Victoria University
of Bangladesh

Final Assessment

Md Bakhtiar Chowdhury

ID: 2121210061

Department: CSE

Semester: Summer 2023

Batch: 21th

Course Title: System Analysis and Design

Course Code: CSI 311

Submitted To:

Umme Khadiza Tithi

Lecturer, Department of Computer Science & Engineering

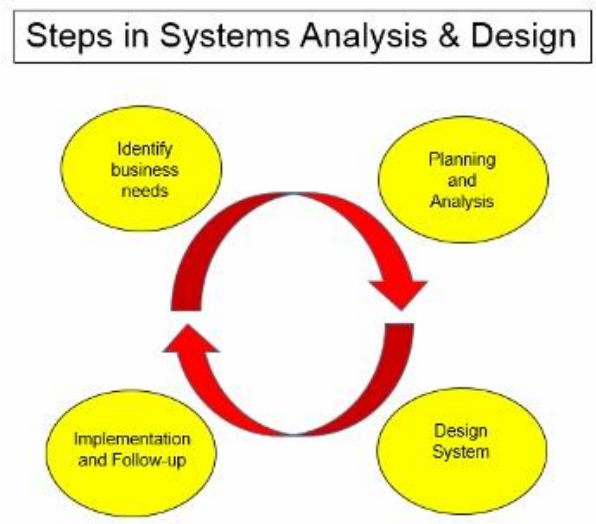
Victoria University of Bangladesh

Submission Date: 10 October, 2023

Answer to the question no 1

 # What is System Design? Differences between system analysis and system design?

System Design is a crucial phase in the software development life cycle, following system analysis. It involves the process of defining how the components of a system will be organized and how they will work together to fulfill the specified requirements. System design transforms the functional requirements identified during system analysis into a detailed technical blueprint for the construction of the system. Here are the key aspects of system design and the differences between system analysis and system design:



Aspect	System Analysis	System Design
Nature of Activity	Focuses on understanding and defining user requirements	Focuses on planning how to implement the system to meet those requirements
Level of Abstraction	Deals with high-level, abstract representations of the system	Deals with low-level, detailed representations specifying how the system will work
Output	Requirements documents, use cases, functional specifications	Detailed technical design documents, architectural diagrams, data models

Aspect	System Analysis	System Design
Purpose	Aims to understand and document user needs and business processes	Aims to provide a technical plan for constructing the system based on defined requirements
Focus	Emphasizes the "what" of the system - defining functional and non-functional requirements	Emphasizes the "how" of the system - specifying how it will be built and how components will interact
Skills and Expertise	Requires skills in requirements gathering, stakeholder interviews, and business process modeling	Requires technical skills in software architecture, database design, and system integration
Timing in the SDLC	Typically one of the initial phases in the Software Development Life Cycle (SDLC), focusing on understanding requirements before design and development.	Follows system analysis in the SDLC, occurring after the requirements are gathered and serves as a blueprint for development.
Abstraction vs. Specificity	Involves abstract and high-level representations of the system, such as use cases, functional diagrams, and requirements.	Involves detailed and specific representations, including architectural diagrams, data models, and technical specifications.

This tabular format provides a clear comparison between the two phases, highlighting their distinct roles and objectives in the software development process.

Answer to the question no 2

 # What is System? Types of Systems? Define System models?

System:
A system is a collection of interconnected and interdependent components or elements that work together to achieve a common objective or purpose. In a system, these components interact with each other to perform specific functions or processes. Systems can be found in various fields, including engineering, biology, information technology, and social sciences. They are characterized by inputs, processes, outputs, and feedback mechanisms. Systems thinking is a methodology that involves analyzing and understanding how these components interact to achieve the system's goals.

Types of Systems:
Systems can be categorized into several types based on their characteristics and application domains:

1. Physical Systems: These involve tangible components, such as machines, vehicles, or buildings. Physical systems are often studied in engineering disciplines.

2. Biological Systems: Biological systems include living organisms, such as humans, animals, and plants. These systems are studied in biology and related fields.

3. Information Systems: Information systems involve the processing and management of data and information. Examples include databases, software applications, and communication networks.

4. Social Systems: Social systems are composed of individuals, groups, or organizations that interact to achieve social or societal goals. Examples include governments, businesses, and communities.

5. Economic Systems: These systems encompass the structures and processes that govern the production, distribution, and consumption of goods and services in an economy. Capitalism and socialism are examples of economic systems.

6. Ecological Systems: Ecological systems involve the interactions between living organisms and their environment. Ecosystems, food chains, and climate systems are examples.

7. Control Systems: Control systems are designed to regulate and control processes or devices. Examples include feedback control systems in engineering and automation.

System Models:

System models are abstract representations or simplifications of real-world systems used for analysis, design, and understanding. They help in studying and simulating complex systems by breaking them down into manageable components. Common types of system models include:

1. Mathematical Models: These models use mathematical equations and formulas to represent the behavior of a system. Differential equations, algebraic models, and statistical models are examples.

2. Block Diagrams: Block diagrams depict a system's components as blocks connected by lines to represent relationships and interactions. They are often used in control systems and engineering.

3. Flowcharts: Flowcharts use graphical symbols and arrows to illustrate the flow of processes within a system. They are commonly used in software development and process analysis.

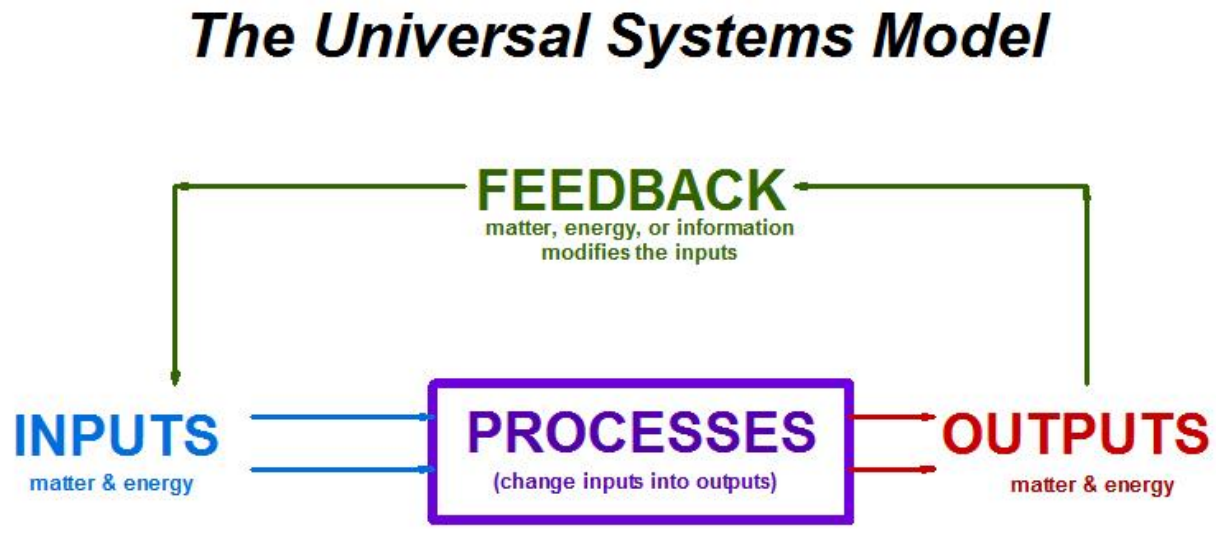
4. State Transition Diagrams: These diagrams depict the different states a system or process can be in and the transitions between them. They are useful for modeling dynamic systems.

5. Simulation Models: Simulation models use computer software to mimic the behavior of a system over time. They are valuable for predicting and analyzing the system's performance.

6. Data Models: Data models represent the structure and relationships within a database or information system. Entity-Relationship Diagrams (ERDs) and relational data models are examples.

7. System Dynamics Models: These models focus on understanding how changes in one part of a system can affect other parts over time. They are often used for studying complex dynamic systems.

System models help in gaining insights into system behavior, predicting outcomes, making decisions, and improving system performance by providing a simplified but meaningful representation of reality.



Answer to the question no 3

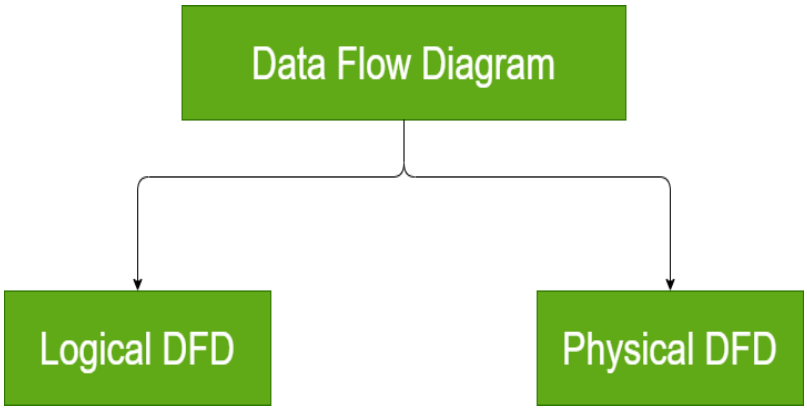
 # Define element and types of DFD?

Data Flow Diagram (DFD) is a graphical representation of data flow in any system. It is capable of illustrating incoming data flow, outgoing data flow and store data. Data flow diagram describes anything about how data flows through the system.

Sometimes people get confused between data flow diagram and flowchart. There is a major difference between data flow diagram and flowchart. The flowchart illustrates flow of control in program modules. Data flow diagrams illustrate flow of data in the system at various levels. Data flow diagram does not have any control or branch elements.

Types of DFD :

DFD is of two types:



Logical DFD:

Logical data flow diagram mainly focuses on the system process. It illustrates how data flows in the system. Logical DFD is used in various organizations for the smooth running of system. Like in a Banking software system, it is used to describe how data is moved from one entity to another.

Physical DFD:

Physical data flow diagram shows how the data flow is actually implemented in the system. Physical DFD is more specific and close to implementation.

Element in DFD (Data Flow Diagram):

In a Data Flow Diagram (DFD), an element refers to a specific component or symbol used to represent different aspects of a system or process. Elements in a DFD help visualize the flow of data within a system and the interactions between various components. The primary elements in a DFD include processes, data stores, data flows, and external entities Here's a brief explanation of these elements:

1. **Processes:** Processes, represented as circles or ovals, depict specific activities, functions, or transformations that occur within the system. They indicate what happens to the data as it moves through the system.
2. **Data Stores:** Data stores, represented as rectangles, represent places where data is stored or retrieved. These can be physical locations like databases or files, or they can be temporary storage within the system.
3. **Data Flows:** Data flows, represented as arrows, represent the movement of data between processes, data stores, external entities, or other elements. They show the direction of data and the paths it takes.
4. **External Entities:** External entities, represented as rectangles with rounded corners, represent entities outside the system that interact with it. These can be users, other systems, or external data sources.

Types of DFD (Data Flow Diagram):

DFDs are typically categorized into different levels or types based on the level of detail and abstraction. The main types of DFDs include:

Certainly, here are the types of Data Flow Diagrams (DFDs) in a shorter format:

- Context Diagram: Provides an overview of the entire system and its interactions with external entities.

- Level 1 DFD: Breaks down the system into major processes or subsystems.

- Lower-Level DFDs: Offer more detailed views of processes, data flows, and data stores with multiple levels of decomposition.

- Full System DFD: Represents the entire system at a detailed level.

- Primitive DFDs: Show processes in detail, often with inputs, outputs, and subprocesses.

The choice of DFD type depends on the purpose of the diagram and the level of detail required for analysis and communication. Context Diagrams are useful for high-level understanding, while lower-level DFDs are valuable for detailed analysis and design.

Answer to the question no 4

 # Write down Bottom-Up strategys advantages and Disadvantage? Objectives of using structural flowcharts?

Bottom-Up Strategy

The Bottom-Up strategy is an approach used in various fields, including software development, problem-solving, and system design. It involves starting with the smallest components or details and gradually building up to create a larger, more complex system. This strategy is often associated with incremental development and can be contrasted with the Top-Down approach, which starts with the overall system and breaks it down into smaller components.

Advantages of Bottom-Up Strategy:

1. Modularity: One of the key advantages of the Bottom-Up strategy is its focus on modularity. By developing and testing small, independent components first, it becomes easier to manage, reuse, and maintain these modules. This modularity enhances system flexibility and scalability.

2. Early Validation: Since each small component is developed and tested individually, validation of these components can occur early in the development process. This helps in identifying and fixing issues at a granular level, reducing the risk of major problems in the final system.

3. Incremental Progress: Bottom-Up development allows for incremental progress. Developers can see tangible results as individual components are completed and integrated, providing a sense of accomplishment and ensuring that progress is visible even before the entire system is complete.

4. Faster Iterations: Bottom-Up allows for faster iterations and frequent updates. As each component is developed, tested, and integrated, it can be refined and improved in subsequent iterations, leading to faster overall development.

5. Flexibility and Adaptability: Bottom-Up strategy lends itself well to adaptability. Changes in requirements or the addition of new features can be integrated more easily into the existing modular structure.

Disadvantages of Bottom-Up Strategy:

1. Integration Complexity: The Bottom-Up strategy, while focused on modularity, can lead to integration complexities. Ensuring that all components work seamlessly together can be challenging, especially if there are unexpected dependencies or interactions.

2. Lack of Early System View: Since the Bottom-Up approach starts with small components, there might be a lack of a comprehensive system view in the early stages of development. This can make it difficult to ensure that the overall system design aligns with the intended goals.

3. Potential Redundancy: Without careful planning and coordination, there's a risk of redundant efforts in developing similar functionalities within different components. This can lead to inefficiencies and increased development time.

4. Difficulty in System-level Decisions: Some system-level decisions may need to be deferred until the components are integrated, making it challenging to make informed architectural choices early in the development process.

5. Testing Complexity: While individual components can be tested effectively, testing the interactions and integration of all components can be complex and time-consuming, especially if integration testing is not well-structured.

In summary, the Bottom-Up strategy offers advantages such as modularity, early validation, incremental progress, and flexibility. However, it can also present challenges in terms of integration complexity, the lack of early system view, and potential redundancy. The choice of strategy depends on the specific project, its requirements, and the development team's preferences and expertise.

>>>>>>END<<<<<<