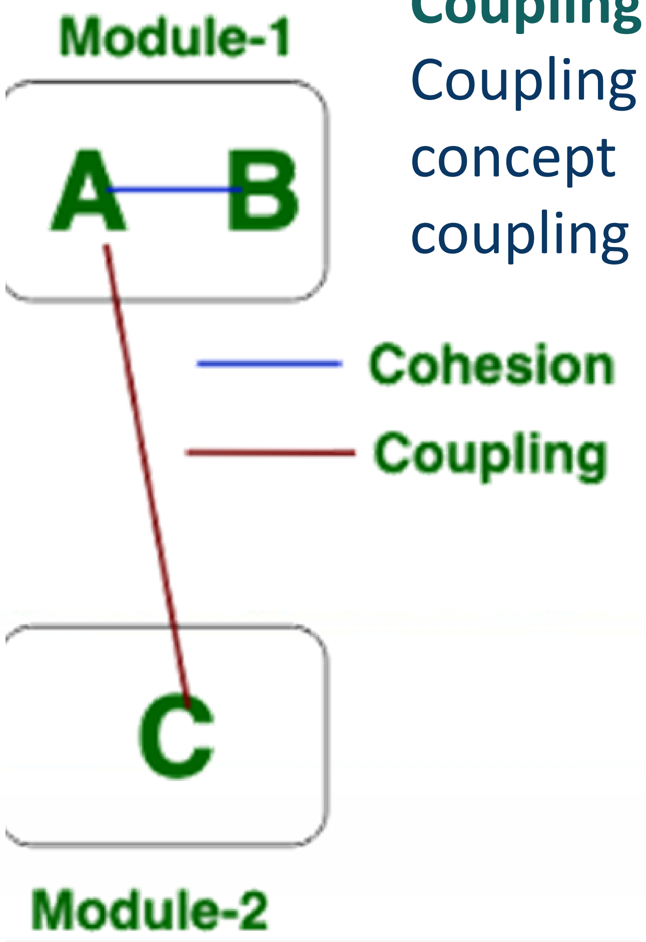
Cohesion & Coupling

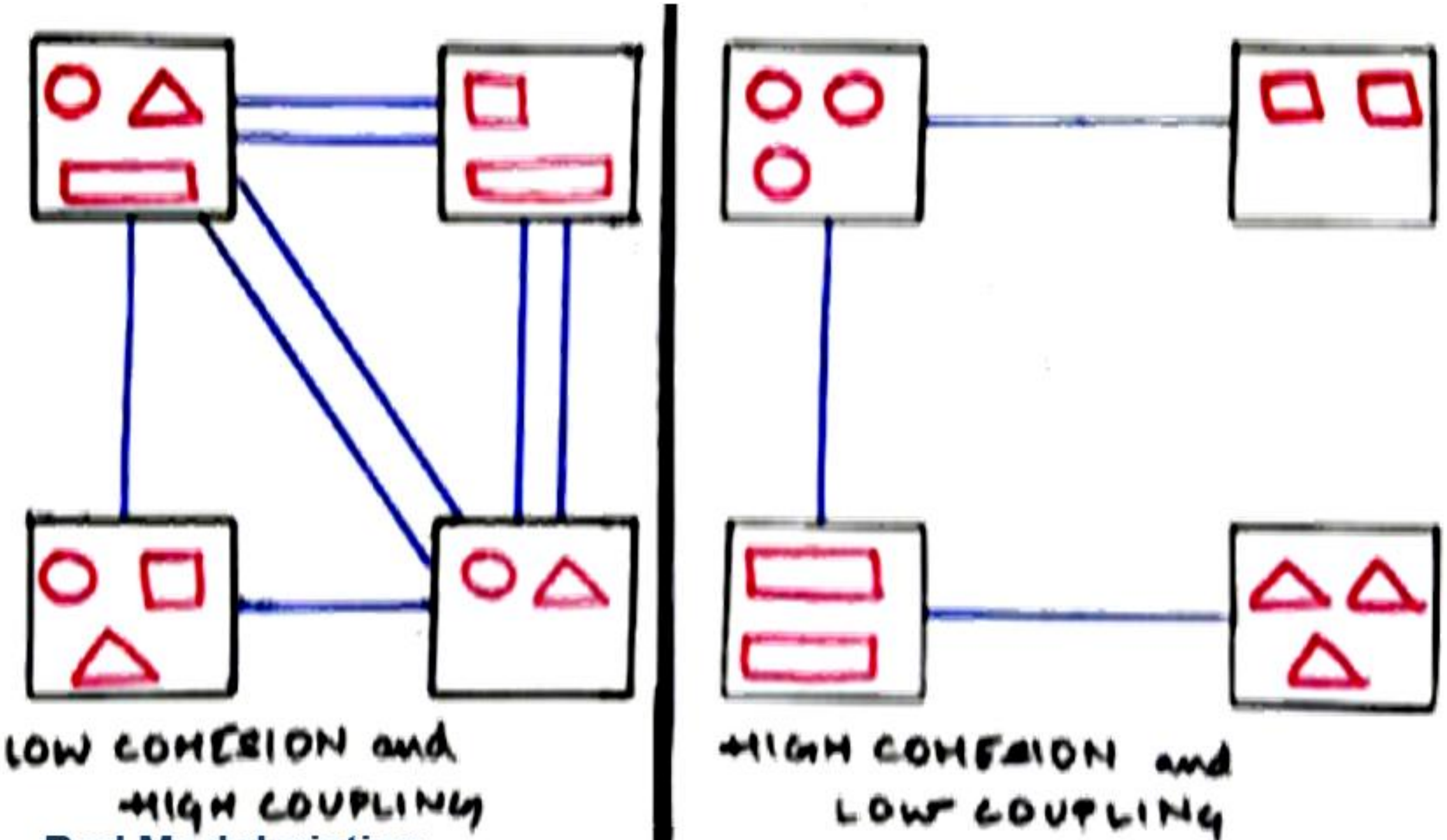
Cohesion:

Cohesion is the indication of the relationship within module. It is concept of intra-module. Cohesion has many types but usually highly cohesion is good for software.

Coupling:

Coupling is also the indication of the relationships between modules. It is concept of Inter-module. Coupling has also many types but usually low coupling is good for software.



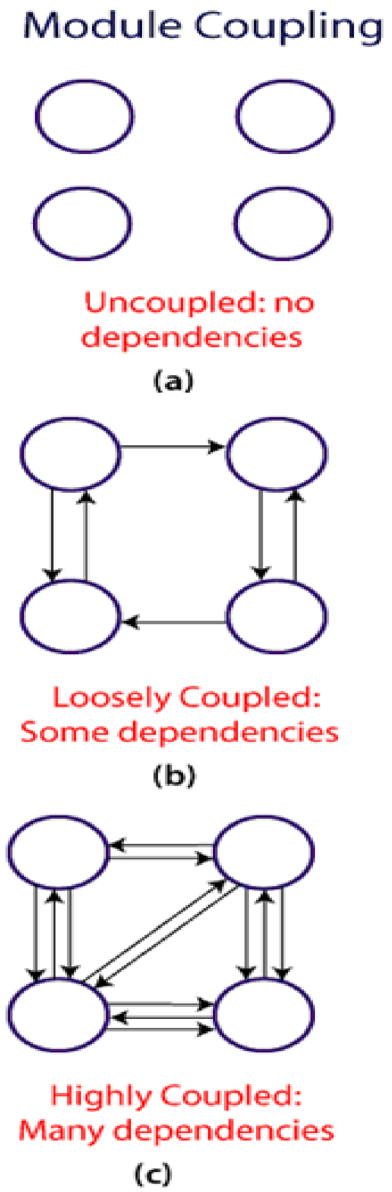

LOW COHESION and HIGH COUPLING
HIGH COHESION and LOW COUPLING

Bad Modulariation

Good Modulariation

Cohesion & Coupling

***Coupling** is about how much one module depends or interacts with other modules.*

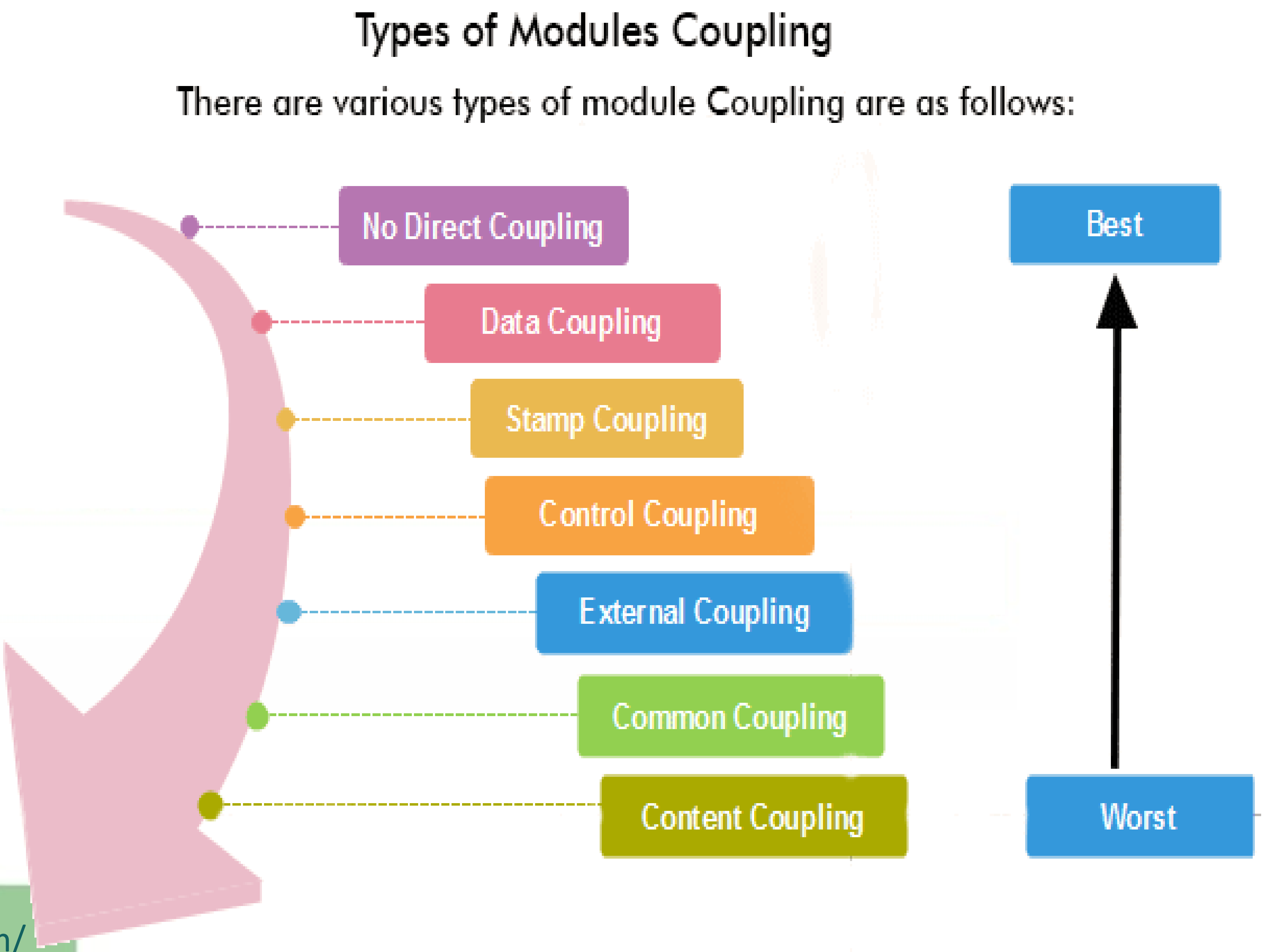


Module Coupling

Uncoupled: no dependencies
(a)

Loosely Coupled: Some dependencies
(b)

Highly Coupled: Many dependencies
(c)



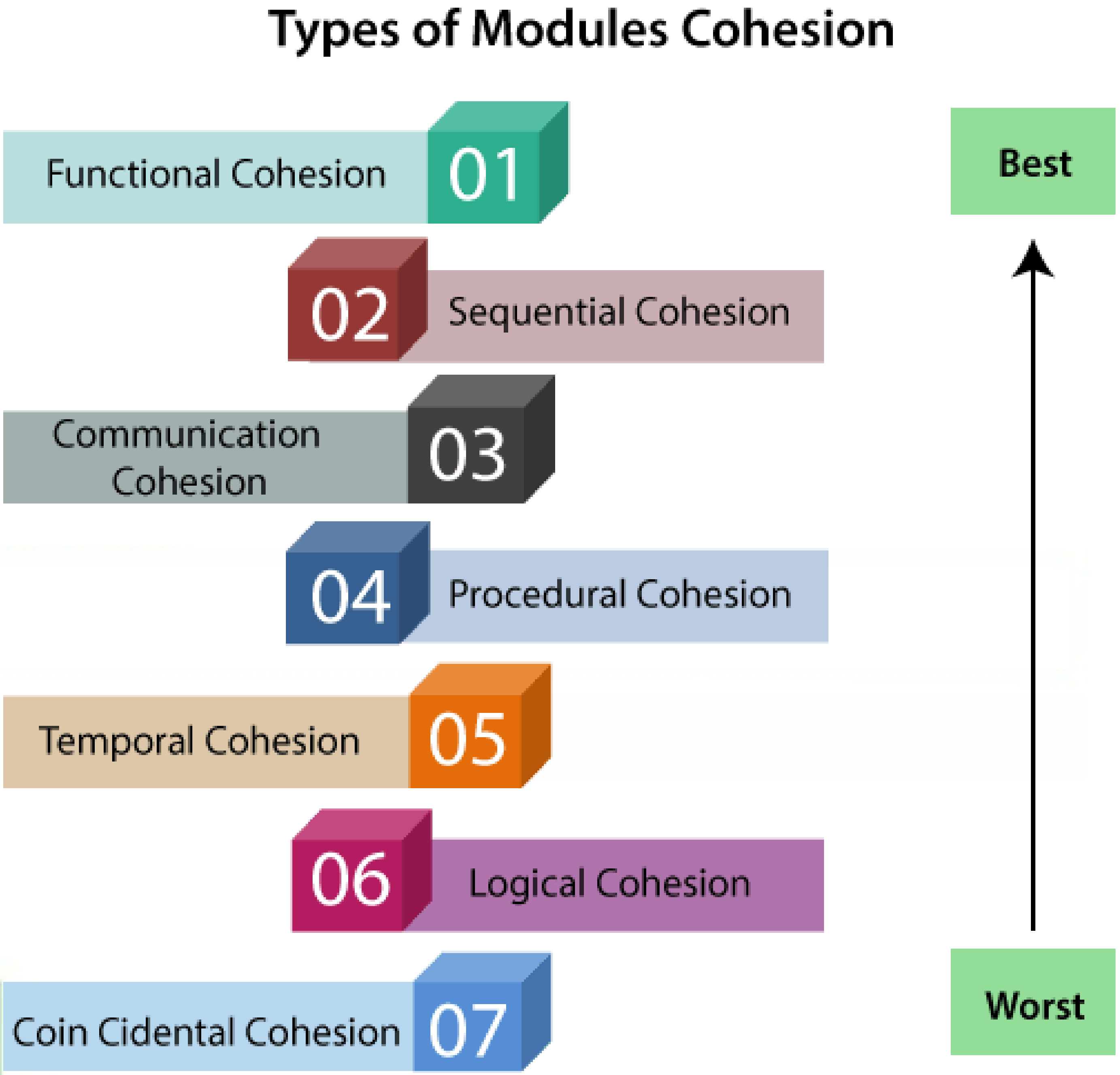
Types of Modules Coupling

There are various types of module Coupling are as follows:

- No Direct Coupling
- Data Coupling
- Stamp Coupling
- Control Coupling
- External Coupling
- Common Coupling
- Content Coupling

Cohesion & Coupling

Cohesion *is about how well elements within a module belong together and serve a common purpose.*



https://ctal-softwareeng.blogspot.com/

Assignment (v v imp.)

Cohesion	Coupling
Cohesion is the concept of **intra module.**	Coupling is the concept of **inter module.**
Cohesion represents the relationship **within** module.	Coupling represents the relationships **between** modules.
Increasing in cohesion is **good** for software.	Increasing in coupling is **avoided** for software.
Cohesion represents the **functional strength** of modules.	Coupling represents the **independence** among modules.
Highly cohesive gives the **best** software.	Where as **loosely coupling** gives the **best** software.
In cohesion, module focuses on the **single thing.**	In coupling, modules are **connected to the other modules**.

Software Design Concepts

7. Refinement - removes impurities

Refinement is a top-down design approach.

- ✓ It is a process of elaboration.
- ✓ A program is established for refining levels of procedural details.
- ✓ A hierarchy is established by decomposing a statement of function in a stepwise manner till the programming language statement are reached.

8. Refactoring

- ✓ It is a reorganization technique which simplifies the design of components without changing its function behaviour.
- ✓ Refactoring is the process of changing the software system in a way that it does not change the external behaviour of the code still improves its internal structure.

9. Design classes

- ✓ The model of software is defined as a set of design classes.
- ✓ Every class describes the elements of problem domain and that focus on features of the problem which are user visible.

Architectural Design

- The design process for identifying the sub-systems making up a system and the framework for sub-system control and communication is **Architectural Design.**
- The output of this design process is a description of the **software architecture.**

- Architectural design is an early stage of the system design process.
- It represents the link between specification and design processes and is often carried out in parallel with some specification activities.
- It involves identifying major system components and their communications.

Architectural Design

Advantages of explicit architecture

- Stakeholder communication

  Architecture may be used as a focus of discussion by system stakeholders.
- System analysis

  Means that analysis of whether the system can meet its non-functional requirements is possible.
- Large-scale reuse

  The architecture may be reusable across a range of systems.

Architectural Design

Architecture and system characteristics

- **Performance**
  Localize critical operations and minimize communications. Use large rather than fine-grain components.
- **Security**
  Use a layered architecture with critical assets in the inner layers.
- **Safety**
  Localize safety-critical features in a small number of sub-systems.
- **Availability**
  Include redundant components and mechanisms for fault tolerance.
- **Maintainability**
  Use fine-grain, replaceable components.

Architectural Design

Architectural Design Decisions

- Is there a generic application architecture that can be used?
- How will the system be distributed?
- What architectural styles are appropriate?
- What approach will be used to structure the system?
- How will the system be decomposed into modules?
- What control strategy should be used?
- How will the architectural design be evaluated?
- How should the architecture be documented?

Architectural Design

System organization

- Reflects the basic strategy that is used to structure a system.
- *Three organisational styles are widely used:*
  - A shared data repository style.
  - A shared services and servers style.
  - An abstract machine or layered style.

i. Repository Model
ii. Client Server Model
iii. Layered Model

System organization

Repository Model

A repository model is a system that will allow interfacing sub-systems to share the same data. Sub-system must exchange data so that they can work together effectively.

This may be done in two ways:

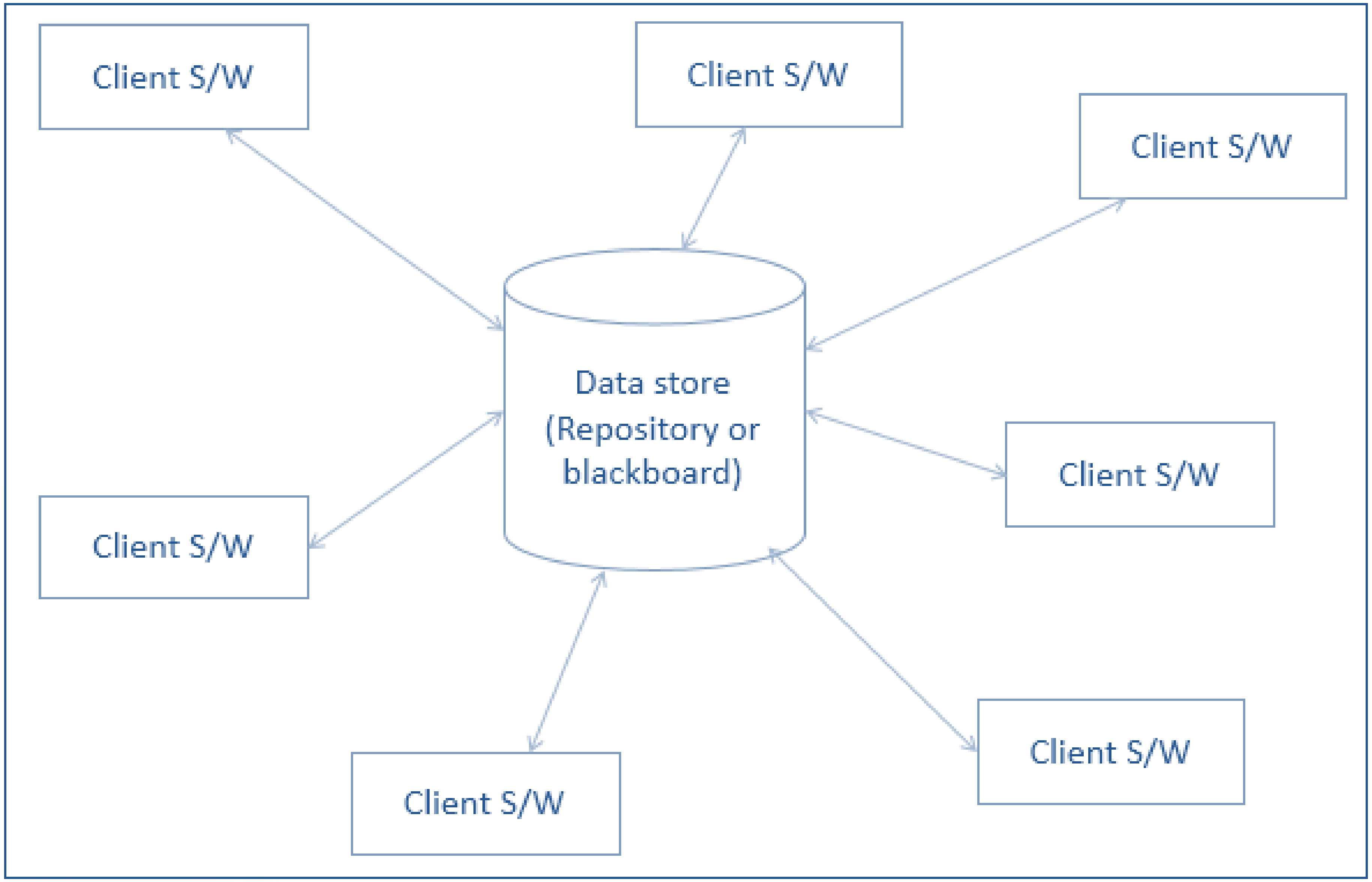
1. All **shared data is held in a central database** that can be accessed by all subsystems. It is called repository model.

2. Each **sub-system maintains its own database**. Data is interchanged with other sub-systems by passing messages to them.

System organization

Repository Model

Example: Repository Model

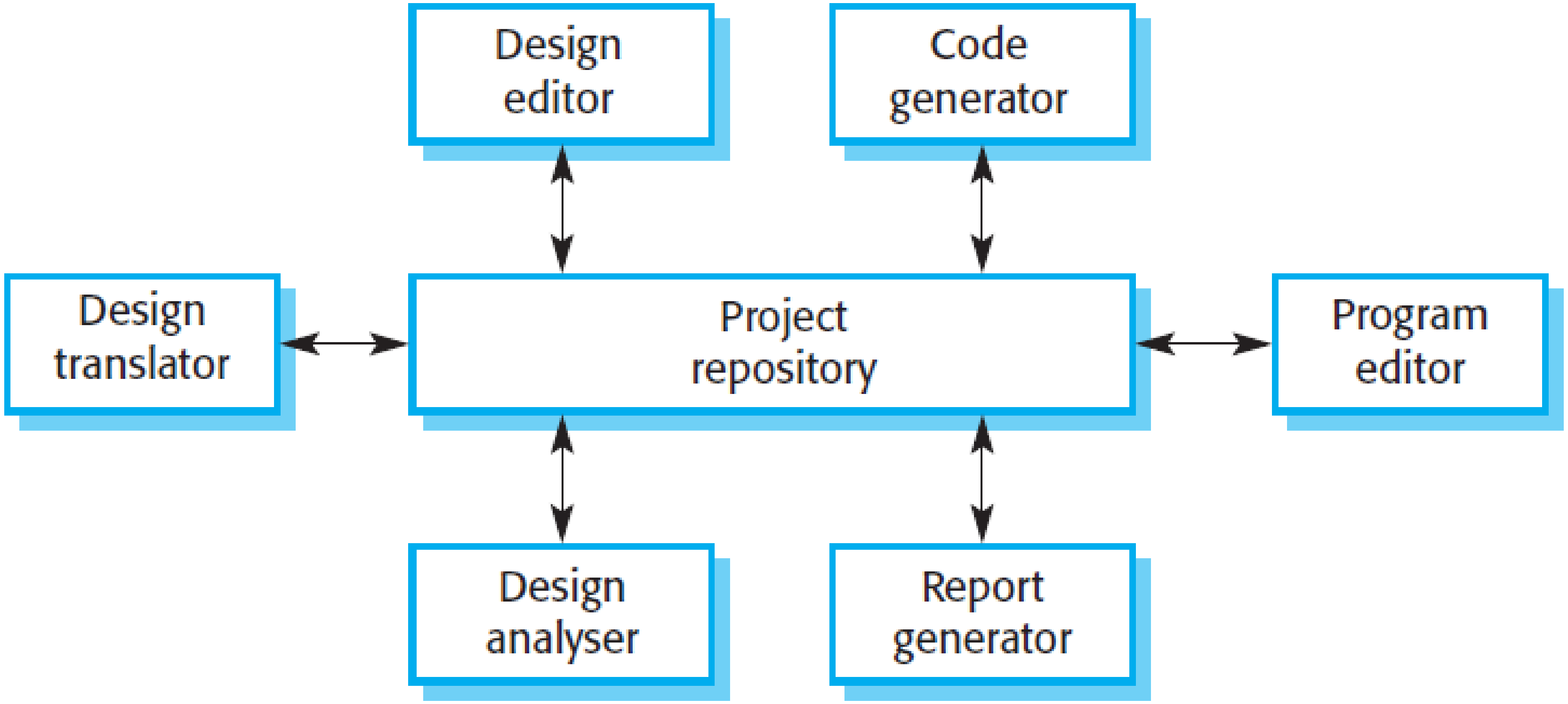


Repository Model

System organization

Repository Model

Example: CASE Toolset



Integrate CASE Toolset

System organization

Repository Model characteristics

Advantages

– Efficient way to share large amounts of data.

– Sub-systems need not be concerned with how data is produced Centralised management e.g. backup, security, etc.

– Sharing model is published as the repository schema.

Disadvantages

– Sub-systems must agree on a repository data model. Naturally a compromise.

– Data evolution is difficult and expensive.

– No scope for specific management policies.

– Difficult to distribute efficiently.

System organization

Client Server Model

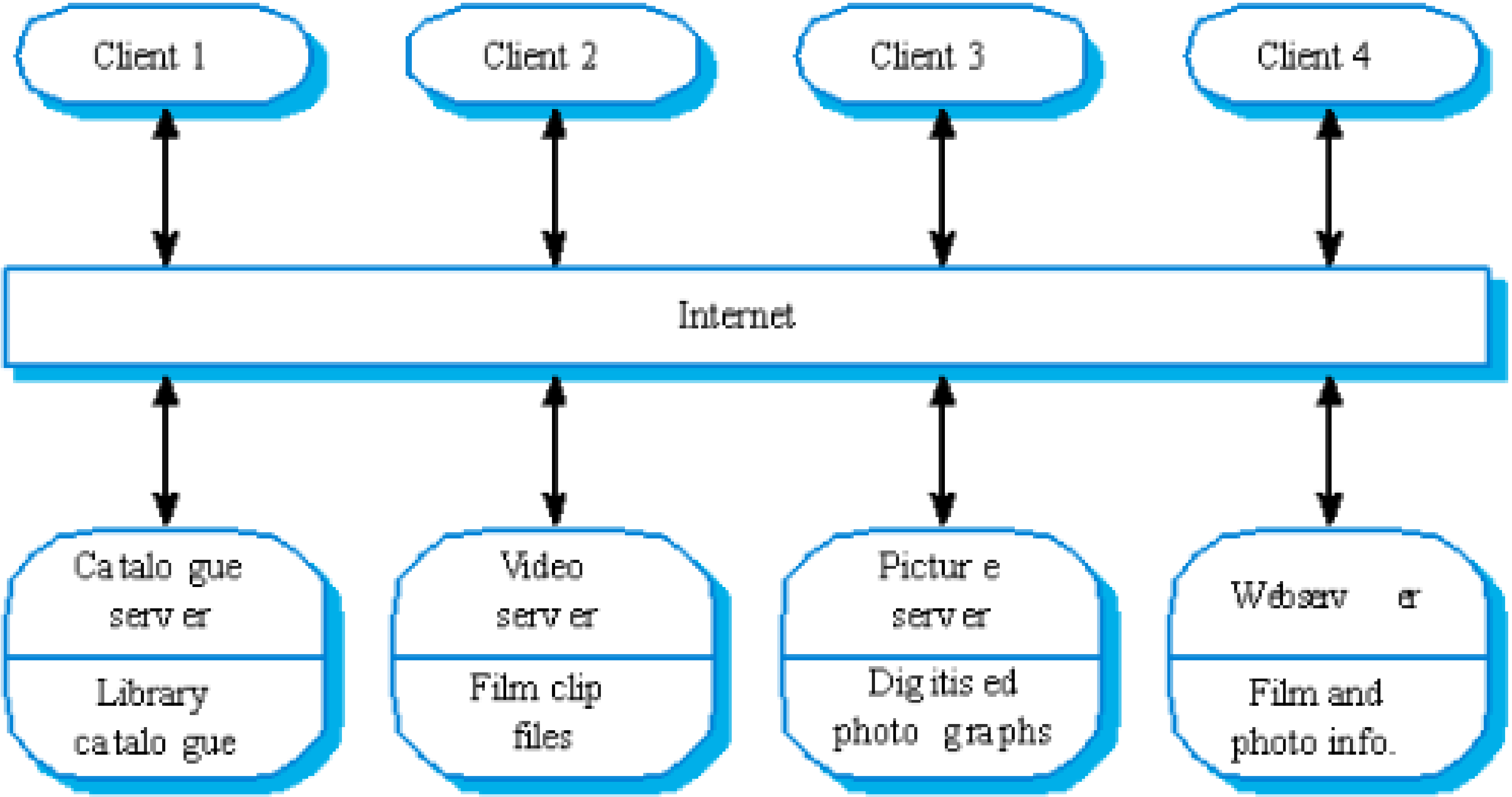
- The client-server architectural model is a system model where the system is organised as a set of services and associated servers and clients that access and use the services
- It is a distributed system model which shows how data and processing is distributed across a range of components.

The major components of this model are

- Set of stand-alone servers which provide specific services such as printing, data management, etc.
- Set of clients which call on these services.
- Network which allows clients to access servers.

System organization

Client Server Model



Film and picture library

System organization

Client Server Model ***characteristics***

Advantages

– Distribution of data is straightforward;

– Makes effective use of networked systems. May require cheaper hardware;

– Easy to add new servers or upgrade existing servers.

Disadvantages

– No shared data model so sub-systems use different data organisation. Data interchange may be inefficient;

– Redundant management in each server;

– No central register of names and services; It may be hard to find out what servers and services are available.

System organization

Layered Model

The layered model sometimes called an abstract machine .

- The layered model organises a system into layers, each of which provide a set of services
- Each layer can be thought of as an abstract machine whose machine language is defined by the services provided by the layer
- This language is used to implement the next level of abstract machine

System organization

Layered Model

For Example

- ○ To implement a programming language
- ○ The first layer is high level language
- ○ Second layer is the compilation layer
- ○ Third layer is translation the compiled language to machine code

When a layer interface changes, only the adjacent layer is affected.

System organization

Layered Model

- Used to model the interfacing of sub- systems
- The layered approach supports the incremental development of sub-systems in different layers
- An Example of layered Model
  - Open System Interconnections OSI reference model of network protocol 1980
  - Ada Programming Support Environment APSE 1980

System organization

Layered Model (Information systems architecture)

- Information systems have a generic architecture that can be organised as a layered architecture.
- ***Layers include:***
  - The user interface
  - User communications
  - Information retrieval
  - System database

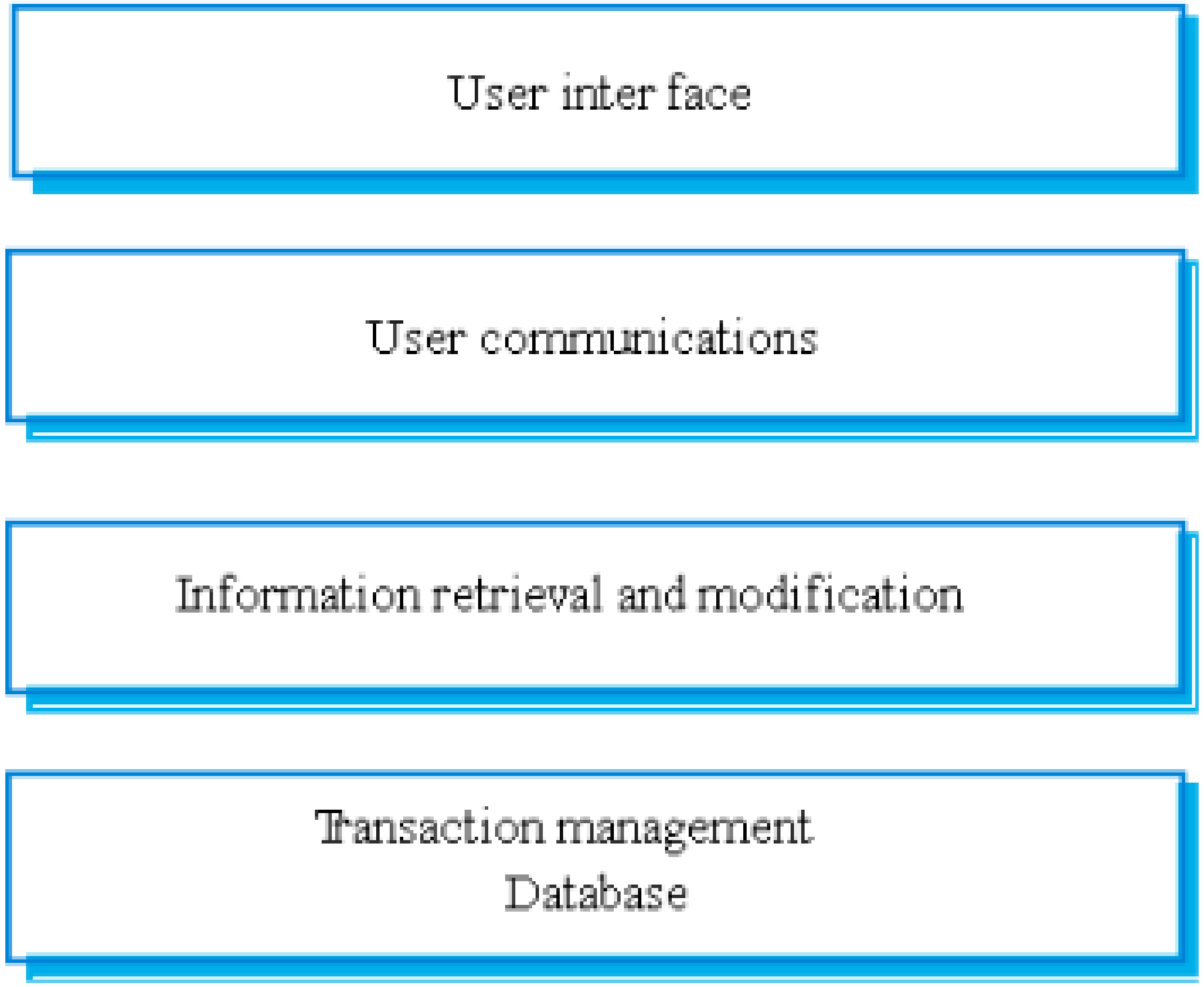


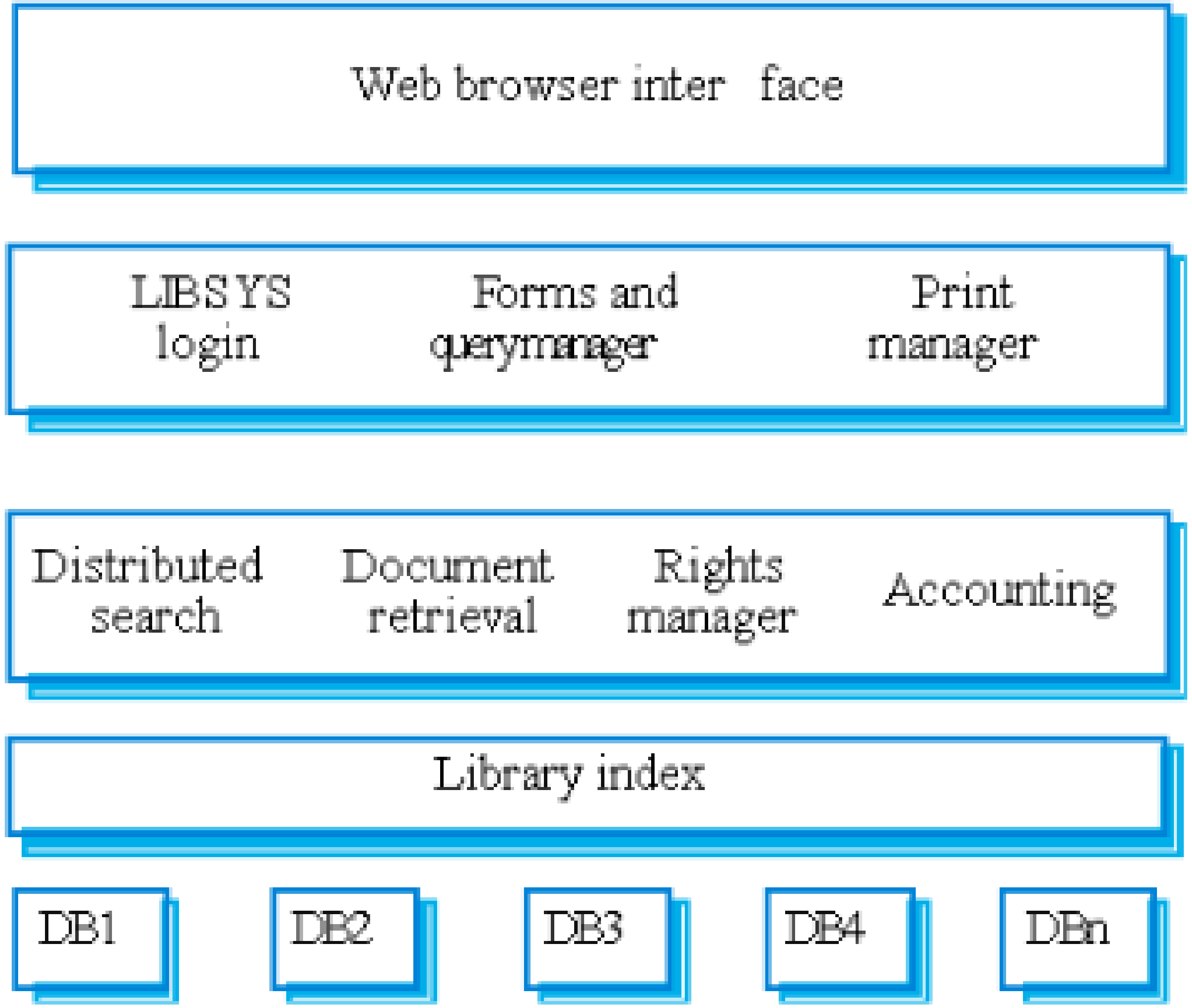
Fig: Information systems structure

System organization

Layered Model (Information systems architecture)

Eg: LIBSYS Orgination

- The library system LIBSYS is an example of an information system.



System organization

Layered Model (Advantage) ***characteristics***

- ✓ The layered model supports the incremental development of systems.
- ✓ As a layer is developed, some of the services provided by the layer may be made available to users
- ✓ This architecture is also changeable and portable, as long as its interface is unchanged, a layer can be replaced by another equivalent layer
- ✓ A layered systems localize machine dependencies in inner layers, So only the inner, machine dependent layers need to be re-implemented to facilitate different operating system or database

System organization

Layered Model

(Dis Advantage) ***characteristics***

- ✓ It structure the system in a way that can be difficult .
- ✓ Inner layers may provide basic facilities, such as file management that are required at all levels .
- ✓ Services required at the top level may have to punch through adjacent layers to get access to services that are provided several levels beneath it .
- ✓ This subverts the model, as the layer in the system does not just depend on its immediate layer.

System organization

Layered Model

(Dis Advantage) ***characteristics***

- ✓ Performance can also be a problem because of it multiple levels of command interpretation .
- ✓ If there are many layers, a services request from a top layer may have to be interpreted several times in different layers before it is processed .
- ✓ To avoid this problem applications may have to communicate directly with inner layers rather than use the services provided by the adjacent layer.

Modular Decomposition

- ✓ After an overall system organization has been chosen, you need to make a decision on the approach to be used in decomposition sub- systems into modules.
- ✓ The component in modules are usually smaller than sub-systems which allows alternative decomposition styles to be used .

Sub-systems and modules

- ✓ *A **sub-system** is a system in its own right whose operation is independent of the services provided by other sub-systems.*
- ✓ *A **module** is a system component that provides services to other components but would not normally be considered as a separate system.*

Modular Decomposition

- ✓ Another structural level where sub-systems are decomposed into modules.
- ✓ There are two main strategies that can be used when decomposing sub- system into module :

  *1. **Object Oriented Decomposition (Object Model)***

  Where the system is decomposed into a set of communicating object.

  2. Function Oriented Pipelining (pipeline/data-flow model)

  Where the system is decomposed into functional modules which transform inputs to outputs.

Modular Decomposition

1. Object models

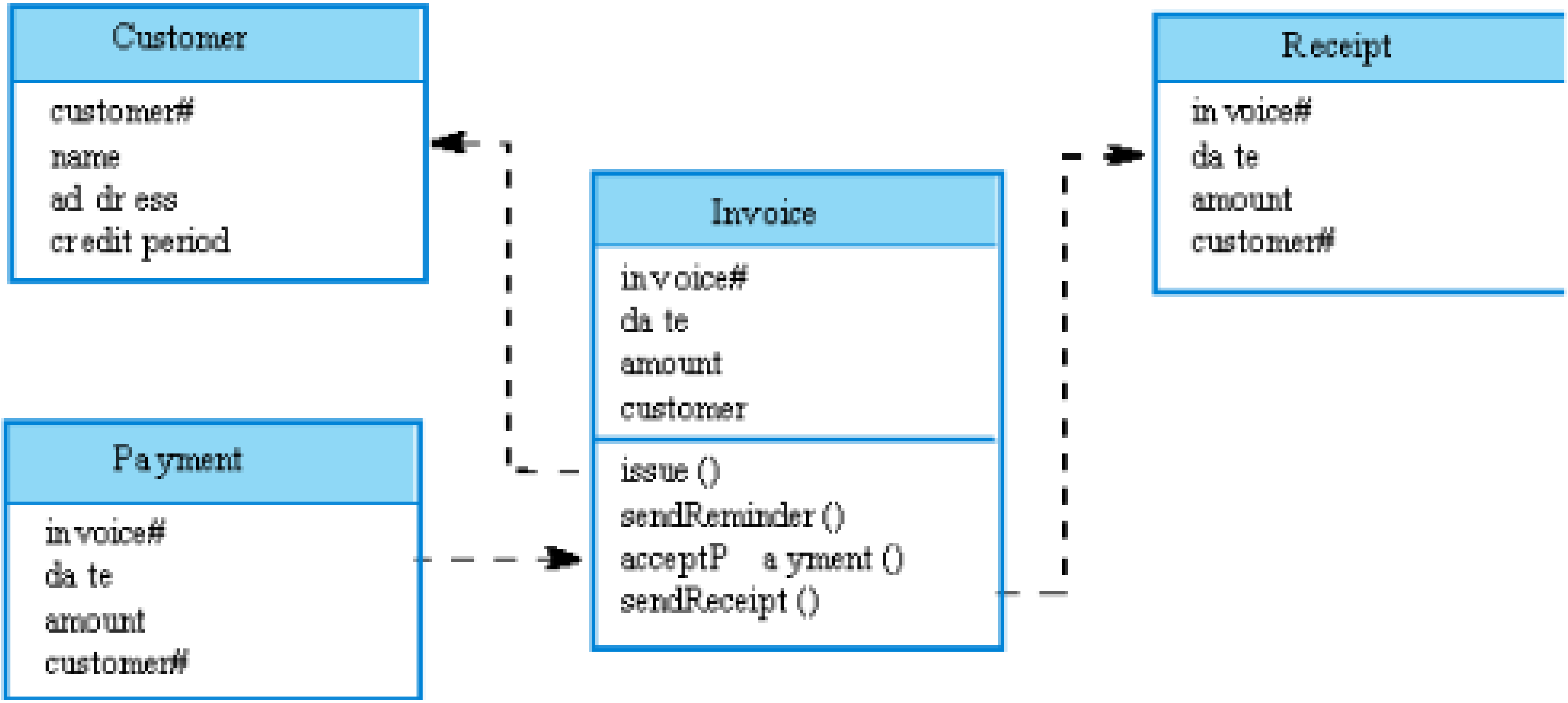
- ✓ Structure the system into a set of loosely coupled objects with well-defined interfaces.
- ✓ Object-oriented decomposition is concerned with identifying object classes, their attributes and operations.
- ✓ When implemented, objects are created from these classes and some control model used to coordinate object operations.

Modular Decomposition

1. Object models

Invoice processing system

This system can issue invoices to customers, receive payment, and issue receipts for these payments and reminders for unpaid invoices



Dashed arrows indicate that an object uses the attributes or services provided by another object.

Modular Decomposition

1. Object models

Object Model Advantages

- Objects are loosely coupled the modifications in one object do not affect the other object.
- Structure of the OO system can well understood.
- Object of the system are reuseable components.

Object Model Disadvantages

- To get any service Object must refer to proper name of interface.
- If interface gets changed then there is a change in the corresponding system.
- However, Complex entities may be hard to represent as objects. Hence, Only small scan entities can be represented by this model.

Modular Decomposition

2. Function Oriented Pipeline Or Data-flow Model

- In a function oriented pipeline or data-flow model, function transformations can applied on the inputs for getting the outputs .
- Data flows from one to another and is transformed as it moves through the sequence.
- Each processing step is called a **transform .**
- The transformations may execute sequentially or in parallel.
- ***May be referred to as a pipe and filter model (as in UNIX shell).***
- When transformations are sequential, this is a ***batch sequential model.***

Modular Decomposition

2. Function Oriented Pipeline Or Data-flow Model

This model is extensively used in data processing systems such as billing systems .

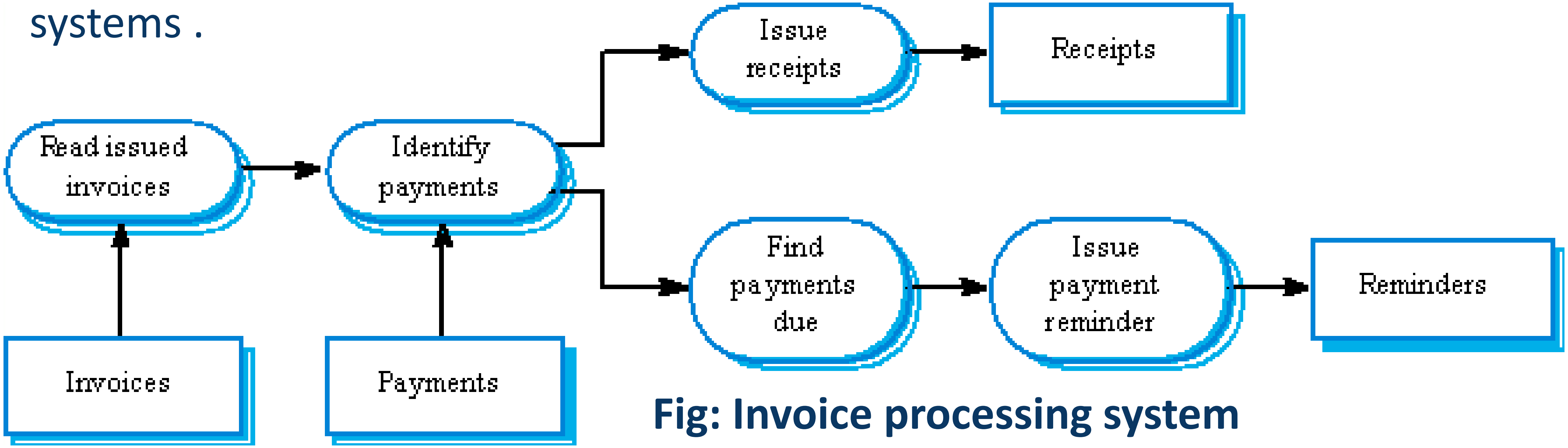


Fig: Invoice processing system

- ❖ An organization has issued invoices to customers.
- ❖ Once a week, payments that have been made are reconciled with the invoices
  - ✓For those invoices that have been paid a receipt is issued.
  - ✓For those invoices that have not been paid within the allowed payment time, a reminder is issued.

Modular Decomposition

2. Function Oriented Pipeline Or Data-flow Model

Advantages

- It provide more detailed representation of the system.
- Transformations can be reused.
- Style is simple to implement.
- Work can be broken interms of input and output.

Disadvantages

- Interactive systems are difficult.
- System containing GUI are difficult.
- It is very complex to break accounding to graphical representation or event-based interaction (mouse event).

Modular Decomposition

2. Function Oriented Pipeline Or Data-flow Model

- In a function oriented pipeline or data-flow model, function transformations can applied on the inputs for getting the outputs .
- Data flows from one to another and is transformed as it moves through the sequence.
- Each processing step is called a **transform .**
- The transformations may execute sequentially or in parallel.
- ***May be referred to as a pipe and filter model (as in UNIX shell).***
- When transformations are sequential, this is a ***batch sequential model.***

Procedal Design

- Software Procedural Design (SPD) converts and translates structural elements into procedural explanations. SPD starts straight after data design and architectural design.
- Procedural design is best used to model programs that have an clear flow of data from input to output. It represents the architecture of a program as a set of interacting processes that pass data from one to another.
- The main objective in Procedural Design is to transform structural components into a procedural description of the software.
- The step occurs after the data and program structures have been established, i.e. after architectural design. Procedural details can be represented in different ways:

Procedal Design

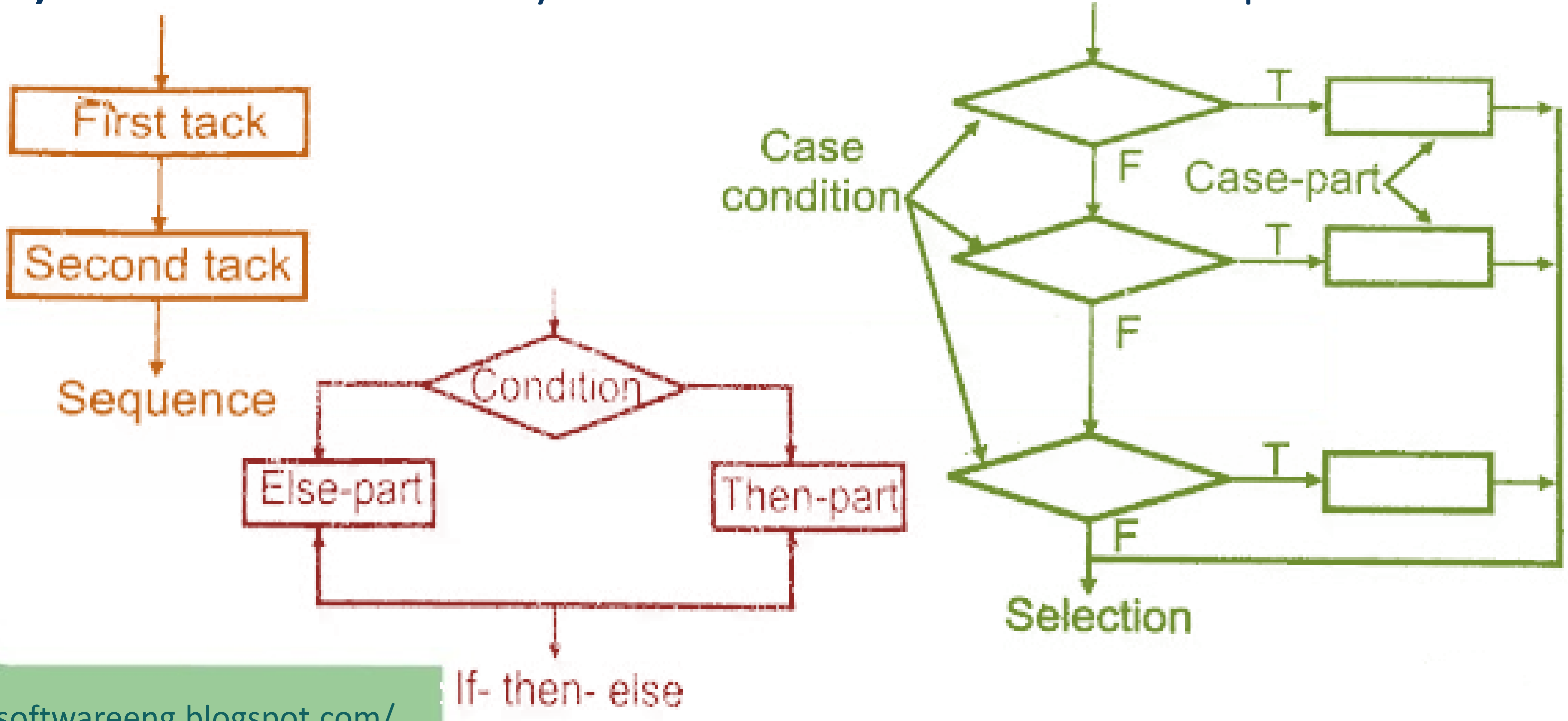
1. Graphical Design Notation: The most widely used notation is the flowchart. Some notation used in flowcharts are:

(i) Boxes to indicate processing steps.

(ii) Diamond to indicate logical condition.

(iii) Arrows to indicate flow of control.

(iv) Two boxes connected by a line of control will indicate a Sequence.

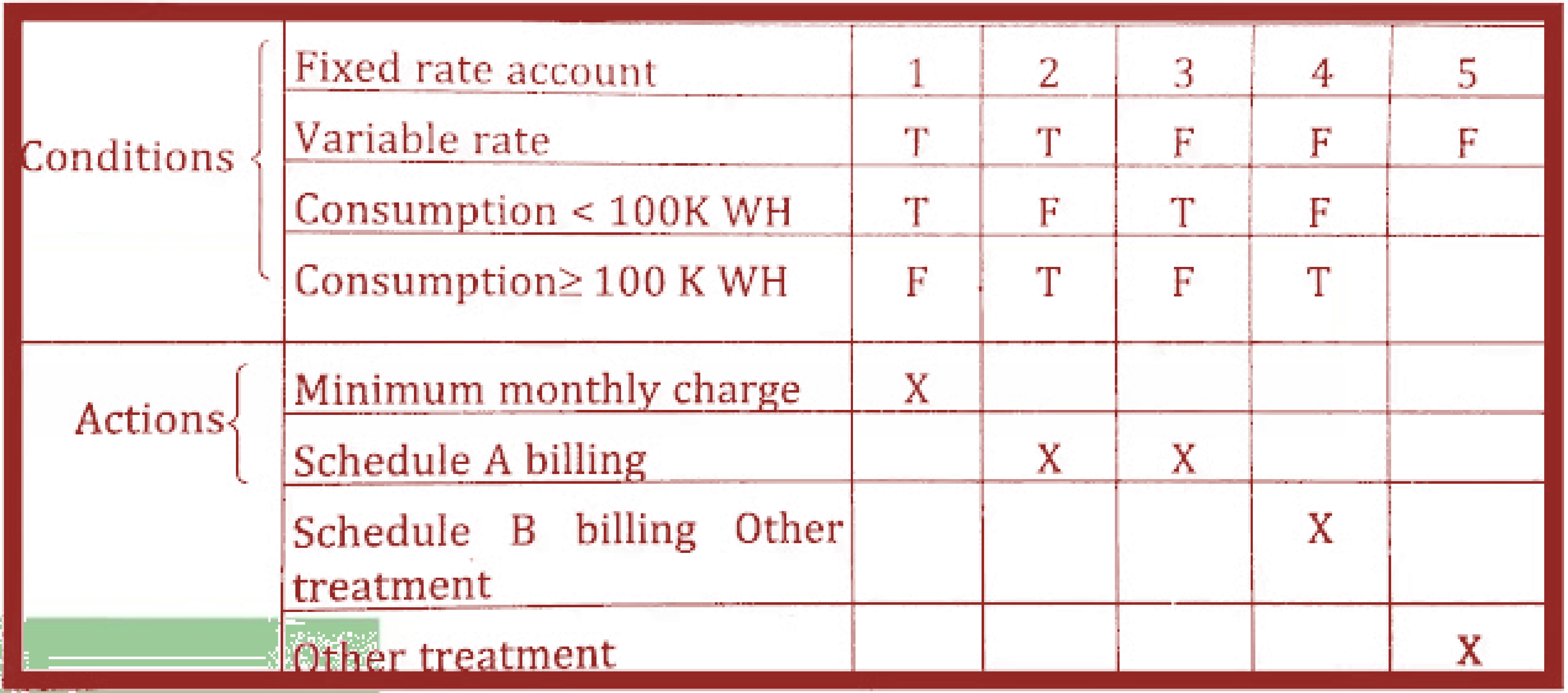


Procedal Design

2. Tabular Design Notation:

(i) Decision tables provide a notation that translates actions and conditions into a tabular form.

(ii) The upper left-hard section contains a list of all conditions. The lower left-hand section lists all actions that are possible based on the conditions. The right-hand sections form a matrix that indicates condition combinations and the corresponding actions that will occur for a specific combination.



Conditions	Fixed rate account	1	2	3	4	5
	Variable rate	T	T	F	F	F
	Consumption < 100K WH	T	F	T	F	
	Consumption≥ 100 K WH	F	T	F	T	
Actions	Minimum monthly charge	X				
	Schedule A billing		X	X		
	Schedule B billing Other treatment				X	
	Other treatment					X

Procedal Design

3. Program Design Language: It is a method designing and documenting methods and procedures in software. It is related to pseudocode, but unlike pseudocode, it is written in plain language without any terms that could suggest the use of any programming language or library.

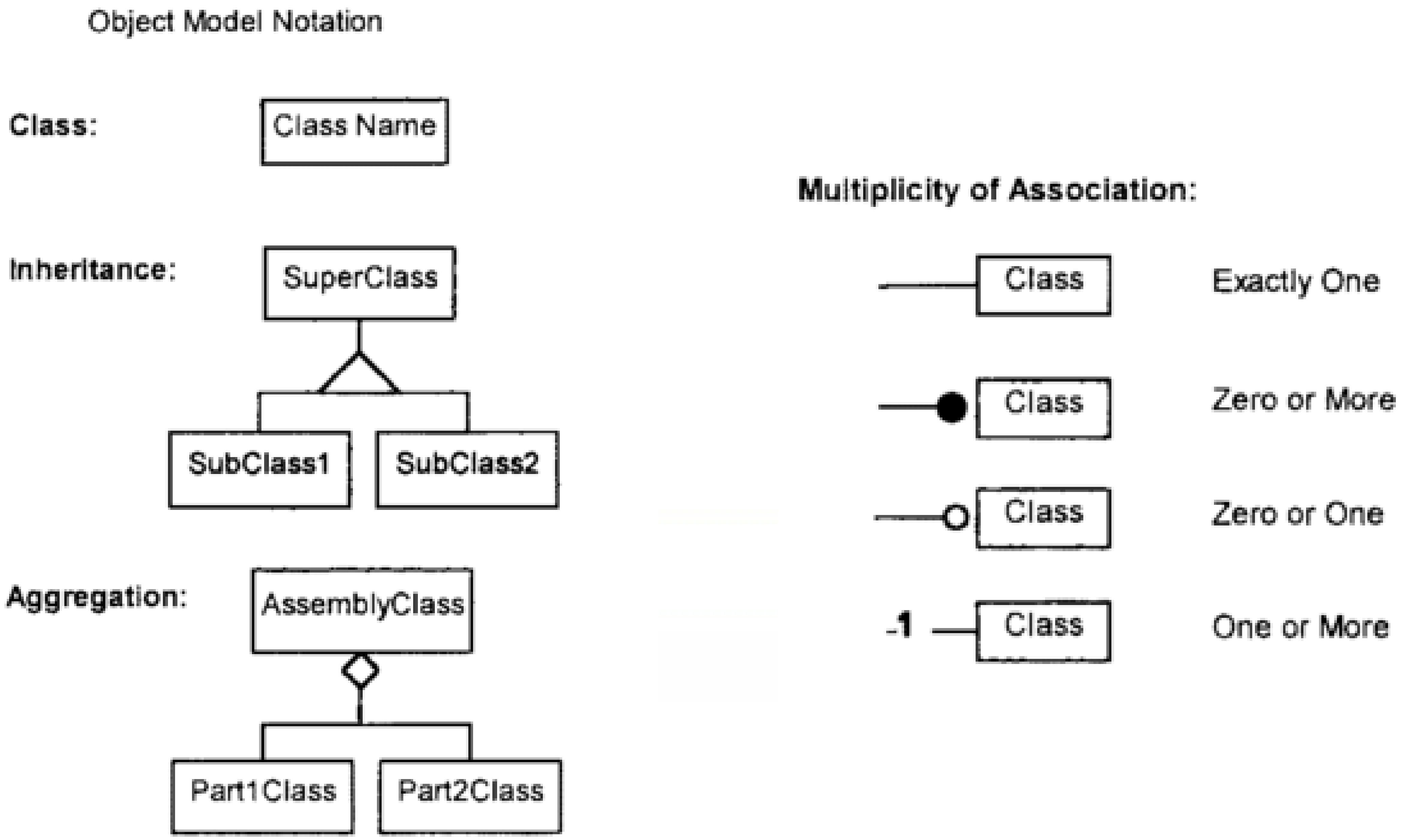
Using Structured Method

A structured method includes a design process model, notations to represent the design, report formats, rules and design guidelines. Structured methods may support some or all of the following models of a system:

- An object model that shows the object classes used in the system and their dependencies.
- A sequence model that shows how objects in the system interact when the system is executing.
- A state transition model that shows system states and the triggers for the transitions from one state to another.
- A data flow model where the system is modelled using the data transformations that take place as it is processed. This is not normally used in object-oriented methods but is still frequently used in real-time and business system design.
- A use-case model that shows the interactions between users and the system.

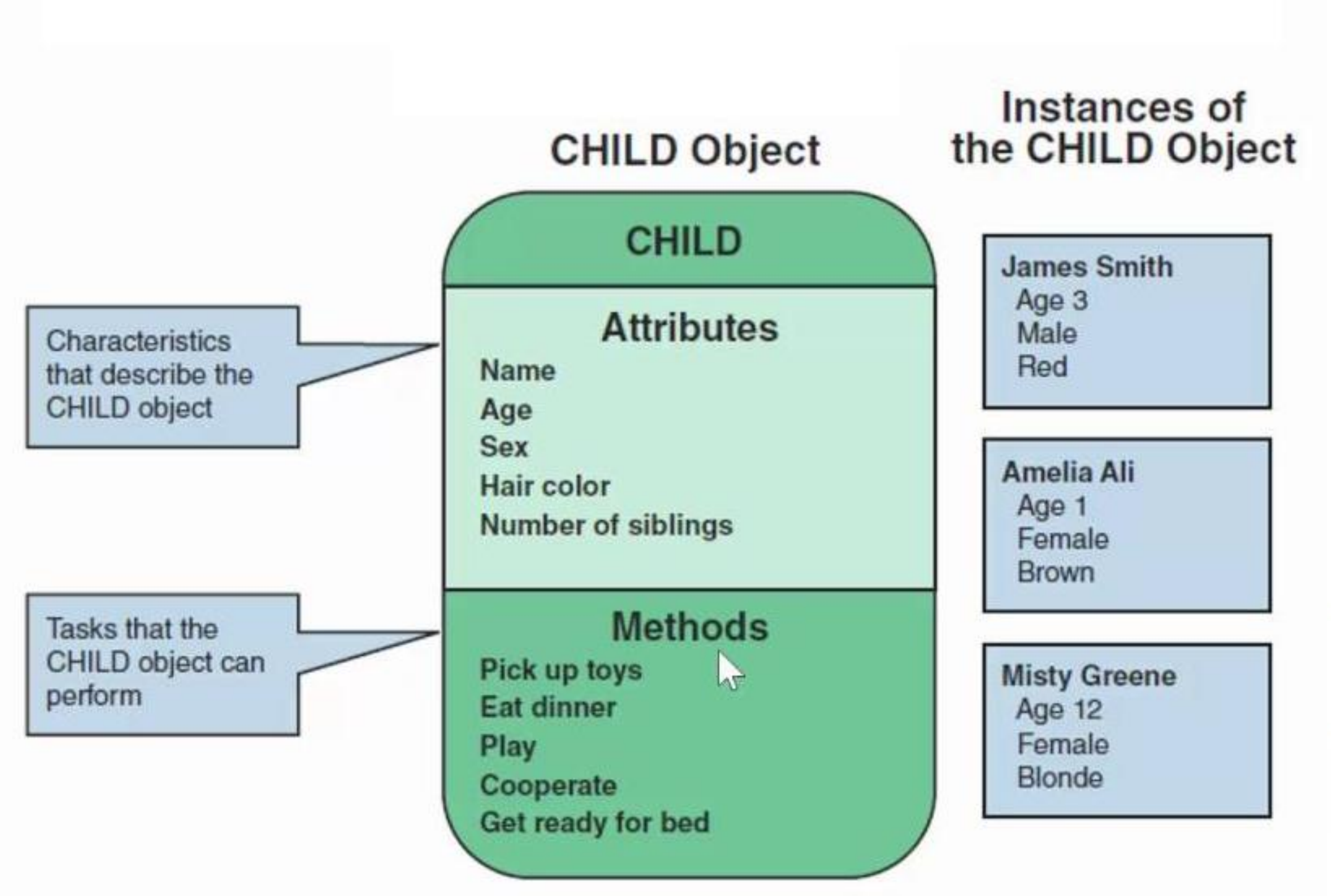
Using Structured Method

- An object model that shows the object classes used in the system and their dependencies.



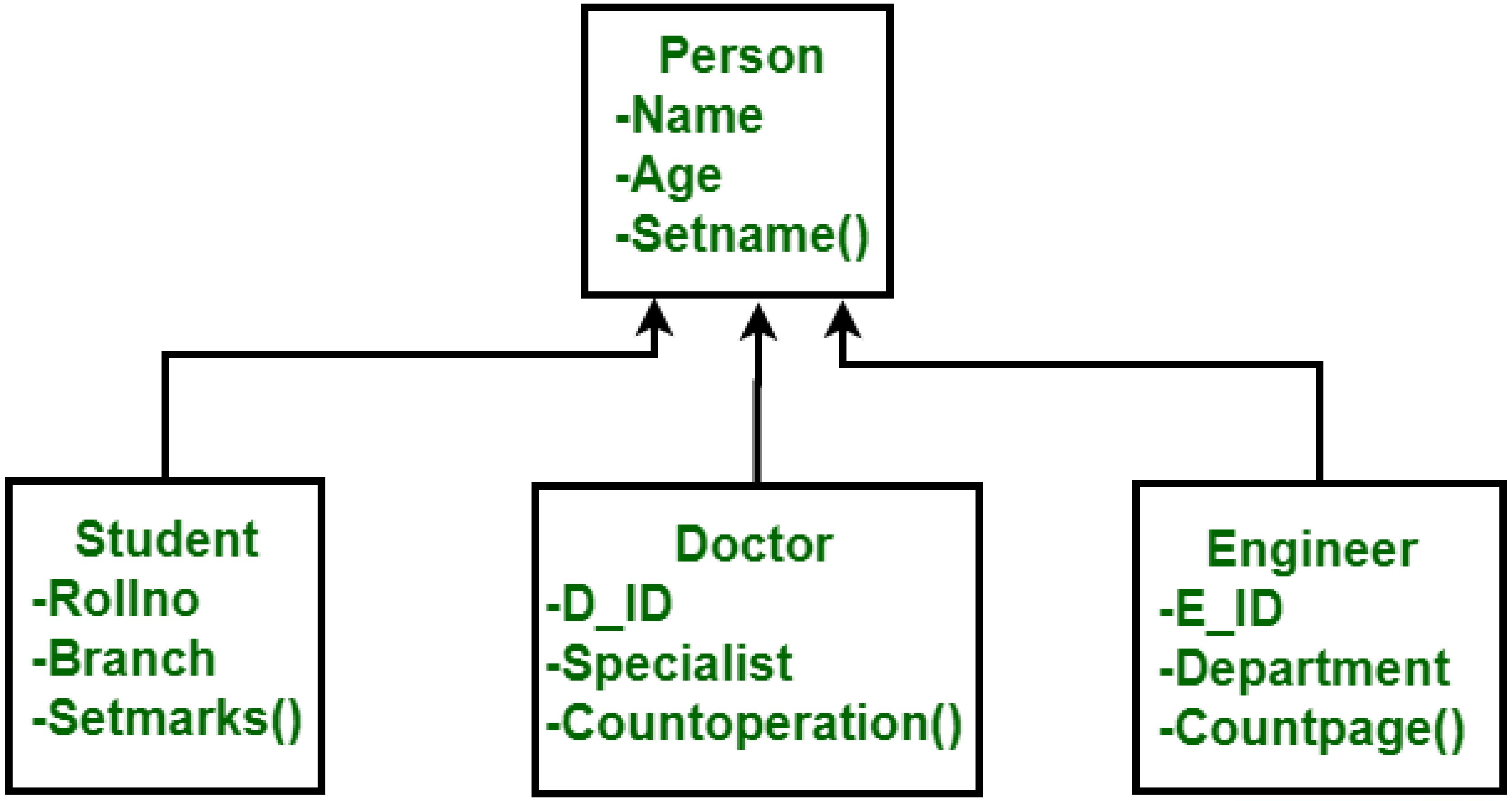
Using Structured Method

- An object model that shows the object classes used in the system and their dependencies.



Using Structured Method

- An object model that shows the object classes used in the system and their dependencies.



Person
-Name
-Age
-Setname()

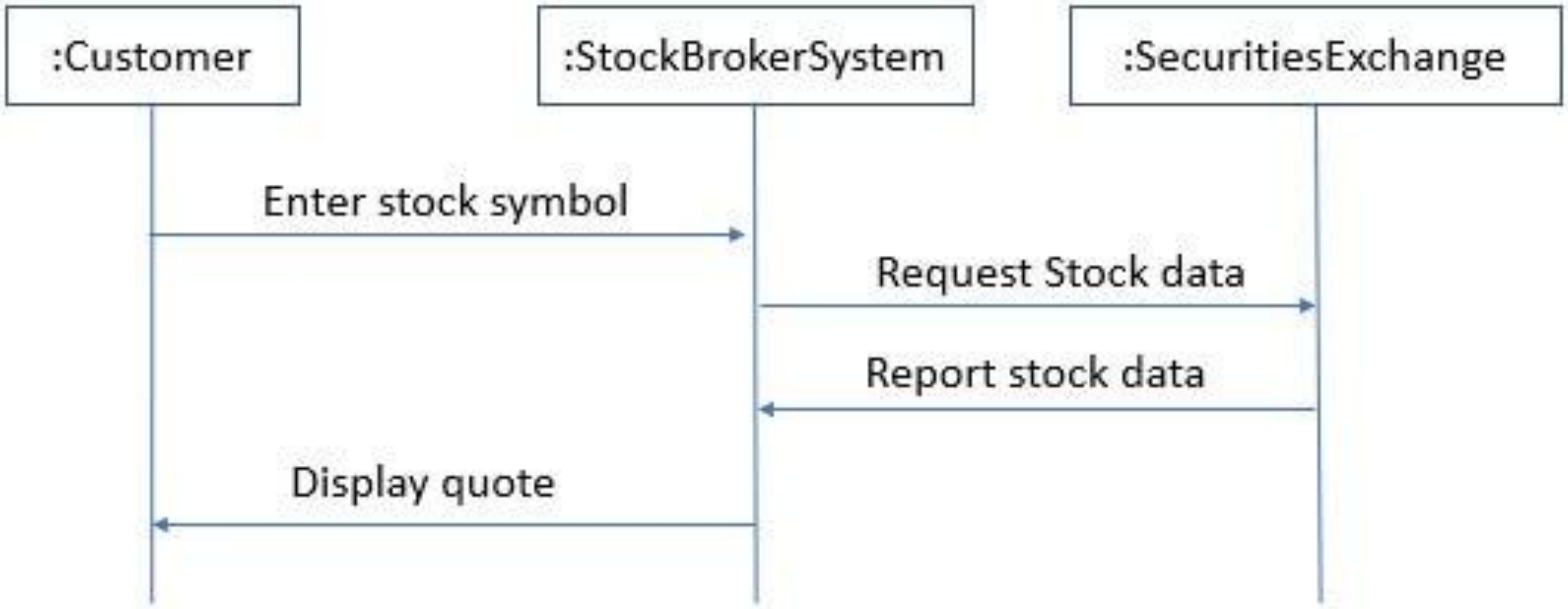
Student
-Rollno
-Branch
-Setmarks()

Doctor
-D_ID
-Specialist
-Countoperation()

Engineer
-E_ID
-Department
-Countpage()

Using Structured Method

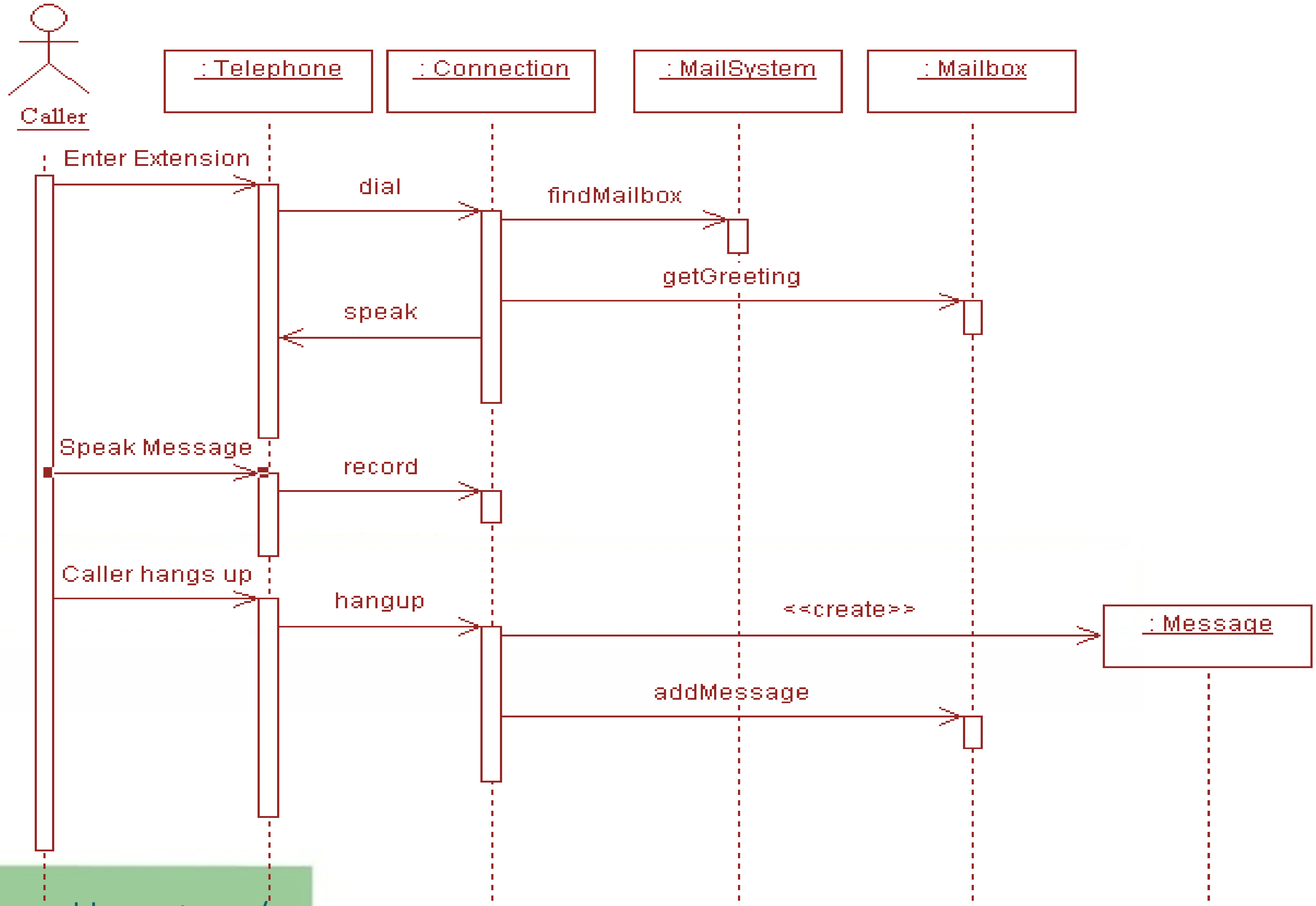
- sequence model



Sequence Diagram for Stock Quote

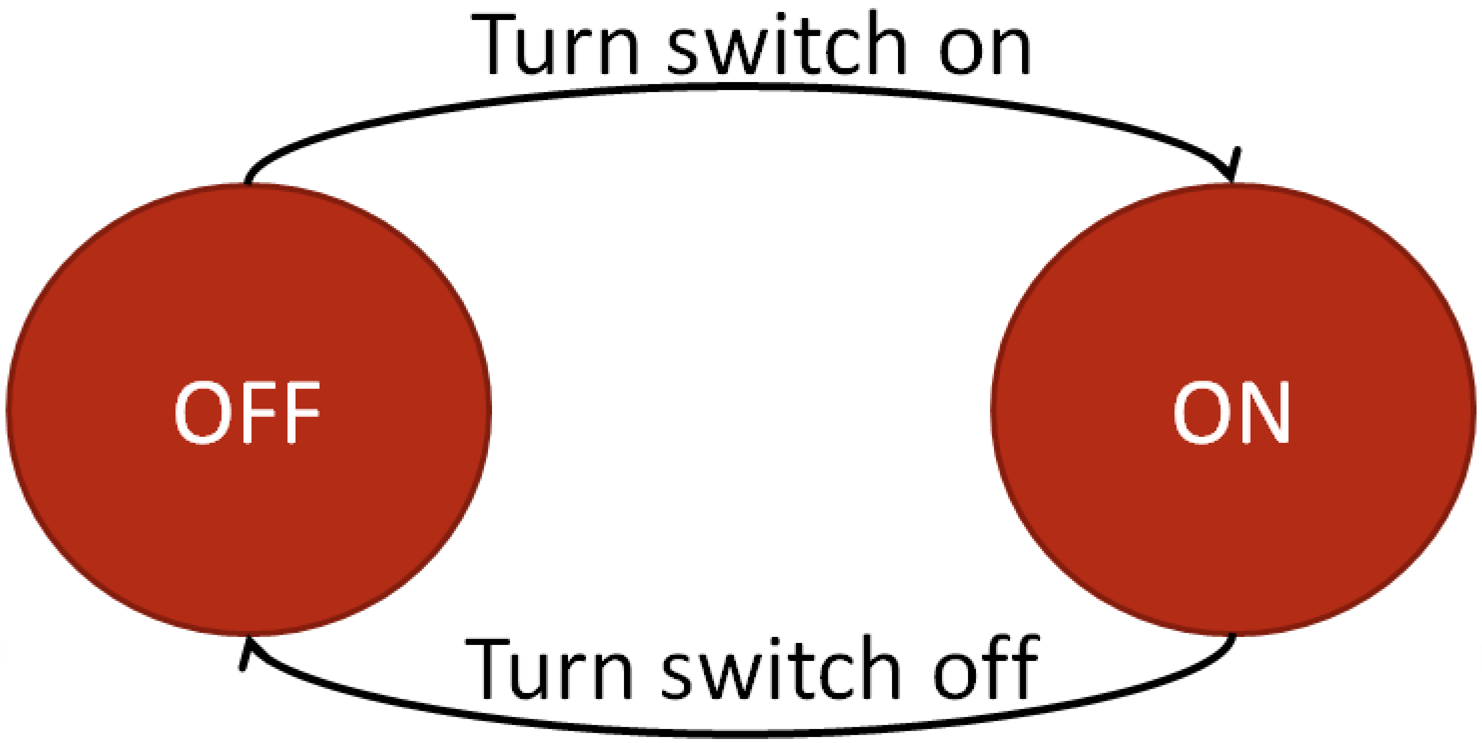
Using Structured Method

- sequence model



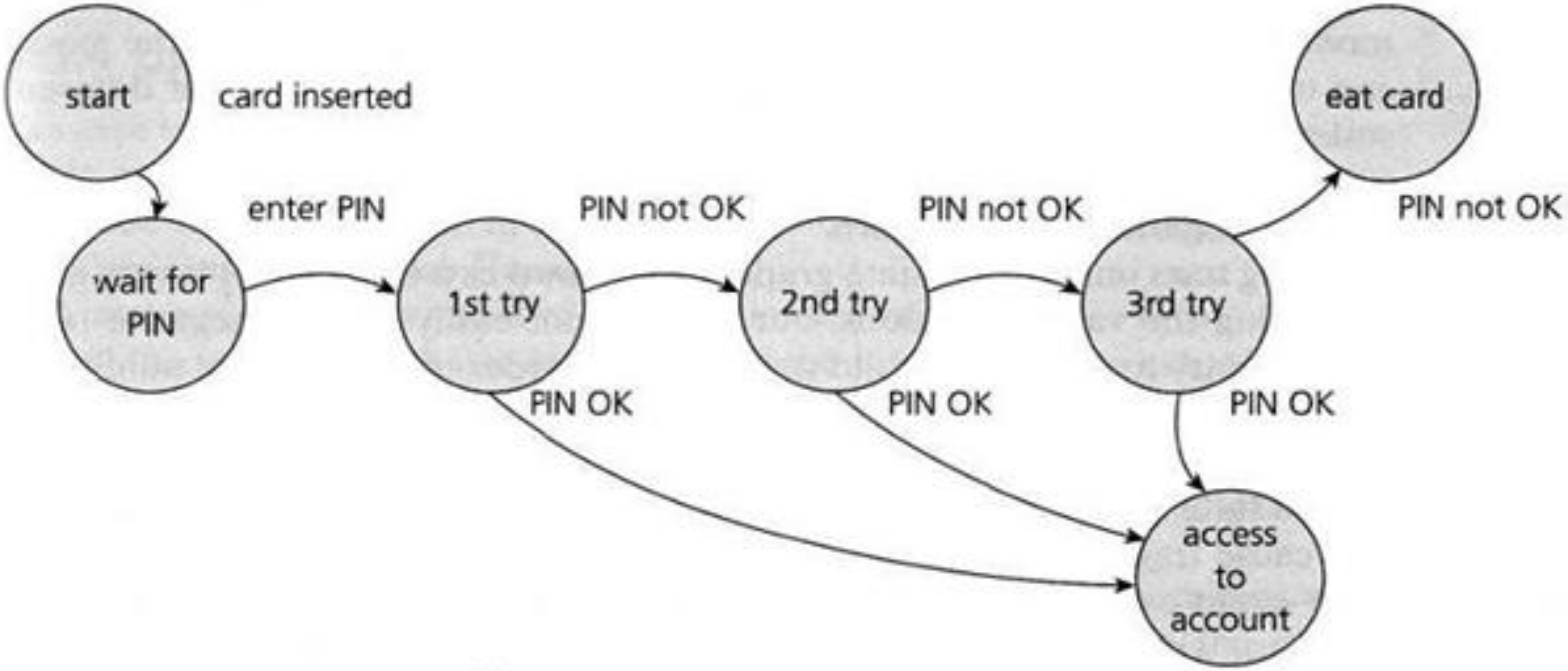
Using Structured Method

- state transition model



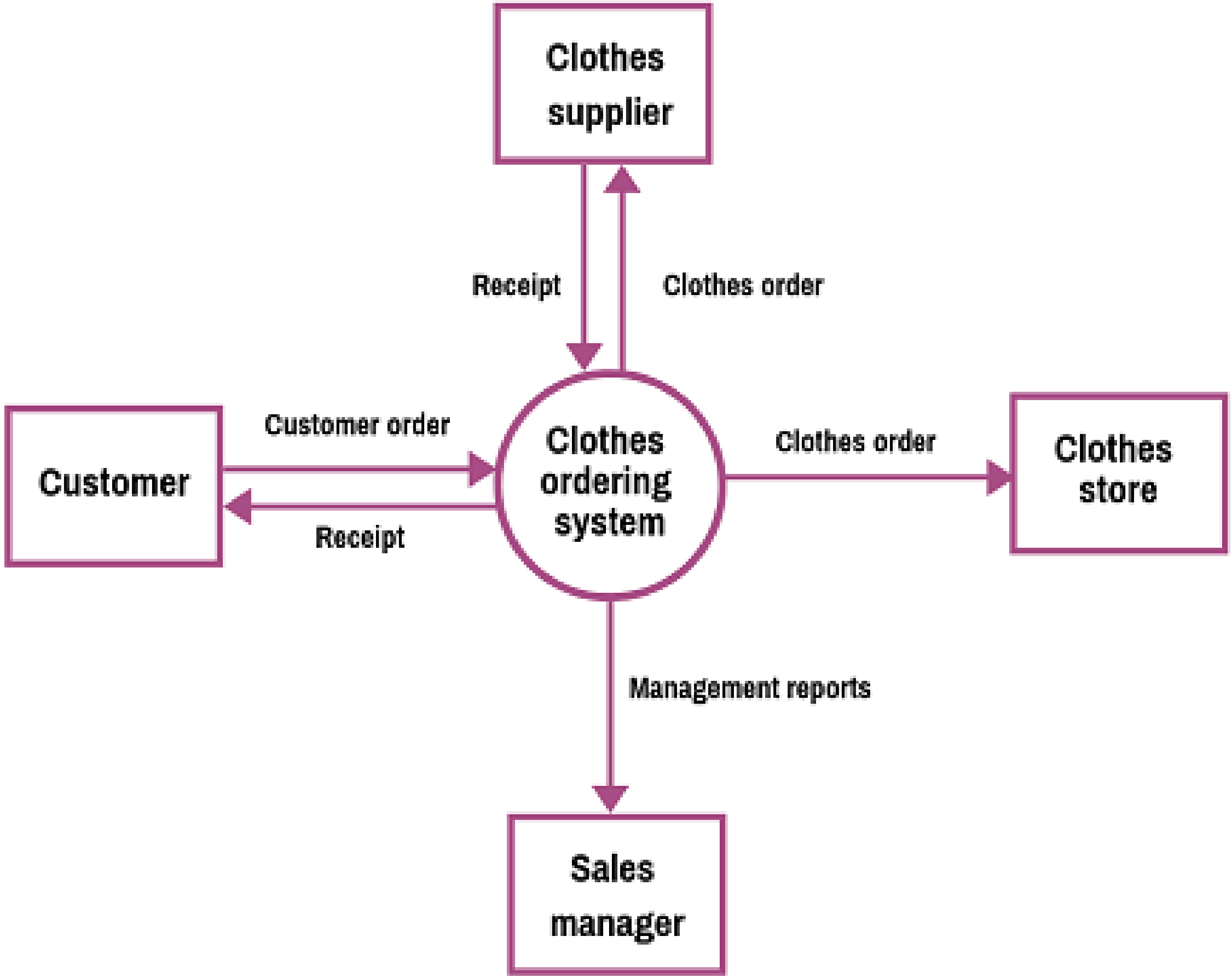
Using Structured Method

- state transition model



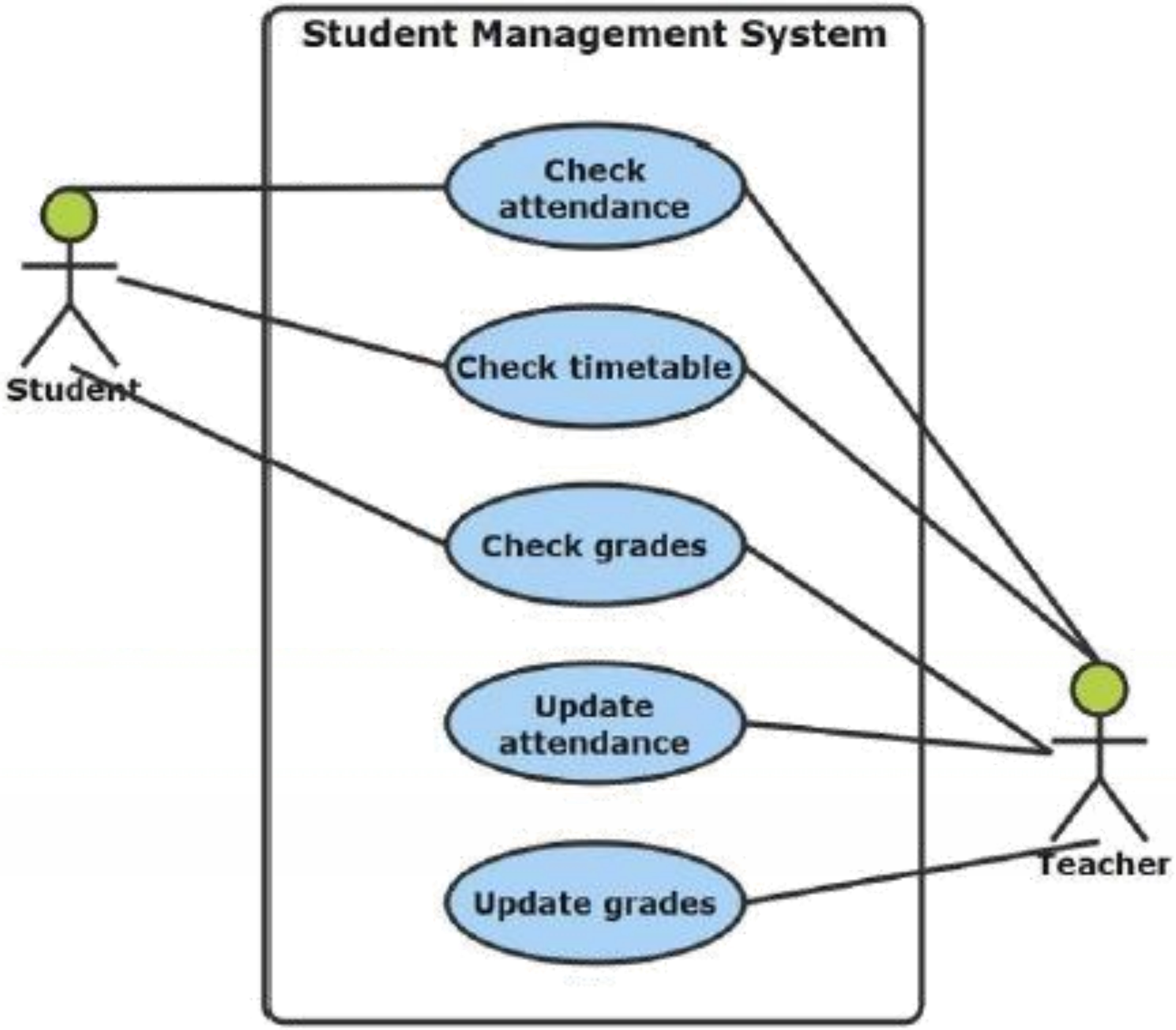
Using Structured Method

- data flow model



Using Structured Method

- A use-case model that shows the interactions between users and the system.



Using Structured Method

Structured Systems Analysis And Design Methodology (SSADM) is a set of standards for systems analysis and application design.

The techniques used in SSADM are logical data modeling, data flow modeling and entity behavior modeling.

Logical Data Modeling: This involves the process of identifying, modeling and documenting data as a part of system requirements gathering. The data are classified further into entities and relationships.
Data Flow Modeling: This involves tracking the data flow in an information system. It clearly analyzes the processes, data stores, external entities and data movement.
Entity Behavior Modeling: This involves identifying and documenting the events influencing each entity and the sequence in which these events happen.

Using Structured Method

SSADM's objectives are to:

- Improve project management & control
- Make more effective use of experienced and inexperienced development staff.
- Develop better quality systems
- Enable projects to be supported by computer-based tools such as computer-aided software engineering systems
- Establish a framework for good communications between participants in a project

SSADM's steps, or stages, are:

- Feasibility
- Investigation of the current environment
- Business systems options
- Definition of requirements
- Technical system options
- Logical design
- Physical 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, ships etc.
- UI provides fundamental platform for human-computer interaction.
- UI can be graphical, text-based, audio-video based, depending upon the underlying hardware and software combination. UI can be hardware or software or a combination of both.

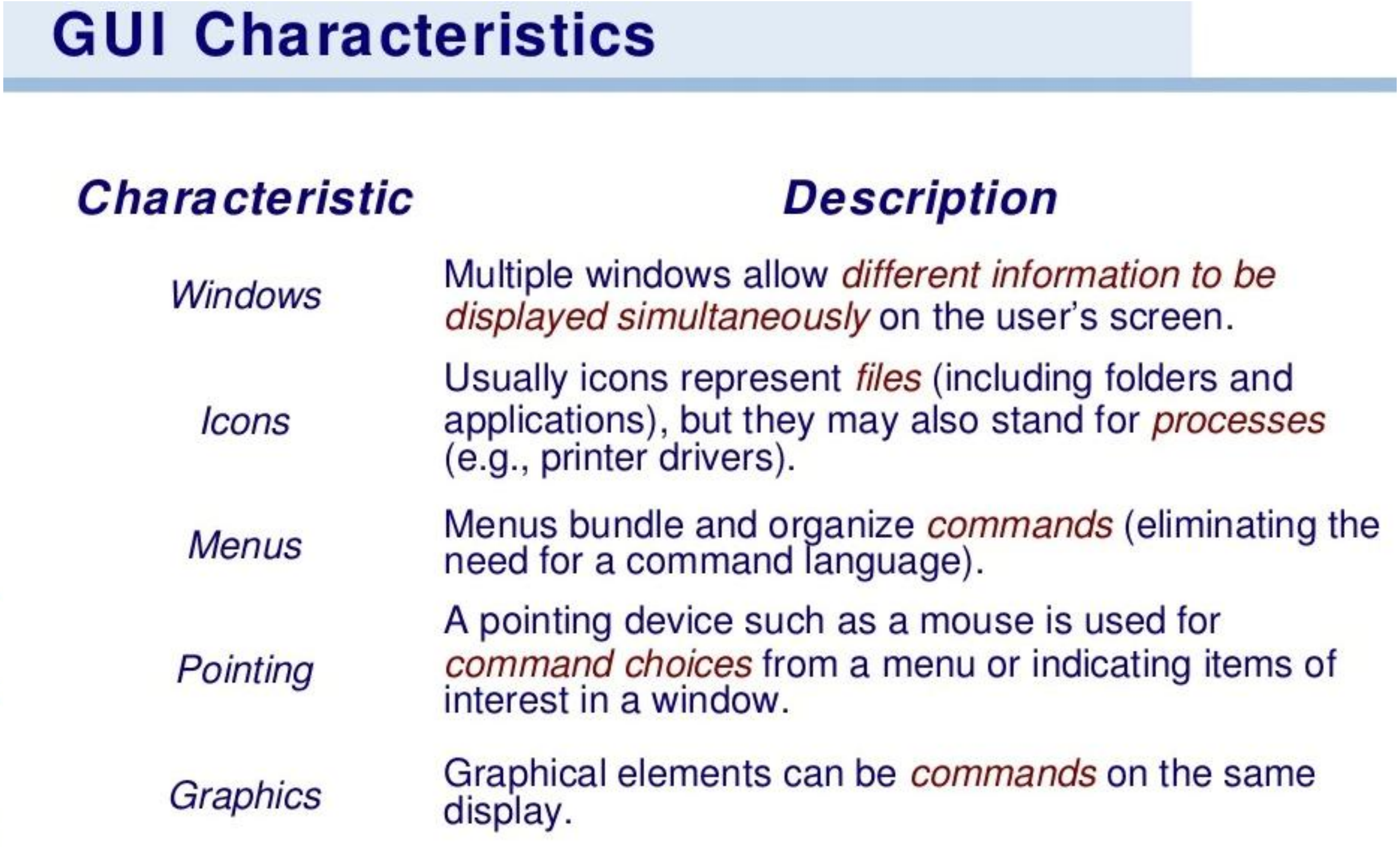
The software becomes more popular if its user interface is:

- □ Attractive
- □ Simple to use
- □ Responsive in short time
- □ Clear to understand

UI is broadly divided into two categories:

1. Command Line Interface
2. Graphical User Interface

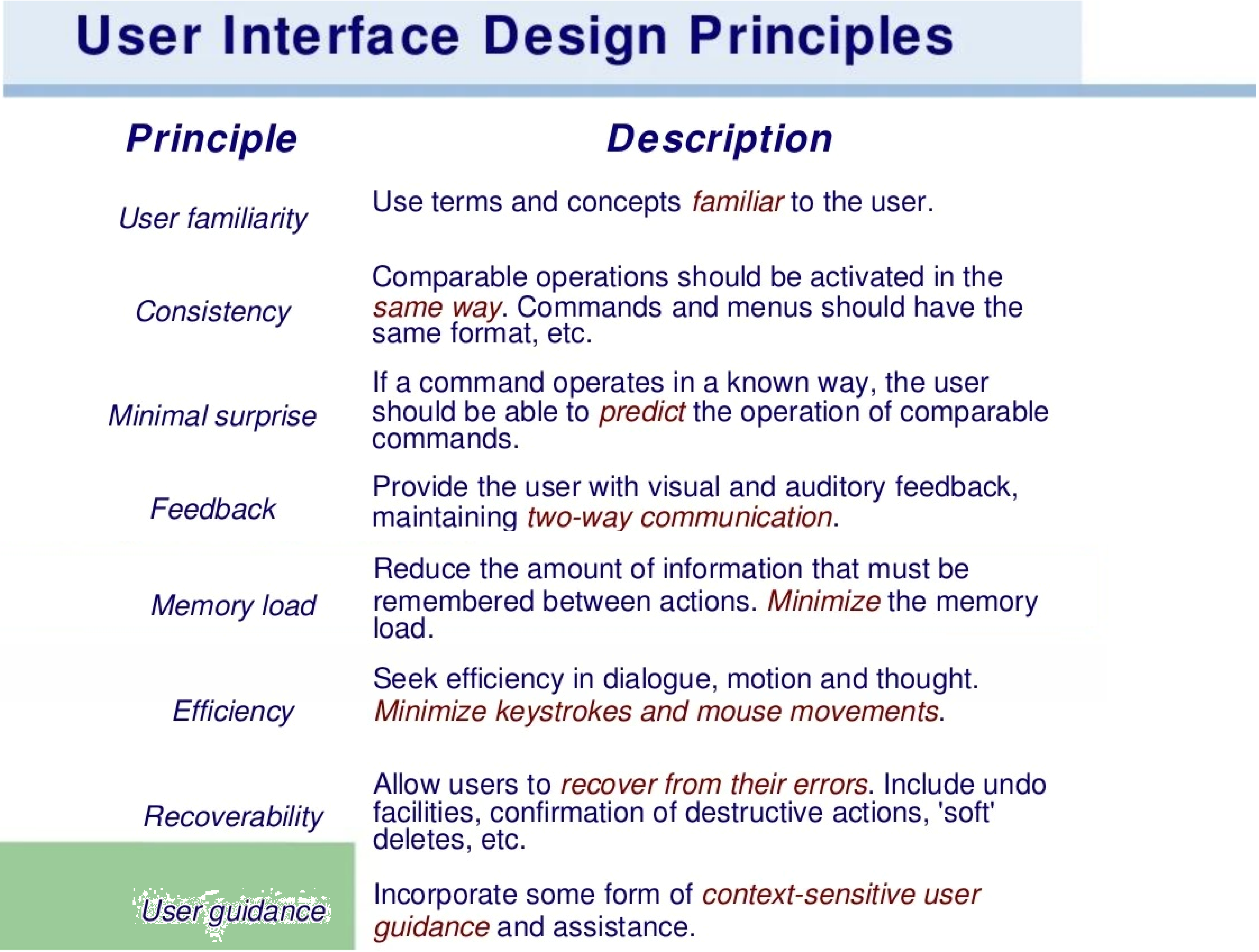
User Interface Design:



GUI Characteristics

Characteristic	*Description*
Windows	Multiple windows allow *different information to be displayed simultaneously* on the user's screen.
Icons	Usually icons represent *files* (including folders and applications), but they may also stand for *processes* (e.g., printer drivers).
Menus	Menus bundle and organize *commands* (eliminating the need for a command language).
Pointing	A pointing device such as a mouse is used for *command choices* from a menu or indicating items of interest in a window.
Graphics	Graphical elements can be *commands* on the same display.

User Interface Design:



User Interface Design Principles

Principle	*Description*
User familiarity	Use terms and concepts *familiar* to the user.
Consistency	Comparable operations should be activated in the *same way*. Commands and menus should have the same format, etc.
Minimal surprise	If a command operates in a known way, the user should be able to *predict* the operation of comparable commands.
Feedback	Provide the user with visual and auditory feedback, maintaining *two-way communication*.
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 Interface Design:

Human-Computer Interaction

Content

1. The Human
2. The Computer
3. The Interaction

User Interface Design:

Human-Computer Interaction

Human

- The end-user
- The members of an organization

Computer

- Hardware
- Software

Interface

- A point where two objects meet.
- A point where the human can tell the computer what to do.
- A point where the computer displays the requested information.

User Interface Design:

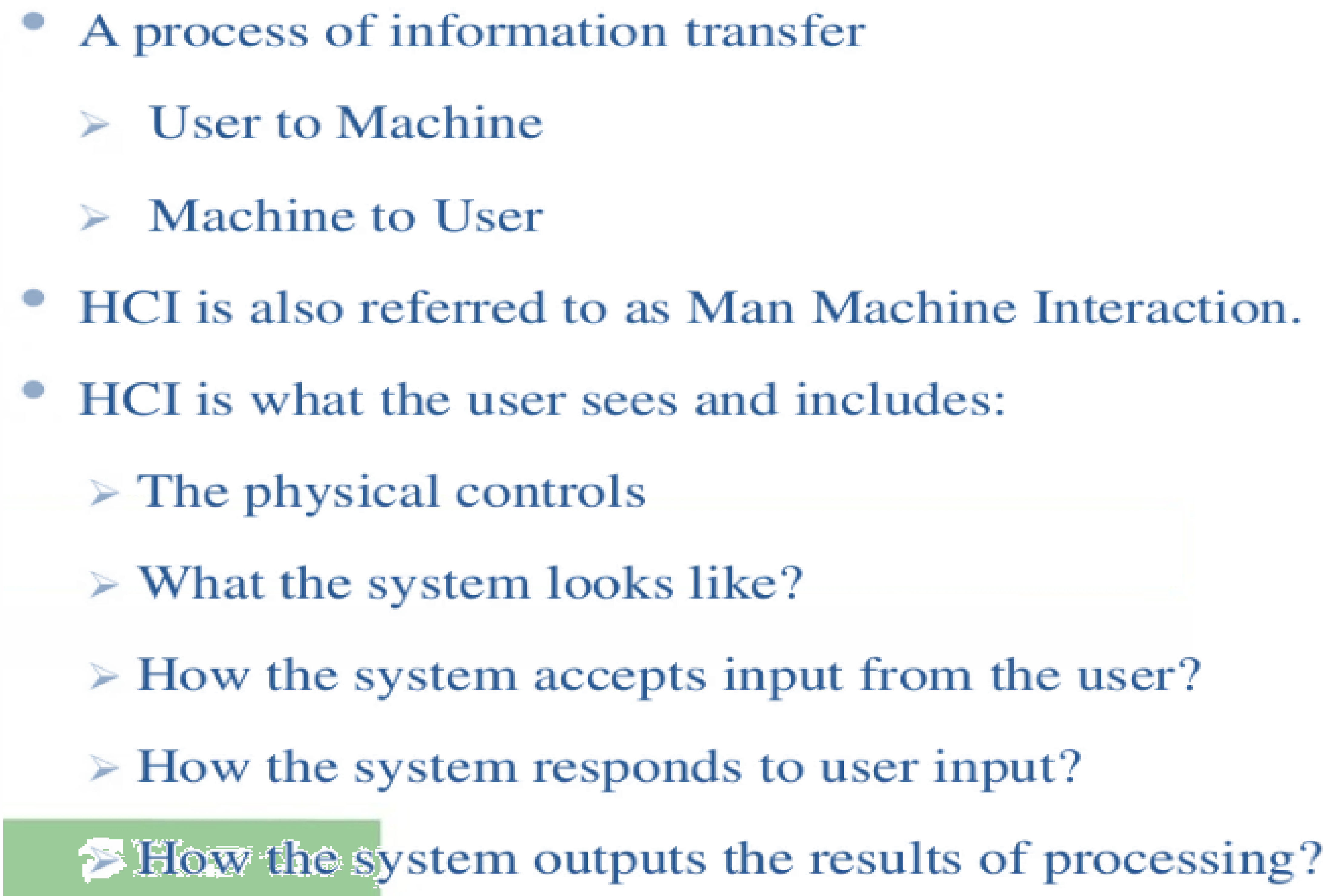
Human-Computer Interaction

What is HCI

- ❑ HCI is the study of interaction between human (users) and computers.
- ❑ The interaction between user(s) and computer(s) is achieved via an interface – user interface

User Interface Design:

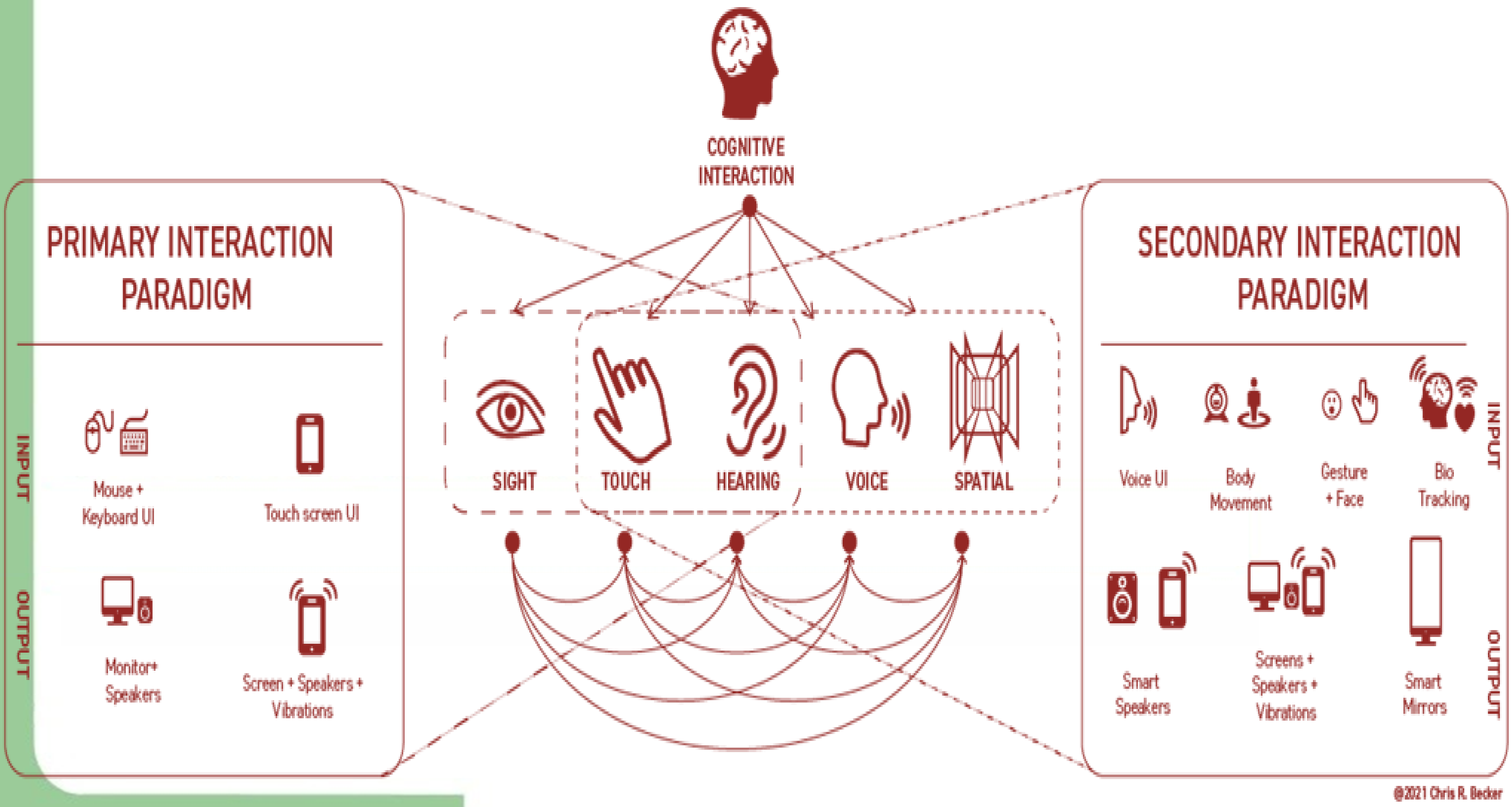
Human-Computer Interaction



- A process of information transfer
  - ➢ User to Machine
  - ➢ Machine to User
- HCI is also referred to as Man Machine Interaction.
- HCI is what the user sees and includes:
  - ➢ The physical controls
  - ➢ What the system looks like?
  - ➢ How the system accepts input from the user?
  - ➢ How the system responds to user input?
  - ➢ How the system outputs the results of processing?

User Interface Design:

Human-Computer Interaction



https://ctal-softwareeng.blogspot.com/

User Interface Design:

Human-Computer Interaction

Four basic method of User Interfaces

1. **Command Line Interface (CLI)** A CLI displays a prompt, the user types a command on the keyboard, the computer executes the command and provides textual output.
2. **Menu Driven Interface** The user has a list of items to choose from, and can make selections by highlighting one.
3. **Graphical User Interface (GUI)** Uses windows, icons, menus and pointers (WIMP) which can be manipulated by a mouse (and often to an extent by a keyboard as well).
4. **Natural Language** Interface Can range from simple command systems to voice activated text processing. Commands are spoken in “normal” language.

6. Information Presentation

- Static information
  - Initialised at the beginning of a session. It does not change during the session.
  - May be either numeric or textual.
- Dynamic information
  - Changes during a session and the changes must be communicated to the system user.
  - May be either numeric or textual.

Easy to learn?

Easy to use?

Easy to understand?

6. Information Presentation

- Static information
  - Initialised at the beginning of a session. It does not change during the session.
  - May be either numeric or textual.
- Dynamic information
  - Changes during a session and the changes must be communicated to the system user.
  - May be either numeric or textual.

6. Information Presentation

- Information presentation is concerned with presenting system information to system users.
- The information may be presented directly (e.g. text in a word processor) or may be transformed in some way for presentation (e.g. in some graphical form).
- The Model-View-Controller approach is a way of supporting multiple presentations of data.

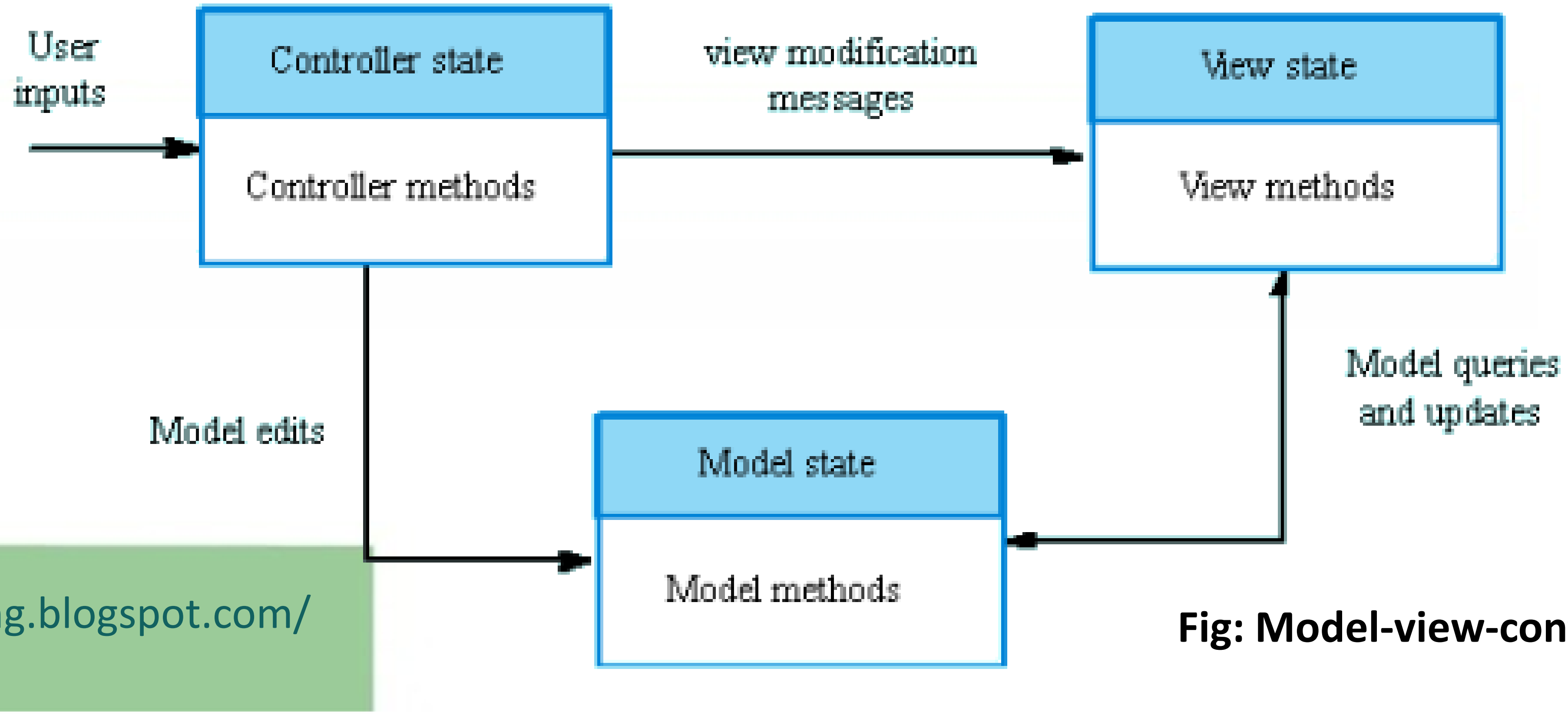
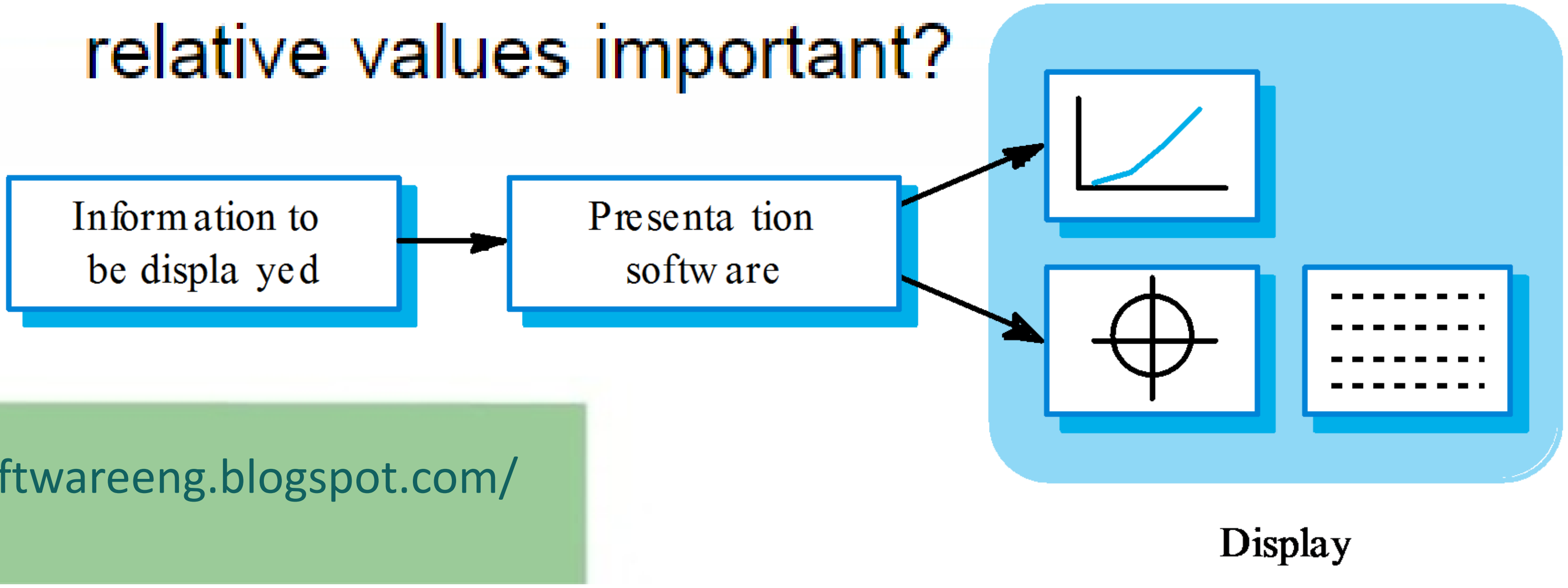


Fig: Model-view-controller

6. Information Presentation

Information display factors

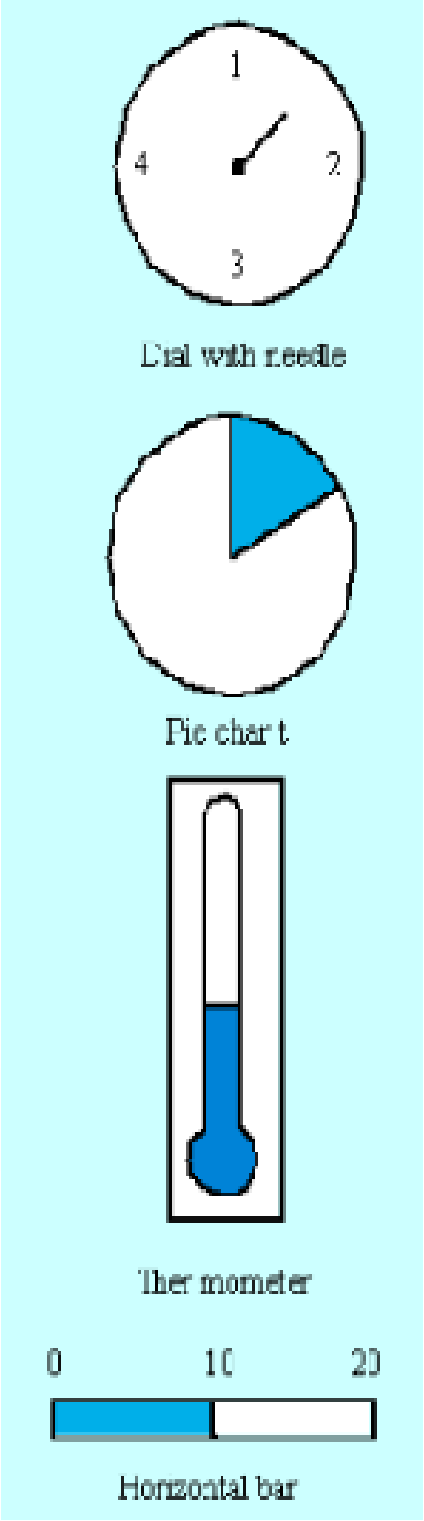
- Is the user interested in precise information or data relationships?
- How quickly do information values change? Must the change be indicated immediately?
- Must the user take some action in response to a change?
- Is there a direct manipulation interface?
- Is the information textual or numeric? Are relative values important?



Information to be displayed

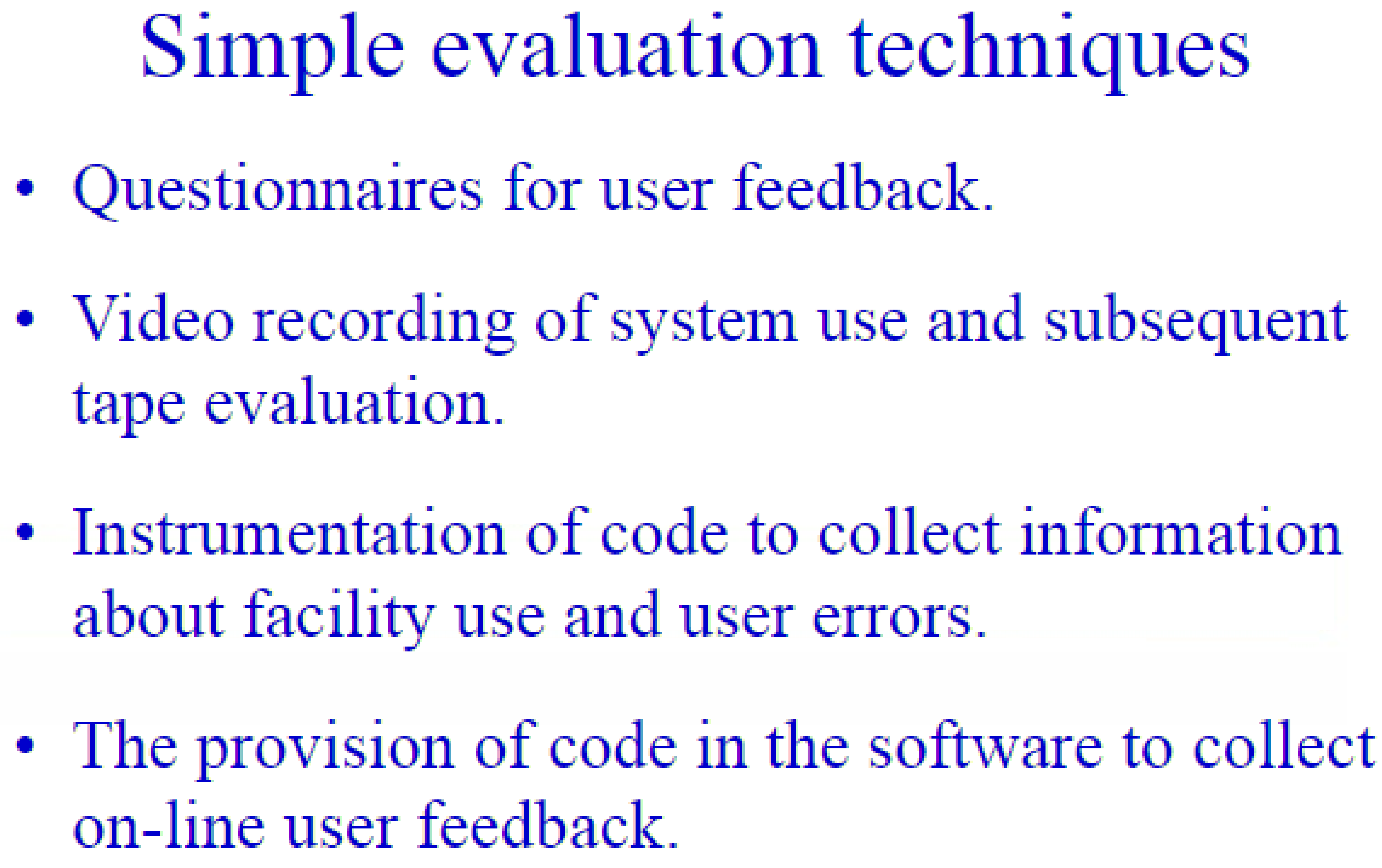
Presentation software

Display



6. Information Presentation

i. Interface Evaluation



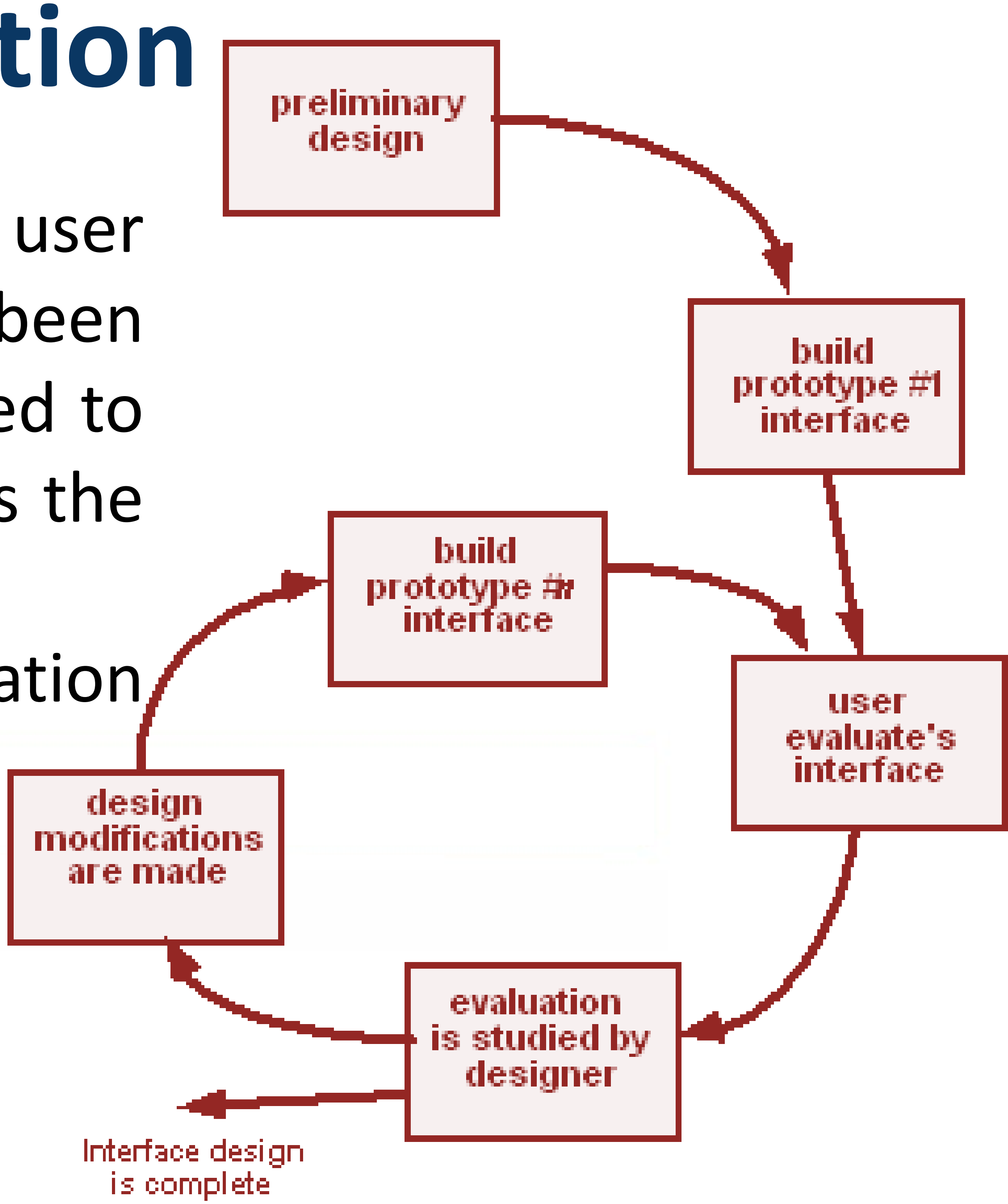
Simple evaluation techniques

- Questionnaires for user feedback.
- Video recording of system use and subsequent tape evaluation.
- Instrumentation of code to collect information about facility use and user errors.
- The provision of code in the software to collect on-line user feedback.

6. Information Presentation

i. Interface Evaluation

- Once an operational user interface prototype has been created, it must be evaluated to determine whether it meets the needs of the user.
- The user interface evaluation cycle.



preliminary design

build prototype #1 interface

build prototype #n interface

user evaluate's interface

design modifications are made

evaluation is studied by designer

Interface design is complete

Fig: Design Evaluation Cycle

6. Information Presentation

i. Interface Evaluation

- The prototyping approach is effective, but is it possible to evaluate the quality of a user interface before a prototype is built?
- If potential problems can be uncovered and corrected early, the number of loops through the evaluation cycle will be reduced and development time will shorten.

6. Information Presentation

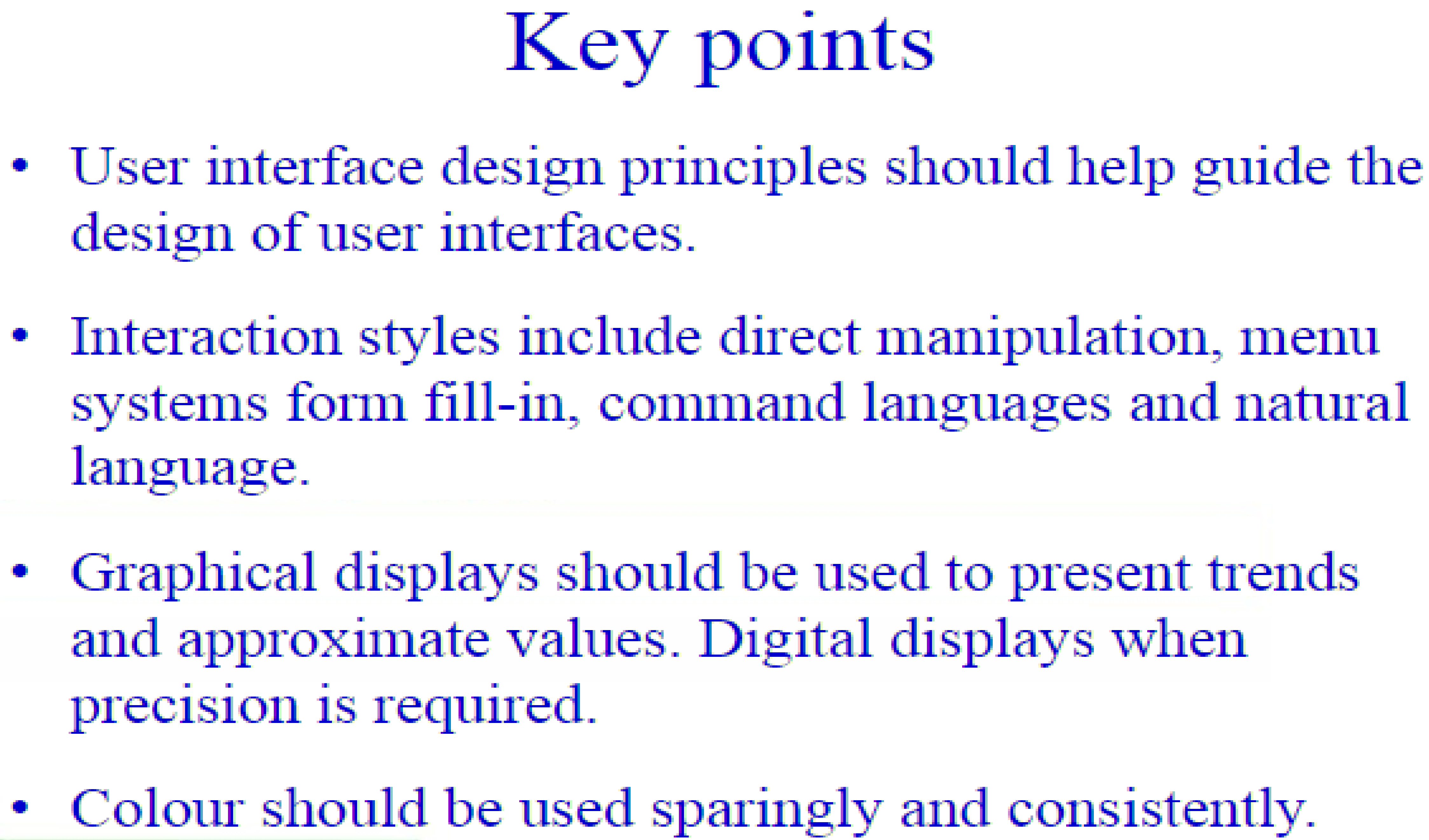
i. Interface Evaluation

Number of evaluation criteria can be applied during early design reviews:

1. The length and complexity of the written specification of the system and its interface provide an indication of the amount of learning required by users of the system.
2. The number of user tasks specified and the average number of actions per task provide an indication of interaction time and the overall efficiency of the system.
3. The number of actions, tasks, and system states indicated by the design model imply the memory load on users of the system.
4. Interface style, help facilities, and error handling protocol provide a general indication of the complexity of the interface and the degree to which it will be accepted by the user.

6. Information Presentation

i. Interface Evaluation

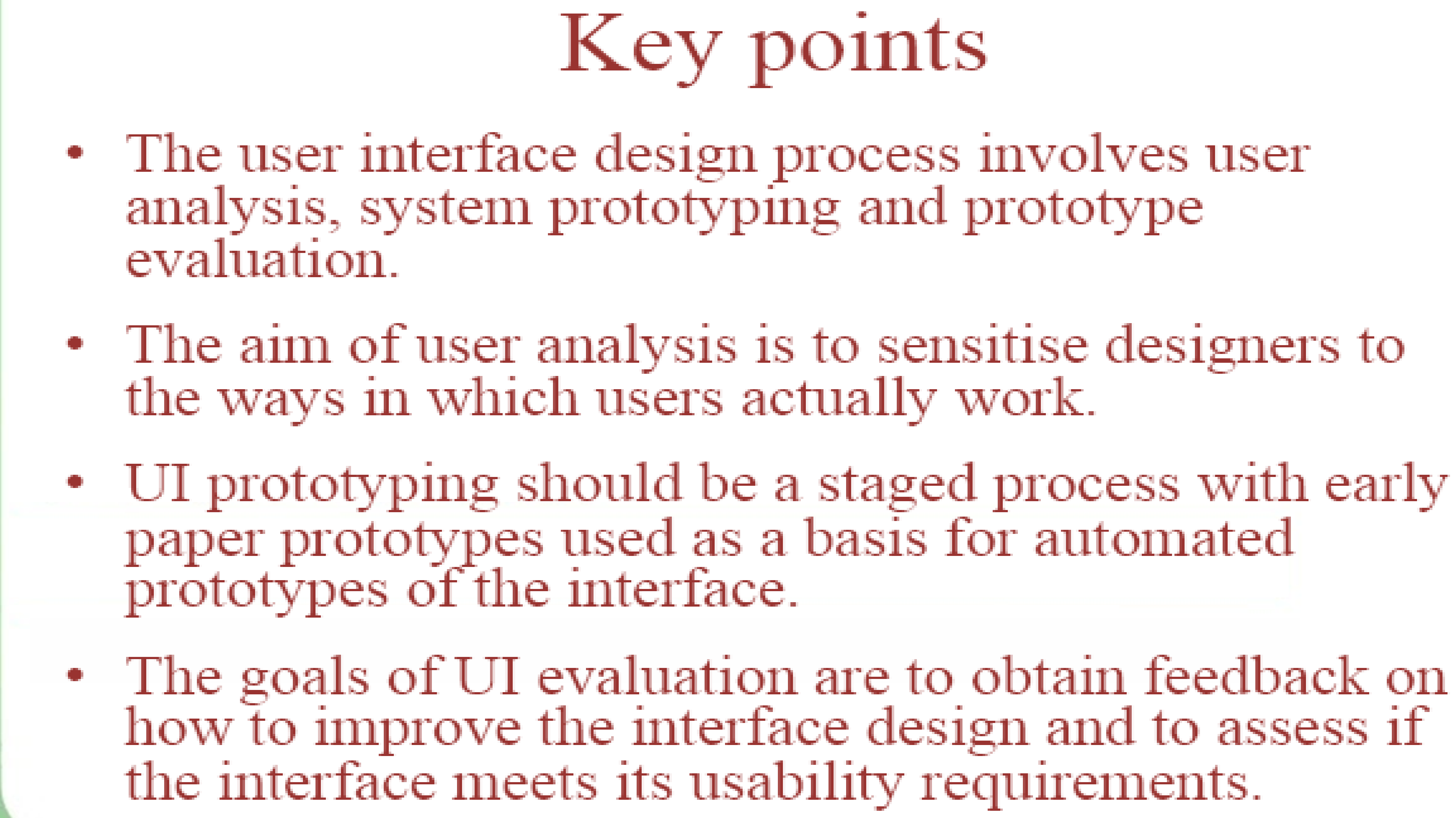


Key points

- User interface design principles should help guide the design of user interfaces.
- Interaction styles include direct manipulation, menu systems form fill-in, command languages and natural language.
- Graphical displays should be used to present trends and approximate values. Digital displays when precision is required.
- Colour should be used sparingly and consistently.

6. Information Presentation

i. Interface Evaluation



Key points

- The user interface design process involves user analysis, system prototyping and prototype evaluation.
- The aim of user analysis is to sensitise designers to the ways in which users actually work.
- UI prototyping should be a staged process with early paper prototypes used as a basis for automated prototypes of the interface.
- The goals of UI evaluation are to obtain feedback on how to improve the interface design and to assess if the interface meets its usability requirements.

6. Information Presentation

i. Design Notation

In software design, as in mathematics, the representation schemes used are of fundamental importance.

Good notation can clarify the interrelationships and interactions of interest, while poor notation can complicate and interface with good design practice.

At least three levels of design specifications exist:

1. External design specifications.
2. Architectural design specifications.
3. Detailed design specifications.

6. Information Presentation

i. Design Notation

In software design, as in mathematics, the representation schemes used are of fundamental importance.

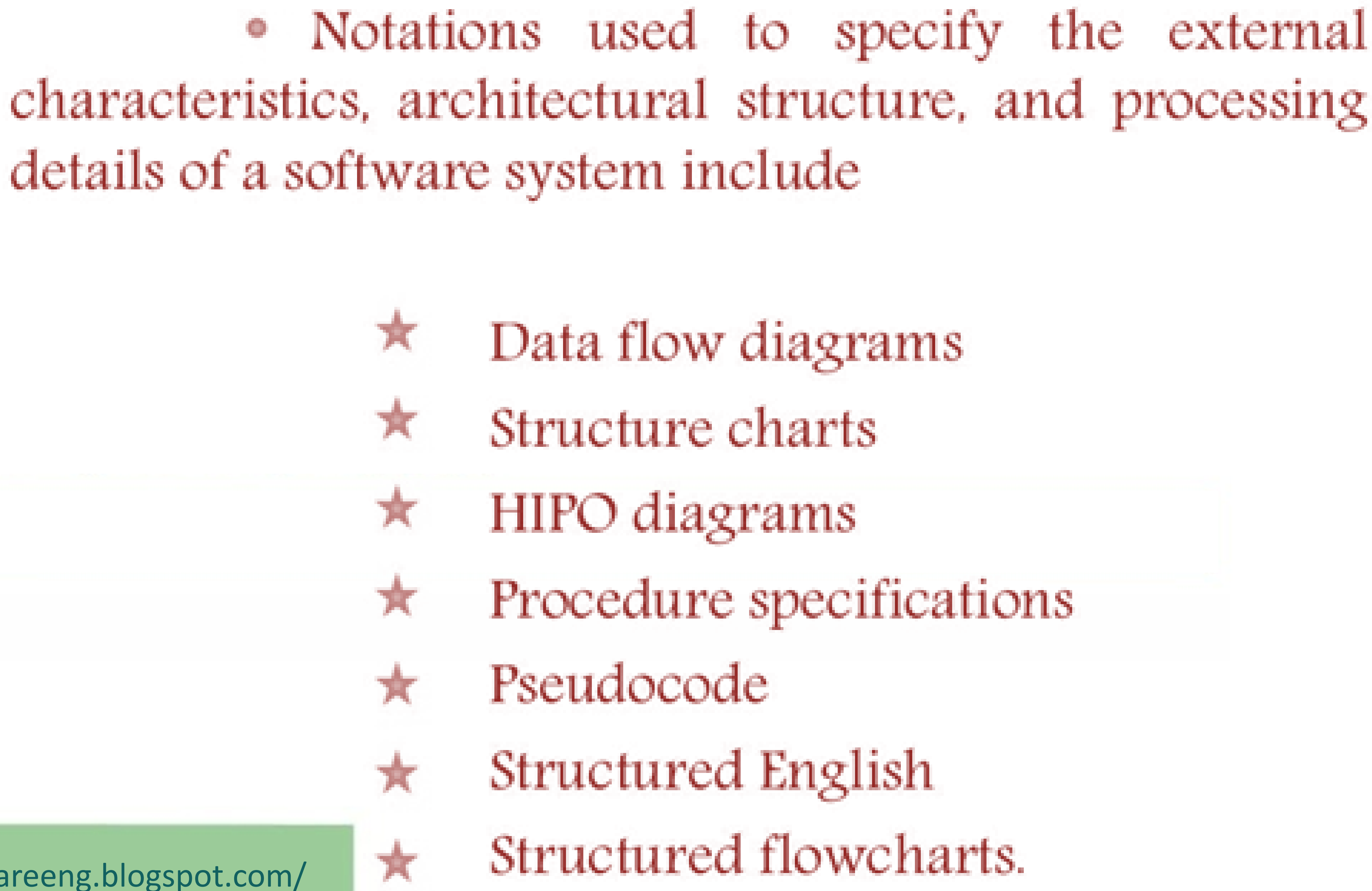
Good notation can clarify the interrelationships and interactions of interest, while poor notation can complicate and interface with good design practice.

At least three levels of design specifications exist:

1. External design specifications.
2. Architectural design specifications.
3. Detailed design specifications.

6. Information Presentation

i. Design Notation

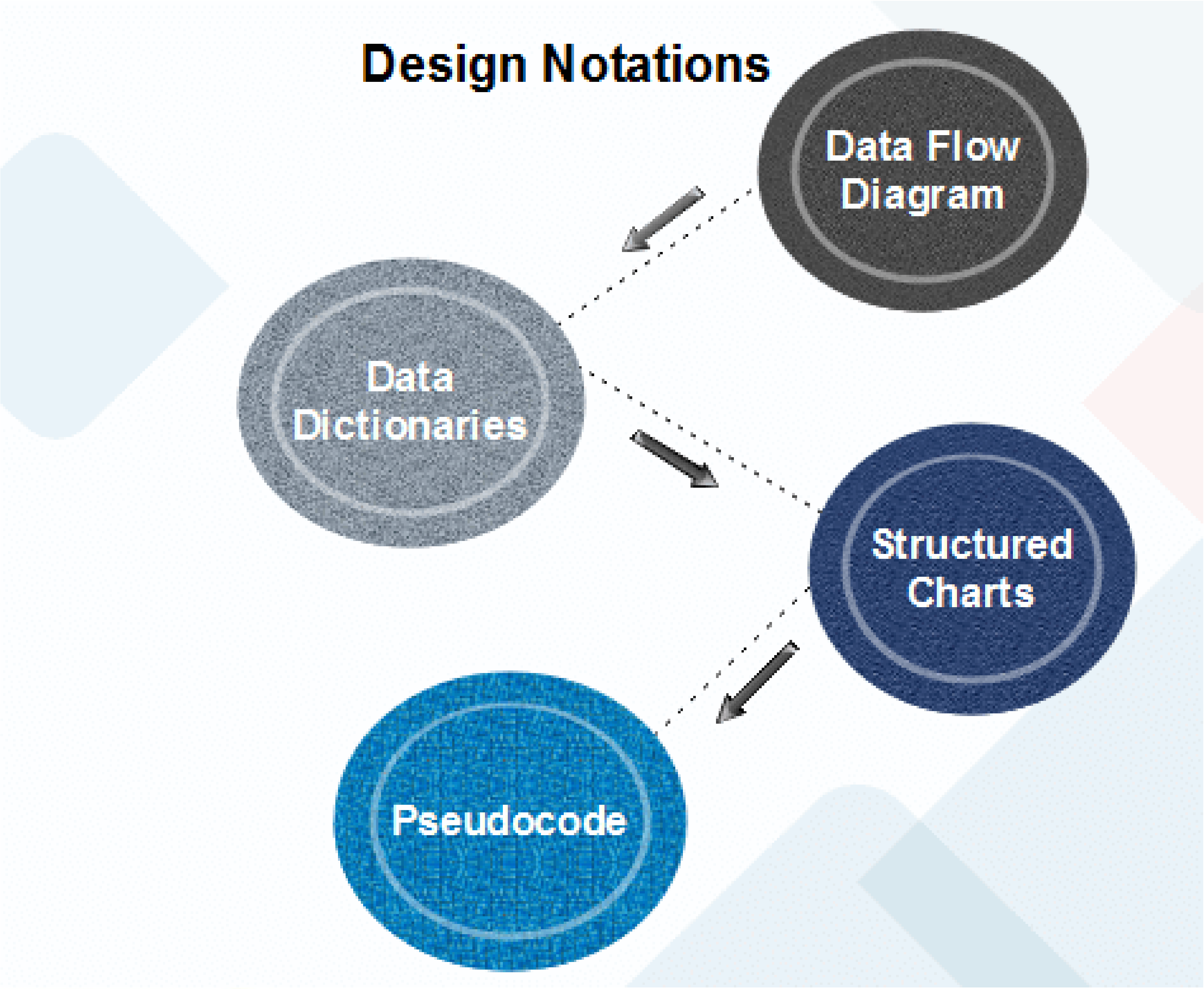


- Notations used to specify the external characteristics, architectural structure, and processing details of a software system include

  ★ Data flow diagrams
  ★ Structure charts
  ★ HIPO diagrams
  ★ Procedure specifications
  ★ Pseudocode
  ★ Structured English
  ★ Structured flowcharts.

6. Information Presentation

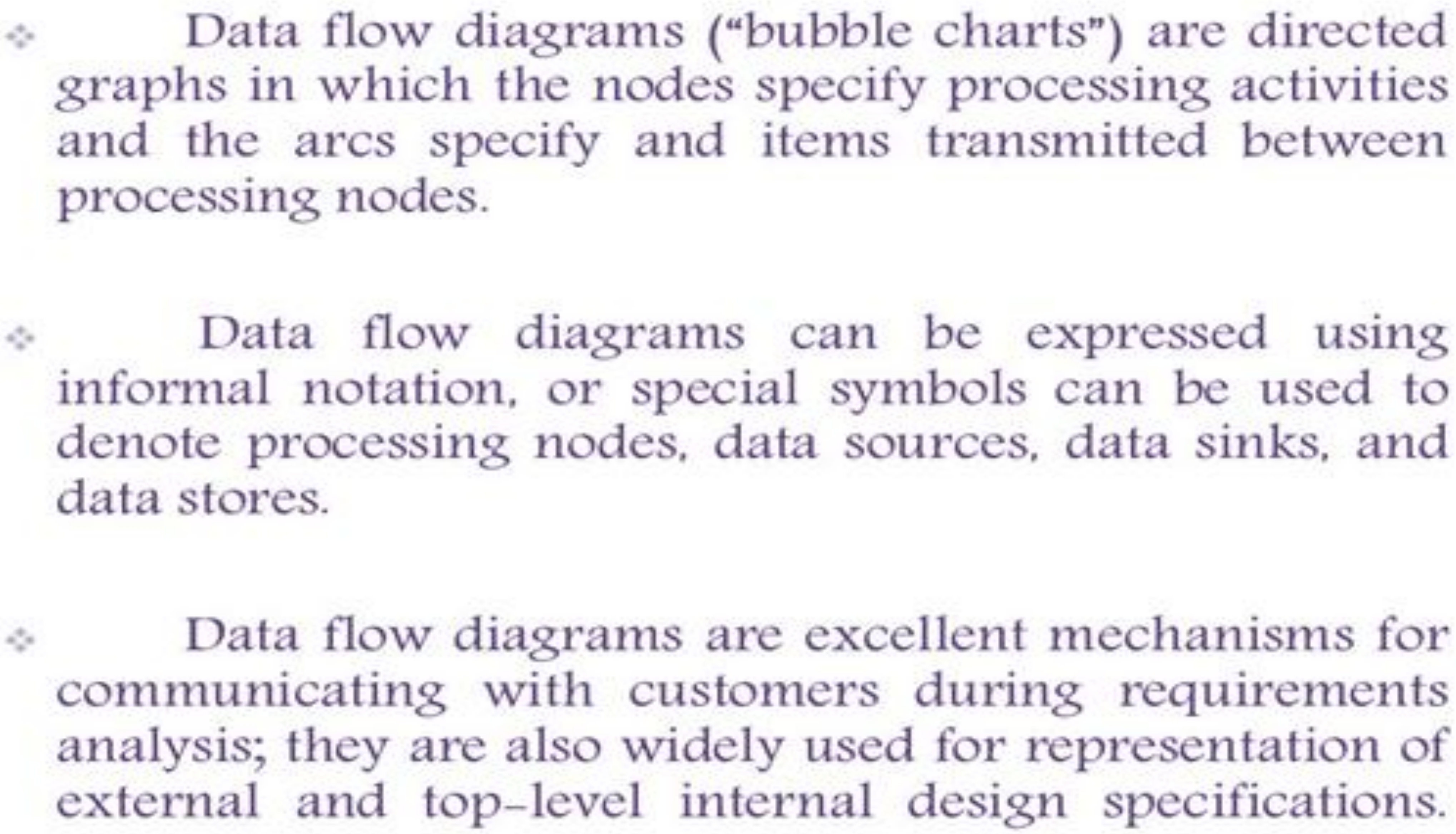
i. Design Notation



6. Information Presentation

i. Design Notation

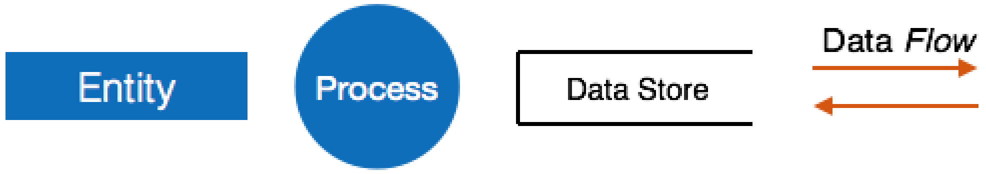
Data Flow Diagram



- Data flow diagrams ("bubble charts") are directed graphs in which the nodes specify processing activities and the arcs specify and items transmitted between processing nodes.

- Data flow diagrams can be expressed using informal notation, or special symbols can be used to denote processing nodes, data sources, data sinks, and data stores.

- Data flow diagrams are excellent mechanisms for communicating with customers during requirements analysis; they are also widely used for representation of external and top-level internal design specifications.



6. Information Presentation

i. Design Notation

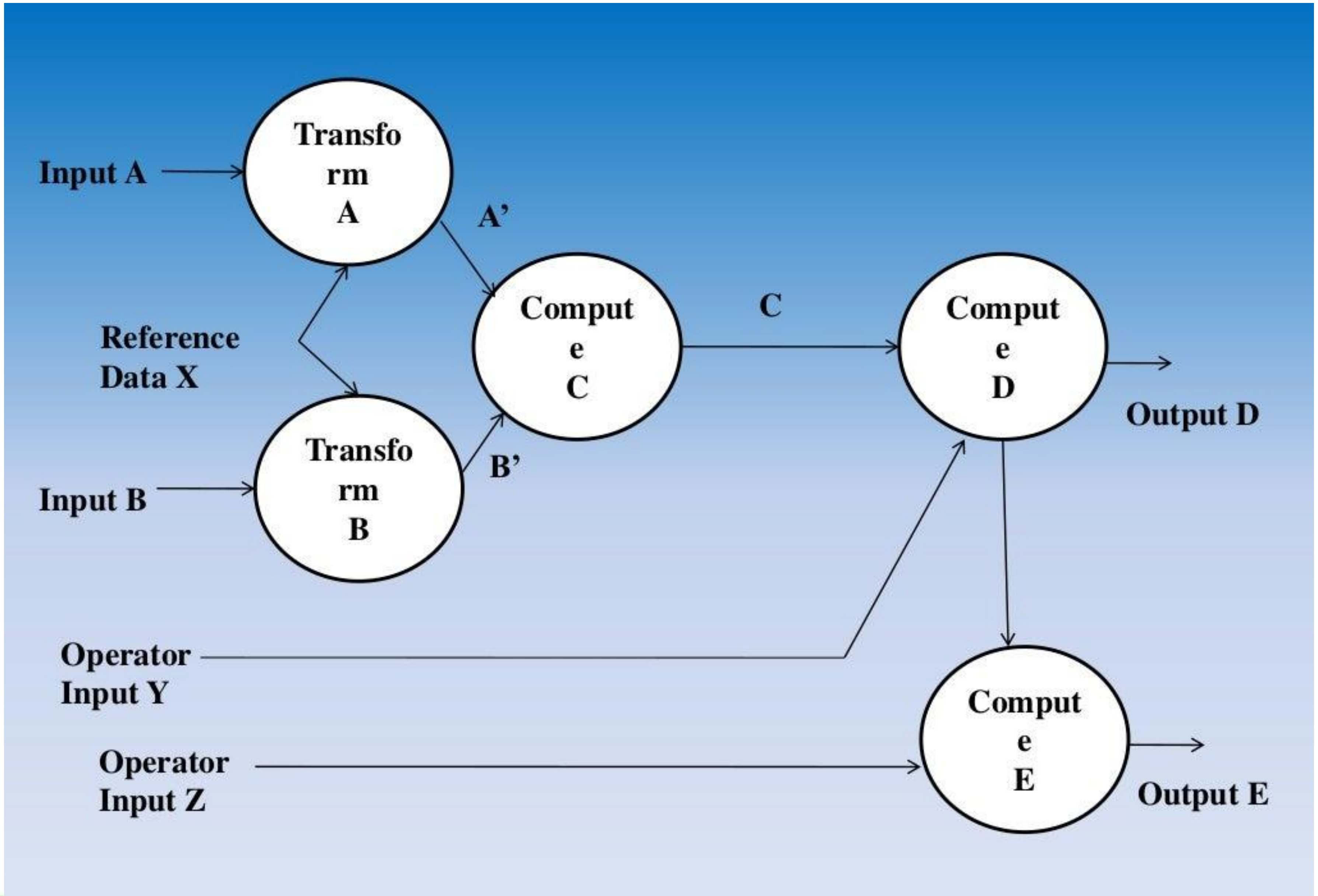
Data Flow Diagram

Symbol	Name	Meaning
🟦	Rounded Rectangle	It represents functions which transforms input to output. The transformation name indicates its function.
🟫	Rectangle	It represents data stores. Again, they should give a descriptive name.
○	Circle	It represents user interactions with the system that provides input or receives output.
→	Arrows	It shows the direction of data flow. Their name describes the data flowing along the path.
"and" and "or"	Keywords	The keywords "and" and "or". These have their usual meanings in boolean expressions. They are used to link data flows when more than one data flow may be input or output from a transformation.

6. Information Presentation

i. Design Notation

Data Flow Diagram



AN INFORMAL DATA FLOW DIAGRAM OR "BUBBLED CHART"

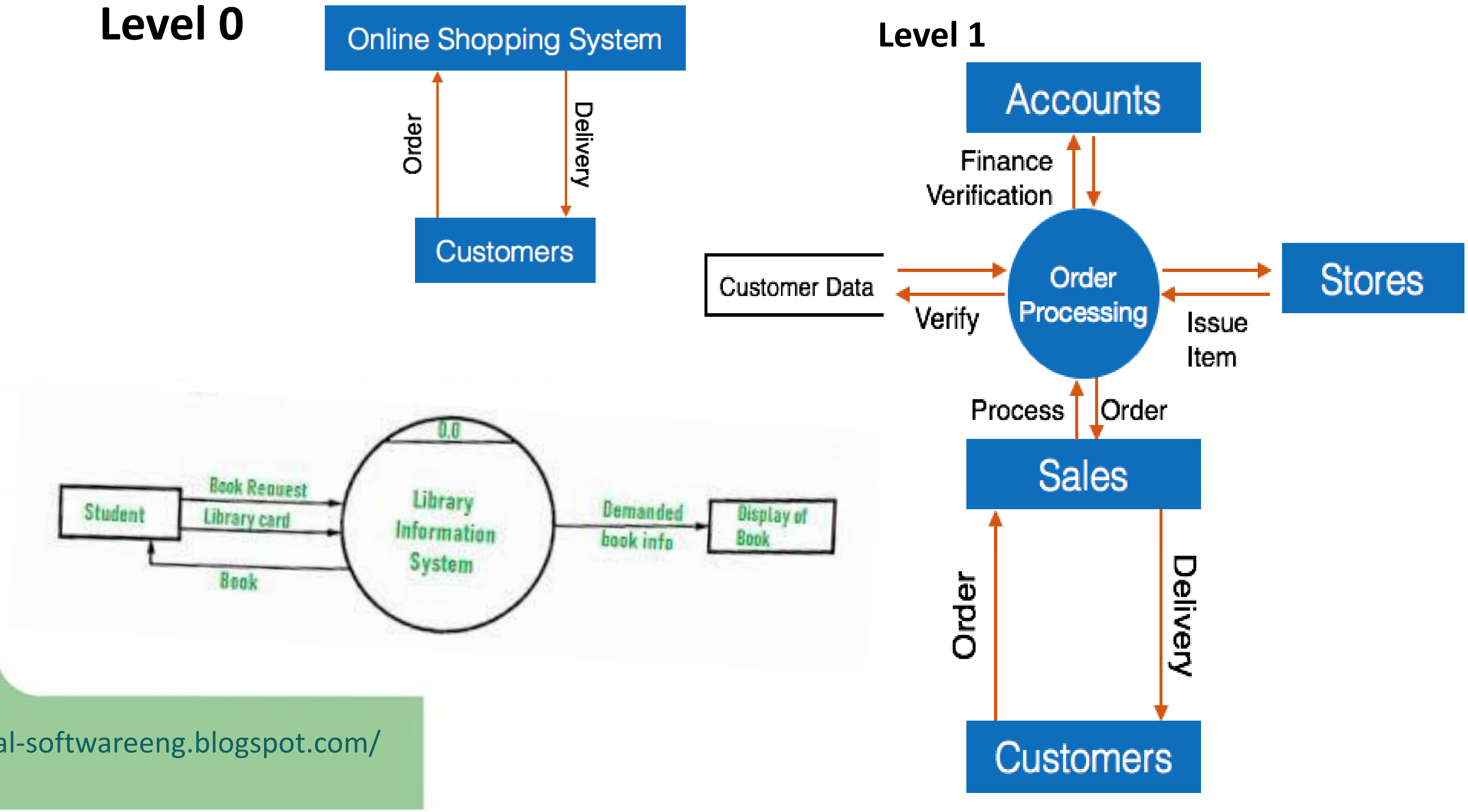
6. Information Presentation

i. Design Notation

Data Flow Diagram

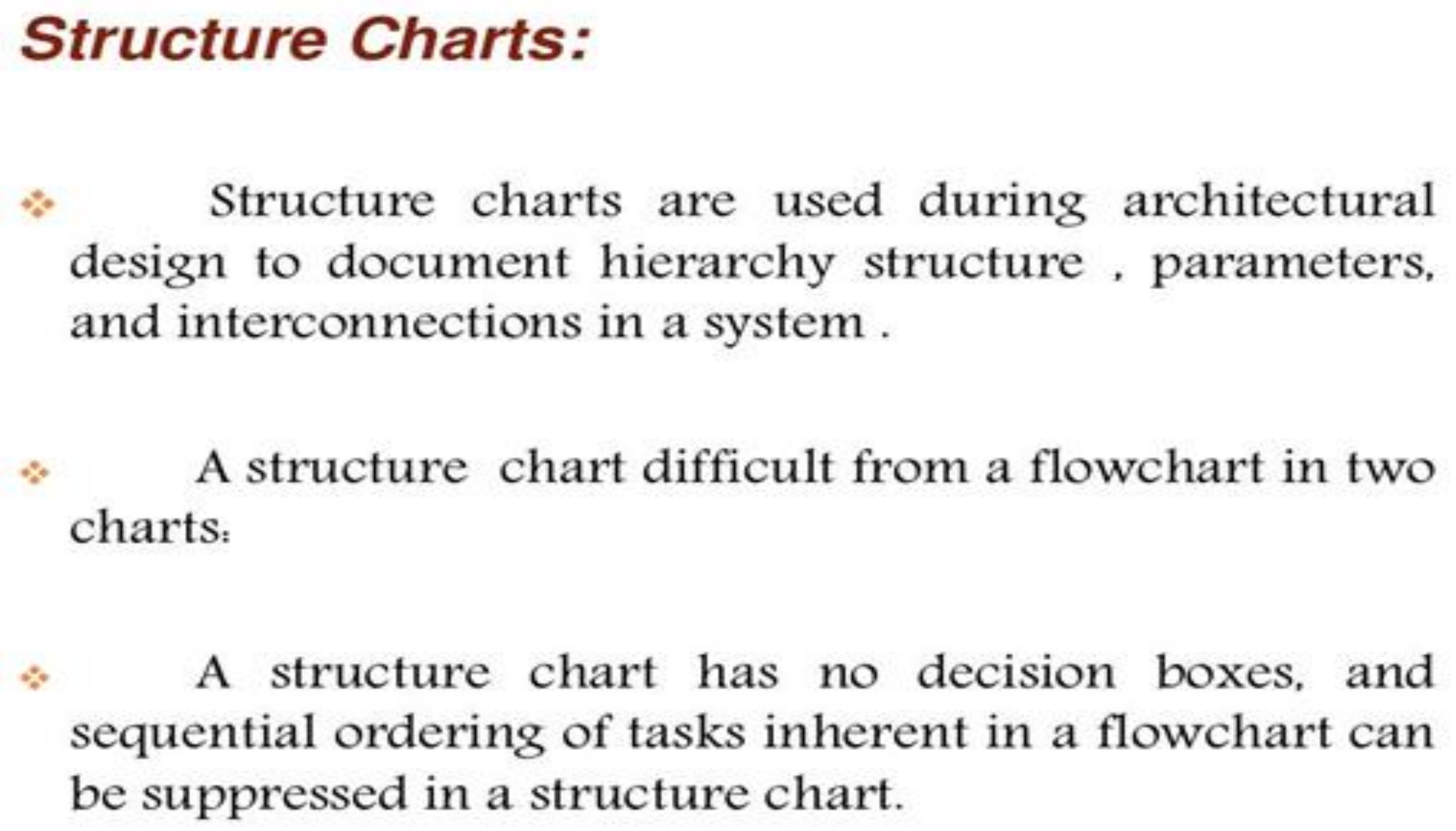
Level 0

Level 1



6. Information Presentation

i. Design Notation



Structure Charts:

- Structure charts are used during architectural design to document hierarchy structure , parameters, and interconnections in a system .

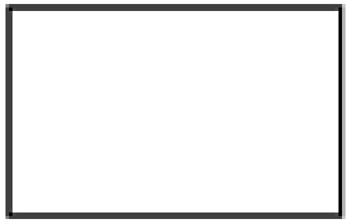
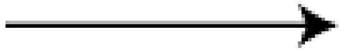
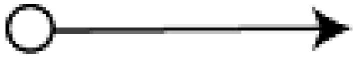
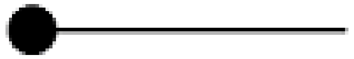
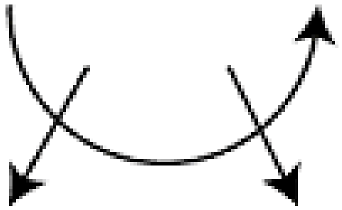
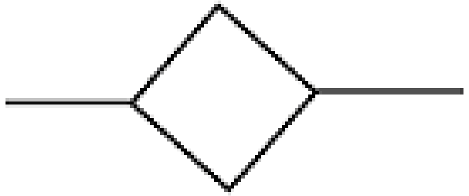
- A structure chart difficult from a flowchart in two charts:

- A structure chart has no decision boxes, and sequential ordering of tasks inherent in a flowchart can be suppressed in a structure chart.

6. Information Presentation

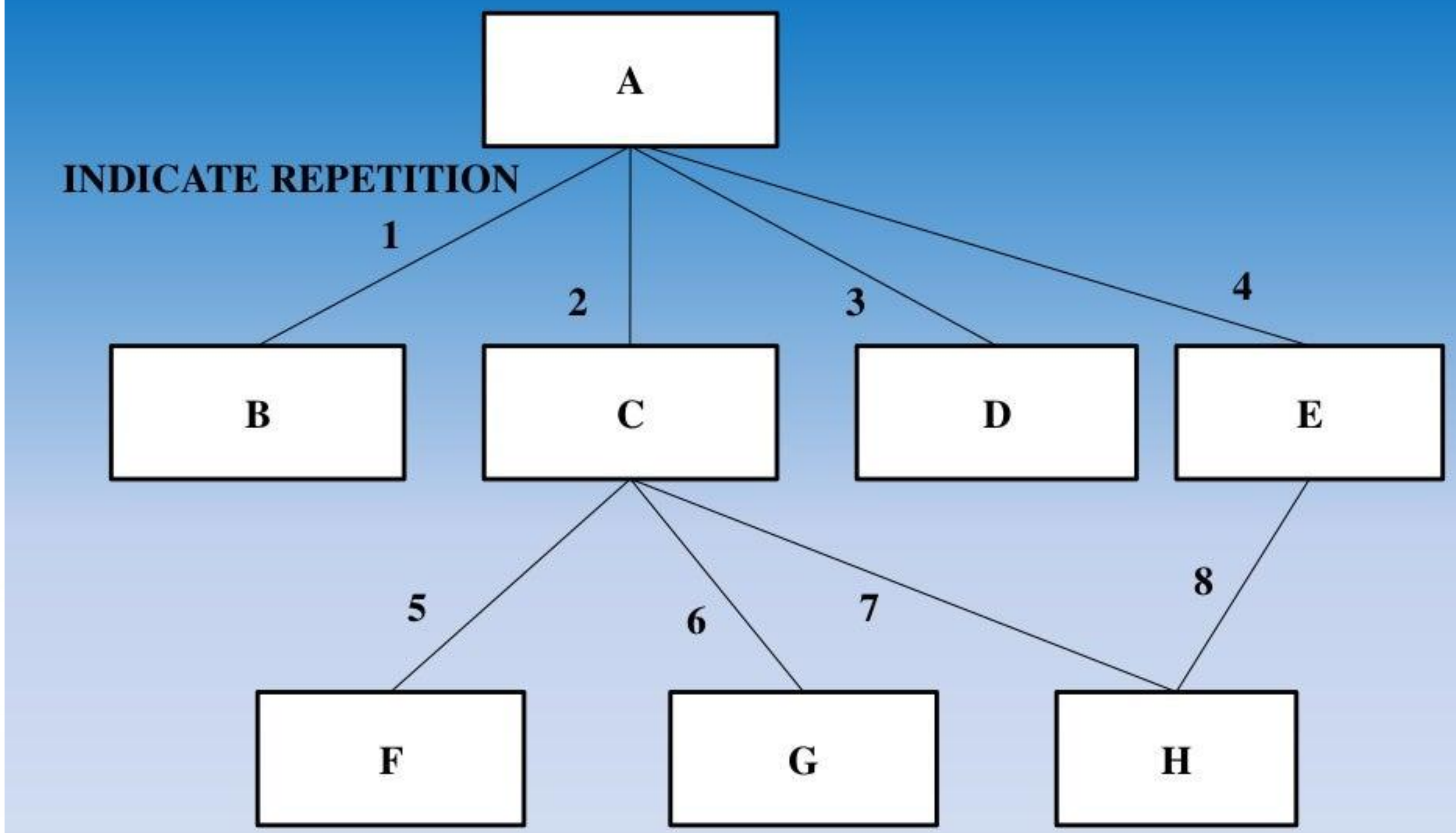
i. Design Notation

Structure Charts:

SYMBOL	DESCRIPTION
	Module
	Arrow
	Data couple
	Control Flag
	Loop
	Decision

6. Information Presentation

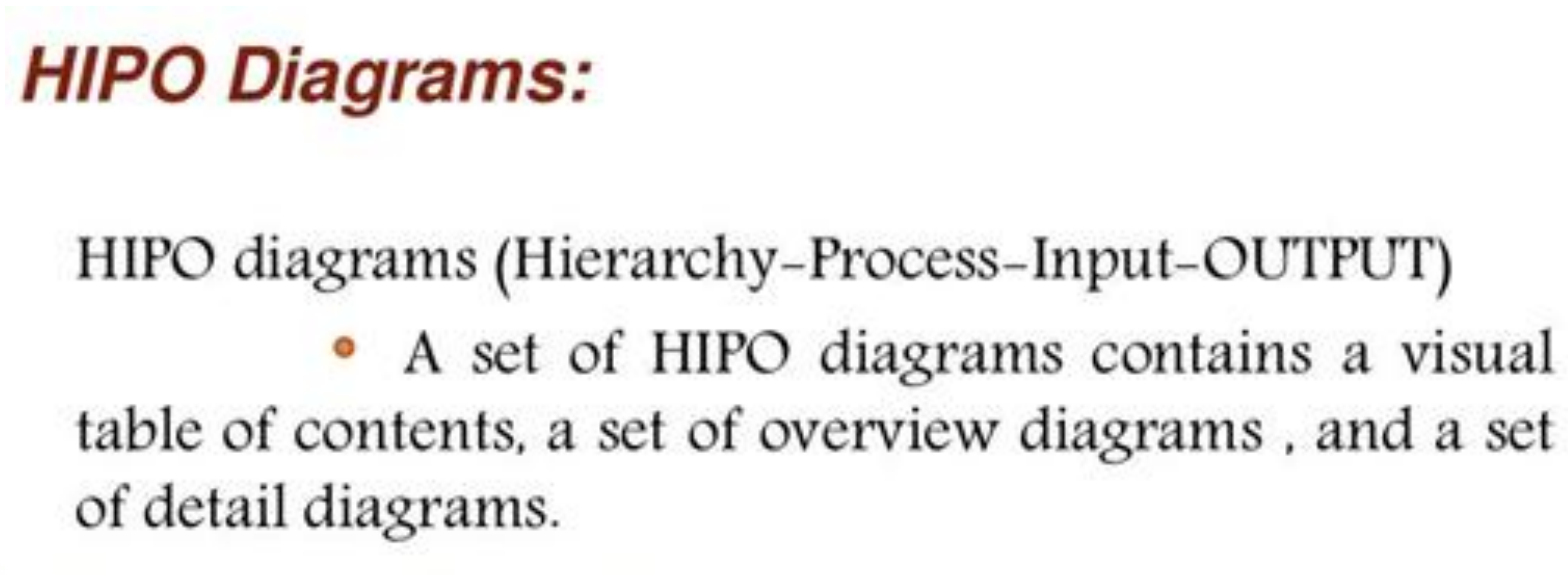
i. Design Notation



FORMAT OF A STRUCTURE

6. Information Presentation

i. Design Notation



HIPO Diagrams:

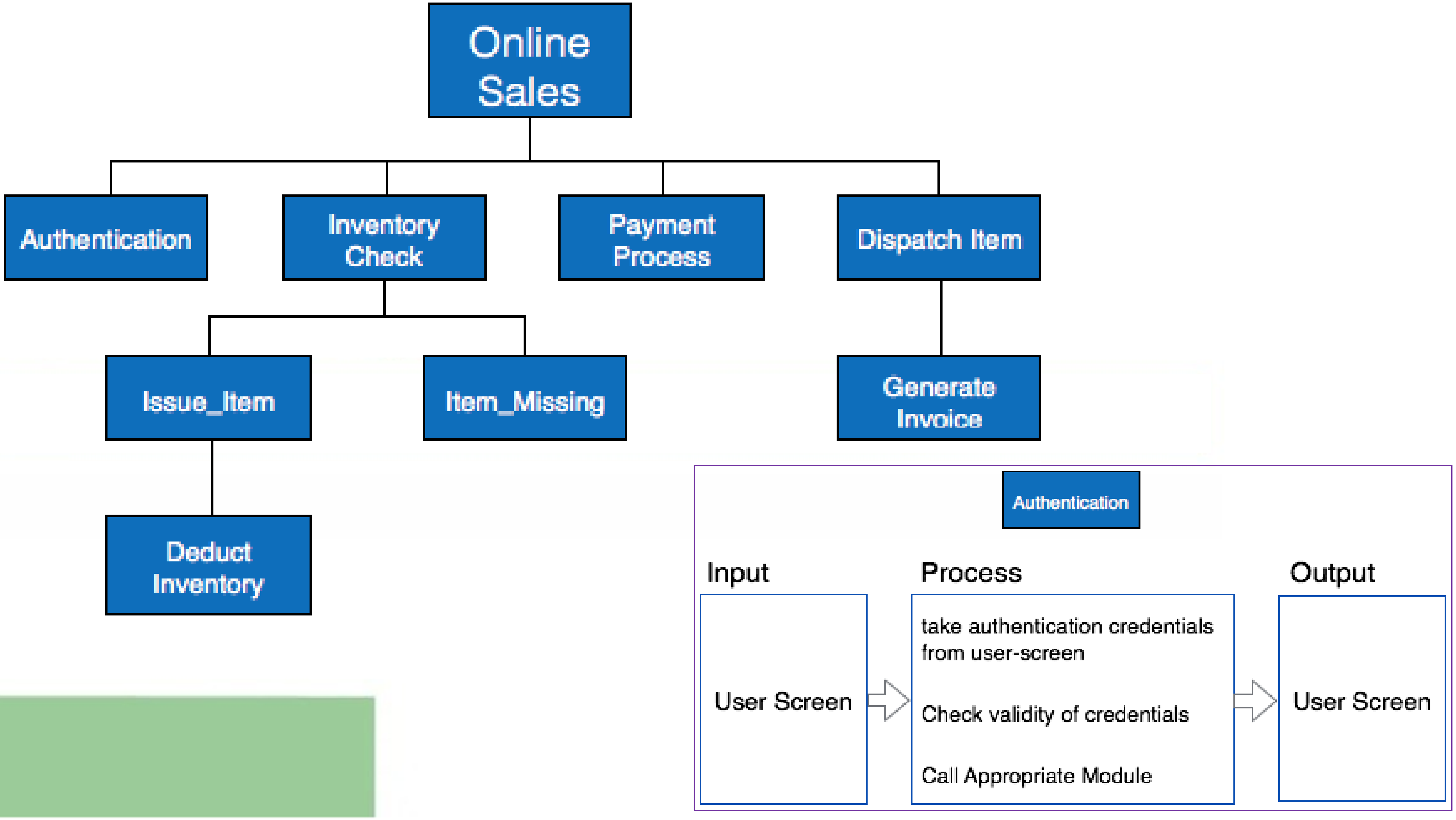
HIPO diagrams (Hierarchy-Process-Input-OUTPUT)

- A set of HIPO diagrams contains a visual table of contents, a set of overview diagrams , and a set of detail diagrams.

6. Information Presentation

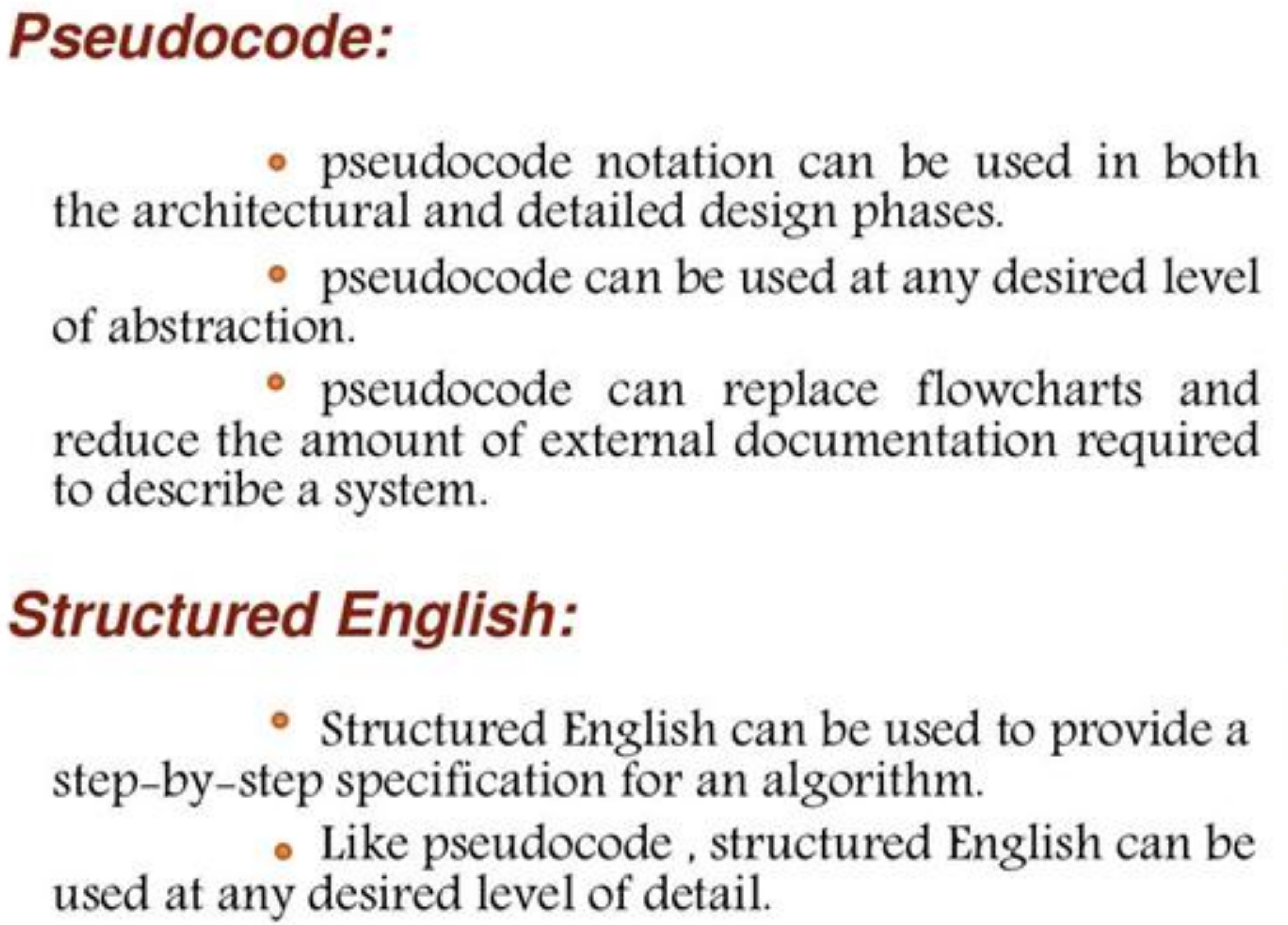
i. Design Notation

HIPO Diagrams:



6. Information Presentation

i. Design Notation



Pseudocode:

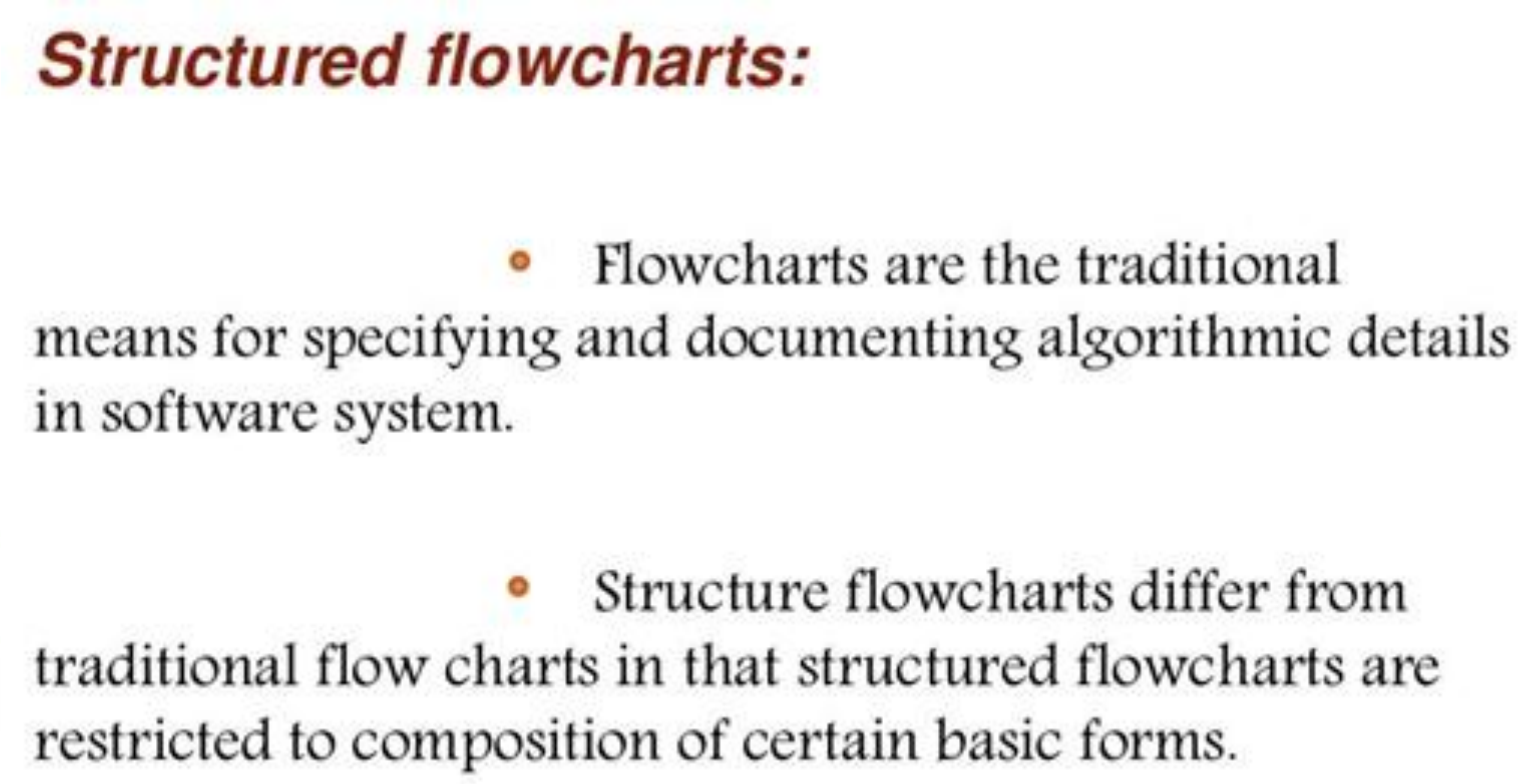
- pseudocode notation can be used in both the architectural and detailed design phases.
- pseudocode can be used at any desired level of abstraction.
- pseudocode can replace flowcharts and reduce the amount of external documentation required to describe a system.

Structured English:

- Structured English can be used to provide a step-by-step specification for an algorithm.
- Like pseudocode , structured English can be used at any desired level of detail.

6. Information Presentation

i. Design Notation



Structured flowcharts:

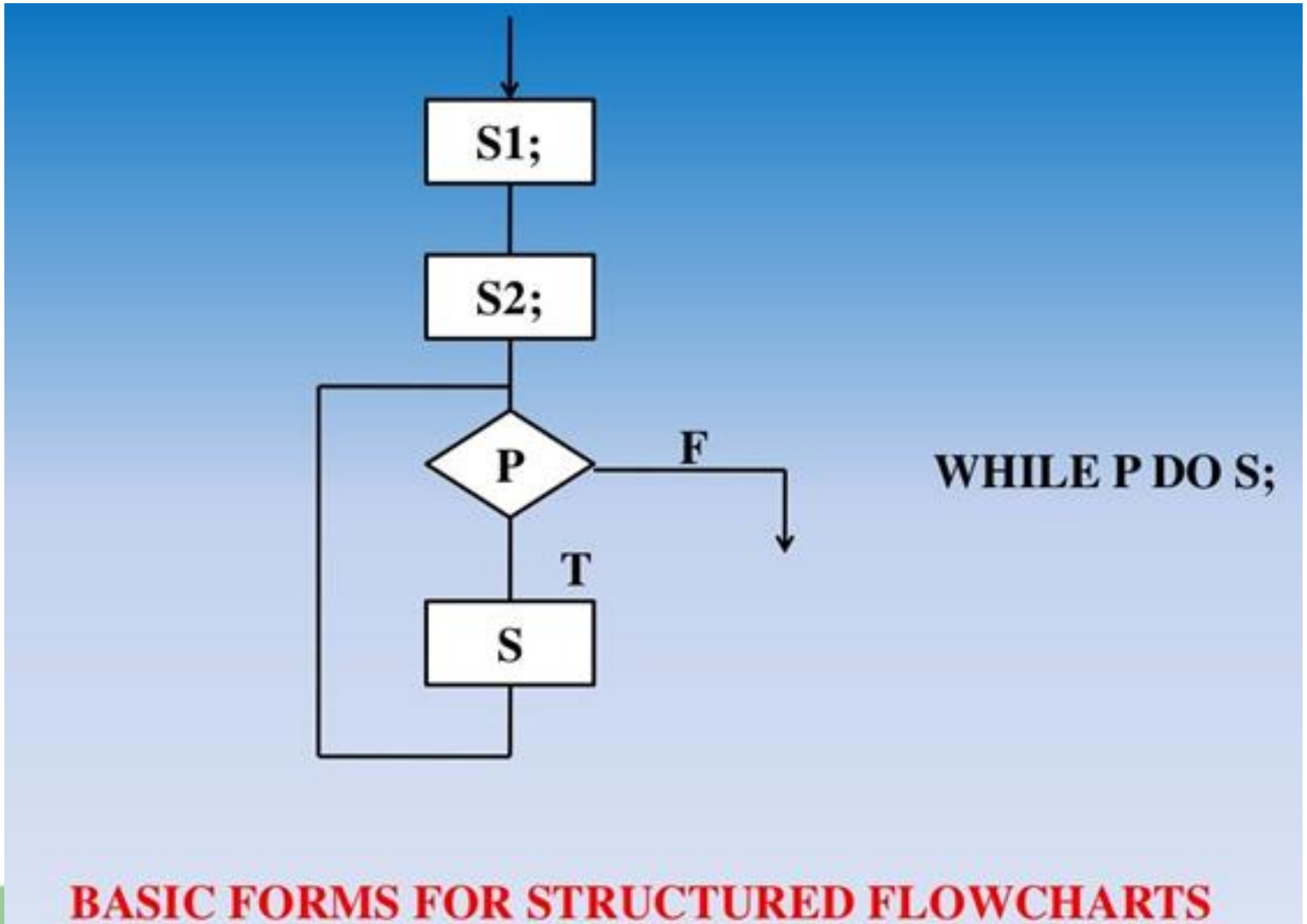
- Flowcharts are the traditional means for specifying and documenting algorithmic details in software system.

- Structure flowcharts differ from traditional flow charts in that structured flowcharts are restricted to composition of certain basic forms.

6. Information Presentation

i. Design Notation

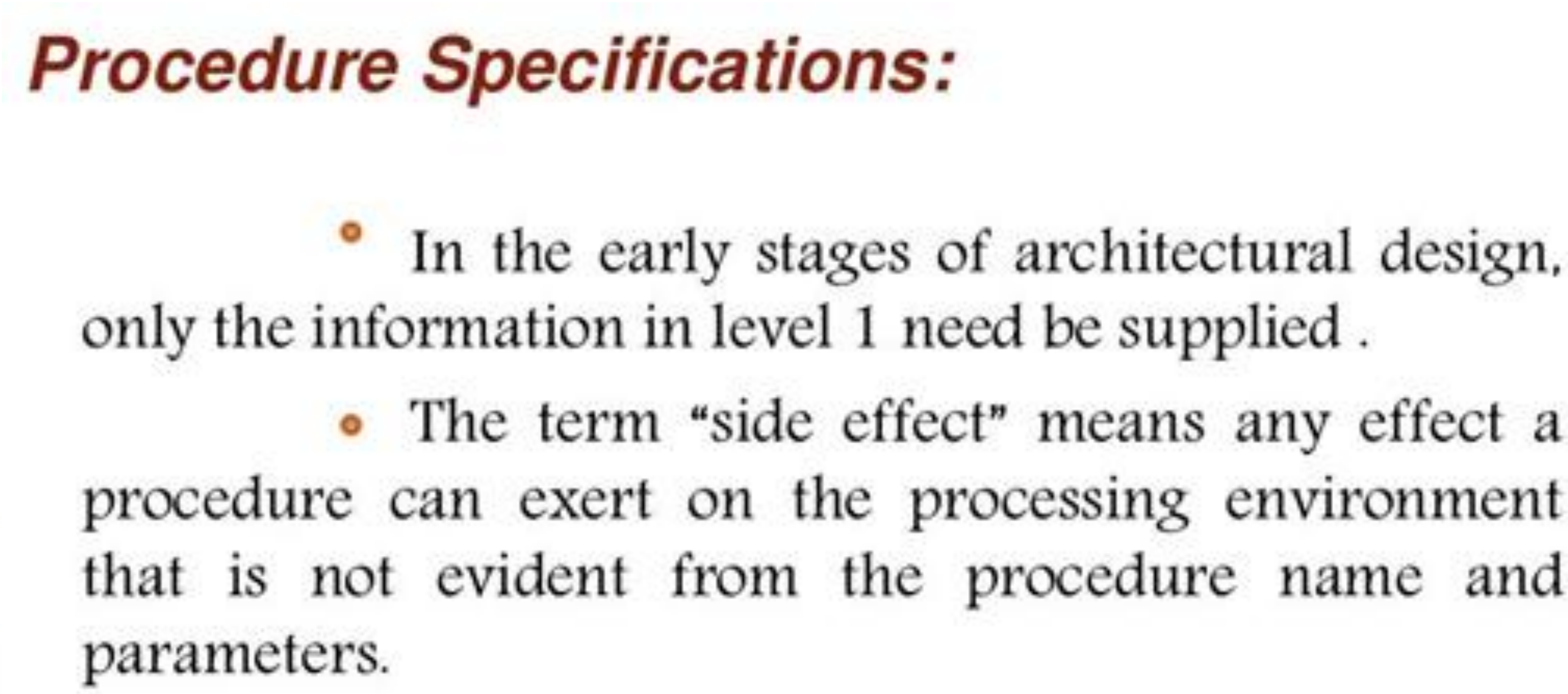
Structured flowcharts:



BASIC FORMS FOR STRUCTURED FLOWCHARTS

6. Information Presentation

i. Design Notation

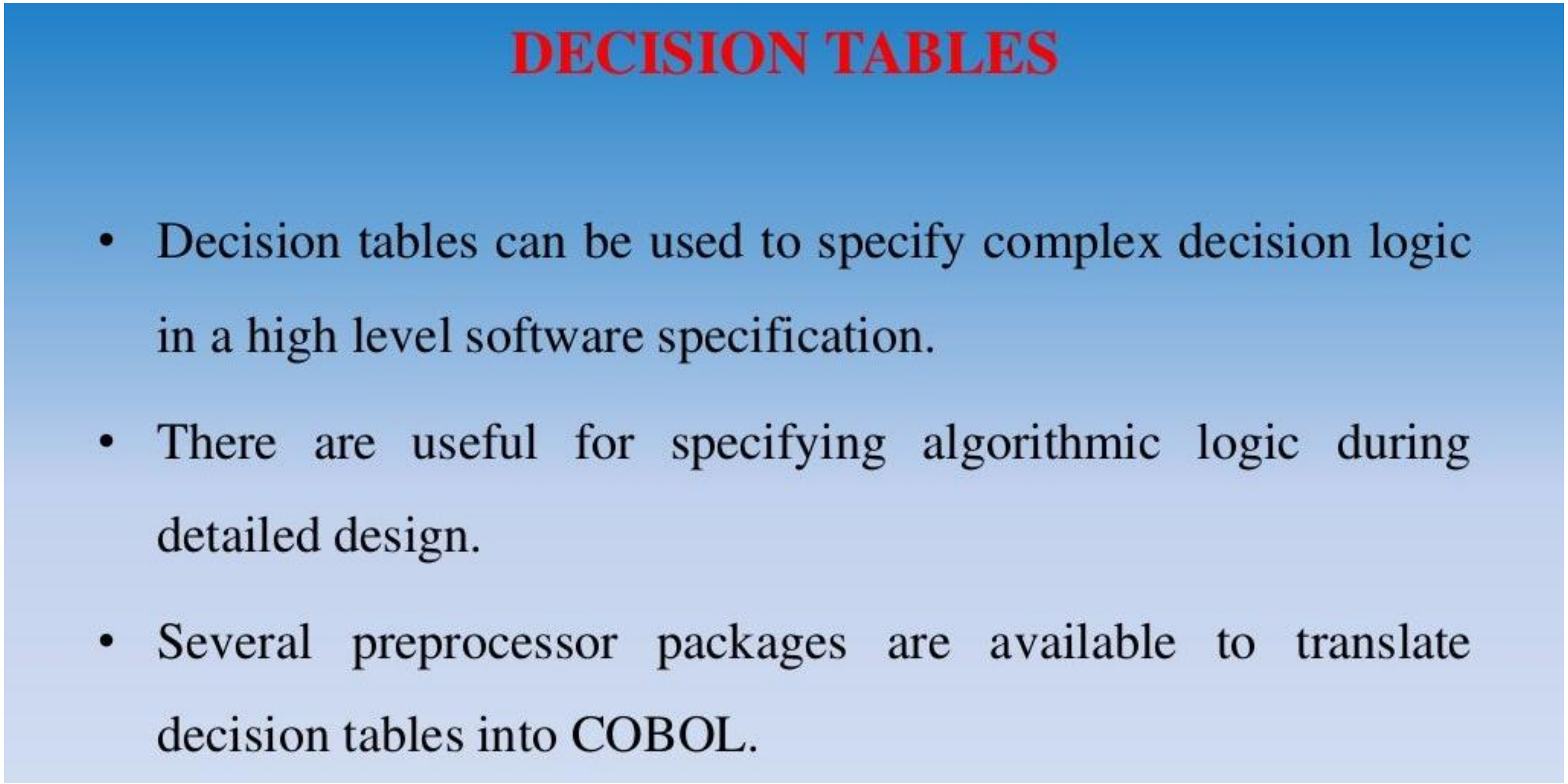


Procedure Specifications:

- In the early stages of architectural design, only the information in level 1 need be supplied .
- The term "side effect" means any effect a procedure can exert on the processing environment that is not evident from the procedure name and parameters.

6. Information Presentation

i. Design Notation



DECISION TABLES

- Decision tables can be used to specify complex decision logic in a high level software specification.
- There are useful for specifying algorithmic logic during detailed design.
- Several preprocessor packages are available to translate decision tables into COBOL.

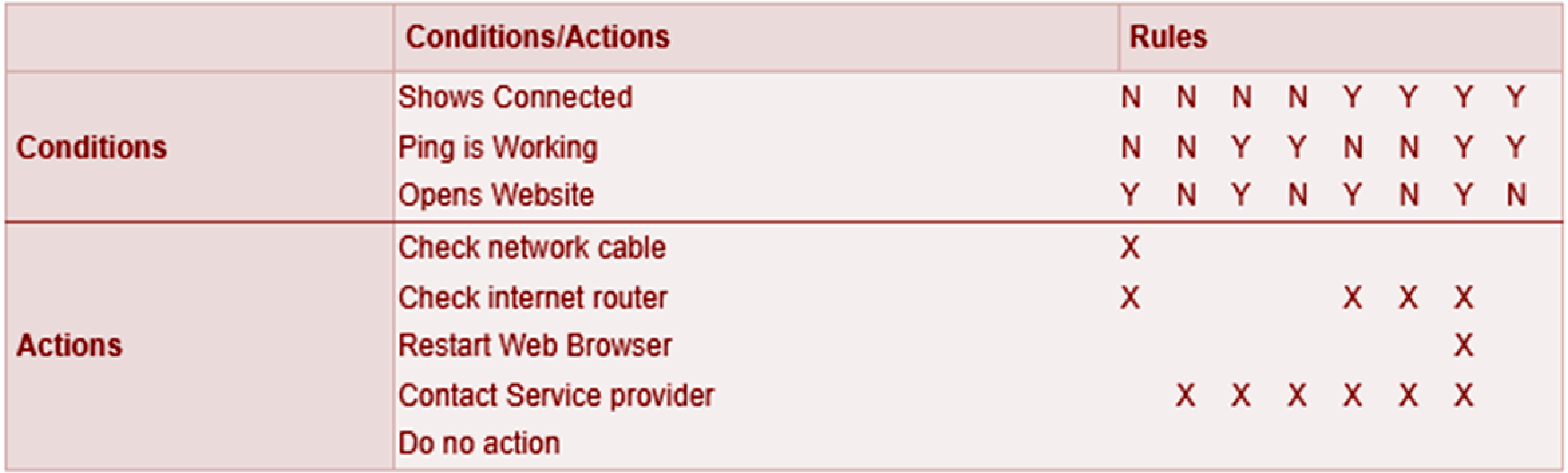
6. Information Presentation

i. Design Notation

Creating Decision Table

Four steps:

1. Identify all possible conditions to be addressed
2. Determine actions for all identified conditions
3. Create Maximum possible rules
4. Define action for each rule



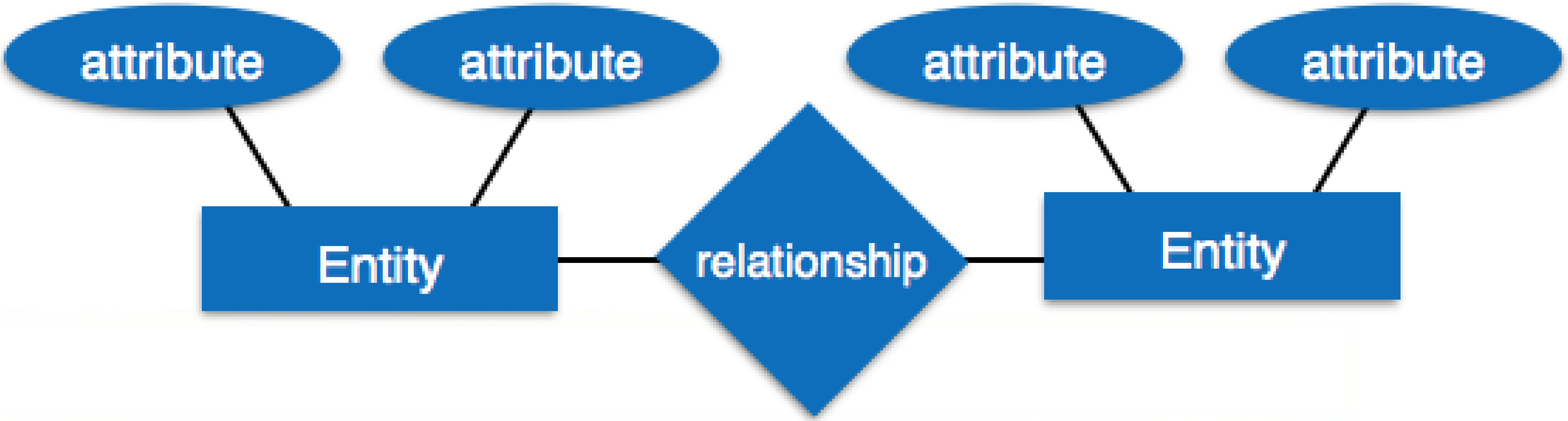
	Conditions/Actions	Rules							
Conditions	Shows Connected	N	N	N	N	Y	Y	Y	Y
	Ping is Working	N	N	Y	Y	N	N	Y	Y
	Opens Website	Y	N	Y	N	Y	N	Y	N
Actions	Check network cable	X							
	Check internet router	X				X	X	X	
	Restart Web Browser							X	
	Contact Service provider		X	X	X	X	X	X	
	Do no action								

Table : Decision Table – In-house Internet Troubleshooting

6. Information Presentation

i. Design Notation

Entity-Relationship Model



6. Information Presentation

i. Design Notation

Data Dictionary

Data dictionary is the centralized collection of information about data. It stores meaning and origin of data, its relationship with other data, data format for usage etc.

Data dictionary is often referenced as meta-data (data about data) repository. It is created along with DFD (Data Flow Diagram) model of software program and is expected to be updated whenever DFD is changed or updated.

Data dictionary should contain information about the following

1. Data Flow
2. Data Structure
3. Data Elements
4. Data Stores
5. Data Processing

=	Composed of
{}	Repetition
()	Optional
+	And
[/]	Or

6. Information Presentation

i. Design Notation

Data Dictionary

Example

Address = House No + (Street / Area) + City + State

Course ID = Course Number + Course Name + Course Level + Course Grades

User Interface

User interface design: User interface design (UI) or user interface engineering is the design of user interfaces for machines and software, such as computers, home appliances, mobile devices, and other electronic devices, with the focus on maximizing the user experience. The goal of user interface design is to make the user's interaction as simple and efficient as possible, in terms of accomplishing user goals (user-centered design).

Interaction design: In design, human–computer interaction, and software development, interaction design, often abbreviated IxD, is defined as "the practice of designing interactive digital products, environments, systems, and services." Like many other design fields interaction design also has an interest in form but its main focus is on behavior. What clearly marks interaction design as a design field as opposed to a science or engineering field is that it is synthesis and imagining things as they might be, more so than focusing on how things are.

User Interface

User experience design: User Experience Design (UXD or UED or XD) is the process of enhancing user satisfaction by improving the usability, accessibility, and pleasure provided in the interaction between the user and the product. User experience design encompasses traditional human–computer interaction (HCI) design, and extends it by addressing all aspects of a product or service as perceived by users.

User-centered design: User-centred design (UCD) is a framework of processes (not restricted to interfaces or technologies) in which the needs, wants, and limitations of end users of a product, service or process are given extensive attention at each stage of the design process. User-centred design can be characterized as a multi-stage problem solving process that not only requires designers to analyse and foresee how users are likely to use a product, but also to test the validity of their assumptions with regard to user behavior in real world tests with actual users at each stage of the process from requirements, concepts, pre-production models, mid production and post production creating a circle of proof back to and confirming or modifying the original requirements. Such testing is necessary as it is often very difficult for the designers of a product to understand intuitively what a first-time user of their design experiences, and what each user's learning curve may look like.