Lecture 10

➔ (Week 10 - March 11, 2024)

Actions in State Machines:

- An action is an atomic execution that completes without interruption.
- Four triggers for actions:
    a. **On Entry:** Action executed when entering a state.
    b. **On Exit:** Action executed when exiting a state.
    c. **Do:** Action executed while remaining in a state.
    d. **On Event:** Action triggered by an event.

State Notations:

- States occupy an interval of time.
- Associated with an abstraction of attribute values satisfying certain conditions.
- Entity changes state not only due to current inputs but also based on past input history.

Transitions:

- Denoted by lines with arrowheads from one state to another.
- May have a trigger, guard, and effect.
- Typically occur in response to an event.

Example: State Machine Diagram for a Printer:

- Illustrates transitions between states with triggers, guards, and effects.

Transition Components:

- **Trigger:** Cause of the transition, such as a signal, event, or change in condition.
- **Guard:** Boolean condition that must be true for the trigger to cause the transition.
- **Effect:** Action invoked on the object owning the state due to the transition.

Multiple Effects:

- Transition notation: trigger [guard] / effect
- Multiple effects can occur for a single transition, separated by commas.

Event:

- Significant occurrence with a location in time and space triggering a state transition.
- Generated by various actions such as asking for inputs, producing outputs, method execution, message sending/receiving, abnormal termination, error handling, and exceptions.

State Machine for a Bank Account:

- Define guard conditions to place constraints on transitions.
- Guards make transitions conditional based on certain criteria.
- Example:
    - Deposit funds only allowed for an account with funds.
    - Withdraw funds allowed only if the balance is greater than zero.

Practical Exercise: Traffic Lights Controller

- Normal sequence: Green light for one direction (~25 seconds) -> Yellow light (~5 seconds) -> Red light (at least 30 seconds).
- Draw a state diagram for the traffic lights controller.

Solution:

- State diagram depicting transitions between states representing different light colors.

Self-Transitions:

- Transition from a state back to itself.
- Useful when an effect is associated with the transition.
    - Example: A state representing a process continually looping until a condition is met.

Compound State:

- State machine diagram may include substate diagrams.
- Example: Printer can be On and either Idle or Working.

Concurrent Regions:

- A state may be divided into regions containing substates executing concurrently.
- Allows parallel execution of different processes within the same state.
- Example: Concurrent regions in a printer state machine diagram.

Choice Pseudo-State:

- Represents a dynamic conditional branch.
- Shown as a diamond with one transition arriving and two or more transitions leaving.
- Decides which transition to take based on certain conditions.
- Example: Used in part of a reporting system state machine diagram to determine the next action based on specific conditions.

Entities:

- University
- Department
- Student
- Instructor
- Course

Relationships:

1. **University - Department:**
    - Each University consists of several Departments.
    - Cardinality: One-to-Many (Each University can have many Departments, but each Department belongs to only one University).
1. **Department - Instructor:**
    - Each Department is assigned several Instructors.
    - Cardinality: One-to-Many (Each Department can have many Instructors, but each Instructor belongs to only one Department).
1. **Instructor - Course:**
    - Each Instructor teaches one or more Courses.
    - Cardinality: One-to-Many (Each Instructor can teach many Courses, but each Course is taught by only one Instructor).
1. **University - Student:**
    - Each University consists of many Students.
    - Cardinality: One-to-Many (Each University can have many Students, but each Student belongs to only one University).
1. **Student - Course:**
    - Each university Student may attend one or more Courses.
    - Cardinality: Many-to-Many (Each Student can attend many Courses, and each Course can have many Students).

Example 1:

- A University comprises multiple Departments.
- Each Department has multiple Instructors who teach one or more Courses.
- Students are members of the University, which consists of many Students.
- Each Student may attend one or more Courses, and each Course is taught by a single Instructor.

Business Rules:

- Business rules are constraints that must be adhered to when the system is operational.
- They ensure the integrity and consistency of data and processes within the system.

Enforcing Referential Integrity:

- Referential integrity ensures that every value of one attribute in a table exists as a value of another attribute in a different table.
- It also ensures that every foreign key value exists as a primary key value.
- Database Management Systems (DBMS) enforce referential integrity automatically once primary and foreign keys are identified by the schema designer.

Building ERD (Entity-Relationship Diagram):

1. **Find Entities:**
    - Identify major categories of information, often derived from process models, data flows, and external entities.
    - Check major inputs, outputs, and instances of entities in the system.
1. **Table Definitions:**
    - Define tables/entities based on the identified categories of information.
    - Add attributes relevant to each entity, considering data flows, data requirements, and existing forms/reports.
    - Assign identifiers (primary keys) to each entity.
1. **Identify Relationships:**
    - Determine relationships between entities, describing them with appropriate verb phrases.
    - Determine cardinality and modality based on business rules and stakeholder discussions.
1. **Validating ERD:**
    - Ensure the ERD accurately represents the entities, attributes, relationships, and constraints of the system.
    - Validate against business rules and requirements.

Tips for ERD Creation:

- Identify all relevant entities and their relationships.
- Each entity should appear only once in a diagram.
- Use precise and appropriate names for entities, attributes, and relationships.
- Prefer simple, familiar terms over technical jargon.
- Attribute names should be meaningful, unique, and easily understandable.
- Remove vague, redundant, or unnecessary relationships.
- Avoid connecting relationships to other relationships.

References:

- Jalote, Pankaj. "A Concise Introduction to Software Engineering."
- Bruegge, Bernd, and Allen H. Dutoit. "Object-Oriented Software Engineering Using UML, Patterns, and Java."
- Entity-Relationship Diagrams (ERD) tutorial.

- Relational Database Design with ERD example (School).

Requirement Specifications:

- Once requirements are elicited, modeled, and analyzed, they should be documented clearly and unambiguously.
- A written requirement document is essential for circulation among stakeholders, including clients, user groups, development teams, and testing teams.

Requirements Specification Document (SRS):

- Describes each essential requirement of the system/software and its external interfaces accurately.
- Defines the scope and boundaries of the system/software.
- Each requirement should be feasible and objectively verifiable.
- Serves as the basis for contractual agreements between contractors/suppliers and customers.
- Elaborated from elicitation notes.

Audience and Templates:

- SRS is intended for a diverse audience including customers, users, analysts, developers, testers, and project managers.
- Different levels of detail and formality are needed for each audience.
- Various templates are available, such as IEEE 830.

Goals of a well-designed SRS:

1. Provide feedback to the customer.
2. Decompose the problem into component parts.
3. Serve as input to design specifications.
4. Serve as a product validation check.

IEEE 830-1998 Standard:

- IEEE Recommended Practice for Software Requirements Specifications.
- Describes the content and qualities of a good SRS.
- Aims to help customers accurately describe their requirements and help suppliers understand customer needs.
- Establishes the basis for agreement between customers and suppliers.
- Helps reduce development effort and provides a basis for realistic estimates of costs and schedules.

Structure of SRS (IEEE 830-1998):

1. Introduction: Purpose, scope, definitions, overview.
2. Overall Description: Product perspective, functions, user characteristics, constraints, assumptions.
3. Specific Requirements: External interfaces, functions, performance requirements, database requirements, design constraints, quality attributes, object-oriented models.
4. Appendices.
5. Index.

Characteristics of a good SRS:

- Correct, unambiguous, complete, consistent, ranked by importance, verifiable, modifiable, and traceable.

Templates (IEEE 830-1998):

- Section 3 (Specific Requirements) can be organized in various ways based on different modes, user classes, concepts, features, stimuli, or organizations