Lecture 8

➔ (Week 8 - Feb 26, 2024)

Purpose of Analysis Phase:

- Specifies the software's operational characteristics.
- Indicates the software's interfaces with other system elements.
- Establishes constraints that the software must meet.
- Provides the software designer with a representation of information, function, and behavior, later translated into architectural, interface, class/data, and component-level designs.
- Provides the developer and customer with the means to assess quality once the software is built.

Analysis Phase Objectives:

1. **Describe Customer Requirements:** Clearly articulate what the customer needs.
2. **Basis for Software Design:** Lay the foundation for creating a software design.
3. **Define Validatable Requirements:** Establish a set of requirements that can be validated post-development.

Analysis Rules of Thumb:

- Focus on visible requirements within the problem or business domain.
- Maintain a relatively high level of abstraction.
- Each element should enhance understanding of requirements and provide insight into the system's information domain, function, and behavior.
- Delay consideration of infrastructure and non-functional models until the design phase.
- Minimize coupling to reduce interconnectedness among functions and classes.
- Provide value to all stakeholders.
- Keep the model as simple as possible.

Domain Analysis:

- **Definition:** Identification, analysis, and specification of common, reusable capabilities within a specific application domain.
- **Sources of Knowledge:** Technical literature, existing applications, customer surveys, expert advice, current/future requirements.
- **Outcome:** Class taxonomies, reuse standards, functional and behavioral models, domain languages.

Models Used in Analysis:

1. **Flow-Oriented Modeling:** Indicates how data objects are transformed by processing functions.
2. **Scenario-Based Modeling:** Represents the system from the user's perspective.
3. **Class-Based Modeling:** Defines objects, attributes, and relationships.
4. **Behavioral Modeling:** Depicts states of classes and the impact of events on these states.

Elements of the Analysis Model:

- **Scenario-Based Elements:** High-level system view from the user's or functional perspective.
- **Class-Based Elements:** Static view showing relationships between different parts of the system.
- **Flow-Oriented Elements:** Show how information/data flows throughout the system.
- **Behavioral Elements:** Depict internal system behavior in response to external events.

Analysis Modeling Approaches:

- **Object-Oriented Analysis (UML):** Focuses on defining classes and their collaborations to fulfill customer requirements.
- **Structured Analysis:** Considers data and processes as separate entities, with processes modeled to show input data, transformation, and resulting output data.

System Context Diagram (SCD)

- **Definition:** A graphical system model providing the most abstract view of a system.
- **Purpose:** Describes system boundary, scope, and interactions with external entities.
- **Data Flow:** Illustrates logical flow of data, not physical.
- **Early Project Use:** Employed to establish agreement on project scope, often included in requirements documents.
- **Stakeholder Communication:** Must be written in plain language for easy understanding by all stakeholders.

Elements of SCD:

1. **Entities (Actors):**
    - Represent external entities interacting with the system.
    - Labeled boxes, with one central box representing the system and additional boxes for each external actor.
1. **Relationships:**
    - Labeled lines connecting entities and the system.
1. **Terminator (Entity):**
    - Represents humans, subsystems, or systems interacting with the system.
    - Always connected to the system and shared stores (if any).
    - Named with singular nouns describing their role (e.g., customer, manager).
    - No connections between terminators.

SCD Building Blocks:

- **System (Single Bubble):**
    - Represents the entire system, concealing processing details and internal stores.
    - Labeled with a general name such as the system name, company name, or department name.
- **Data Flow:**
    - Describes the flow of data in and out of the system.
    - Connects the single bubble to terminators and shared stores.
    - Labels indicate data items or collections passing between entities.

Notations:

- **Yourdon and Coad Notation:** A graphical notation used for depicting system context diagrams, emphasizing clarity and simplicity.

System Context Diagrams serve as foundational artifacts in software development, aiding in defining system boundaries, scope, and external interactions. They facilitate stakeholder understanding and agreement on project requirements.

F-Event and ERF

- **F-Event:** Stands for "Flow Event". Not every incoming flow to the system is considered an F-Type event.
- **ERF (Event Related Flow):**

- When two events occur with a relatively short pause between them, the second event is termed an ERF.
- It occurs when the response for the first event cannot be completed unless the second event occurs.
- ERFs are not listed on the event table.

Example:

- **Scenario:**
  - A graduate school application process involves interactions with various entities.
- **Entities Involved:**
  a. Grad School
  b. Student
  c. Employee
  d. Old School
  e. Library
- **Events:**
  a. Application (F-Type Event)
  b. Respond Letter (F-Type Event)
- **Event Related Flows (ERFs):**
  a. Transcript
  b. Reference Letter
  c. List of Publications

Structured Analysis: Behavioral Model

Elements of Structured Analysis and Design:

- **Environmental Model**
- **Behavioral Model**
- **Implementation Model**
- **Essential Model**

Behavioral Model Components:

1. **Data Dictionary**
2. **Data Flow Diagram (DFD)**
3. **Entity Relationship Diagram**
4. **Process Specification**
5. **State Transition Diagram**

Data Flow Diagram (DFD)

- **Introduction:**
  - Proposed by Larry Constantine, the original developer of structured design.
  - Popularized in the 1970s to visualize software system processes.
- **Purpose:**
  - Graphically represents the flow of data through an information system, focusing on process aspects.
- **Usage:**
  - Used as a preliminary step to create an overview of the system.
  - Helps visualize data processing and structured design.
- **Key Features:**

- Illustrates data input, output, sources, destinations, and storage.
- Does not depict process timing or sequencing details.

Data Flow Diagrams serve as valuable tools for visualizing data flow within a system, aiding in understanding system processes and designing system architecture.

Source or Sink Rules in Data Flow Diagrams (DFD)

- **Purpose:** Source or Sink rules govern the connections and interactions within Data Flow Diagrams (DFDs), ensuring consistency and coherence in the representation of data flow.

Rule Summary (DFD):

- **External Entity:**
  - Represents sources or destinations of data outside the system.
  - External entities interact with the system through data flows.
- **Data Flow:**
  - Represents the movement of data between processes, external entities, and data stores.
  - Data flows must originate from a source and end at a sink.
- **Process:**
  - Represents the transformation of data within the system.
  - Processes must consume input data flows and produce output data flows.

DFD Levels:

- **Top-Level DFD (Context Diagram - DFD Level 0):**
  - Shows system boundaries, external entities interacting with the system, and major information flows.
  - Contains a single process (process 0) representing the system as a whole.
- **Level-1 DFD:**
  - Provides a high-level overview of the system's major processes, data flows, and data stores.
  - Expanded from the Context Diagram (DFD Level 0), retaining all connections flowing into and out of level 0.

Practical Exercise:

- DFDs may become large and complex, requiring hierarchical organization.
- **Hierarchical Structure:**
  - Start with the top-level DFD (Context Diagram - DFD Level 0).
  - Expand into lower-level DFDs (e.g., Level-1 DFD) to provide more detailed views of processes, data flows, and data stores.

Source or Sink rules help maintain consistency and clarity in DFDs by defining the roles and relationships of external entities, data flows, and processes within the system.

Creating a Multi-Layer DFD

- **Build the Context Diagram:**
    - Define system boundaries, external entities, and major data flows.
    - Represent the system as a single process (Process 0).
- **Level 0 Diagram:**
    - Use requirements definition and business activity summaries.
    - Illustrate major processes and their interactions.
    - Break down the system into its primary functions.
- **Decompose Level 0 Processes:**
    - Identify processes requiring further elaboration.
    - Create Level 1 DFDs for each decomposed process.
- **Decompose Level 1 Processes:**
    - If necessary, further decompose Level 1 processes into Level 2 DFDs.
    - Continue decomposition iteratively to capture more detailed processes.
- **Balance and Validate:**
    - Ensure completeness and correctness across all generated diagrams.
    - Validate against requirements and business activities.
    - Maintain balance between diagrams, ensuring consistency in representing processes, data flows, and data stores.

Considerations:

- **Identify Data Transformation Processes:**
    - Focus on processes that transform data within the system.
- **Maintain Balance:**
    - Ensure each DFD level provides a balanced representation of the system's processes and data flows.
- **Assess Decomposition Need:**
    - Decompose processes if they involve multiple logical functions or tasks.
- **Follow Naming Conventions:**
    - Use standardized naming conventions for processes, data flows, and data stores.
- **Conceptualize Data Flows and Stores:**
    - Consider data flows as "data in motion" and data stores as "data at rest."
- **Ignore External Entity Data Handling:**
    - Focus on how data flows within the system, rather than how external entities handle data.

DFD Evaluation:

- **Validating Processes:**
    - Ensure each process has a unique name (verb-noun), inputs, outputs, and a clear description.
- **Validating External Entities:**
    - Ensure each external entity has a unique name and at least one input or output data flow.
- **Validating Data Flows:**
    - Ensure each data flow has a unique name, connects to at least one process, and flows in a single direction without crossed lines.
- **Validating Data Stores:**
    - Ensure each data store has a unique name and identification number, with at least one input and output data flow.

Documentation:

- **Formal Definition of Components:**
    - Provide detailed descriptions for each component (process, data flow, data store) on the DFD to ensure clarity and understanding.