CSE0613-2201 System Analysis & Design

Systems Analysis and Design

Partho Sarathi Sarker
Asst. Professor
Dept. of Computer Science and Engineering(CSE)
University of Global Village (UGV)

System Analysis & Design

Course Code:	CSE 0613-2201	Credits:	02
		CIE Marks:	60
Exam Hours:	02	SEE Marks:	40

Course Learning Outcome (CLOs): After Completing this course successfully, the student will be able to…

CLO1	Understand fundamental principles and methodologies of system analysis and design.
CLO2	Apply system analysis techniques to gather requirements and develop models.
CLO3	Design system architecture, interfaces, and data structures using appropriate tools and strategies.
CLO4	Evaluate and implement system development methodologies, scalability, performance, and security mechanisms.
CLO5	Develop, test, and deploy system designs through practical projects and real-world applications.

Summary of Course Content

Sl. No.	COURSE CONTENT	HRs	CLOs
1	System Design & Analysis: SDLC stages, system development methodologies	2	CLO1 & CLO2
2	System Elements & Types: Elements, types of systems, and system categories	2	CLO1
3	Information Systems & Role of Analyst and Planning: MIS, DSS, TPS, and role of system analyst & analysis strategies	2	CLO1, CLO2 & CLO3
4	Structured Analysis & Models: DFDs, ER diagrams & data dictionaries	2	CLO4
5	Implementation & Testing: System coding, unit testing, system integration	2	CLO5

1. “Systems Analysis and Design” by Elias M Awad
2. "Object-Oriented Systems Analysis and Design Using UML" by Simon Bennett, Steve McRobb and Ray Farmer, 2nd Edition
3. "Systems Analysis and Design" by Kenneth E. Kendall and Julie E. Kendall, 10th Edition

Assessment Pattern

Bloom's Category Marks (out of 90)	Tests (30)	Assignments (10)	Quizzes (10)	Attendance (10)
Remember	04	03		
Understand	05	02	04	
Apply	06	03	03	
Analyze	05			
Evaluate	05	02	03	
Create	05			

SEE- Semester End Examination (40 Marks)

Bloom's Category	Test
Remember	05
Understand	05
Apply	13
Analyze	07
Evaluate	05
Create	05

Course Plan

Week	Topics	Teaching Strategy	Assessment Strategy	Mapped CLO(s)
1	Introduction to Systems	Lecture, multimedia, discussions	Feedback, Q&A, assessment	CLO1
2	System Elements and Categories	Lecture, discussions	Q&A, Midterm assessments	CLO1
3	Introduction to Systems Analysis	Lecture, multimedia	Midterm assessments	CLO1, CLO2
4	Techniques in Systems Analysis	Lecture, discussions	Midterm assessments	CLO2
5	System Design Principles	Lecture, discussions	Feedback, Q&A, assessment	CLO1, CLO3
6	System Design Tools	Lecture, multimedia, discussions	Q&A, assignments	CLO3
7	SDLC Overview	Lecture, multimedia, discussions	Ethical analysis, Final assessments	CLO1, CLO2
8	SDLC - System Design	Interactive lectures, examples from real-world applications	Final term assessments	CLO3

Course Plan

Week	Topics	Teaching Strategy	Assessment Strategy	Mapped CLO(s)
9	Implementation Strategies	Lecture, multimedia, discussions	Final term assessments	CLO4
10	Testing in Systems Design	Revision through Q&A, group activities	Participation, group evaluation	CLO4
11	Scalability and Performance	Lecture, multimedia, discussions	Feedback, Q&A, assessment	CLO4
12	Security in System Design	Lecture, discussions	Q&A, Midterm assessments	CLO4
13	Advanced Analysis Techniques	Lecture, multimedia	Midterm assessments	CLO2, CLO4
14	Challenges in System Design	Lecture, discussions	Midterm assessments	CLO4
15	Roles and Responsibilities	Lecture, discussions	Feedback, Q&A, assessment	CLO1, CLO2
16	Real-World Applications	Lecture, multimedia, discussions	Q&A, assignments	CLO5
17	Capstone Project and Review	Lecture, multimedia, discussions	Ethical analysis, Final assessments	CLO5

week 1&2
Systems Analysis and Design Basics

Systems Analysis and Design

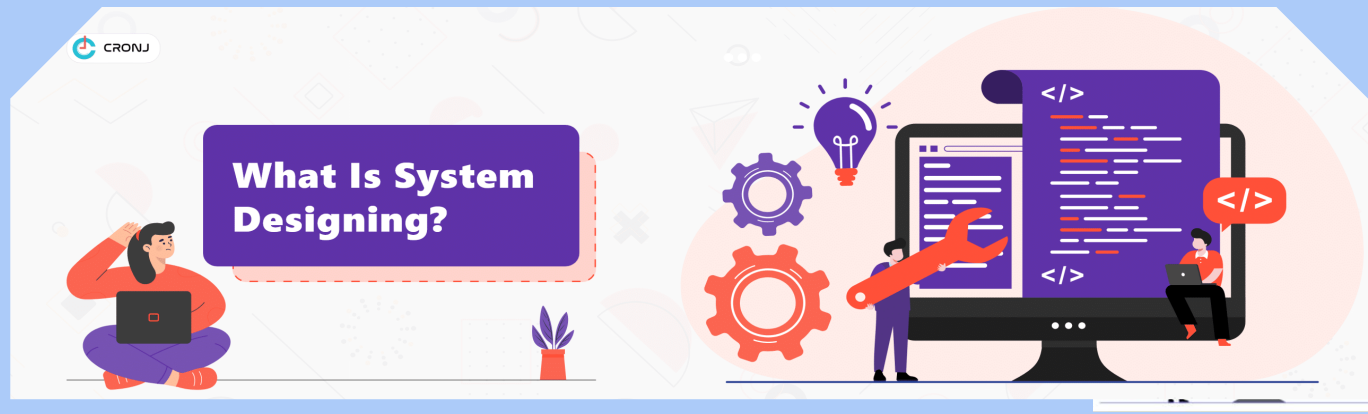
- Systems development is systematic process which includes phases such as planning, analysis, design, deployment, and maintenance.
- Systems analysis
- Systems design

Systems Analysis

- It is a process of collecting and interpreting facts, identifying the problems, and decomposition of a system into its components.
- System analysis is conducted for the purpose of studying a system or its parts in order to identify its objectives. It is a problem solving technique that improves the system and ensures that all the components of the system work efficiently to accomplish their purpose.
- Analysis specifies what the system should do.

Systems Design

- It is a process of planning a new business system or replacing an existing system by defining its components or modules to satisfy the specific requirements.
- Before planning, you need to understand the old system thoroughly and determine how computers can best be used in order to operate efficiently.
- System Design focuses on how to accomplish the objective of the system.



Properties of Systems

- Organization
- Interaction
- Interdependence
- Integration
- Integration
- Central Objective

Properties of Systems

Organization

- Organization implies structure and order. It is the arrangement of components that helps to achieve predetermined objectives.

- An **organization** or **organisation** is an entity—such as a company, or corporation or an institution (formal organization), or an association—comprising one or more people and having a particular purpose.

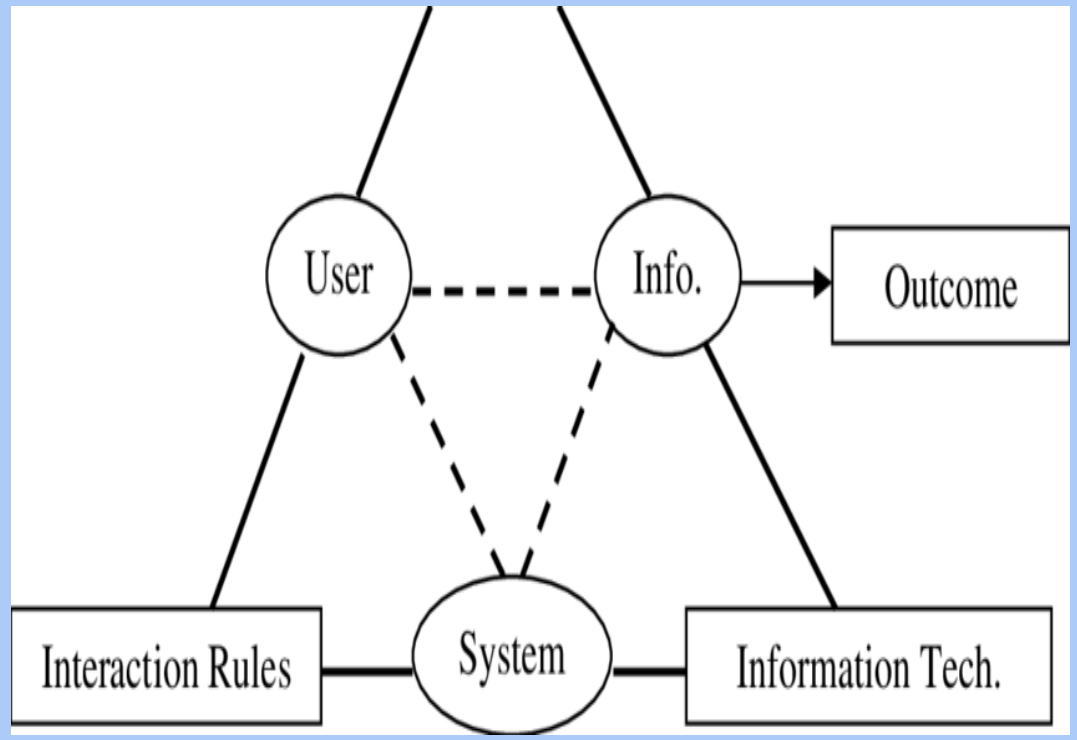
Properties of Systems

Interaction

- Interaction defined by the manner in which the components operate with each other. For example, in an organization, purchasing department must interact with production department and payroll with personnel department.

Interdependence

- Interdependence means how the components of a system depend on one another. For proper functioning, the components are coordinated and linked together according to a specified plan. The output of one subsystem is the required by other subsystem as input



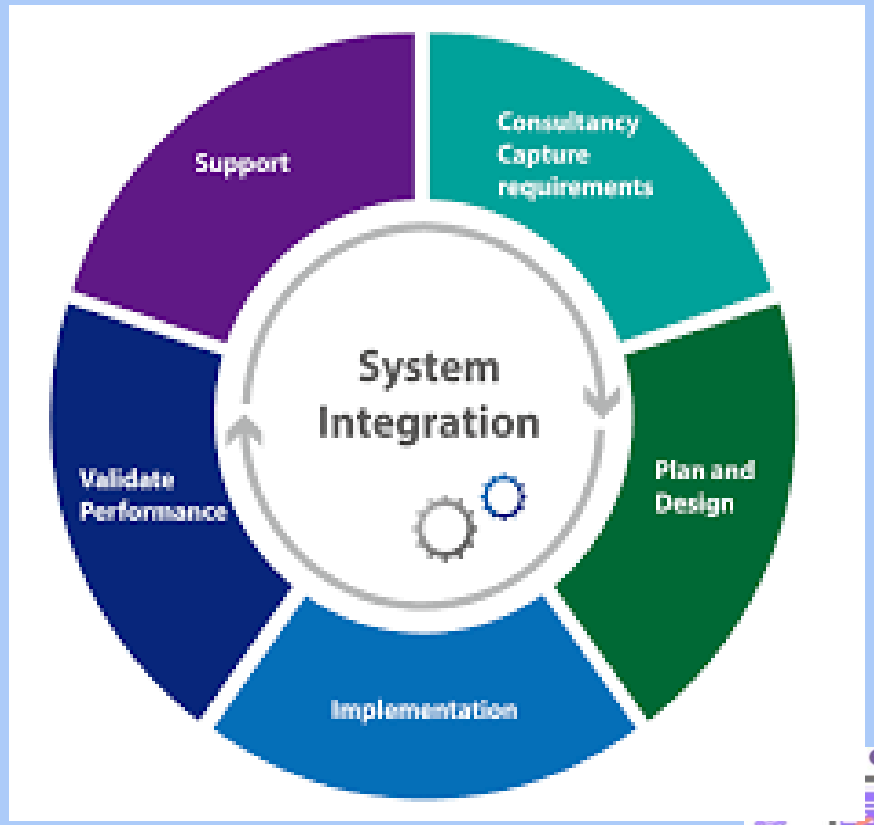
Properties of Systems

Integration

- Integration is concerned with how a system components are connected together. It means that the parts of the system work together within the system even if each part performs a unique function.

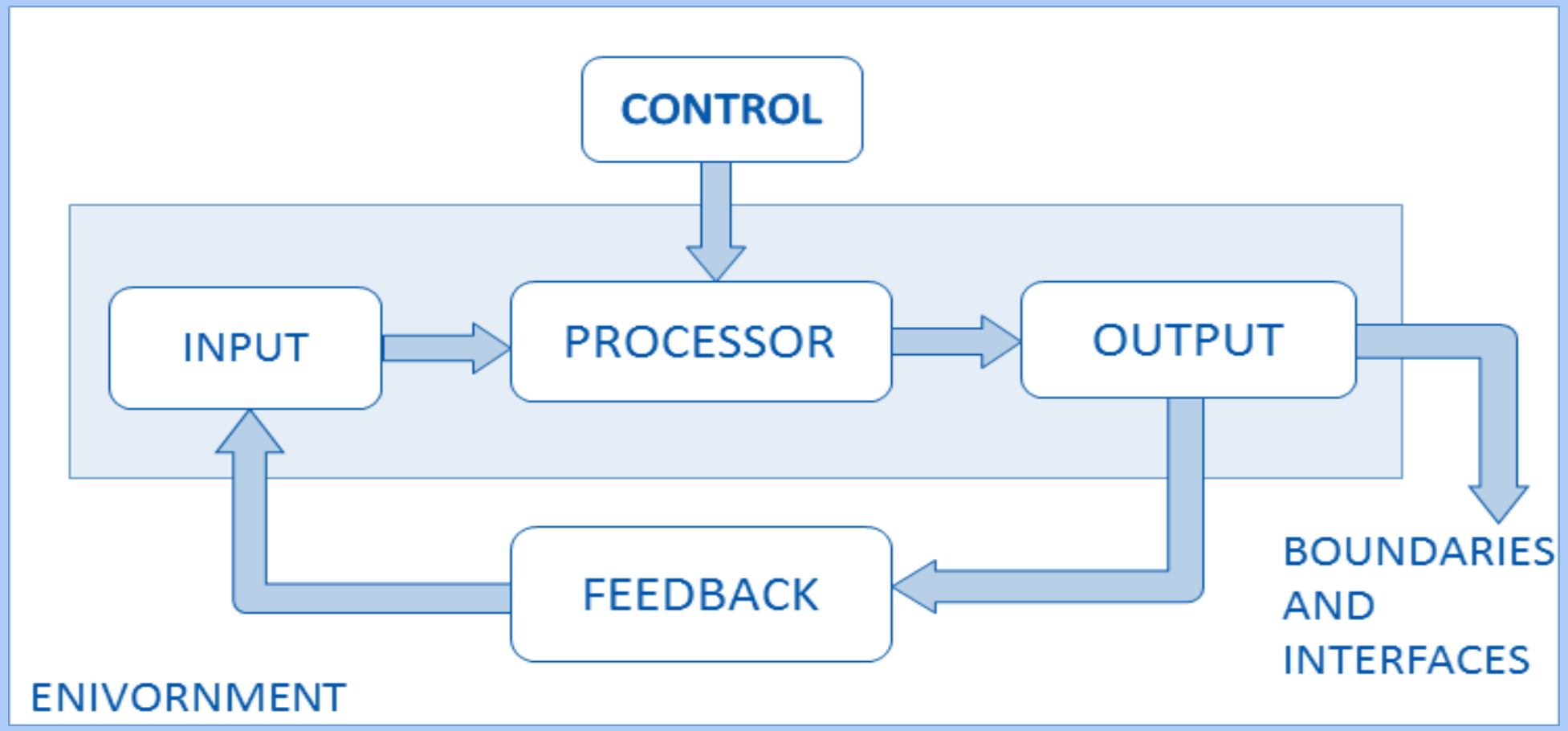
Central Objective

- The objective of system must be central. It may be real or stated. It is not uncommon for an organization to state an objective and operate to achieve another.
- The users must know the main objective of a computer application early in the analysis for a successful design and conversion.



Week 3&4
Element of System & Type of System, Categories of Information

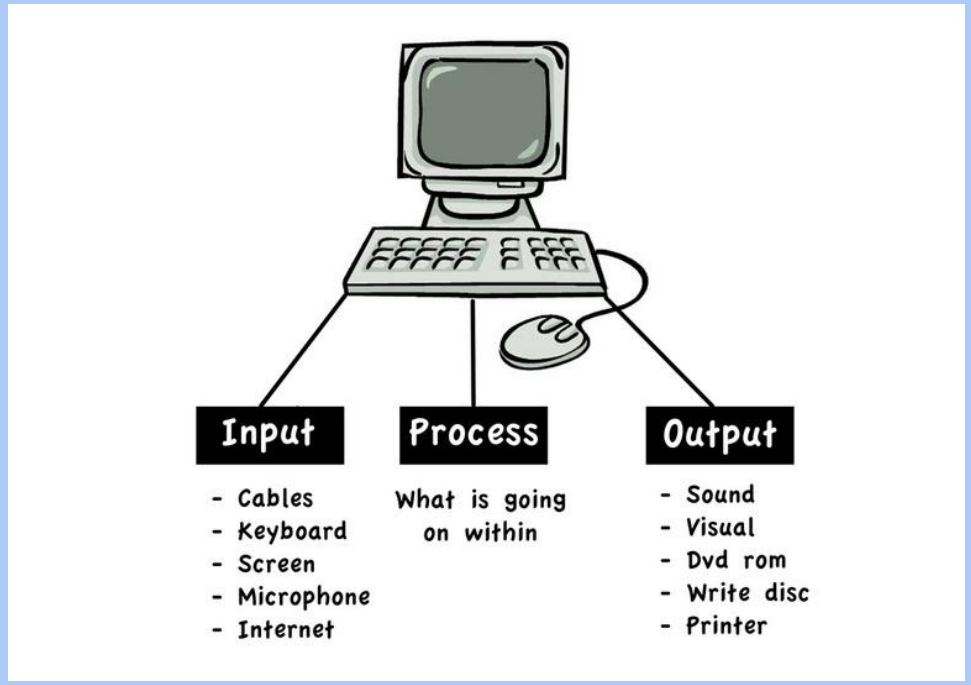
Elements of System



Elements of System

Outputs and Inputs

- The main aim of a system is to produce an output which is useful for its user.
- Inputs are the information that enters into the system for processing.
- Output is the outcome of processing.



Elements of System

Processor(s)

- The processor is the element of a system that involves the actual transformation of input into output.
- It is the operational component of a system. Processors may modify the input either totally or partially, depending on the output specification.
- As the output specifications change, so does the processing. In some cases, input is also modified to enable the processor for handling the transformation.

Elements of System

Control

- The control element guides the system.
- It is the decision–making subsystem that controls the pattern of activities governing input, processing, and output.
- The behaviour of a computer System is controlled by the Operating System and software. In order to keep system in balance, what and how much input is needed is determined by Output Specifications.

Feedback

- Feedback provides the control in a dynamic system.
- Positive feedback is routine in nature that encourages the performance of the system.
- Negative feedback is informational in nature that provides the controller with information for action.

Elements of System

Control

- The Environment
- The environment is the “super system” within which an organization operates.
- It is the source of external elements that strike on the system.
- It determines how a system must function. For example, vendors and competitors of organization’s environment, may provide constraints that affect the actual performance of the business.

Boundaries and Interface

- A system should be defined by its boundaries. Boundaries are the limits that identify its components, processes, and interrelationship when it interfaces with another system.
- Each system has boundaries that determine its sphere of influence and control.
- The knowledge of the boundaries of a given system is crucial in determining the nature of its interface with other systems for successful design.

Types of Systems

- Physical or Abstract Systems
- Open or Closed Systems
- Adaptive and Non Adaptive System
- Permanent or Temporary System
- Natural and Manufactured System
- Man–Made Information Systems

Types of Systems

Physical or Abstract Systems

- Physical systems are tangible entities. We can touch and feel them.
- Physical System may be static or dynamic in nature. For example, desks and chairs are the physical parts of computer centre which are static. A programmed computer is a dynamic system in which programs, data, and applications can change according to the user's needs.
- Abstract systems are non-physical entities or conceptual that may be formulas, representation or model of a real system.

Open or Closed Systems

- An open system must interact with its environment. It receives inputs from and delivers outputs to the outside of the system. For example, an information system which must adapt to the changing environmental conditions.
- A closed system does not interact with its environment. It is isolated from environmental influences. A completely closed system is rare in reality.

Types of Systems

Adaptive and Non Adaptive System

- Adaptive System responds to the change in the environment in a way to improve their performance and to survive. For example, human beings, animals.
- Non Adaptive System is the system which does not respond to the environment. For example, machines

Permanent or Temporary System

- Permanent System persists for long time. For example, business policies.
- Temporary System is made for specified time and after that they are demolished. For example, A DJ system is set up for a program and it is dissembled after the program.

Types of Systems

Natural and Manufactured System

- Natural systems are created by the nature. For example, Solar system, seasonal system.
- Manufactured System is the man-made system. For example, Rockets, dams, trains.

Man–Made Information Systems

- It is an interconnected set of information resources to manage data for particular organization, under Direct Management Control (DMC).
- This system includes hardware, software, communication, data, and application for producing information according to the need of an organization.

Categories of Information

Volume of Information	Type of Information	Information Level	Management Level	System Support
Low Consensed	Unstructured	Strategic Information	Upper	DSS
Medium Moderately Processed	Moderately Structured	Management Control Information	Middle	MIS
Large Detail Reports	Highly Structured	Operational Information	Lower	DPS

Categories of Information

Strategic Information

- This information is required by topmost management for long range planning policies for next few years. For example, trends in revenues, financial investment, and human resources, and population growth.
- This type of information is achieved with the aid of Decision Support System (DSS).

Managerial Information

- This type of Information is required by middle management for short and intermediate range planning which is in terms of months. For example, sales analysis, cash flow projection, and annual financial statements.
- It is achieved with the aid of Management Information Systems (MIS).

Operational information

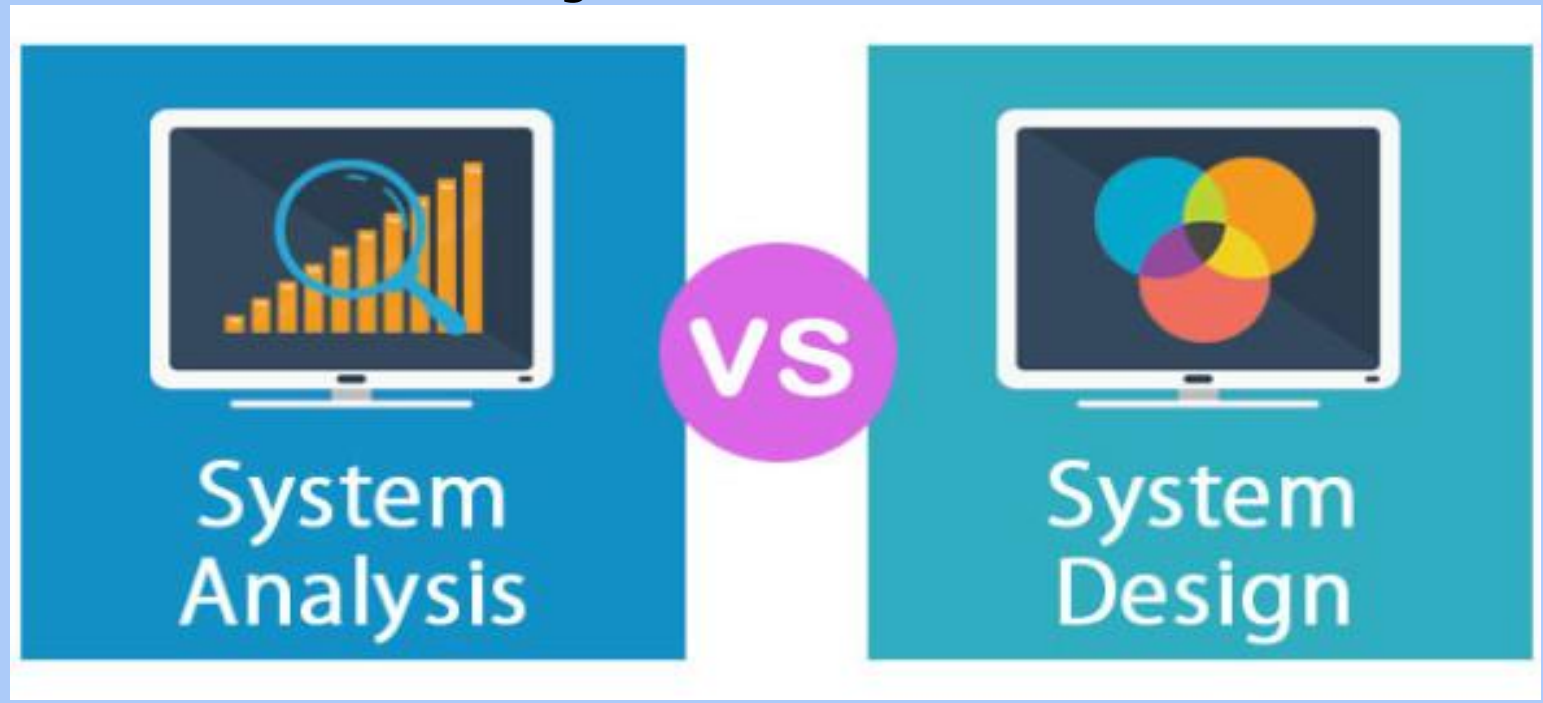
- This type of information is required by low management for daily and short term planning to enforce day-to-day operational activities.
- For example, keeping employee attendance records, overdue purchase orders, and current stocks available.
- It is achieved with the aid of Data Processing Systems (DPS).

Week 5,6

Differences between System Analysis and System Design, Role of System Analyst, Qualities of the System Analyst,

Differences between System Analysis and System Design

- System analysis and system design are two critical phases in the development lifecycle of a software system. While they are often used interchangeably, they serve distinct purposes and involve different methodologies.



Differences between System Analysis and System Design

- **System Analysis**
  - System analysis is the initial phase of a software development project where the requirements of the system are gathered, analyzed, and documented. It involves understanding the problem domain, identifying the stakeholders, and defining the scope and objectives of the system.

Key Activities in System Analysis

- **Requirement Gathering**– Identifying the needs and expectations of the users and stakeholders.
- **Requirement Analysis**– Analyzing the gathered requirements to ensure consistency, feasibility, and completeness.
- **Feasibility Study**– Assessing the technical, economic, and operational feasibility of the proposed system.
- **Process Modelling**– Creating diagrams and models to represent the current and proposed business processes.
- **Data Modelling**– Defining the data entities, attributes, and relationships within the system.

Role of System Analyst

- The analyst plays a key role in information system development projects.
- Must understand how to apply technology to solve business problems
- Analyst may serve as a change agents who identify the organizational improvement

Qualities of the System Analyst

- Problem solver
- Communicator
- Strong personal and professional ethics
- Self-disciplined and self -motivated

System Analyst Recommend, Design, and Maintain Many Types of Systems for Users

A system analyst may be involved with any or all of these systems at each organization level.

OPERATIONAL LEVEL

Transaction Processing System (TPS)

It is a process of large amounts of data for routine business transactions.

- **Boundary-Spanning**
  Its *concerned with the detection of information*. It has **two primary sources** and **two main sources**.

Primary sources of Information

(1) Detect information
(2) Send information into the environment presenting the company in a favorable light.

Main sources of Information

(I) ***Business intelligence.***
(2) ***Competitive information***

- **Support the day-to-day operations of the company**
  Example: Payroll Processing, Inventory Management.

KNOWLEDGE LEVEL

Office Automation System (OAS)

- Supports data workers who share information, but do not usually create new knowledge

Examples: Word processing, Spreadsheets etc.

Knowledge Work System (KWS)

- Supports professional workers such as scientists, engineers, and doctors

Examples: computer-aided design systems, virtual reality systems, investment workstations

Higher Level

- **Management Information System (MIS)**

To supports data worker who share information but do not usually create new knowledge.

Example: Word processing, Spreadsheets, Desktop publishing, Email Electronic scheduling, Communication through voice mail, Email, Video

- **Decision Support System (DSS)**

Aids decision makers in the making of decisions

Examples: financial planning with what-if analysis, budgeting with modeling

- **Expert System (ES)**

Captures and uses the knowledge of an expert for solving a particular problem which leads to a conclusion or recommendation.

Examples: **MYCIN** (an early ***xpert system*** that used **artificial intelligence** ;

XCON (e**X**pert **CON**figurer)

Strategic Level

- **Executive Support System (ESS)**
  - Helps executives to make unstructured strategic decisions in an informed way
    Examples: drill-down analysis, status access

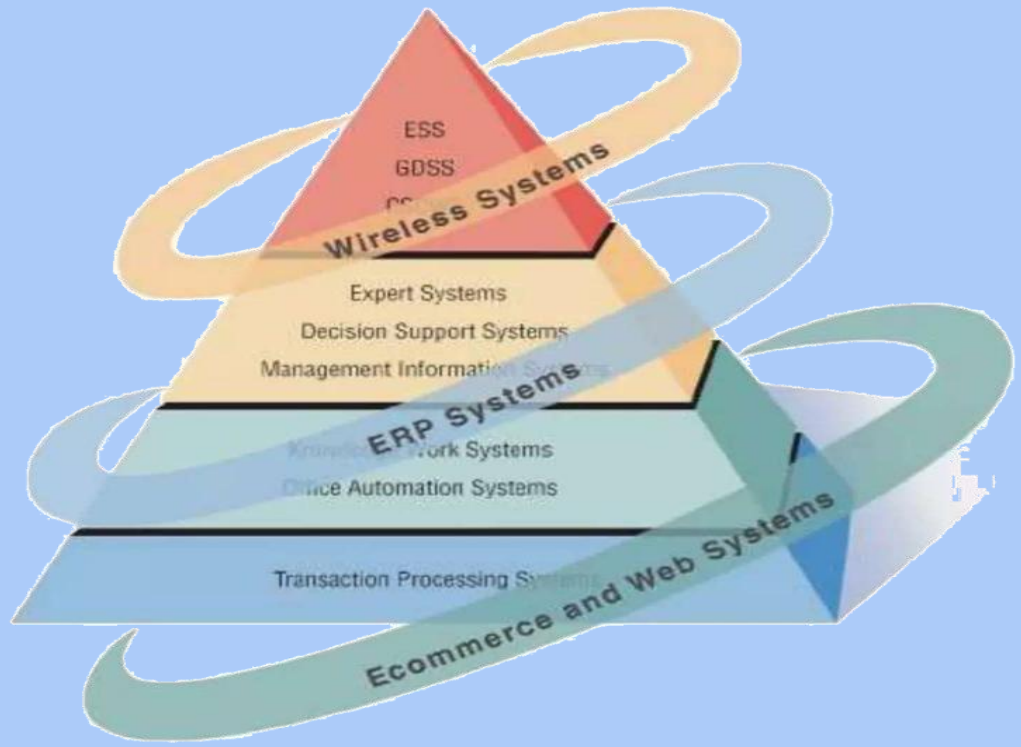
- **Group Decision Support System (GDSS)**
  - Permit group members to interact with electronic support
    Examples: email, Lotus Notes

- **Computer-Supported Collaborative Work System (CSCWS)**
  - CSCWS is a more general term of GDSS. It may include software support called "*groupware*" for team collaboration via network computers.
    Example: video conferencing, Web survey system

Integrating New Technologies into Traditional Systems

- Ecommerce and Web Systems.
- Enterprise Resource Planning Systems.
- Wireless Systems.
- Open Source Software.
- Need for Systems Analysis and Design.

Systems analysts need to be aware that integrating technologies affects all types of system



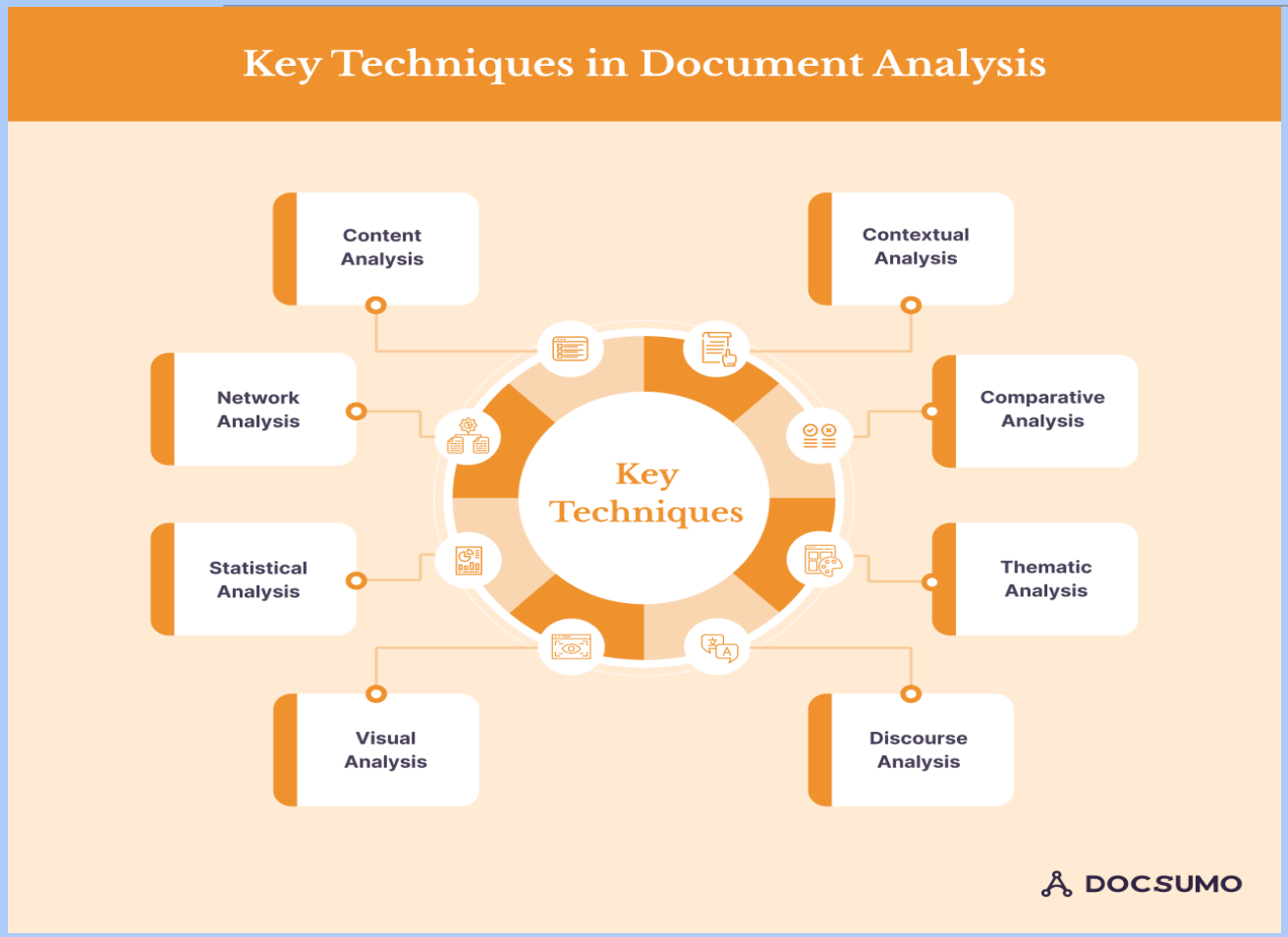
Week 7,8
Techniques Used in System Analysis, System Design Definition, Key Activities in System Design, Difference between system analysis and design

Techniques Used in System Analysis

- **Interviews**– Gathering information from stakeholders through face-to-face or online interviews.

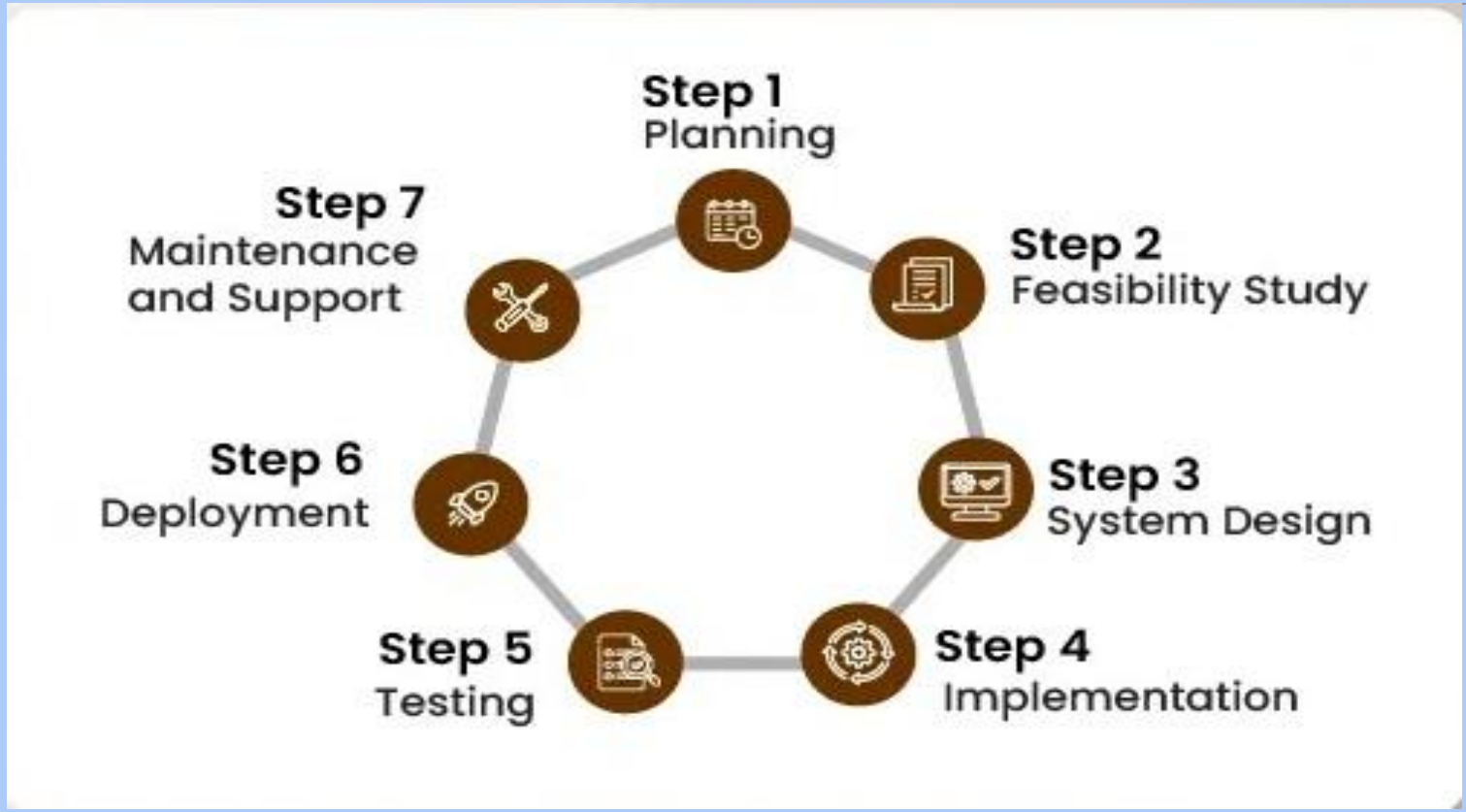
- **Surveys**– Collecting data from a large number of respondents using questionnaires.

Techniques Used in System Analysis



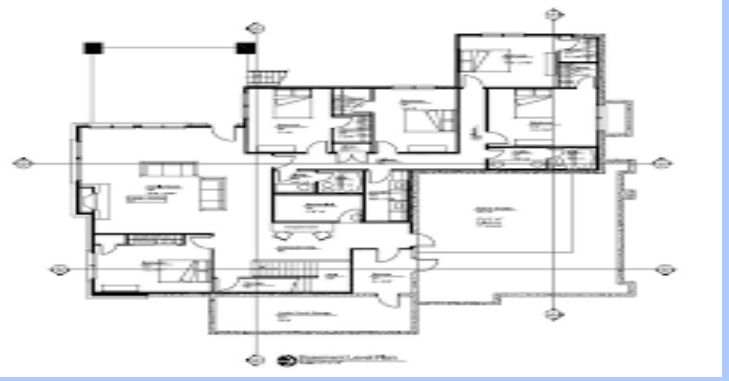
- **Observation**– Observing the current system in operation to understand its processes and workflows.
- **Document Analysis**– Examining existing documents, reports, and manuals.
- **Prototyping**– Creating simplified models or mock-up's of the system to gather feedback and refine requirements.

System Design



System design is the subsequent phase where the detailed specifications of the system are developed. It involves designing the architecture, components, interfaces, and data structures that will implement the requirements defined in the analysis phase.

Key Activities in System Design



- **Architectural Design**– Determining the overall structure and components of the system.
- **Component Design**– Designing individual components and their interactions.
- **Interface Design**– Specifying the interfaces between components and with external systems.
- **Data Design**– Designing the database schema and data structures.
- **Detailed Design**– Creating detailed specifications for each component, including algorithms and data flow.

Techniques Used in System Design

- **Unified Modelling Language (UML)**– A standardized modelling language used to visualize, specify, construct, and document software systems.
- **Data Flow Diagrams (DFDs)**– Diagrams that illustrate the flow of data through a system.
- **Entity-Relationship Diagrams (ERDs)**– Diagrams that represent the entities and relationships between them in a database.
- **Decision Trees**– Diagrams that show the possible outcomes and decisions in a process.
- **State Transition Diagrams**– Diagrams that represent the different states a system can be in and the transitions between them.

Key Differences Between System Analysis and System Design

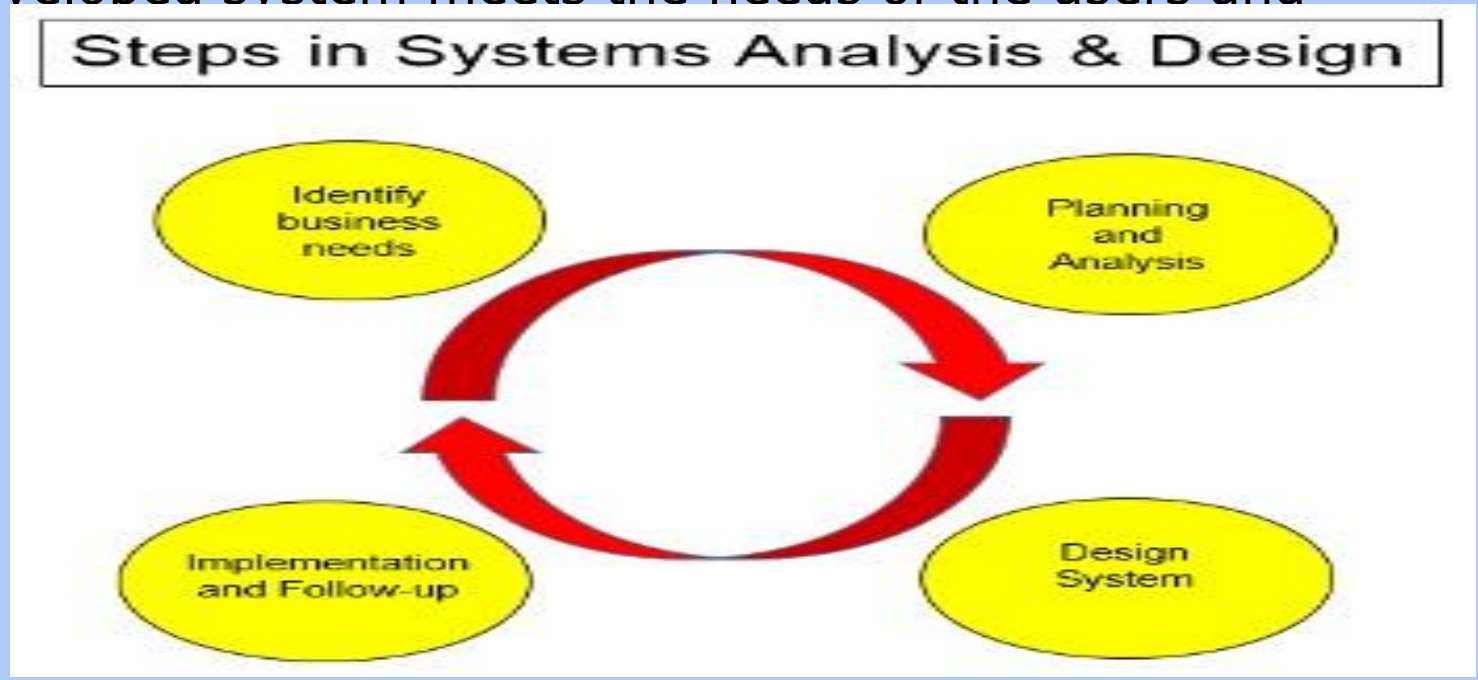
Sr.No.	Feature	System Analysis	System Design
1	Focus	Understanding the problem domain and gathering requirements.	Specifying the solution and designing the system.
2	Output	Requirements document	System design specifications
3	Techniques	Interviews, surveys, observation, document analysis	UML, DFDs, ERDs, decision trees, state transition diagrams
4	Level of Detail	High-level understanding	Detailed specifications

Week 7,8

The Relationship Between System Analysis and System Design, Horizontal and Vertical Scaling in System Design, Benefits & Limitations of Horizontal, Vertical Scaling

The Relationship Between System Analysis and System Design

System analysis and system design are closely interconnected. The output of the analysis phase (the requirements document) serves as the input for the design phase. The design specifications must align with the requirements to ensure that the developed system meets the needs of the users and stakeholders.



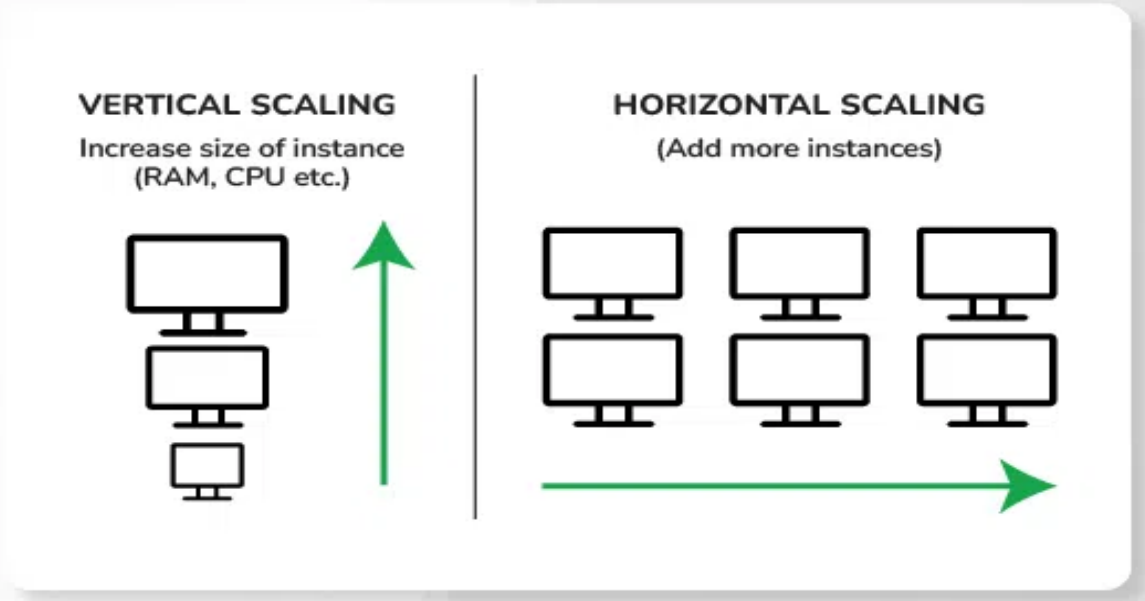
Horizontal and Vertical Scaling in System Design

- What is Scaling?
  - Before diving into the specifics of horizontal and vertical scaling, it is essential to understand what scaling entails.

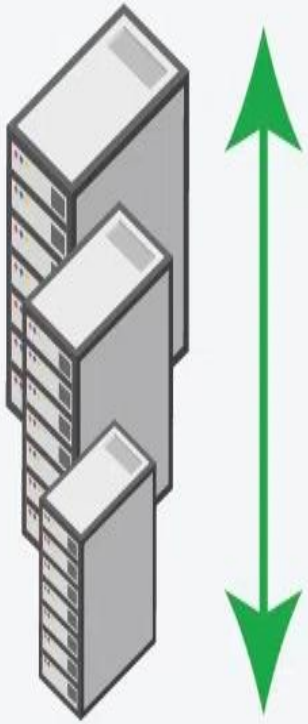
Scaling refers to the process of adjusting resources—such as computing power, storage, or network capabilities—to ensure that an application can handle increased demand without sacrificing performance. As systems grow and face more users, more transactions, or increased data throughput, scaling ensures that the system maintains its efficiency and does not become a bottleneck.

Horizontal and Vertical Scaling

System Design



Vertical Scaling

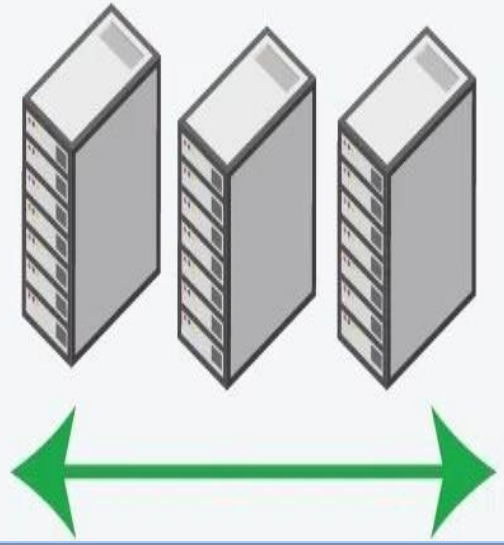


Increase or decrease the capacity of existing services/instances

- No changes have to be made to the application code
- Less complex network
- Less complicated maintenance

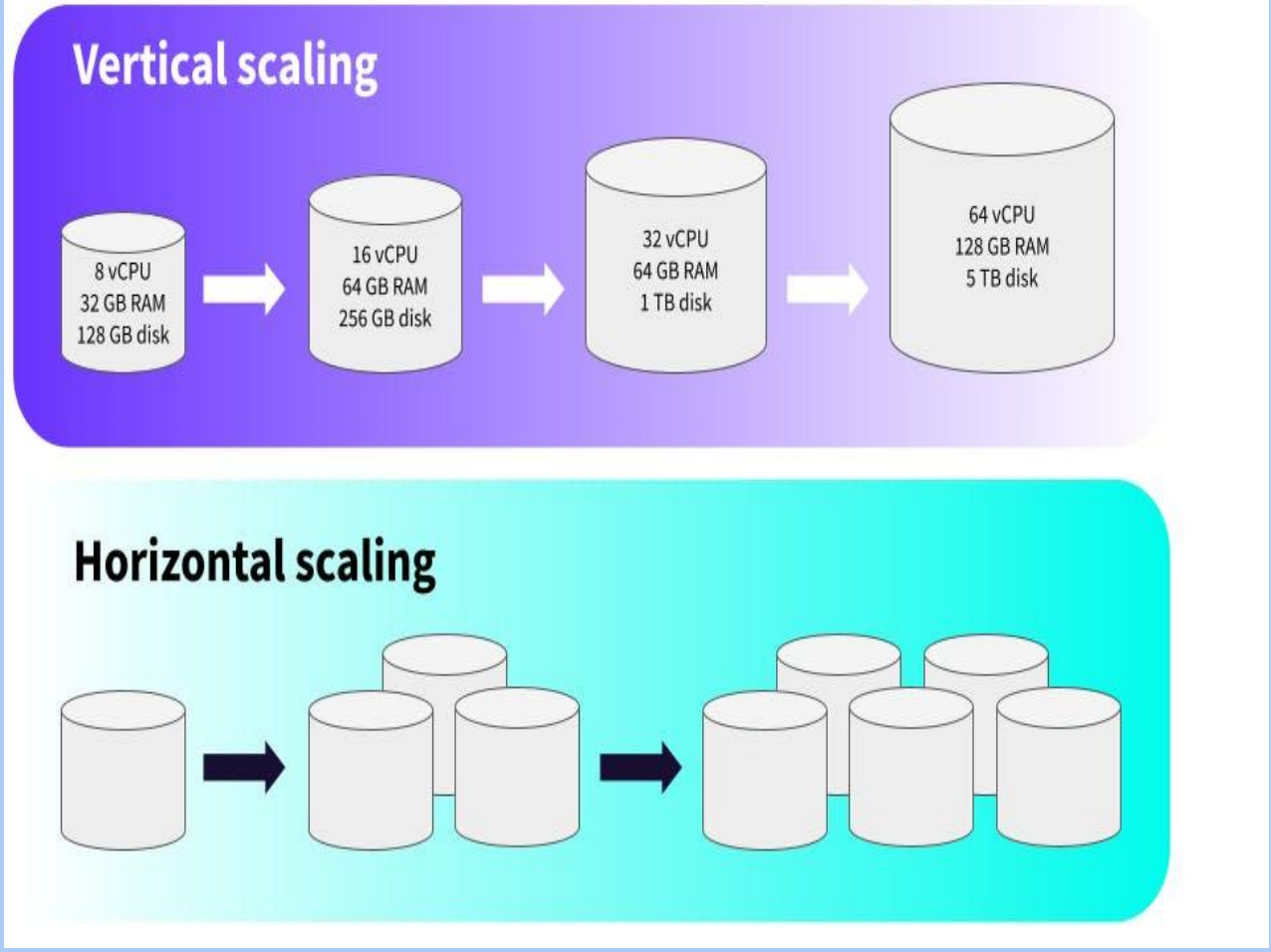
Horizontal Scaling

Add more resources like virtual machines to your system to spread out the workload across them.



- Increases high availability
- Fewer periods of downtime
- Easy to resize according to your needs

Example...



- *Imagine you have a computer at home that you use for various tasks such as web browsing, word processing, and light gaming. Over time, you find that your computer struggles to keep up with more resource-intensive tasks like video editing or running advanced software.*

- *Imagine you own a small delivery service company that initially operates with a single delivery van. As your business grows, you start receiving more orders and delivering to a larger area. However, you quickly realize that the single van is not sufficient to handle the increasing demand efficiently.*

Benefits & Limitations of Vertical Scaling

Benefits	Limitations
Simplicity– Vertical scaling is generally straightforward	**Single Point of Failure**– If the server crashes or faces hardware issues, the whole system may go down.
Reduced Complexity– Managing a single server reduces the complexity of operations, including maintenance and monitoring.	**Resource Limits**– Eventually, a physical server can only be upgraded so much. There is a limit to how much CPU, RAM, or storage can be added to a single machine.
Consistency– Since the system runs on a single machine, data consistency is easier to maintain.	**Cost**– High-end servers and components are expensive.
Ideal for Monolithic Applications– Monolithic applications, which are tightly coupled and difficult to break down into smaller components,	**Downtime During Upgrades**– Depending on the system, upgrading hardware (e.g., adding RAM or storage) might require shutting down the server, which leads to downtime

Benefits & Limitations of Horizontal Scaling

Benefits	Limitations
No Theoretical Limit– As demand grows, you can continue adding machines, thus providing potentially infinite scalability.	**Increased Complexity**– Managing multiple servers is inherently more complex than managing a single server.
Fault Tolerance– Since multiple machines are involved, if one node fails, the system can continue to operate using the remaining nodes.	**Data Consistency Issues**– In distributed systems, maintaining data consistency across multiple nodes can be challenging
Cost Efficiency at Scale– Instead of investing in one high-end machine, horizontal scaling allows the use of many lower-end machines.	**Network Overhead**– With more servers, communication between nodes increases, potentially leading to network latency and overhead.
Better for Cloud and Distributed Applications– Horizontal scaling is ideal for cloud-native and distributed systems like microservices architectures, where different parts of the application can run independently on different servers.	**Scaling the Entire Stack**– For certain workloads, scaling horizontally might require that the entire application stack be designed for horizontal distribution, which may require significant refactoring.

When to Choose Horizontal or Vertical Scaling

- **Application Architecture**– Distributed systems or microservices naturally lend themselves to horizontal scaling, whereas monolithic applications are easier to scale vertically.
- **Budget**– Horizontal scaling may be more cost-effective in the cloud, where additional instances can be spun up as needed. Vertical scaling may result in high costs due to expensive hardware.
- **Consistency Requirements**– Applications requiring strict data consistency (like banking systems) may favour vertical scaling due to simpler data management.
- **Expected Growth**– If your application is expected to grow rapidly, horizontal scaling may be more appropriate since it offers theoretically unlimited scaling potential.

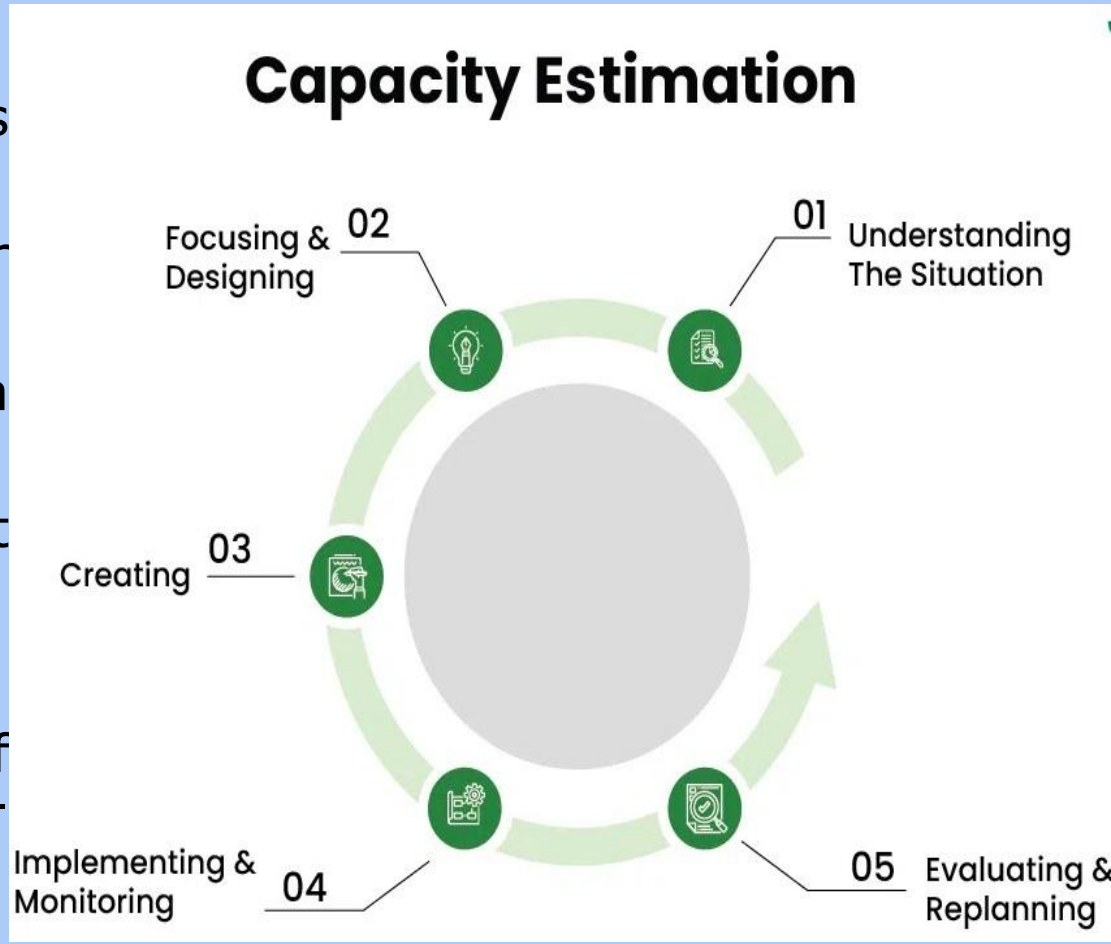
Week 9

Understanding Capacity Estimation, Steps in Capacity Estimation, Clustering and Load Balancing

Understanding Capacity Estimation

Key Metrics

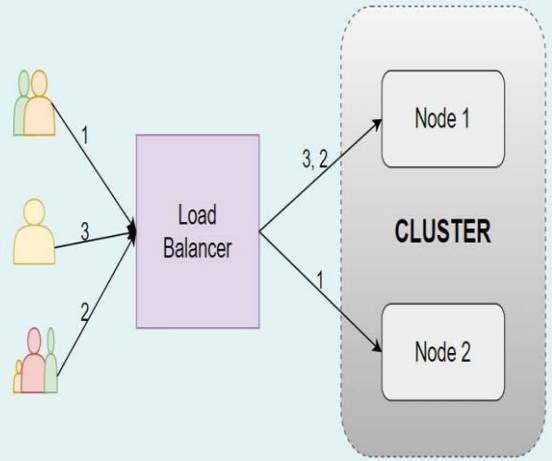
- **Throughput**– Transactions per second or requests per second.
- **Latency**– Time to complete a transaction or request.
- **Response Time**– The total time a user waits for a response.
- **Load and Concurrency**– The number of concurrent users or operations.
- **Utilization**– Percentage of capacity used.
- **Business Impact**– Outline the cost implications of over-provisioning and the risk of under-provisioning.



Steps in Capacity Estimation

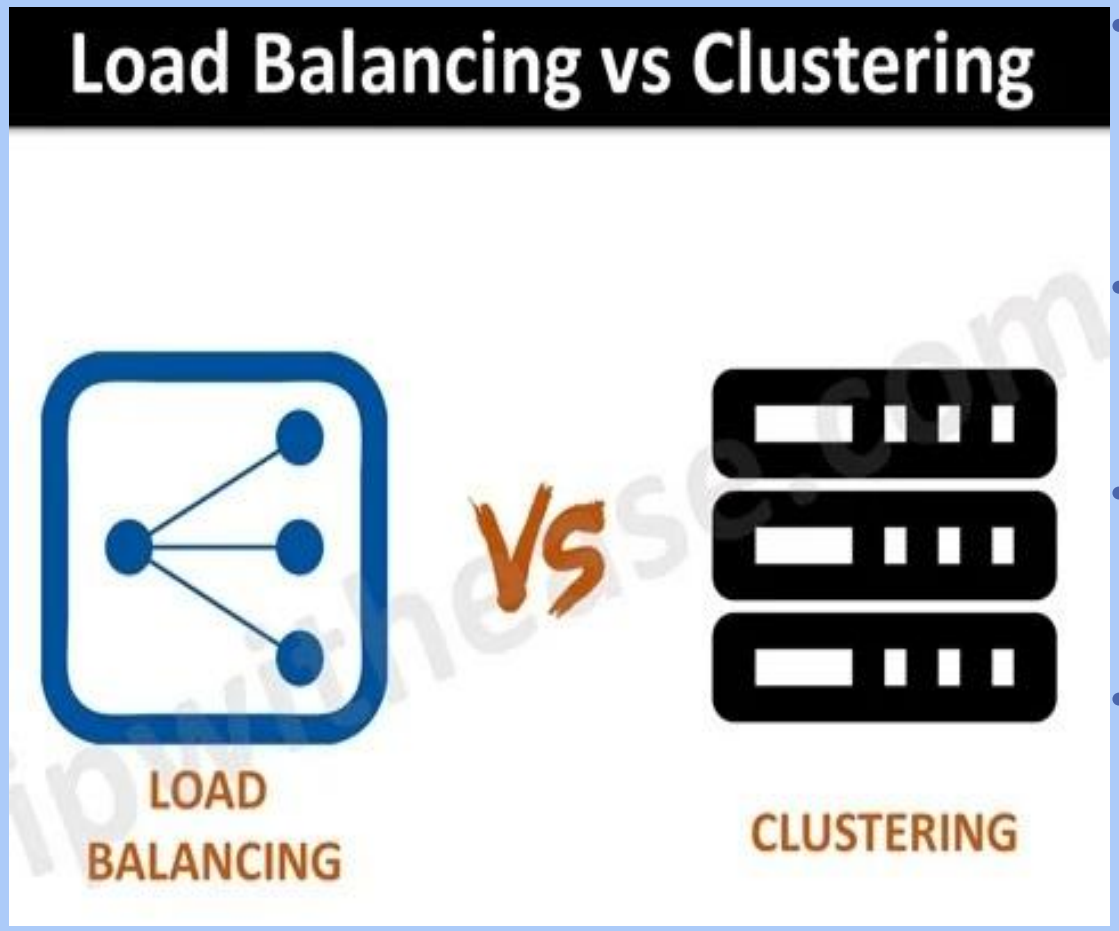
- **Define Requirements**– Identify the expected workload, peak traffic, and availability needs.
- **Analyze Historical Data**– Use historical system data to find patterns and identify trends.
- **Model the System**–
  - **Workload Modelling**– Characterize the types and intensity of workloads (e.g., read-heavy vs. write-heavy operations).
  - **Resource Consumption Modelling**– Quantify resource usage for each workload (CPU, memory, disk I/O).
  - **Concurrency and Scaling Factors**– Include factors for concurrency and examine how each resource is affected.
- **Conduct Load Testing**– Perform stress and load tests to validate models and identify bottlenecks.
- **Provision Resources**– Calculate the required resources for the projected capacity with a margin for peak usage.
- **Estimate Growth**– Forecast workload growth based on business expectations.

Clustering and Load Balancing



- Clustering and load balancing are essential for modern applications to ensure they are scalable, highly available, and perform well under varying loads. Here's why they are significant.
  - **High Availability**– Clustering ensures that if one server goes down, others can take over, minimizing downtime and ensuring continuous availability.
  - **Scalability**– By adding more nodes to a cluster, applications can handle more users and more data without performance degradation.
  - **Fault Tolerance**– Clusters are designed to continue operating even when individual nodes fail, which enhances the resilience of the application.
  - **Resource Management**– Distributes workloads across multiple nodes, optimizing resource usage and preventing any single node from becoming a bottleneck.

Load Balancing



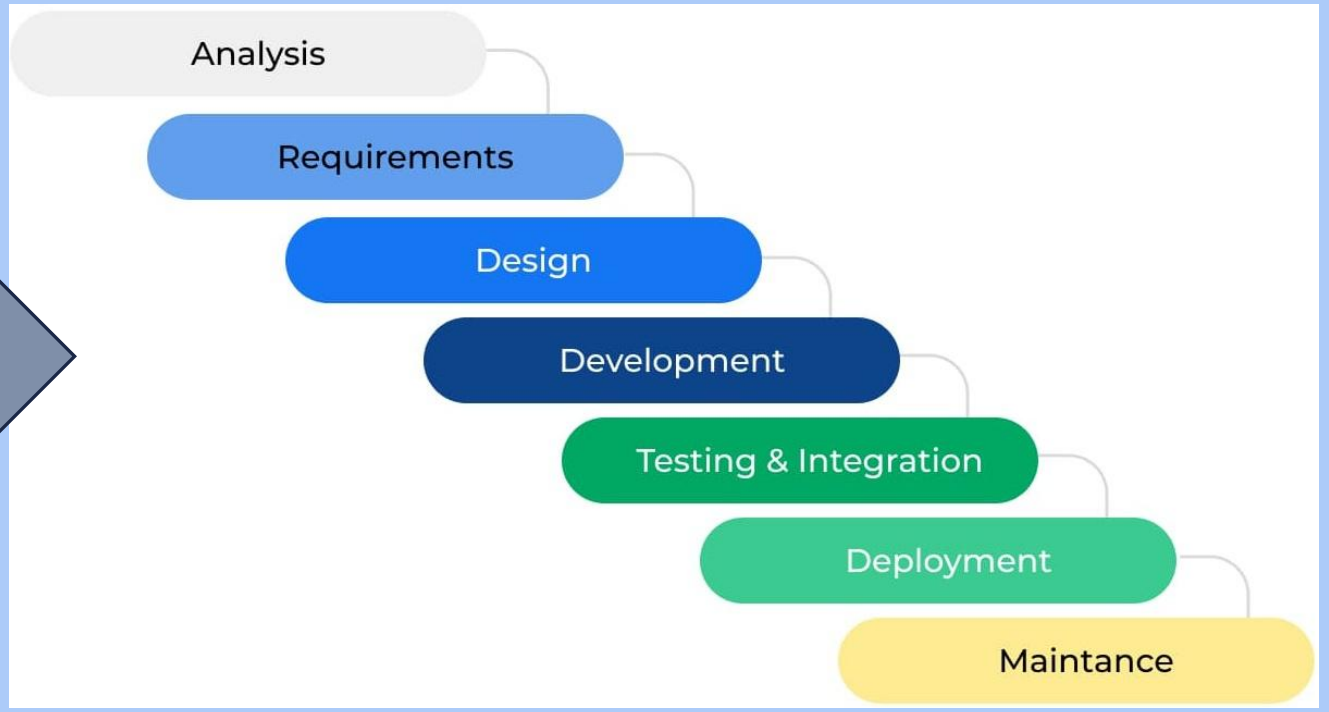
- **Efficient Resource Utilization–** Load balancing distributes incoming traffic across multiple servers, ensuring that no single server is overwhelmed, which optimizes resource utilization.
- **Improved Performance–** By balancing the load, applications can respond faster to user requests, enhancing the overall user experience.
- **Redundancy–** Load balancing ensures that if one server fails, traffic can be redirected to other operational servers, providing redundancy.
- **Scalability–** Easily scales by adding more servers to the pool, allowing applications to handle increasing traffic seamlessly.

Week 10
System Development Life Cycle

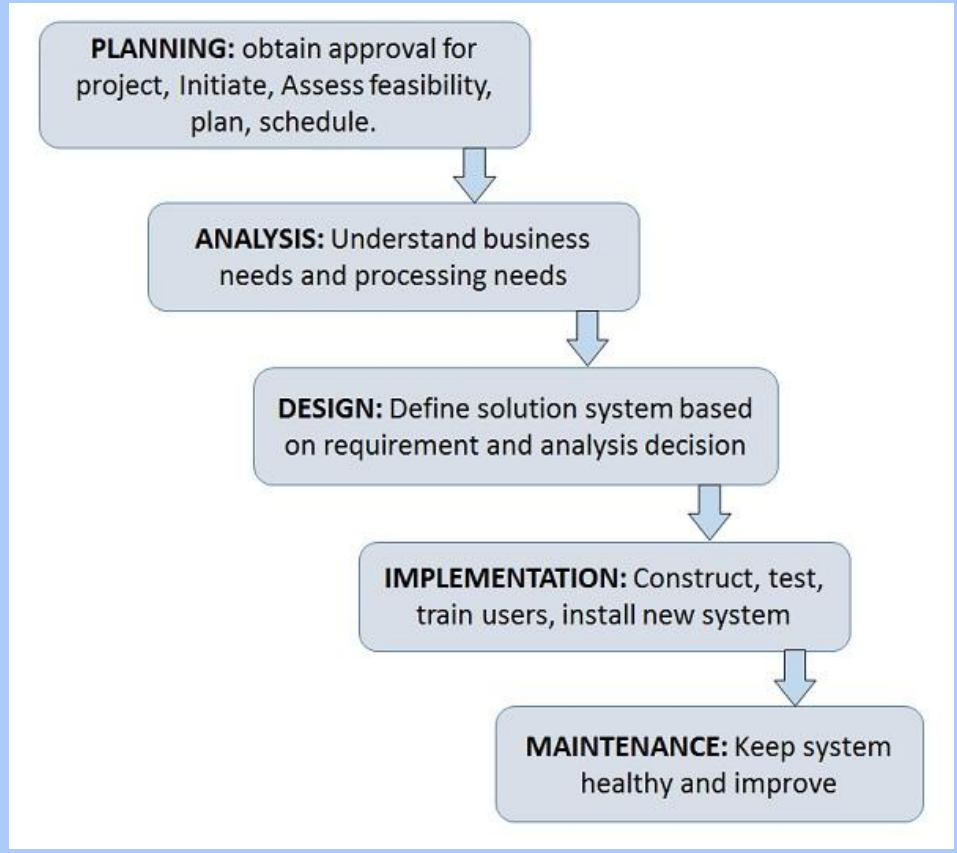
System Development Life Cycle

- System Development Life Cycle (SDLC) is a conceptual model which includes policies and procedures for developing or altering systems throughout their life cycles.

SDLC Activities



Phases of SDLC



Feasibility Study or Planning

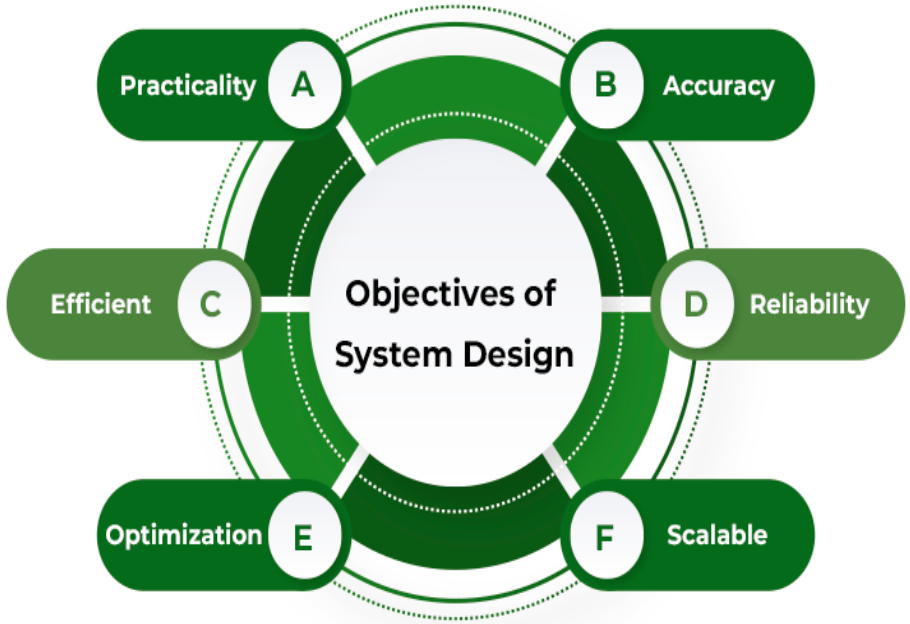
- Define the problem and scope of existing system.
- Overview the new system and determine its objectives.
- Confirm project feasibility and produce the project Schedule.
- During this phase, threats, constraints, integration and security of system are also considered.
- A feasibility report for the entire project is created at the end of this phase.

Analysis and Specification

- Gather, analyze, and validate the information.
- Define the requirements and prototypes for new system.
- Evaluate the alternatives and prioritize the requirements.
- Examine the information needs of end-user and enhances the system goal.
- A Software Requirement Specification (SRS) document, which specifies the software, hardware, functional, and network requirements of the system is prepared at the end of this phase.

System Design

- Includes the design of application, network, databases, user interfaces, and system interfaces.
- Transform the SRS document into logical structure, which contains detailed and complete set of specifications that can be implemented in a programming language.
- Create a contingency, training, maintenance, and operation plan.
- Review the proposed design. Ensure that the final design must meet the requirements stated in SRS document.
- Finally, prepare a design document which will be used during next phases.



Implementation

- Implement the design into source code through coding.
- Combine all the modules together into training environment that detects errors and defects.
- A test report which contains errors is prepared through test plan that includes test related tasks such as test case generation, testing criteria, and resource allocation for testing.
- Integrate the information system into its environment and install the new system.

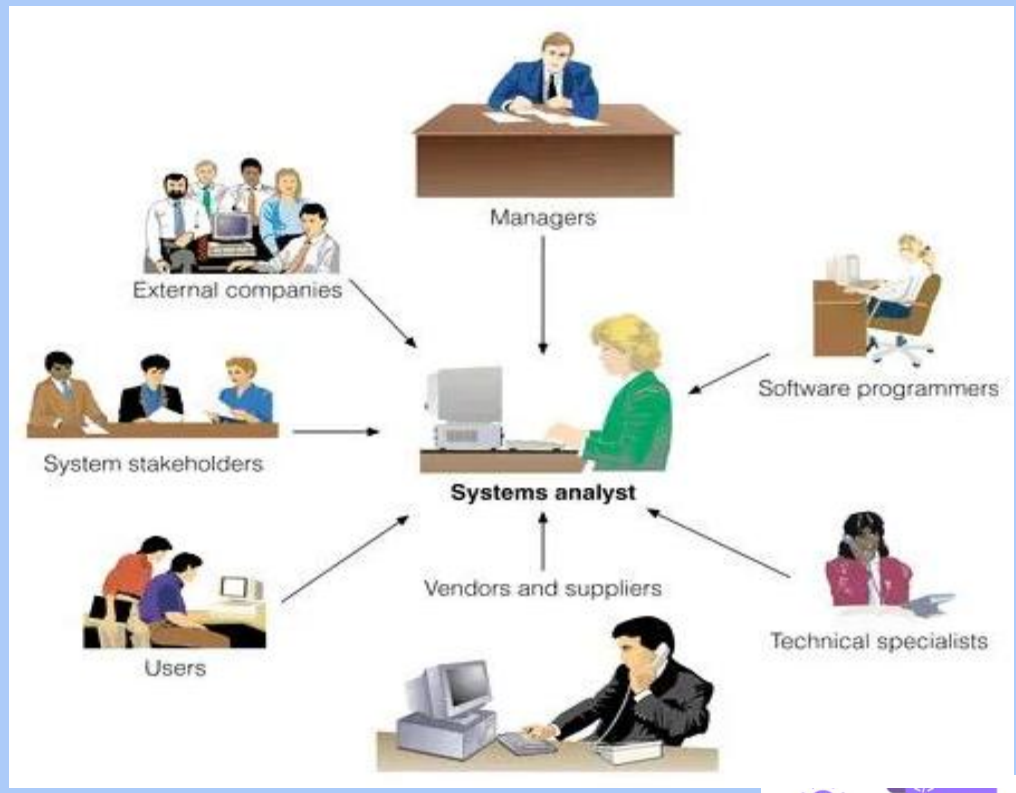
Maintenance/Support

- Include all the activities such as phone support or physical on-site support for users that is required once the system is installing.
- Implement the changes that software might undergo over a period of time, or implement any new requirements after the software is deployed at the customer location.
- It also includes handling the residual errors and resolve any issues that may exist in the system even after the testing phase.
- Maintenance and support may be needed for a longer time for large systems and for a short time for smaller systems.

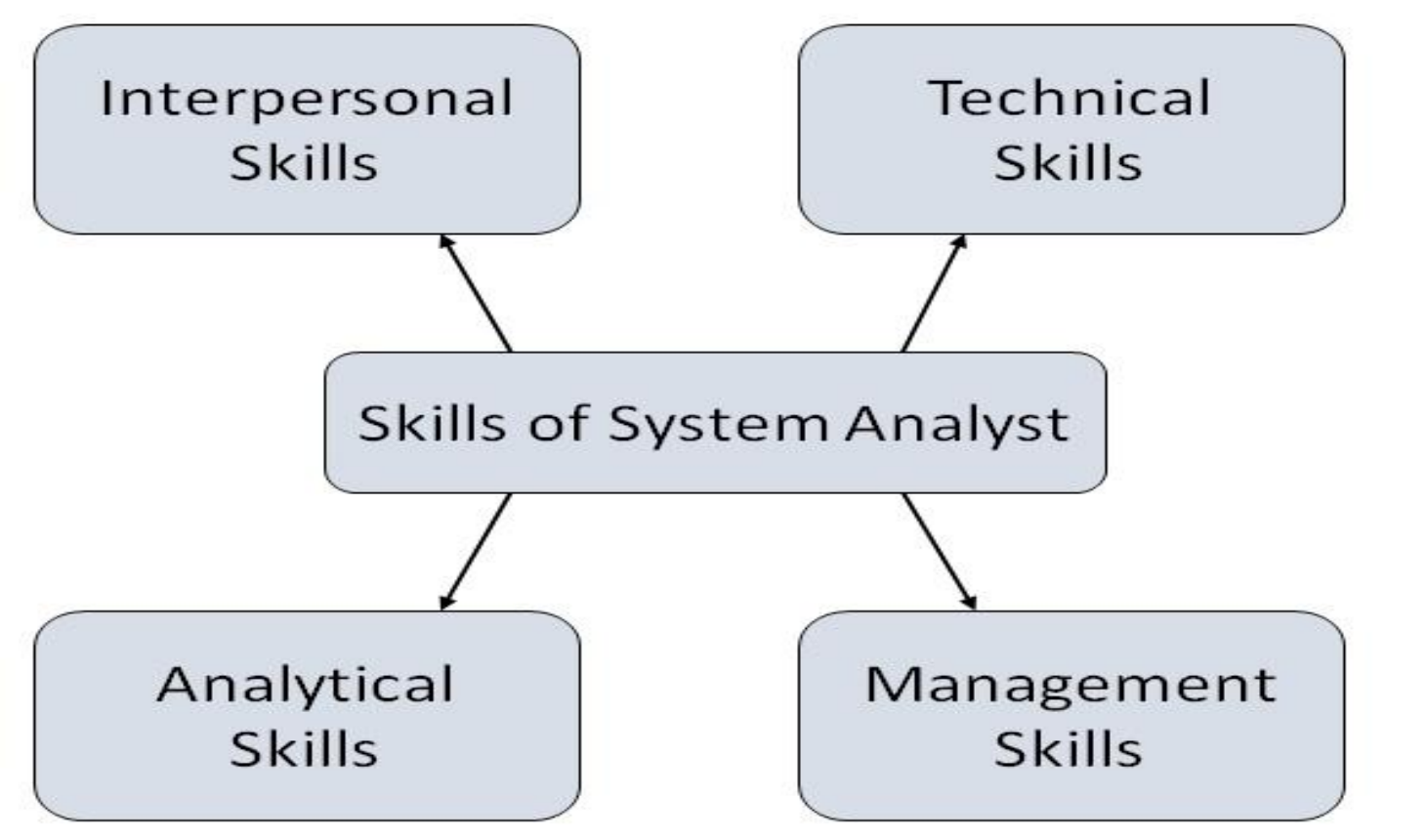
Week 11
Role of System Analyst, Attributes of a Systems Analyst, Requirement Determination

Role of System Analyst

- Defining and understanding the requirement of user through various Fact finding techniques.
- Prioritizing the requirements by obtaining user consensus.
- Gathering the facts or information and acquires the opinions of users.
- Maintains analysis and evaluation to arrive at appropriate system which is more user friendly.
- Suggests many flexible alternative solutions, pick the best solution, and quantify cost and benefits.
- Draw certain specifications which are easily understood by users and programmer in precise and detailed form.
- Implemented the logical design of system which must be modular.
- Plan the periodicity for evaluation after it has been used for some time, and modify the system as needed.



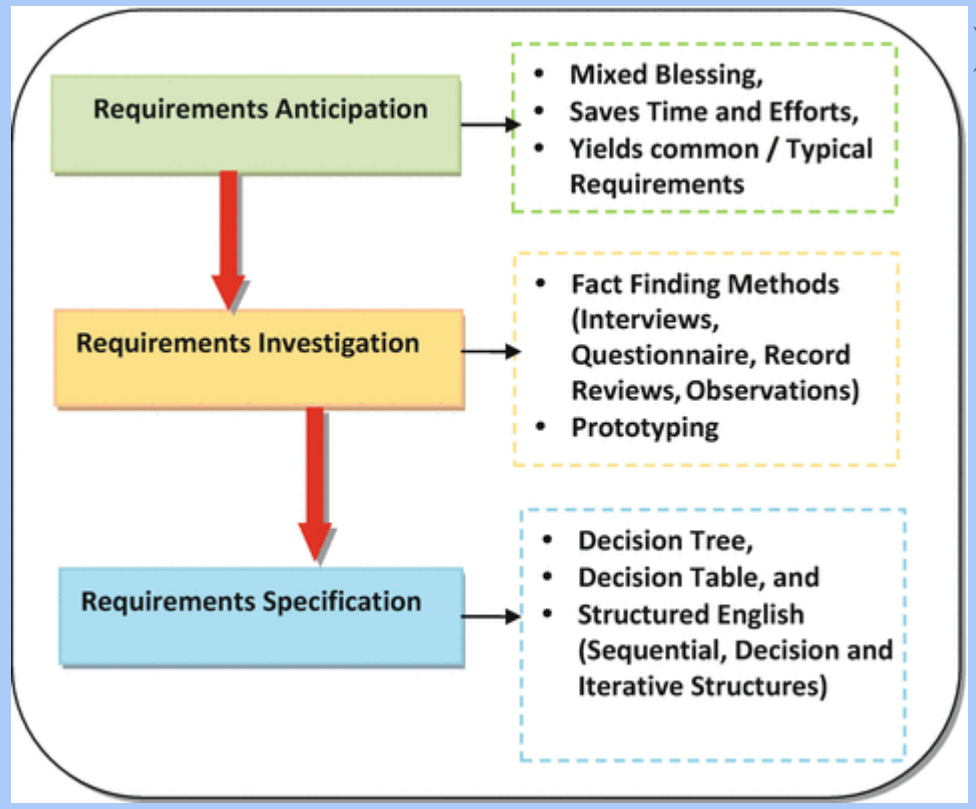
Attributes of a Systems Analyst



Assignment… !

Write short Paragraph of Every skill on the Skills of System Analyst of a specific Given System

Requirement Determination



- In the realm of systems analysis and design, requirement determination is a critical phase that sets the foundation for successful software development. It involves gathering, analyzing, and documenting the needs and expectations of stakeholders to ensure that the final system meets its intended purpose.

Importance of Requirement Determination

- **Clarity of Purpose**– Clearly defined requirements help stakeholders understand the system's purpose and functionality, reducing ambiguity.
- **Stakeholder Satisfaction**– Engaging stakeholders early and accurately capturing their needs leads to greater satisfaction with the final product.
- **Cost and Time Efficiency**– Well-documented requirements minimize the risk of costly changes during later development stages, leading to a more efficient project lifecycle.
- **Risk Management**– Identifying potential issues early allows teams to devise strategies to mitigate risks before they escalate.
- **Framework for Development**– Requirements serve as a guide for system design, coding, testing, and implementation, ensuring alignment throughout the development process.

Home Task.. !

- **Analyze the Methodologies for Requirement Determination of A Specific System**

Week 12
Systems Implementation, Structured Analysis

Systems Implementation

Systems implementation is the process of:

1. defining how the information system should be built (i.e., physical system design),
2. ensuring that the information system is operational and used,
3. ensuring that the information system meets quality standard (i.e., quality assurance).

Choosing the Right Implementation Approach

➢Big Bang Approach

- Replacing old systems with new ones at once.
- Pros and cons of immediate transition.

➢Phased Implementation

- Gradual deployment in stages.
- Benefits of controlling scope and user adaptation.

➢Pilot Implementation

- Deploying the system in a limited area to assess performance.
- Benefits in risk reduction before full-scale roll-out.

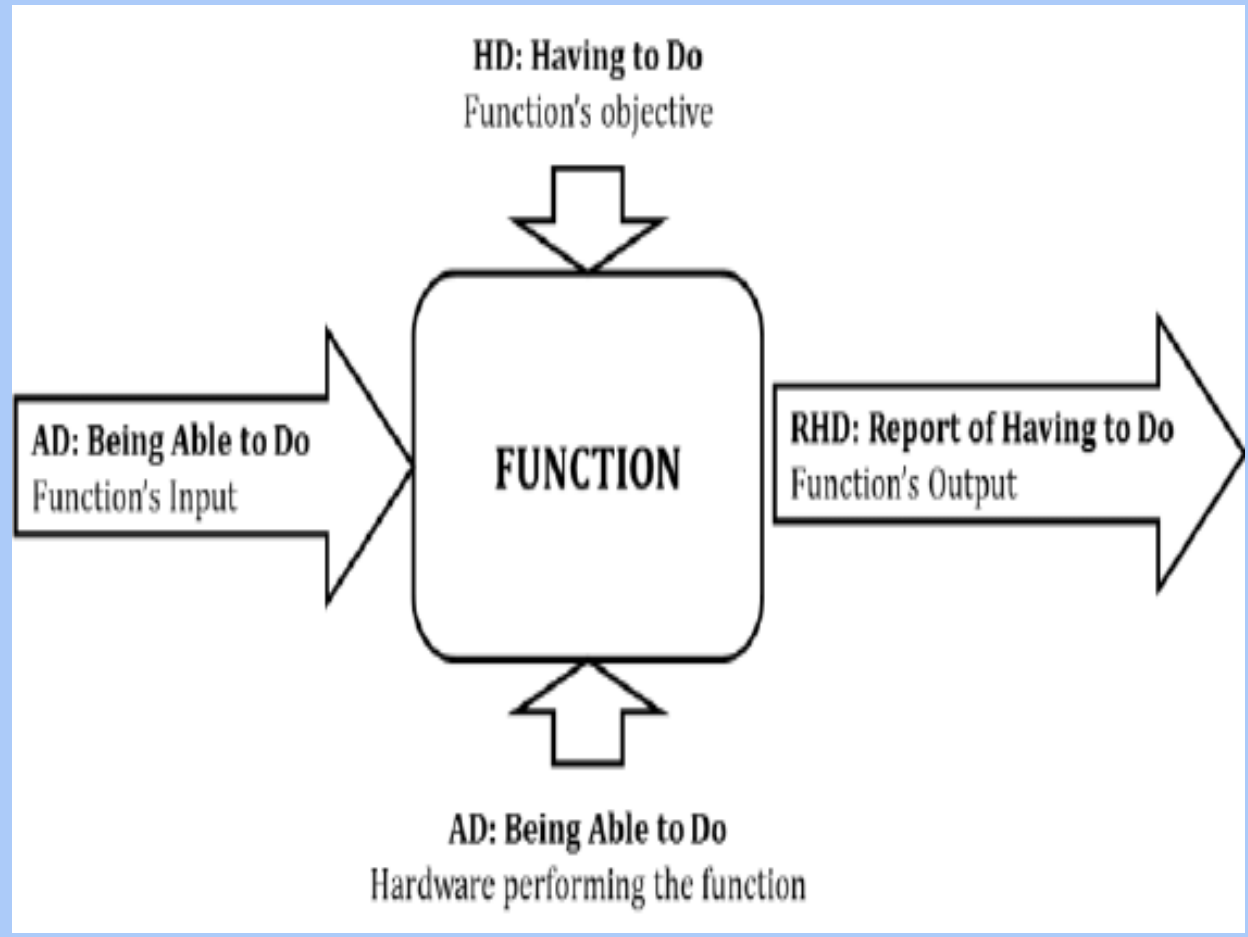
➢Parallel Implementation

- Running old and new systems concurrently.
- Advantages for validation and testing.

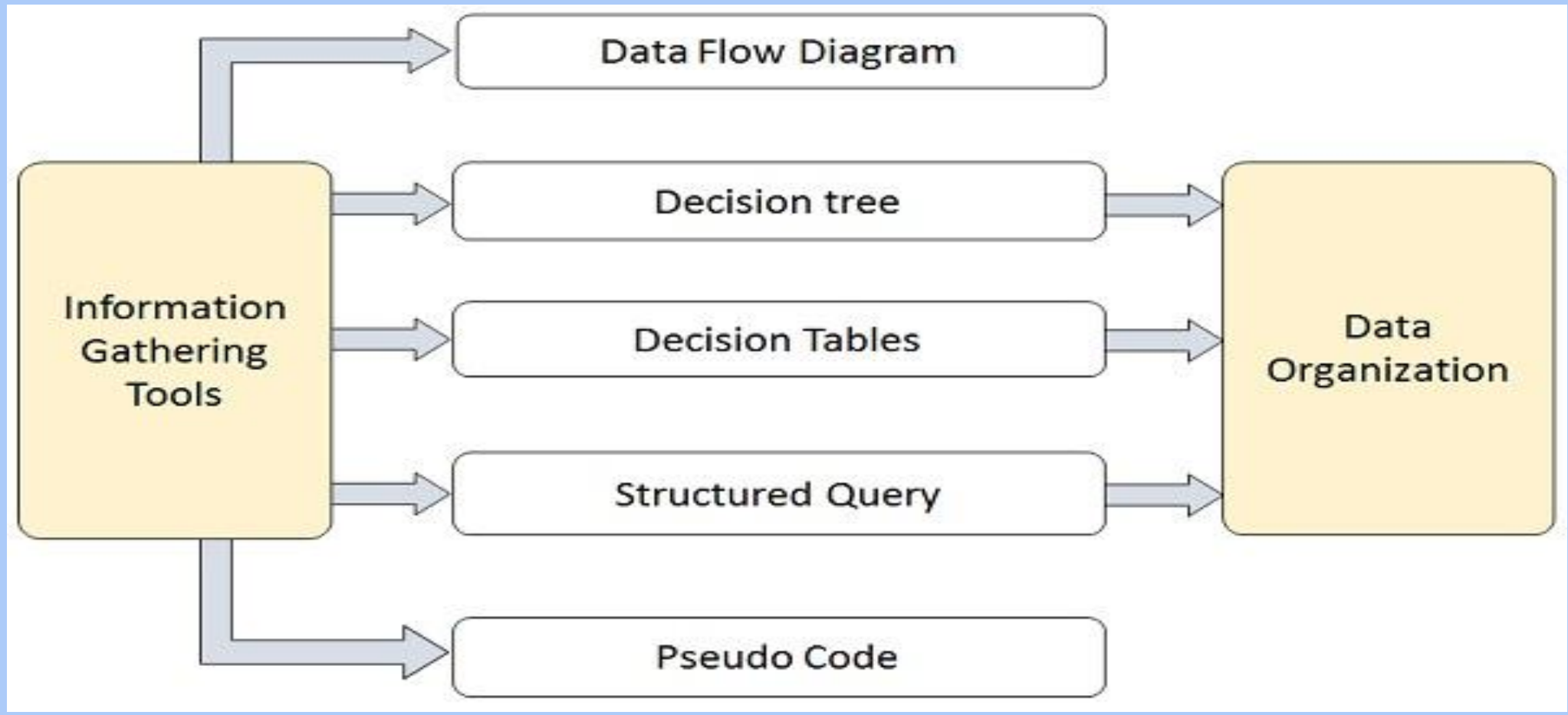
Structured Analysis

Structured Analysis is a development method that allows the analyst to understand the system and its activities in a logical way.

It is a systematic approach, which uses graphical tools that analyze and refine the objectives of an existing system and develop a new system specification which can be easily understandable by user.

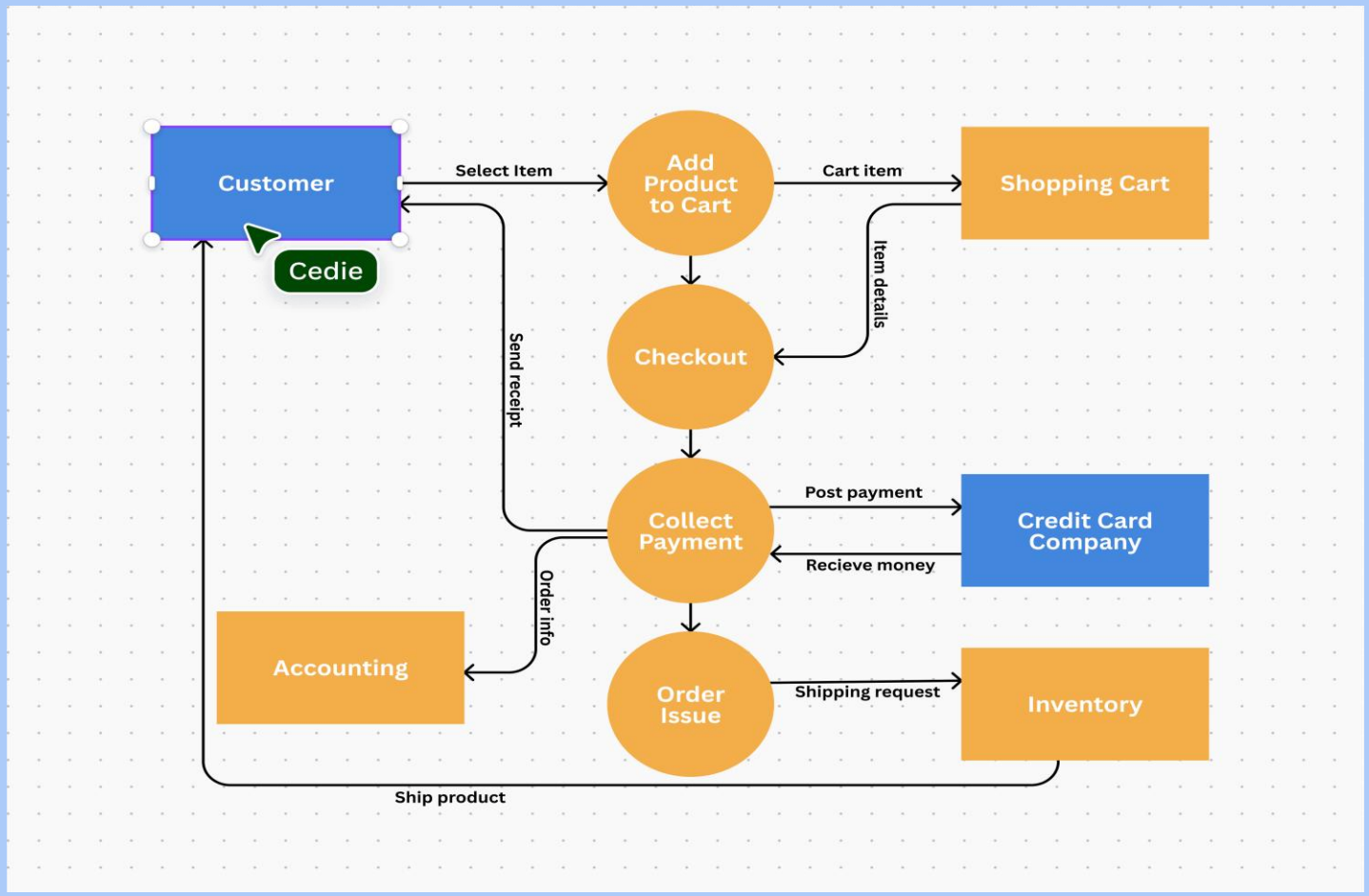


Structured Analysis Tools



Week 13
Data Flow Diagram

Data Flow Diagrams (DFD)

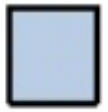


- **Data Flow Diagram (DFD)** represents the flow of data within information systems. DFD provide a graphical representation of the data flow of a system that can be understood by both technical and non-technical users. The models enable software engineers, customers, and users to work together effectively during the analysis and specification of requirements.

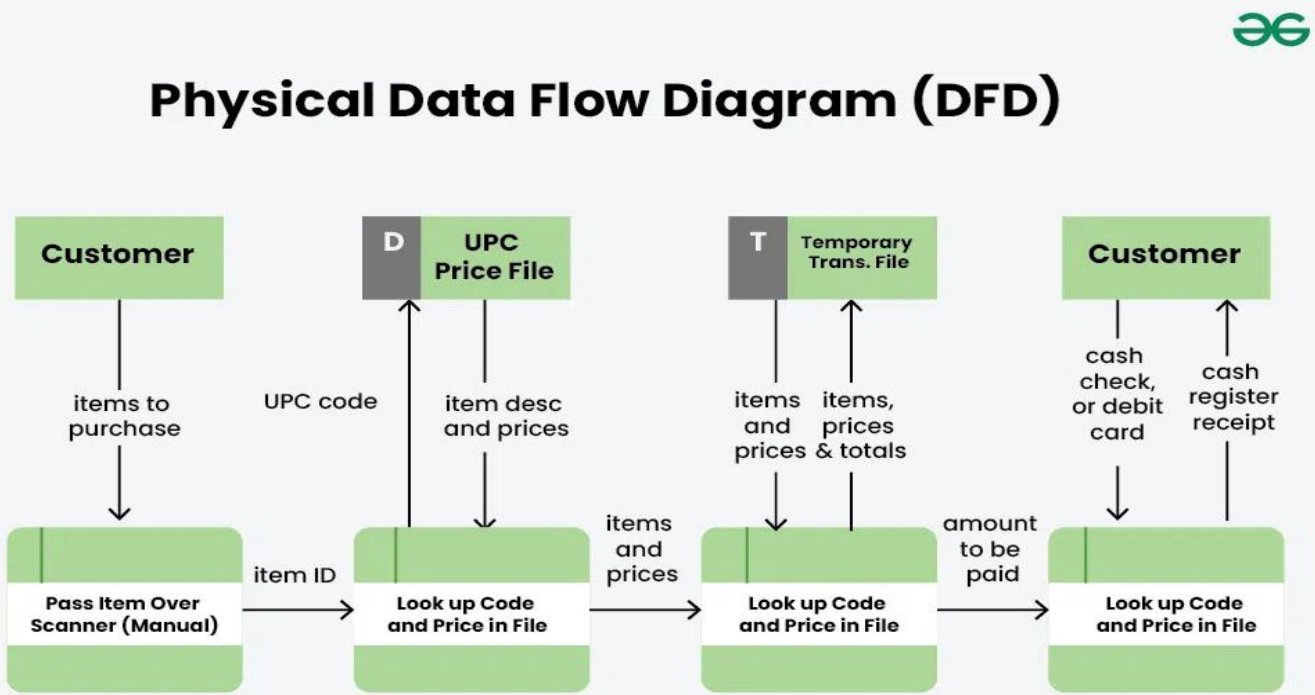
Data Flow Diagrams (DFD)

- It shows the flow of data between various functions of system and specifies how the current system is implemented.
- It is an initial stage of design phase that functionally divides the requirement specifications down to the lowest level of detail.
- Its graphical nature makes it a good communication tool between user and analyst or analyst and system designer.
- It gives an overview of what data a system processes, what transformations are performed, what data are stored, what results are produced and where they flow.

Basic Elements of DFD

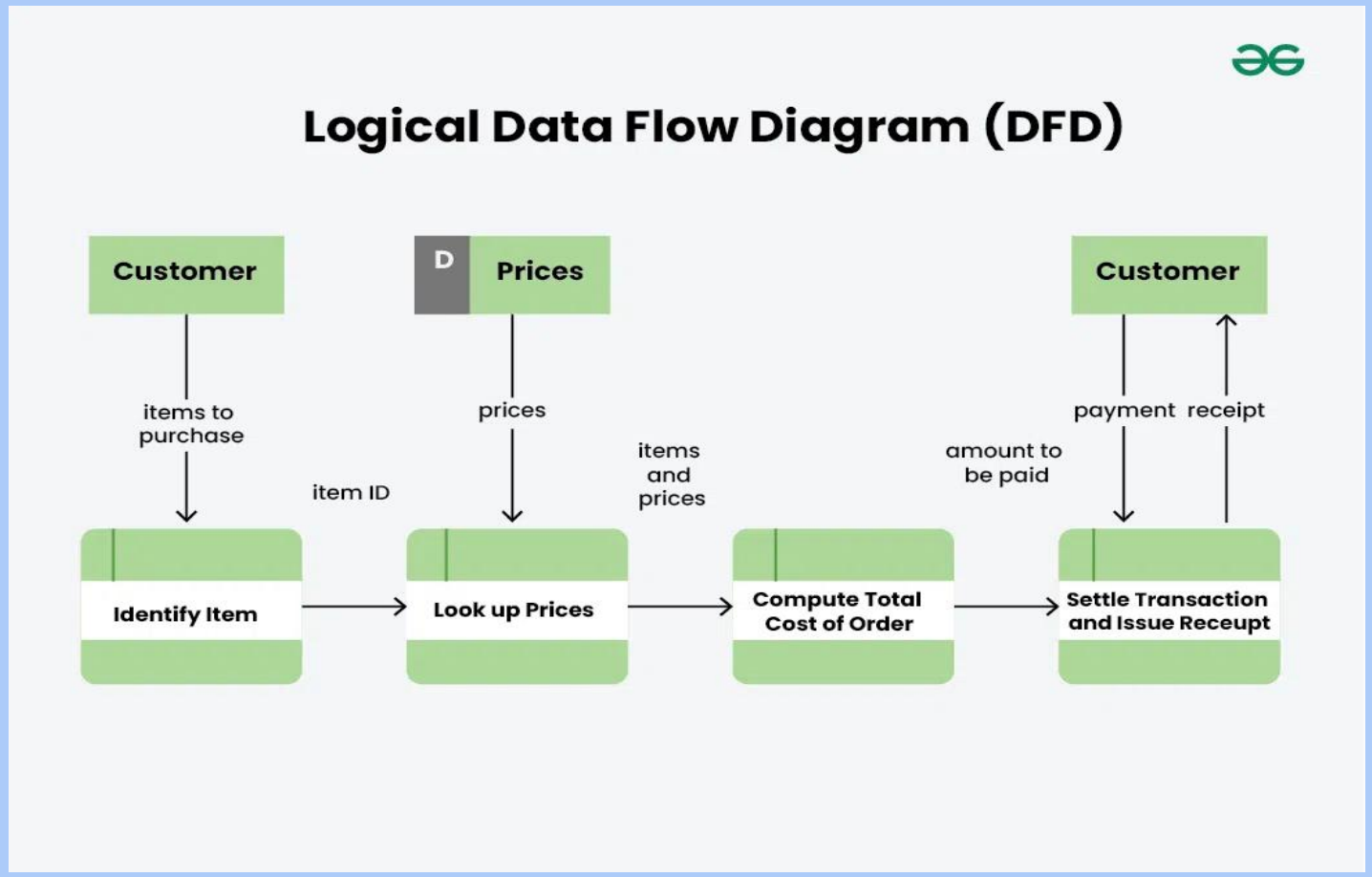
Symbol Name	Symbol	Meaning
Square		Source or Destination of Data
Arrow		Data flow
Circle		Process transforming data flow
Open Rectangle		Data Store

Physical DFD



- Physical DFD depicts how the system will be implemented (or how the current system operates).
- The processes represent the programs, program modules, and manual procedures.
- The data stores represent the physical files and databases, manual files.
- It show controls for validating input data, for obtaining a record, for ensuring successful completion of a process, and for system security.

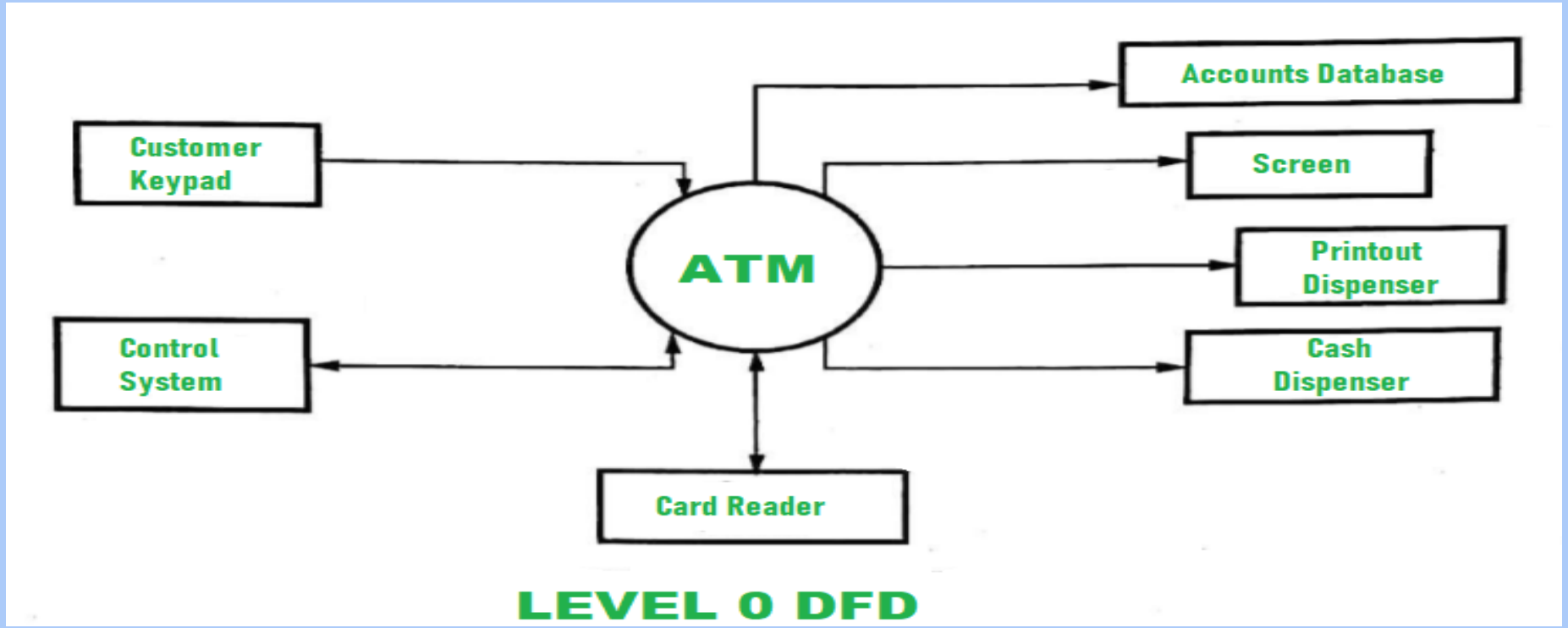
Logical DFD



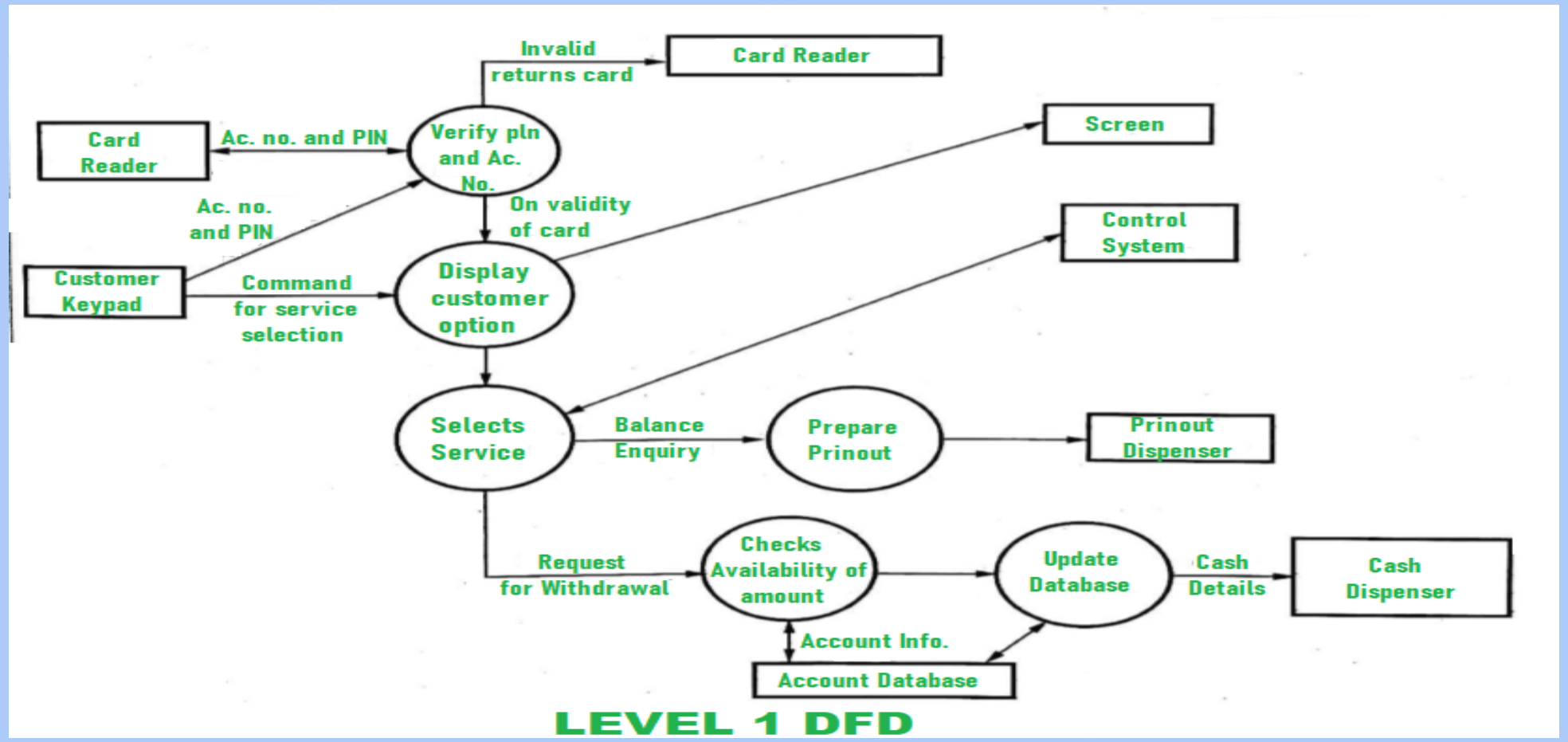
- Logical DFD depicts how the business operates.
- The processes represent the business activities.
- The data stores represent the collection of data regardless of how the data are stored.
- It s how business controls.

Types of Data Flow Diagrams

- Level 0 DFD– Context Diagram
  - **Purpose**– It provides a broad overview of the system, showing the major entities (or external systems) that interact with it and the primary flow of information.



Types of Data Flow Diagrams



- Level 1 DFD– Decomposition of the Main System
  - **Purpose**– To break down the main process into smaller sub-processes, showing how the system functions internally.

The Development Process of Data Flow Diagrams

Step 1: Define System Boundaries and Scope

- Identify all external entities interacting with the system.
- Define what lies within and outside the scope of the DFD.

Step 2: Identify Core Processes

- Pinpoint the main processes that handle data.
- Consider breaking down complex processes to increase clarity.

Step 3: Identify Data Stores

- Determine where the data will be stored within the system.
- Classify these stores based on how data is managed.

The Development Process of Data Flow Diagrams

Step 4: Identify Data Flows

- Establish the data flows between entities, processes, and stores.
- Verify that all necessary inputs and outputs are represented.

Step 5: Construct Context Diagram (Level 0 DFD)

- Create the highest-level DFD showing a single process and external entities.
- Connect entities with the main process through data flows.

The Development Process of Data Flow Diagrams

Step 6: Develop Detailed Levels (Level 1, Level 2)

- Break down the main process in the context diagram into sub-processes.
- Add detail with each level, ensuring accuracy in data flows and connections.

Step 7: Validation and Review

- Validate the DFD with stakeholders to ensure completeness.
- Adjust the diagram based on feedback to address any gaps.

Assignment...!

- **Give detailed Description of DFD**

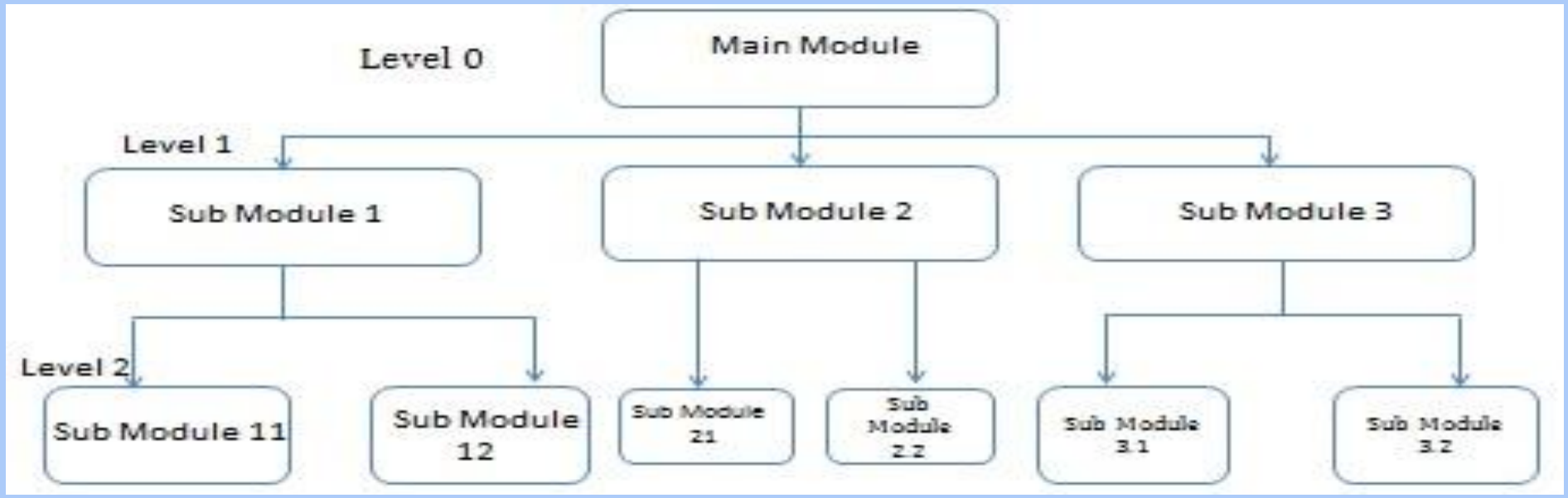
- **Consider any kind of system.**
  - **Now analyze and Draw Physical DFD**
  - **Logical DFD**
  - **Level 0 DFD**
  - **Level 1 DFD**
  - **Level 2 DFD**

Week 14
Design Strategies, Bottom-Up Strategy, Factors Affecting System Complexity

Design Strategies

Top-Down Strategy

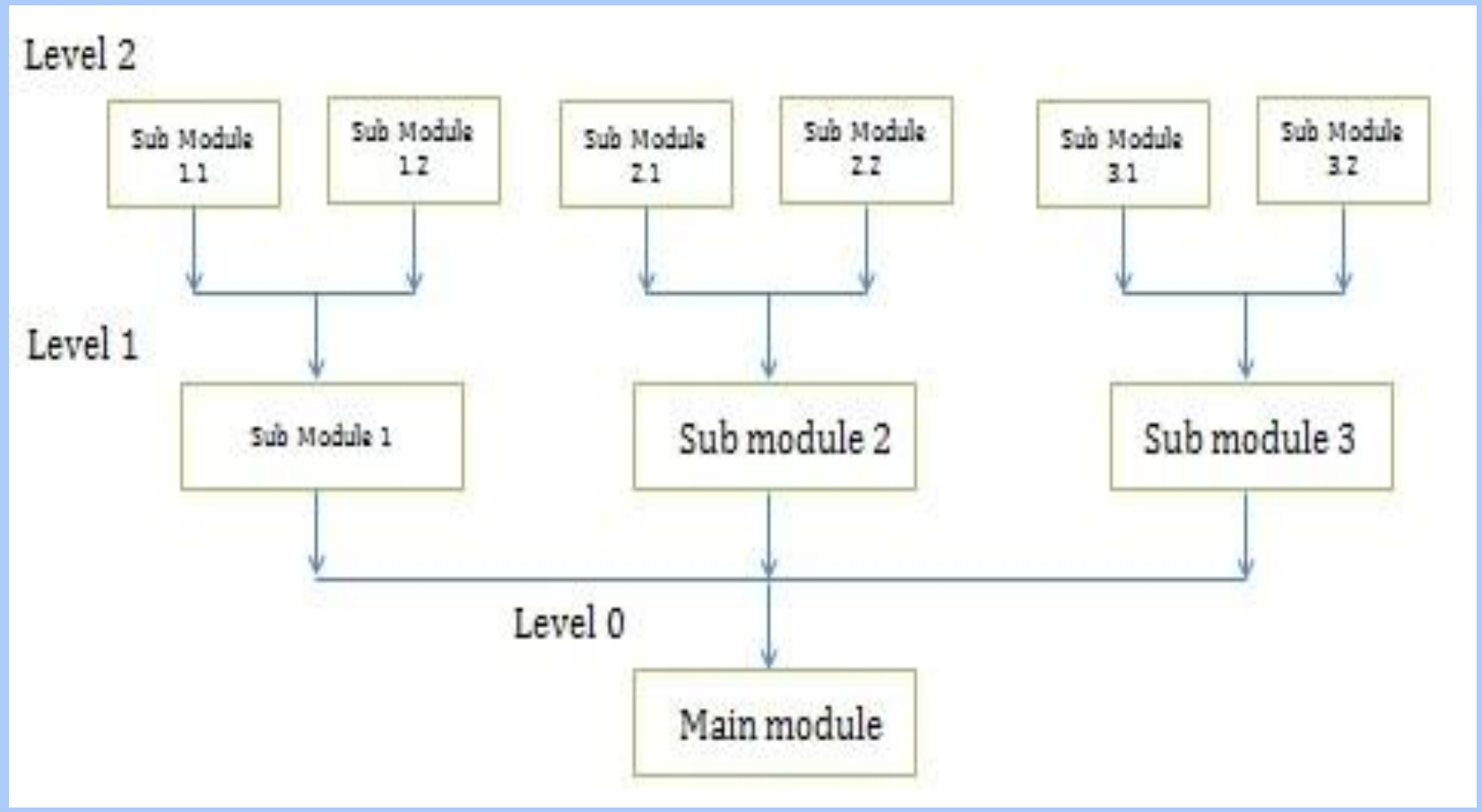
The top-down strategy uses the modular approach to develop the design of a system. It is called so because it starts from the top or the highest-level module and moves towards the lowest level modules.



Design Strategies

- In this technique, the highest-level module or main module for developing the software is identified.
- The main module is divided into several smaller and simpler submodules or segments based on the task performed by each module. Then, each submodule is further subdivided into several submodules of next lower level.
- This process of dividing each module into several submodules continues until the lowest level modules, which cannot be further subdivided, are not identified.

Bottom-Up Strategy



- Bottom-Up Strategy follows the modular approach to develop the design of the system.
- It is called so because it starts from the bottom or the most basic level modules and moves towards the highest level modules.

Bottom-Up Strategy

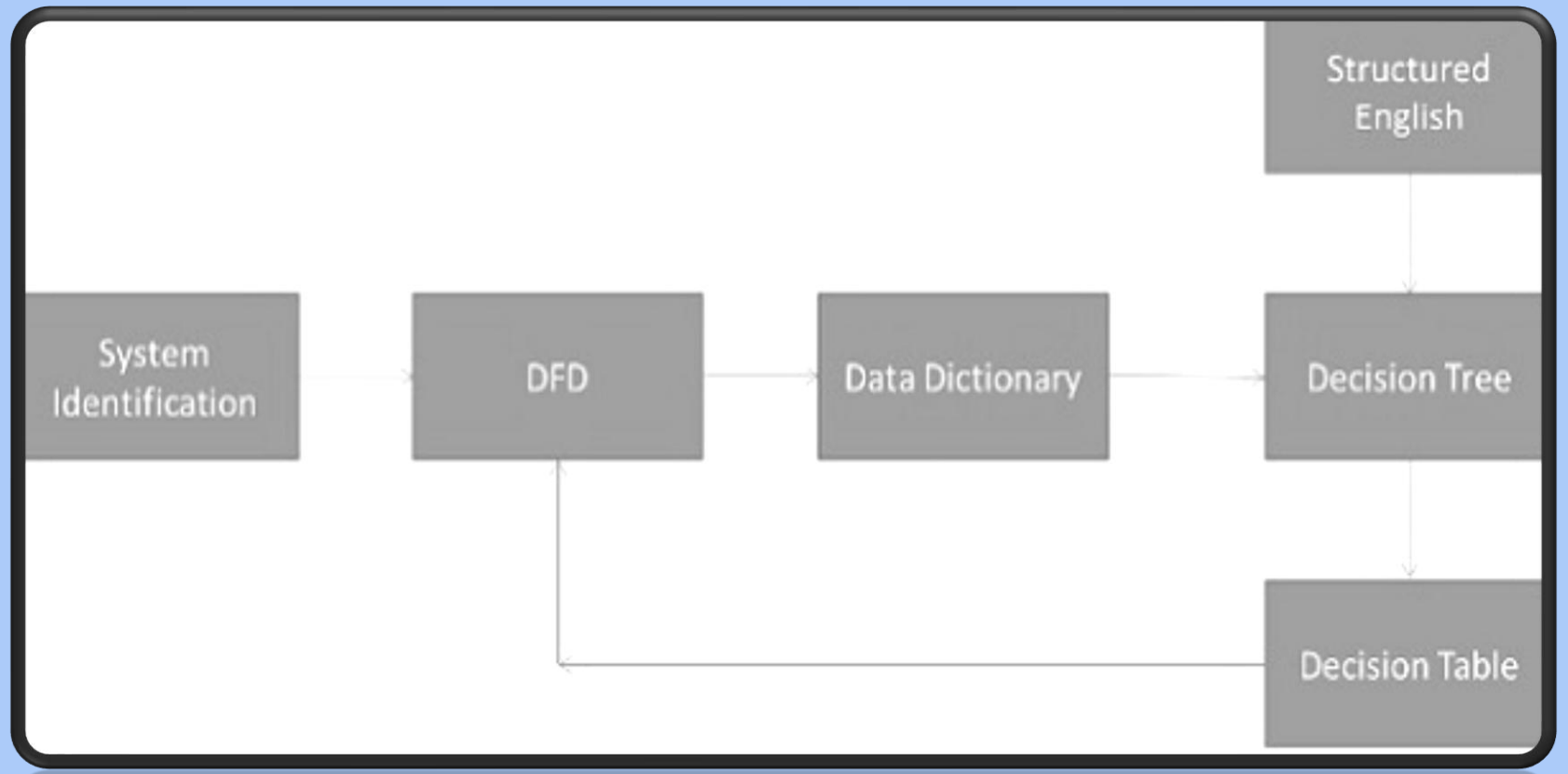
- The modules at the most basic or the lowest level are identified.
- These modules are then grouped together based on the function performed by each module to form the next higher-level modules.
- Then, these modules are further combined to form the next higher-level modules.
- This process of grouping several simpler modules to form higher level modules continues until the main module of system development process is achieved.

Week 15,16
Structured Design, Coupling, Cohesion, System Design vs. Software Design

Structured Design

- Structured design is a data-flow based methodology that helps in identifying the input and output of the developing system.
- The main objective of structured design is to minimize the complexity and increase the modularity of a program.
- Structured design also helps in describing the functional aspects of the system.

Structured Design



- In structured designing, the system specifications act as a basis for graphically representing the flow of data and sequence of processes involved in a software development with the help of DFDs.
- After developing the DFDs for the software system, the next step is to develop the structure chart.

Factors Affecting System Complexity

The two important concepts related to the system development that help in determining the complexity of a system are **coupling** and **cohesion**.

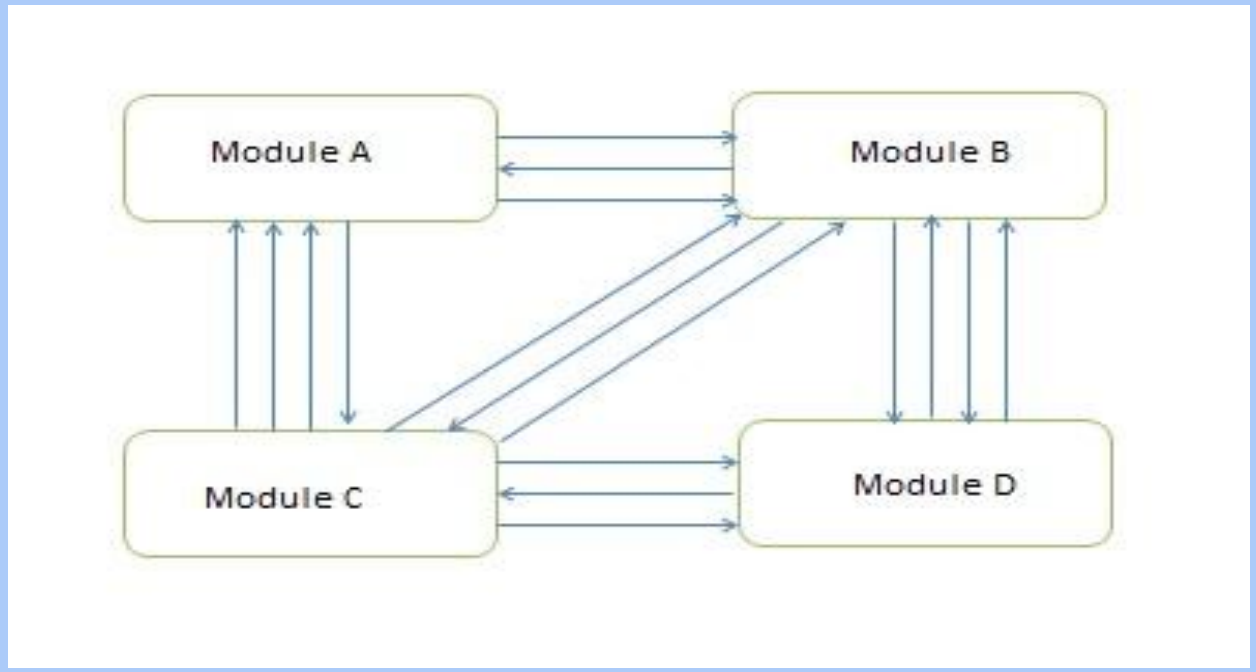
Coupling

Coupling is the measure of the independence of components. It defines the degree of dependency of each module of system development on the other. In practice, this means the stronger the coupling between the modules in a system, the more difficult it is to implement and maintain the system.

coupling

High Coupling

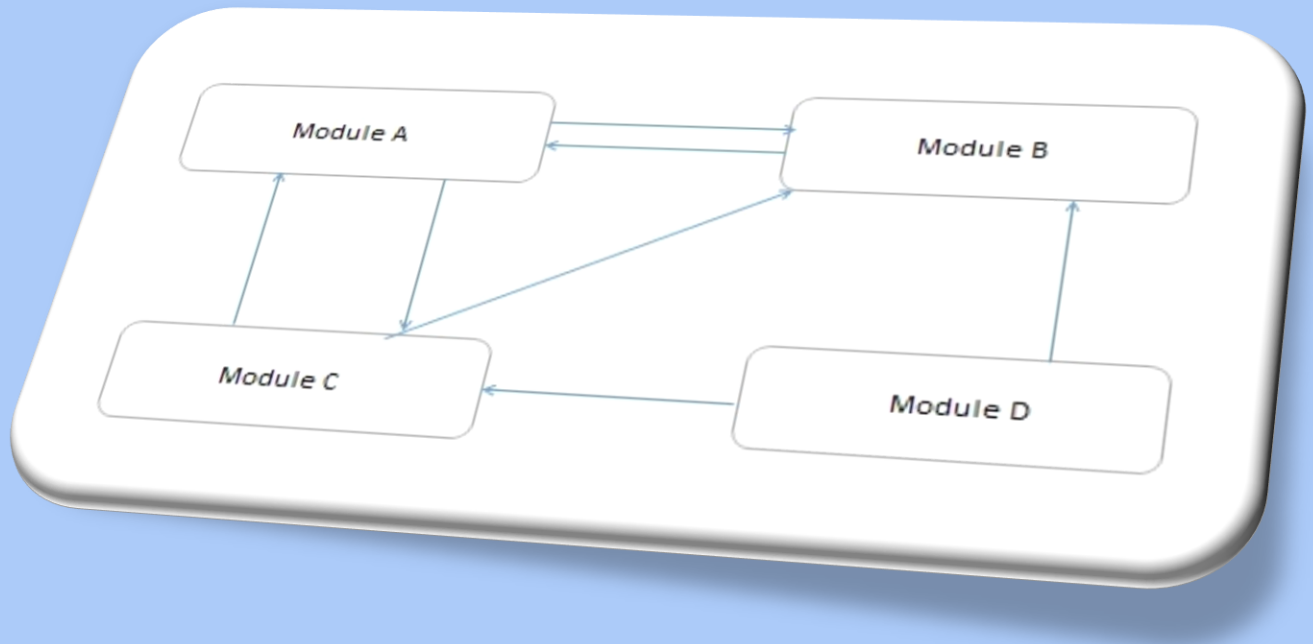
These type of systems have interconnections with program units dependent on each other. Changes to one subsystem leads to high impact on the other subsystem.



coupling

Low Coupling

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



Coupling Measures

- **Content Coupling** – When one component actually modifies another,then the modified component is completely dependent on modifying one.
- **Common Coupling** – When amount of coupling is reduced somewhat by organizing system design so that data are accessible from a common data store.

Coupling Measures

- **Control Coupling** – When one component passes parameters to control the activity of another component.
- **Stamp Coupling** – When data structures is used to pass information from one component to another.
- **Data Coupling** – When only data is passed then components are connected by this coupling.

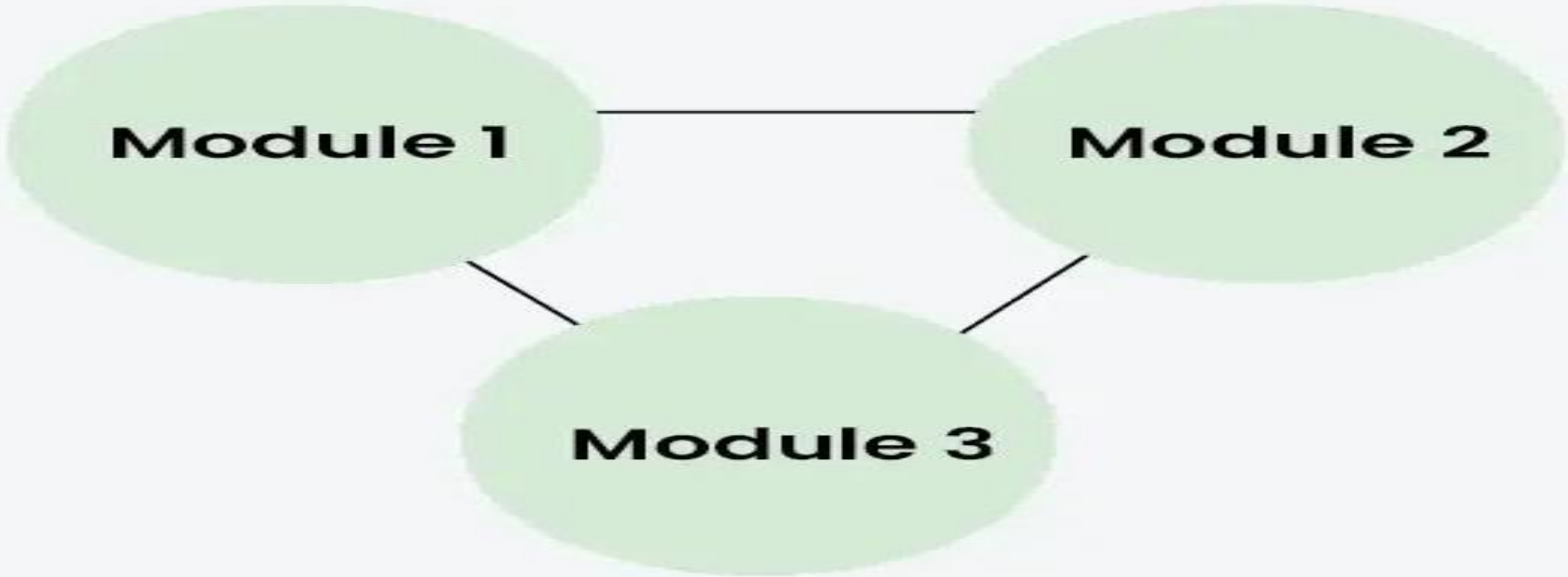
Cohesion

Cohesion is the measure of closeness of the relationship between its components. It defines the amount of dependency of the components of a module on one another. In practice, this means the systems designer must ensure that –

- They do not split essential processes into fragmented modules.
- They do not gather together unrelated processes represented as processes on the DFD into meaningless modules.

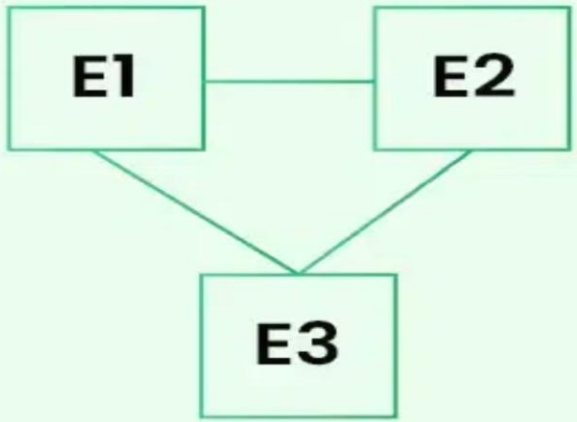
The best modules are those that are functionally cohesive. The worst modules are those that are coincidentally cohesive.

Coupling in System Design



Coupling refers to the degree of interdependence between modules

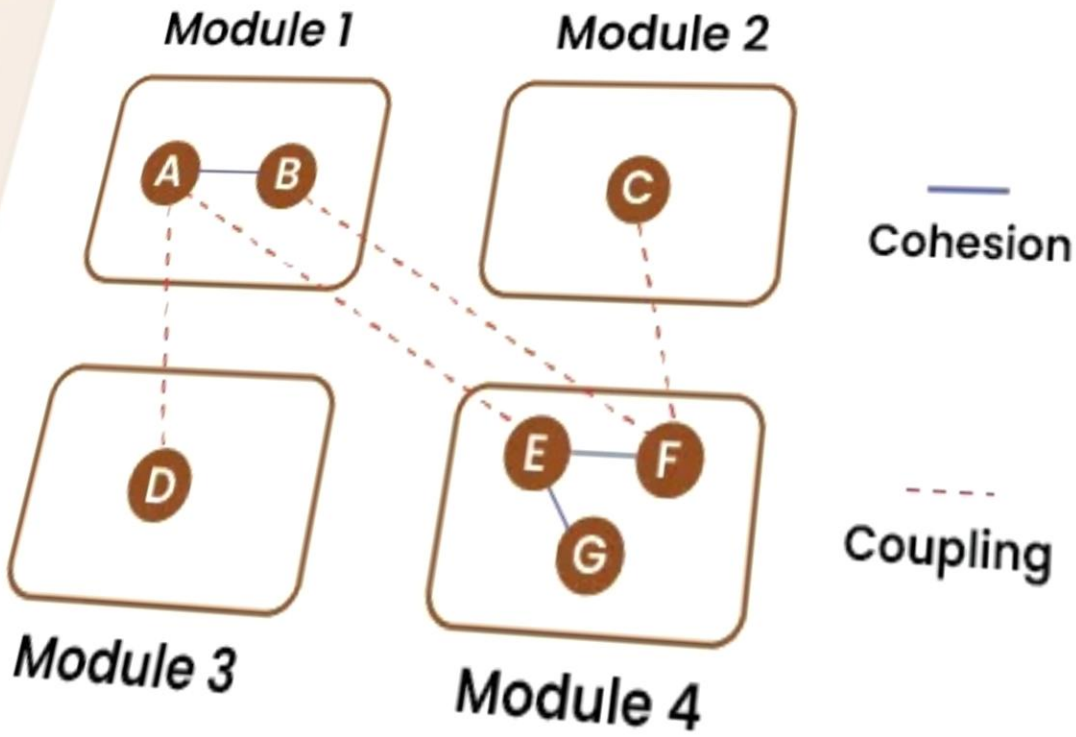
Cohesion in System Design



Cohesion refers to the degree to which elements within a module work together to fulfill a single, well-defined purpose

Coupling & Cohesion

in System Design



Types of Coupling

1. **Content Coupling (Highest Coupling):** One module directly modifies or relies on the internal workings of another module (e.g., accessing local data of another module). This type of coupling is highly undesirable because changes in one module can have significant and unpredictable effects on another.

2. **Common Coupling:** Multiple modules share global data or variables. This is also undesirable as changes to the global data can affect all the modules that use it, leading to potential side effects and difficulties in tracking and managing changes.

3. **External Coupling:** Modules share an externally imposed data format, communication protocol, or interface. This type of coupling occurs when modules are dependent on external systems or hardware, making the system vulnerable to changes in those external entities.

Types of Coupling

4. **Control Coupling:** One module controls the behavior of another by passing it information on what to do (e.g., passing a control flag). This type of coupling is less desirable because it implies that one module is dictating the flow of control in another module.

5. **Stamp (Data-Structured) Coupling:** Modules share a composite data structure and use only a part of it. This is less tightly coupled than control or common coupling but can still lead to dependencies and the need to understand the entire data structure.

6. **Data Coupling (Lowest Coupling):** Modules share data through parameters. Each data item is an elementary piece of data, and no control data is passed. This is the most desirable form of coupling because it minimizes dependencies and makes modules more independent and reusable.

Types of Cohesion

1. **Coincidental Cohesion (Lowest Cohesion):** Elements are grouped arbitrarily and have little to no meaningful relationship to each other. This type of cohesion is undesirable because the module's purpose is unclear, making it difficult to understand and maintain.

2. **Logical Cohesion:** Elements are grouped because they perform similar kinds of activities (e.g., several functions performing input operations). This type is better than coincidental cohesion but still not ideal, as it can lead to less clarity about the module's primary purpose.

3. **Temporal Cohesion:** Elements are grouped because they are involved in activities that are related in time (e.g., initialization tasks that must occur at system startup). While more related than logical cohesion, it still mixes unrelated functionalities tied only by the timing of their execution.

4. **Procedural Cohesion:** Elements are grouped because they always follow a certain sequence of execution (e.g., a sequence of steps in a process). This is better than temporal cohesion but still not ideal, as it focuses on the order of execution rather than the functional relationship.

5. **Communicational (Informational) Cohesion:** Elements are grouped because they operate on the same data or contribute to the same data structure. This type of cohesion is stronger as it

Types of Cohesion

5. ensures that all elements in the module are functionally related through the data they manipulate.

6. **Sequential Cohesion:** Elements are grouped because the output from one part serves as input to another part (e.g., a series of steps in data processing). This type is better than procedural cohesion as it emphasizes the functional relationship between the steps.

7. **Functional Cohesion (Highest Cohesion):** Elements are grouped because they all contribute to a single, well-defined task or function. This is the most desirable form of cohesion. It ensures that each module performs one task or function, making it easy to understand, maintain, and reuse.

System Design vs. Software Design

What is System Design?

System design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. It involves translating user requirements into a detailed blueprint that guides the implementation phase. The goal is to create a well-organized and efficient structure that meets the intended purpose while considering factors like scalability, maintainability, and performance.

- **What is Software Design?**

- Software design encompasses various aspects, including choosing appropriate algorithms, data structures, modules, interfaces, and patterns to ensure the software's functionality, maintainability, scalability, and performance.

- It also involves considering factors such as usability, security, and compatibility to create a fast and user-friendly software solution.

Components of System Design

- **Hardware Components:** This includes physical devices such as servers, computers, storage devices, networking equipment, sensors, and other peripherals necessary for system operation.

- **Software Components:** These encompass the programs, applications, and operating systems required to control and manage hardware resources, process data, and facilitate communication between system elements.

- **Network Infrastructure:** This comprises the network architecture, protocols, and communication channels that enable data transfer and interaction between system components, both within the local environment and across distributed networks.

Components of System Design

- **Data Management:** This involves designing databases, data models, data storage, and retrieval mechanisms to ensure efficient handling, processing, and storage of data within the system.
- **Processes and Workflows:** This includes defining the sequence of tasks, activities, and operations required to accomplish specific goals or functions within the system, often represented through flowcharts, diagrams, or business process models.
- **Security Mechanisms:** These encompass measures such as access control, encryption, authentication, and authorization to protect system assets, data, and resources from unauthorized access, manipulation, or breaches.
- **Scalability and Performance Considerations:** This involves designing the system architecture and infrastructure to accommodate increasing loads, user bases, and data volumes while maintaining optimal performance levels.

Components of System Design

- **Reliability and Fault Tolerance:** This includes implementing redundancy, failover mechanisms, and error-handling strategies to ensure uninterrupted operation and resilience against hardware failures, software errors, or external disruptions.

- **User Interface (UI) and User Experience (UX):** These aspects focus on designing intuitive interfaces, workflows, and interactions to enhance user satisfaction, productivity, and accessibility within the system.

- **Integration and Interoperability:** This involves ensuring seamless interaction and compatibility between different system components, external systems, and third-party services through standardized interfaces, APIs, and protocols.

Components of Software Design

- **Architectural Design:** This involves defining the overall structure and organization of the software system, including high-level components, their interactions, and the distribution of responsibilities among them. Common architectural patterns include layered architecture, client-server architecture, microservices architecture, and more.

- **Module Design:** Modularization involves breaking down the system into smaller, cohesive units or modules, each responsible for specific functionalities. This promotes code reusability, maintainability, and scalability. Design principles such as cohesion and coupling guide the creation of effective modules.

- **Data Design:** This focuses on designing the data structures, databases, and data models required to represent and manage the application's data effectively. It includes considerations such as data integrity, normalization, indexing, and optimization for efficient storage and retrieval.

Components of Software Design

- **User Interface (UI) Design:** UI design involves creating intuitive and user-friendly interfaces that enable users to interact with the software easily. It encompasses layout design, navigation flows, visual elements, and usability considerations to enhance the user experience.

- **Algorithm Design:** Algorithms are at the core of software systems, determining how tasks are performed efficiently. Software design involves selecting and designing appropriate algorithms for various operations, considering factors such as time complexity, space complexity, and optimization techniques.

- **Error Handling and Exception Design:** This component focuses on designing mechanisms to detect, report, and handle errors and exceptions gracefully within the software system. It includes strategies such as exception handling, logging, and recovery mechanisms to ensure robustness and reliability.

Components of Software Design

- **Security Design:** Security design involves incorporating measures to protect the software system against unauthorized access, data breaches, and malicious attacks. This includes authentication, authorization, encryption, input validation, and other security controls to mitigate security risks.

- **Performance Design:** Performance design aims to optimize the software system's speed, responsiveness, and resource utilization. It involves profiling, benchmarking, and optimizing critical components, algorithms, and database queries to achieve desired performance goals.

- **Concurrency and Multithreading Design:** For concurrent and parallel execution of tasks, software design includes designing concurrent algorithms, synchronization mechanisms, and thread-safe data structures to ensure correctness and avoid race conditions.

Challenges in System Design

- **Scalability:** Designing a system that can handle increasing loads, data volumes, and user bases without compromising performance or functionality.
- **Reliability:** Ensuring continuous and dependable operation of the system, even in the face of hardware failures, software errors, or external disruptions.
- **Availability:** Designing redundancy and failover mechanisms to minimize downtime and ensure that critical services are always accessible to users.
- Performance Optimization: Balancing resource utilization, response times, and throughput to achieve optimal system performance under varying workloads and conditions.

Challenges in System Design

- **Complex Systems Integration:** Integrating diverse hardware and software components, legacy systems, and third-party services while maintaining compatibility and interoperability.
- **Security:** Implementing robust security measures to protect sensitive data, prevent unauthorized access, and mitigate cybersecurity threats such as hacking, malware, and data breaches.
- **Legacy Systems:** Upgrading or migrating legacy systems to modern architectures while minimizing disruption and ensuring compatibility with existing infrastructure and processes.
- **Evolving Requirements:** Adapting the system design to accommodate changing business needs, technological advancements, and regulatory requirements over time.

Challenges in Software Design

- **Requirements Management:** Gathering, understanding, and managing requirements from stakeholders can be complex. Ensuring that requirements are clear, complete, and consistent is essential to successful software design.

- **Scalability:** Designing software to handle increasing amounts of data, users, or transactions can be challenging. Ensuring that the architecture and design can scale efficiently without sacrificing performance is crucial.

- **Maintainability:** Writing code that is easy to understand, modify, and extend is essential for long-term success. Poorly designed software can become difficult and costly to maintain over time.

Challenges in Software Design

- **Flexibility and Extensibility:** Designing software that can adapt to changing requirements and environments is vital. Building a flexible and extensible architecture allows for easier integration of new features and technologies.

- **Performance Optimization:** Balancing performance requirements with other design considerations can be challenging. Optimizing algorithms, data structures, and system architecture to meet performance goals without sacrificing other qualities such as maintainability and scalability is crucial.

Challenges in Software Design

- **Security:** Ensuring that software is secure against various threats, such as unauthorized access, data breaches, and malicious attacks, is essential. Designing robust security measures into the architecture and implementing secure coding practices are critical aspects of software design.

- **Integration and Interoperability:** Integrating software components with external systems, APIs, and third-party services can be complex. Ensuring compatibility, reliability, and seamless interaction between different components is essential for interoperability.

Week 16, 17
System Design - High Level Design, Low Level Design, System Implementation, Testing

System Design - High Level Design

Definition

High-Level Design (HLD) provides a macro view of the system. It outlines the architecture, subsystems, modules, and how they interact. Unlike Low-Level Design, which deals with specific implementations, HLD focuses on–

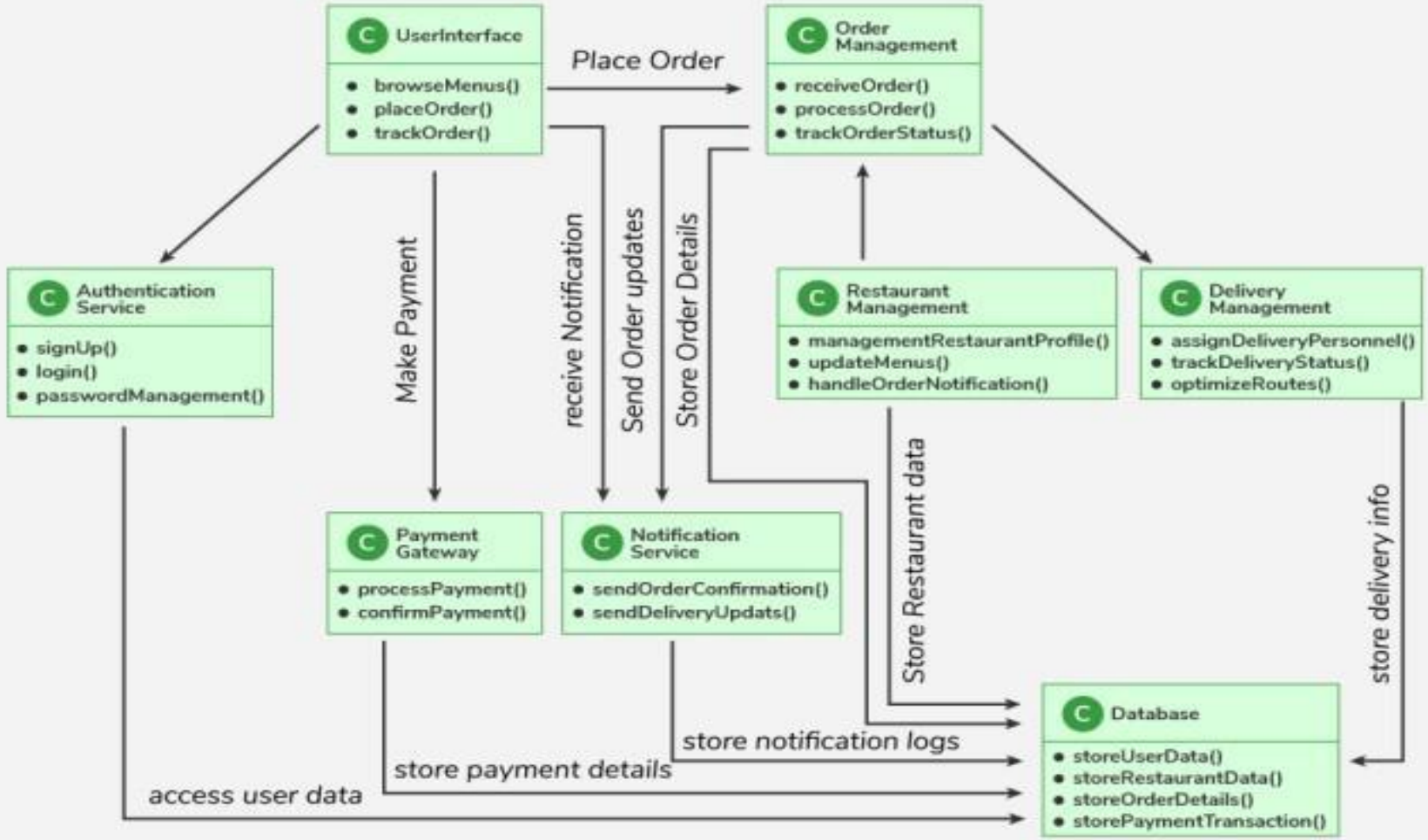
- The system's major components.
- Communication protocols between components.
- Scalability and performance considerations.

Goals of High-Level Design

- **Clarity**– Provide a clear and shared understanding of the system's architecture.
- **Direction**– Guide developers by offering an architectural roadmap.
- **Scalability**– Anticipate future growth and design for it.
- **Alignment**– Ensure that technical decisions align with business objectives.
- **Key Components of High-Level Design**
  - System Architecture
  - Subsystems and Modules
  - Data Flow
  - Database Design

Online Food Delivery System

(High-Level Design)



What is Low-Level Design (LLD)?

Low-Level Design refers to the process of designing the internal workings of individual components in a software system. It breaks down the abstract architectural ideas from HLD into concrete, implementable details.

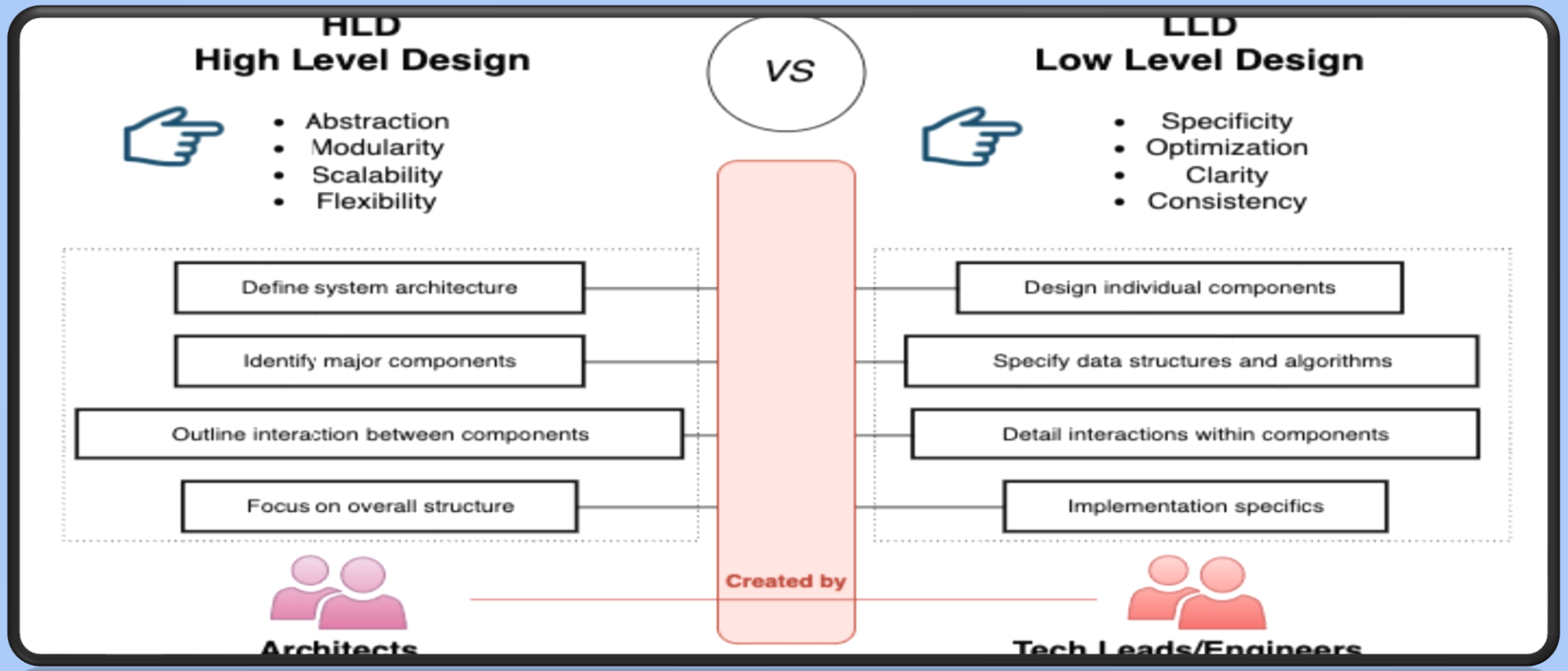
Key characteristics of LLD

- Detailed documentation of classes, methods, and interactions.
- Definitions of how system components communicate internally.
- Inclusion of diagrams like UML (Unified Modeling Language) to represent relationships and workflows.

Low Level vs High Level

	High-Level Design	Low-Level Design
Level of detail	Abstract and broad, provides a bird's-eye view of the system	Detailed and specific, focuses on the inner workings of each component
Focus	System as a whole and its major components	Implementation details of each component
Documentation	Less detailed, more conceptual	More detailed, provides specific guidance for implementation
Timing	Created early in the design process to establish the overall architecture	Created after the HLD, during the detailed design phase to guide development

Low Level vs High Level

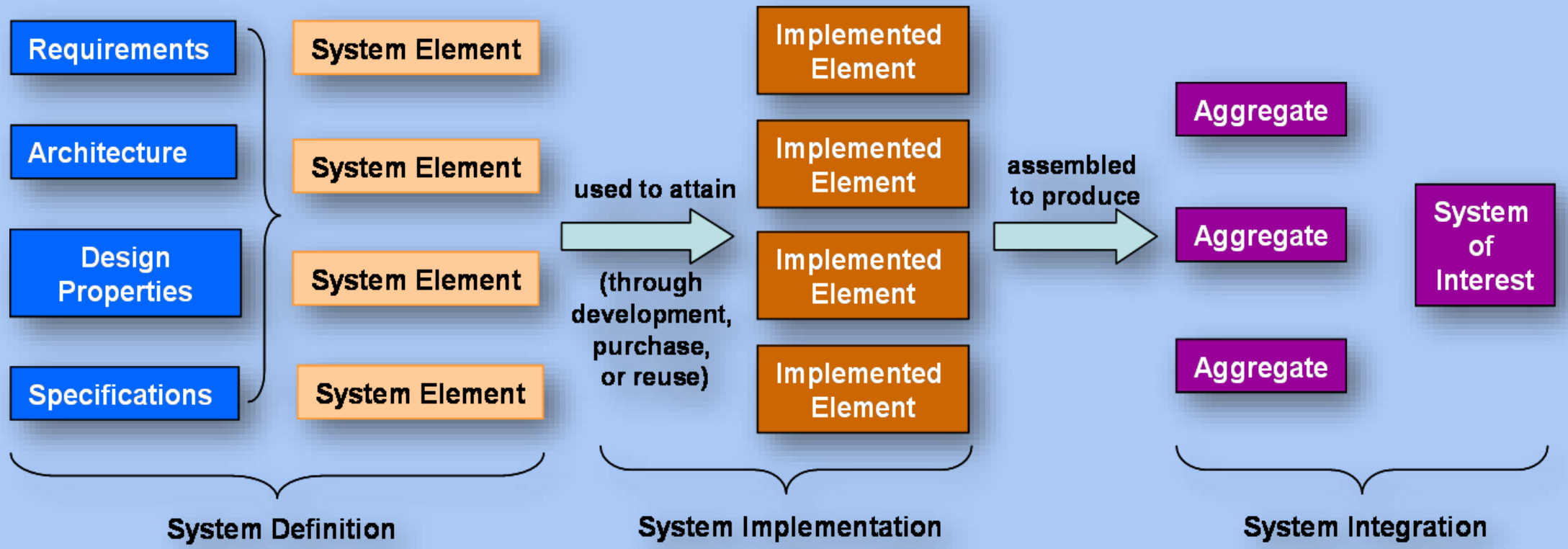


System Implementation

- **System Implementation** uses the structure created during architectural design and the results of system analysis to construct system elements that meet the stakeholder needs and requirements and system requirements developed in the early life cycle phases.
- These system elements are then integrated to form intermediate aggregates and finally the complete system-of-interest (SoI).
- The purpose of the implementation process is to design and create (or fabricate) a system element conforming to that element's design properties and/or requirements.
- The element is constructed employing appropriate technologies and industry practices.
- This process bridges the system definition processes and the integration process.

System Implementation

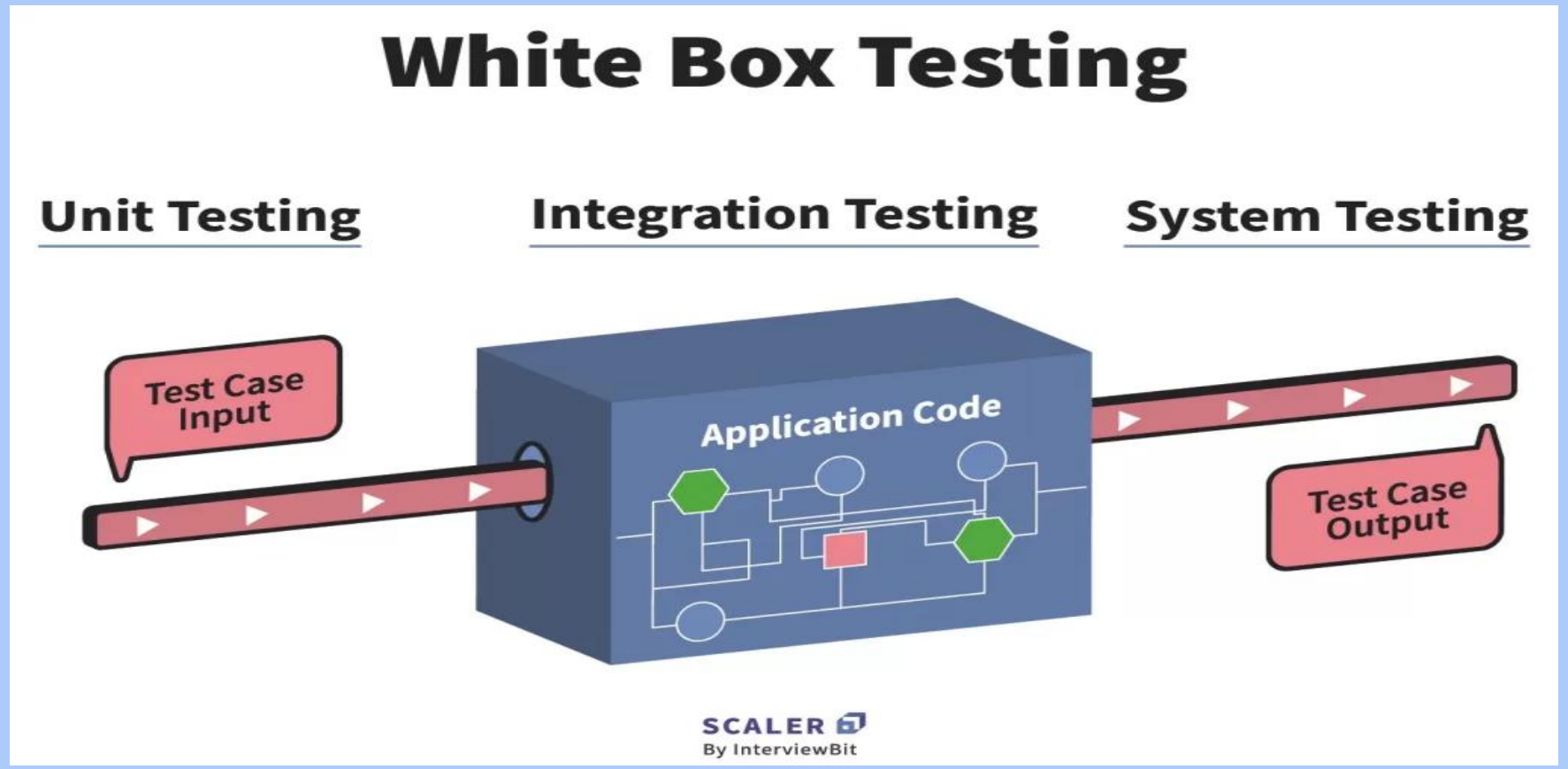
Following Figure portrays how the outputs of system definition relate to system implementation, which produces the implemented (system) elements required to produce aggregates and the SoI.



System Testing

- **What is System Testing?**
  - **System Testing** is a level of testing that validates the complete and fully integrated software product. The purpose of a system test is to evaluate the end-to-end system specifications.

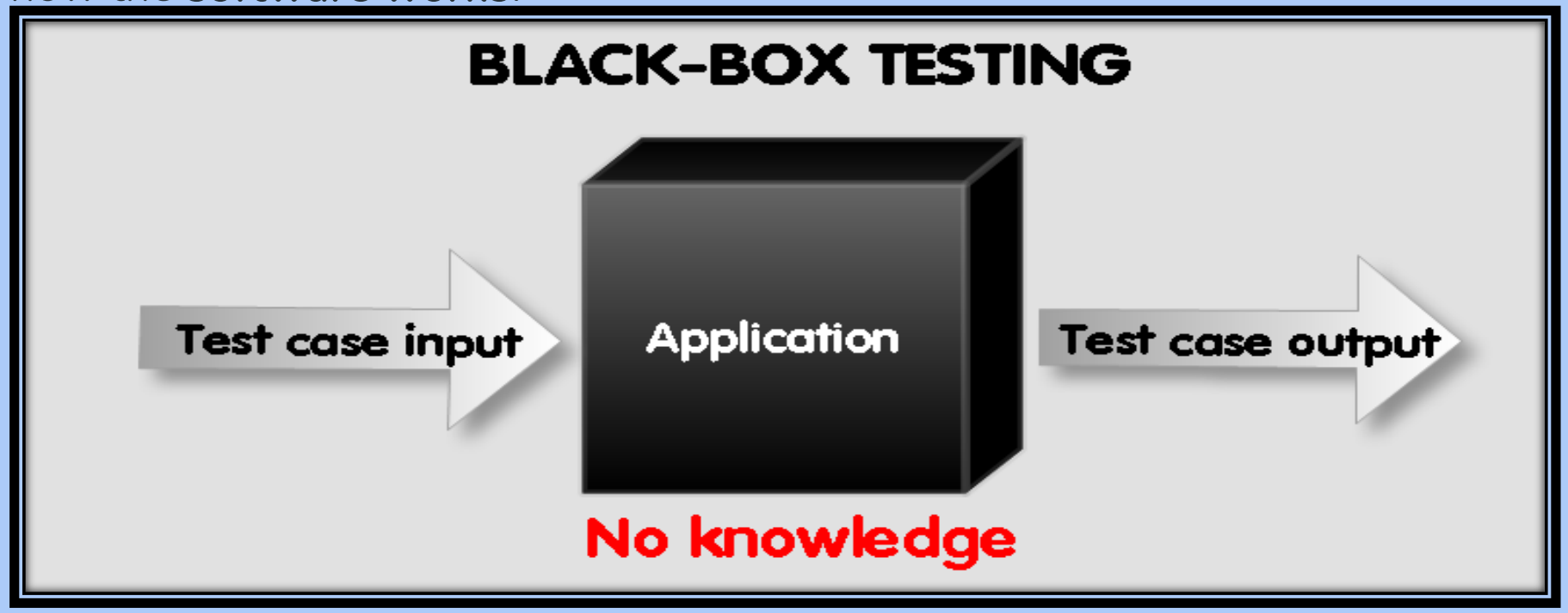
- Black Box Testing
- White Box Testing



- **White box testing** is the testing of the internal workings or code of a software application. In contrast, black box or System Testing is the opposite. System test involves the external workings of the software from the user's perspective.

Black Box Testing

Black Box Testing is an important part of making sure **software** works as it should. Instead of exploring the **code**, testers check how the **software behaves** from the outside, just like **users** would. This helps catch any **issues** or **bugs** that might affect how the **software works**.



Software Testing Hierarchy

- **Unit testing** performed on each module or block of code during development. Unit Testing is normally done by the programmer who writes the code.
- **Integration testing** done before, during and after integration of a new module into the main software package. This involves testing of each individual code module. One piece of software can contain several modules which are often created by several different programmers. It is crucial to test each module's effect on the entire program model.

Testing Hierarchy

- **System testing** done by a professional testing agent on the completed software product before it is introduced to the market.
- **Acceptance testing** – beta testing of the product done by the actual end users.