|  |
| --- |
| Sister Nivedita University |
| *DG 1/2 New Town, Kolkata – 700156*  *www.snuniv.ac.in* |

A logo with a black text

Description automatically generated with medium confidence

SOFTWARE ENGINEERING Lab manual

School of Engineering

**Department of Computer Science**

**Bachelor of Technology (B. Tech)**

**[ 5th Semester]**

|  |  |
| --- | --- |
| **- - - - - - - - - -- - - - -- - - - - - - - - -- - - - - - - - - - - -**  **Signature**  **Dr. Parthasarathi De**  **Course Instructor**  **Department of Computer Science**  **Sister Nivedita University**  **Kolkata** | **- - - - - - - - - -- - - - -- - - - - - - - - - - - - - - - - - - - -**  **Signature**  **Dr. Anirban Mitra**  **Head of the Department**  **School of Engineering**  **Sister Nivedita University**  **Kolkata** |

**Index**

**1. Introduction**

* 1.1 Overview of Software Engineering Lab
* 1.2 Objectives
* 1.3 Tools Used
* 1.4 Lab Plan

**2. Project Planning & Management**

* 2.1 Project Planning
* 2.2 Gantt Chart
  + 2.2.1 Types of Task Dependencies
  + 2.2.2 Task Constraints
* 2.3 Tracking Progress
  + 2.3.1 Gantt Chart
  + 2.3.2 PERT Chart
* 2.4 PERT Chart
  + 2.4.1 Terminology
* 2.5 Exercises
* 2.6 Deliverables

**3. Software Requirement Specification (SRS)**

* 3.1 Understanding SRS
* 3.2 Requirements Engineering Process
  + 3.2.1 Elicitation
  + 3.2.2 Analysis
  + 3.2.3 Validation
  + 3.2.4 Management
* 3.3 Writing Requirements
* 3.4 Writing Specifications
* 3.5 CASE Tools
* 3.6 Exercises
* 3.7 Deliverables

**4. Software Design: Introduction to UML & Use Case Diagram**

* 4.1 What is UML?
* 4.2 Use Case Diagrams
  + 4.2.1 Actors
  + 4.2.2 Use Cases
  + 4.2.3 Use Case Relationships
* 4.3 CASE Tools
* 4.4 Exercises
* 4.5 Deliverables

**5. Software Design: Class Diagram**

* + 5.1 Objectives
  + 5.2 Outline
  + 5.3 Background
    - 5.3.1 Design Concepts
    - 5.3.2 Visibility
    - 5.3.3 Relationship
    - 5.3.4 Multiplicity
  + 5.4 CASE Tools
  + 5.5 Example
  + 5.6 Deliverables

**6. Software Design: Interaction Diagrams**

* + 6.1 Objectives
  + 6.2 Outline
    - 6.2.1 Interaction Diagrams
      * 6.2.1.1 Sequence Diagrams
      * 6.2.1.2 Collaboration Diagrams
  + 6.3 Background
    - 6.3.1 Analyzing a System’s Behavior
    - 6.3.2 Sequence Diagrams
    - 6.3.3 Creating Sequence Diagrams
    - 6.3.4 Collaboration Diagrams
    - 6.3.5 Notes
  + 6.4 CASE Tools
  + 6.5 Exercises
  + 6.6 Deliverables

**7. Software Design: Activity Diagrams**

* + 7.1 Objectives
  + 7.2 Outline
  + 7.3 Activity Diagrams
    - 7.3.1 Activity Diagram Notation
  + 7.4 CASE Tools
  + 7.5 Exercises
  + 7.6 Deliverables

**8. Software Design: State Transition Diagram**

* + 8.1 Objectives
  + 8.2 Outline
    - 8.2.1 UML State Diagrams
    - 8.2.2 Creating State Transition Diagrams
    - 8.2.3 State Transition Details
  + 8.3 CASE Tools
  + 8.4 Example
  + 8.5 Deliverables

**9. System Modeling**

* + 9.1 Objective
  + 9.2 Outline
    - 9.2.1 System Analysis Model Elements
      * 9.2.2 Data Model: Entity-Relationship Diagram (ERD)
      * 9.2.3 Functional Model: Data Flow Diagram (DFD)
  + 9.3 Background
    - 9.3.1 Why Model Software?
    - 9.3.2 Analysis Model Objectives
    - 9.3.3 The Elements of the Analysis Model
      * 9.3.3.1 Entity Relationship Diagram
      * 9.3.3.2 Data Flow Diagram
  + 9.4 CASE Tools
  + 9.5 Exercises
  + 9.6 Deliverables

1. **Software Testing**

* 10.1 Objectives
* 10.2 Outline
* 10.3 Background
  + 10.3.1 Basic Definitions
  + 10.3.2 Good Test Attributes
  + 10.3.3 Unit Testing
* 10.4 CASE Tools
  + 10.4.1 JUnit Terminology
  + 10.4.2 How JUnit Works
* 10.5 In-Class Demo
* 10.6 Exercises
* 10.7 Deliverables

1. **Appendix**

* Sample 1: Use Case Diagram for Library Management System
* Sample 2: Class Diagram for Online Shopping System
* Sample 3: Sequence Diagram for Online Exam System
* Sample 4: Activity Diagram for ATM Machine System

### Objective of the Lab

In the Software Engineering Lab, the following objectives are to be achieved through the design and execution of a project proposal:

1. **Project Schedule Preparation**: Utilize tools such as MS Project to create a comprehensive project schedule.
2. **Work Breakdown Structure (WBS)**: Organize the project into a structured Work Breakdown Structure (WBS).
3. **Gantt Chart Design**: Develop a Gantt chart based on the project schedule.
4. **PERT Chart Design**: Create a PERT (Program Evaluation and Review Technique) chart and estimate project duration.
5. **Project Management Plan**: Prepare a detailed Project Management Plan using a standard format.
6. **Requirements Specification**: Draft a Software Requirements Specification (SRS) document for the proposed project in a standard format.
7. **UML Diagrams**: Design and illustrate the following UML diagrams:
   * Use Case Diagram
   * Class Diagram
   * Sequence Diagram
   * Activity Diagram
8. **Software Design Document**: Develop a Software Design Document that outlines the architectural and design aspects of the project.
9. **Test Plan Design**: Formulate a Test Plan incorporating both Black Box and White Box testing approaches.

### Introduction to the Lab

The lab is equipped with 36 systems, each featuring the following specifications:

* **Processor**: Pentium 4™ 2.4 GHz
* **RAM**: 1 GB
* **Hard Disk**: 40 GB
* **Mouse**: Optical
* **Network Interface Card**: Present

**Software Environment**:

* **Operating System**: Windows XP / Linux (Fedora 13)
* **Installed Software**:
  + MS Project 2007
  + NetBeans 6.9.1
  + MS Word 2007
  + MS Excel 2007

**Additional Features**:

* **LAN Connections**: Available on client machines
* **Student Access**: Each student is provided with a dedicated system, maintaining a 1:1 ratio. Systems are assigned specific numbers and students use the same system for each lab session.

### 1. Introduction and Project Definition

#### Objectives

* Introduce the lab environment and the tools used in the software engineering lab.
* Understand the concept of a project and learn how to write a project definition.

#### 1.1 Outline

* Familiarization with software projects.
* Introduction to CASE (Computer-Aided Software Engineering) tools.
* Overview of the lab plan and objectives.

#### 1.2 Background

The role of a software engineer involves analyzing business needs, identifying opportunities for improvement, and designing information systems to meet these needs. Mastery of these skills through practice is essential for the successful design and implementation of software systems.

**Introduction**  
In this lab, you will practice various phases of the software development life cycle, including project management, requirements engineering, systems modeling, software design, prototyping, and testing. This will be done using CASE tools in a collaborative environment. UML (Unified Modeling Language) will be used as the modeling language for analysis and design.

**Tools Used in the Lab**

Software engineering requires proficiency with several tools. Some will be introduced during the lab, while others are expected to be known beforehand or learned individually.

* **MS Project**: For project planning and management.
* **NetBeans**: For creating UML diagrams (object-oriented analysis and design).
* **Microsoft Word**: For documenting the Software Requirements Specification (SRS).
* **JUnit**: For software testing.

#### 1.3 Software Engineering Lab Objectives

* Understand and practice the phases of the software life cycle, including project management, requirements engineering, software design, prototyping, and testing.
* Apply these phases to a project using various CASE tools.
* Gain familiarity with UML as a modeling language for analysis and design.

#### 1.4 Lab Plan

| **Lab #** | **Lab Content** | **Deliverables** | **Tool** |
| --- | --- | --- | --- |
| Lab 1 | Project Planning: Gantt Chart Diagram | Diagram | MS Project 2007 |
| Lab 2 | Project Planning: Gantt Chart Diagram | Diagram | MS Project 2007 |
| Lab 3 | Project Planning: PERT Chart Diagram | Diagram | MS Project 2007 |
| Lab 4 | Project Planning: PERT Chart Diagram | Diagram | MS Project 2007 |
| Lab 5 | SRS Design | Plan Document | Microsoft Word |
| Lab 6 | Introduction to UML and Use Case Diagrams | Diagram | NetBeans 6.9.1 |
| Lab 7 | Class Diagrams | Diagram | NetBeans 6.9.1 |
| Lab 8 | Sequence Diagrams | Diagram | NetBeans 6.9.1 |
| Lab 9 | Activity Diagrams | Diagram | NetBeans 6.9.1 |
| Lab 10 | State Transition Diagram | Diagram | NetBeans 6.9.1 |
| Lab 11 | System Modeling (DFD and ERD) | Diagram | Microsoft Word |
| Lab 12 | Software Testing | Testing Report | JUnit |

#### 1.5 Exercises

* Review the concept of software project management using a well-regarded textbook.
* List five potential project titles.
* Select one project from your list and write a project definition for it.

#### 1.6 Deliverables

* Propose a project and write a detailed project definition or problem statement.

### 2. Project Planning and Management

#### Objectives

* Understand project management principles.
* Learn how to prepare project plans.
* Identify and manage project risks.
* Gain proficiency with MS Project as a CASE tool.

#### 1. Outline

* Project planning essentials.

#### 2. Background

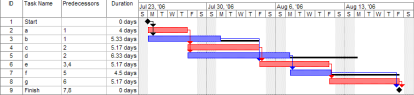
Project management involves planning and controlling the development of a system within a specified timeframe and budget, while ensuring the system meets its intended functionality.

**2.1 Project Work Plan** Create a detailed work breakdown structure (WBS) that includes:

* **Duration of tasks**: Time required to complete each task.
* **Task dependencies**: Relationships between tasks.
* **Task constraints**: Restrictions affecting task scheduling.
* **Milestone dates**: Significant dates marking progress points.
* **Deadline dates**: Final dates for task completion.

**2.2 Tracking Progress**

* **Gantt Chart**:

Fig: Gantt chart of a simple project

* + A bar chart that visually represents a project schedule.
  + Useful for monitoring the status of a project and understanding task timelines and dependencies.
* **PERT Chart**:
  + A flowchart format that illustrates task dependencies and the critical path.
  + Helps in analyzing and representing the tasks involved in completing a project.

**2.3 Gantt Chart**

* Developed by Henry Gantt in 1910, this bar chart displays project schedules, including start and finish dates for tasks.
* Shows dependencies between tasks and highlights the sequence of events.

**2.4 Types of Task Dependencies**

* **Finish-to-Start (FS)**: Task B cannot start until Task A is complete. For example, the user manual (B) can be printed only after it is written (A).

Finish to-start (FS)

* **Start-to-Start (SS)**: Task B cannot start until Task A begins, though they do not need to start simultaneously. For example, score commentary (B) begins when the match (A) starts.

Start-to start (SS)

* **Finish-to-Finish (FF)**: Task B cannot be completed until Task A is completed. For example, the entire book (B) is finished when the last chapter (A) is done.

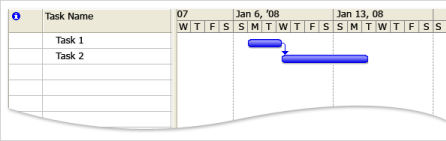
Finish to-finish (FF)

* **Start-to-Finish (SF)**: Task B cannot finish until Task A begins. For example, the previous shift (B) ends when a new shift (A) starts.

Start-to finish (SF)

**2.5 Task Dependencies in MS Project**

* Utilize MS Project to manage and visualize task dependencies and relationships.

****

A screenshot of a computer

Description automatically generated

A screenshot of a calendar

Description automatically generated

A screenshot of a computer

Description automatically generated

**2.6 Task Constraints**

* **Flexible Constraints**: No specific dates; allows tasks to start as early or as late as possible while meeting other constraints.
* **Semi-Flexible Constraints**: Requires associated dates for earliest or latest start/finish dates.
* **Inflexible Constraints**: Requires fixed dates for start or finish, considering external factors like resource availability or deadlines.

**2.7 Types of Constraints**

| **Constraint Type** | **Constraint Name** | **Description** |
| --- | --- | --- |
| **Flexible** | As Late As Possible (ALAP) | Schedules the task to end as late as possible without delaying subsequent tasks. |
| **Flexible** | As Soon As Possible (ASAP) | Schedules the task to start as early as possible. |
| **Semi-Flexible** | Start No Earlier Than (SNET) | Task starts on or after a specified date. |
| **Semi-Flexible** | Finish No Earlier Than (FNET) | Task finishes on or after a specified date. |
| **Semi-Flexible** | Start No Later Than (SNLT) | Task starts on or before a specified date. |
| **Semi-Flexible** | Finish No Later Than (FNLT) | Task finishes on or before a specified date. |
| **Inflexible** | Must Finish On (MFO) | Task must finish on a specified date. |
| **Inflexible** | Must Start On (MSO) | Task must start on a specified date. |

**2.8 Project Scheduling Terminology**

* **Minimum Time (MT)**: The maximum time required across all paths from start to finish.
* **Earliest Start (ES)**: The earliest time a task can start based on preceding tasks.
* **Latest Start (LS)**: Latest time a task can start without delaying the project.
* **Earliest Finish (EF)**: Earliest start time plus the task duration.
* **Latest Finish (LF)**: Latest start time plus the task duration.
* **Slack Time (ST)**: The time a task can be delayed without affecting the project end date.

**2.9 Deadline Date for a Task** Set deadlines to monitor task completion without enforcing inflexible constraints. MS Project will update the schedule and indicate if a task is overdue.

A screenshot of a computer

Description automatically generated

**2.10 Milestone for a Task** Milestones are tasks with zero duration that represent significant events or achievements in the project, such as completing a project phase or meeting a critical deadline.

A screenshot of a calendar

Description automatically generated

**2.11 PERT Chart**

* **PERT Activity**: A task that consumes time and resources.
* **Optimistic Time (O)**: Best-case time scenario for a task.
* **Pessimistic Time (P)**: Worst-case time scenario for a task.
* **Most Likely Time (M)**: Most probable time for a task.
* **Expected Time (E)**: Weighted average time calculated as E=O+4M+P6E = \frac{O + 4M + P}{6}E=6O+4M+P​.
* **Slack Time**: Time a task can be delayed without affecting the project.
* **Critical Path**: The longest path through the project, determining the minimum project duration.

#### 3. CASE Tools

* **Microsoft Project 2007**: Used for project planning, scheduling, and management.

#### 4. Exercises

1. Use MS Project 2007 to create a project plan.
2. Include:
   * Start and end dates for tasks.
   * A comprehensive task list with dependencies.
   * Subtasks and their links.
   * Time estimates for each task.
   * Resource assignments.
   * Detailed task information.

#### 5. Deliverables

Submit the project plan and time schedule created in MS Project 2007, including all tasks, dependencies, constraints, and resources.

### 3. Software Requirement Specification (SRS)

#### Objectives

* Develop a comprehensive understanding of Software Requirement Specification (SRS).
* Learn how to effectively write requirements and specifications for software systems.

#### 1. Outline

* Overview of the requirements engineering process.
* Techniques for writing effective requirements and specifications.
* Creating a Software Requirement Specification (SRS) document.

#### 2. Background

A requirement is a statement that defines a behavior or attribute a system must have to be acceptable to stakeholders. The Software Requirement Specification (SRS) document details what a computer system must accomplish from the user's perspective. It includes:

* **System Behavior**: Input data, required processing, output data, operational scenarios, and interfaces.
* **System Attributes**: Performance, security, maintainability, reliability, availability, safety requirements, and design constraints.

**2.1 Requirements Engineering Process** The process of requirements engineering involves four key phases:

* **Requirements Elicitation**: Gathering detailed requirements from stakeholders.
* **Requirements Analysis**: Evaluating the requirements for consistency, feasibility, and quality.
* **Requirements Validation**: Ensuring the requirements accurately represent what the customer needs.
* **Requirements Management**: Handling changes in requirements during development and identifying any missing or additional requirements.

**2.2 Writing Requirements**

Requirements should be:

* **Correct**: Reflect what is necessary for the system.
* **Unambiguous**: Clearly defined with only one interpretation.
* **Complete**: All necessary functionalities and constraints are included.
* **Consistent**: No conflicts within the requirements.
* **Testable**: There should be a clear method to verify the requirements.

**2.3 Writing Specifications**

Specifications describe the operations and attributes of the system. They can be presented as documents, databases, prototypes, diagrams, or a combination. Specifications differ from requirements in that they are:

* **Complete**: They provide a thorough description of the system as it will be built.
* **Conflict-Free**: They resolve any conflicting requirements.

**2.4 Software Requirement Specification (SRS) Characteristics**

While there is no perfect SRS, it should be:

* **Correct**: Each requirement should accurately represent something required by the system.
* **Unambiguous**: Each requirement must have a single, clear interpretation.
* **Complete**: The SRS should encompass everything the system needs to do.
* **Verifiable**: There should be a way to check that the final system meets the requirements.
* **Consistent**: The document should be free from contradictions.
* **Understandable**: Written in a way that stakeholders can easily comprehend.
* **Modifiable**: Changes should be easy to implement, complete, and consistent.
* **Design Independent**: It should not specify design or algorithm details.
* **Concise**: The SRS should be as brief as possible while still being comprehensive.
* **Organized**: Requirements should be easy to locate, and related requirements grouped together.
* **Traceable**: Each requirement should be easy to reference later.

#### 3. CASE Tools

* **RequisitePro**: A CASE tool that combines the features of a database with Microsoft Word. It integrates with other Rational Suite products for managing requirements effectively.

#### 4. Exercises

Perform the following tasks using RequisitePro:

* Create a new project.
* Set up a new package.
* Add and define requirements within RequisitePro.
* Generate a requirement document.
* Create a new view for managing requirements.

#### 5. Deliverables

* Submit the completed assignments demonstrating proficiency with RequisitePro.
* Ensure you have executed all tasks as outlined.
* Include the requirements documentation created during the requirements phase of your term project.

### 4. Software Design: Introduction to UML & Use Case Diagram

#### Objectives

* Understand the benefits of UML (Unified Modeling Language).
* Learn to create use case diagrams by identifying actors and use cases.
* Practice designing use case diagrams using NetBeans.

#### 1. Outline

* Overview of software design.
* Introduction to UML.
* Using UML for software design.
* Creating and understanding use case diagrams.

#### 2. Background

Software design involves problem-solving and planning to create a solution that meets specified requirements. Once the purpose and specifications of the software are defined, developers or designers create a plan for the solution.

**2.1 What is UML?** Unified Modeling Language (UML) is the standard language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. UML can be applied throughout the software development life cycle and across various implementation technologies.

**2.2 Using UML: Use Case Diagram** A use case diagram is a visual representation of user interactions with a system and outlines the specifications of each use case. It illustrates the different types of users (actors) and their interactions with the system.

**2.3 Actors**

* **Definition**: Actors are entities that interact with the system but are not part of the system itself.
* **Types of Interactions**: Actors can:
  + Input information into the system.
  + Receive information from the system.
  + Both input and receive information.
* **Representation**: Actors are depicted as stick figures in UML diagrams.

**2.4 Use Case**

* **Definition**: A use case represents a sequence of actions performed by a system to achieve a specific goal. It typically represents a major piece of functionality that is complete from start to finish.

**2.5 Use Case Relationships**

* **Actor to Use Case**:
  + **Association/Communication**: Lines connecting actors to use cases show interaction. Arrows indicate the direction of communication.
* **Use Case to Use Case**:
  + **Generalization**: Indicates shared functionality between use cases.
  + **Include**: Represents a required use case that must be performed as part of another use case.
  + **Extend**: Represents optional behavior that occurs under specific conditions.
  + **Dependency**: Indicates that one use case must be completed before another can start.

#### 3. CASE Tools

* **NetBeans**: A comprehensive tool for visual modeling in software development, useful for creating robust, efficient solutions in various environments. NetBeans supports both business process modeling and application logic modeling.

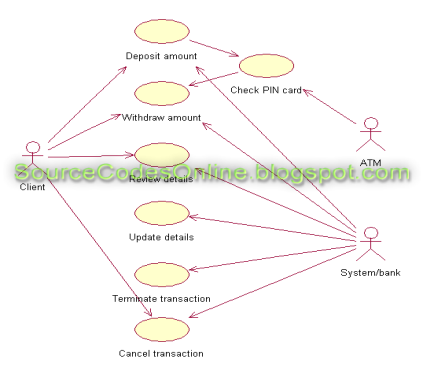
#### 4. Exercises

Read the following problem statement and complete the tasks:

**Problem Statement:** An automated teller machine (ATM) allows customers to withdraw cash at any time from any location without involving a banking clerk. Customers must insert their ATM card and authenticate themselves by entering their Personal Identification Number (PIN). If the PIN is incorrect, no facilities are available. Authenticated customers can change their PIN, deposit cash into their account, and transfer funds to other accounts.

**Tasks:**

1. Identify the **actors** in the system.
2. List the **use cases** for each actor.
3. Determine any **extended use cases** (if applicable).
4. Draw the main **use case diagram** based on the identified actors and use cases.



#### 5. Deliverables

* Submit the completed use case diagrams as per the assignment.
* Utilize NetBeans 6.9.1 to create use case diagrams for your term project.

### 5. Software Design: Class Diagram

#### Objectives

* Understand object-oriented software design principles.
* Get introduced to class diagrams.
* Learn how to identify and model relationships between classes to create UML class diagrams.

#### 1. Outline

* Software design concepts and principles.
* Software architecture.
* Defining attributes, operations, and class relationships.
* Creating UML class diagrams.
* Documenting software design.

#### 2. Background

A class diagram in UML is a type of static structure diagram that represents the structure of a system by showing its classes, attributes, operations (methods), and the relationships between the classes. It provides a static view of the system’s architecture.

**2.1 Design Concepts** In UML class diagrams:

* **Classes** are represented by boxes divided into three sections:
  + **Upper Section**: Contains the class name.
  + **Middle Section**: Lists the attributes of the class.
  + **Lower Section**: Details the methods or operations of the class.

**2.2 Visibility**

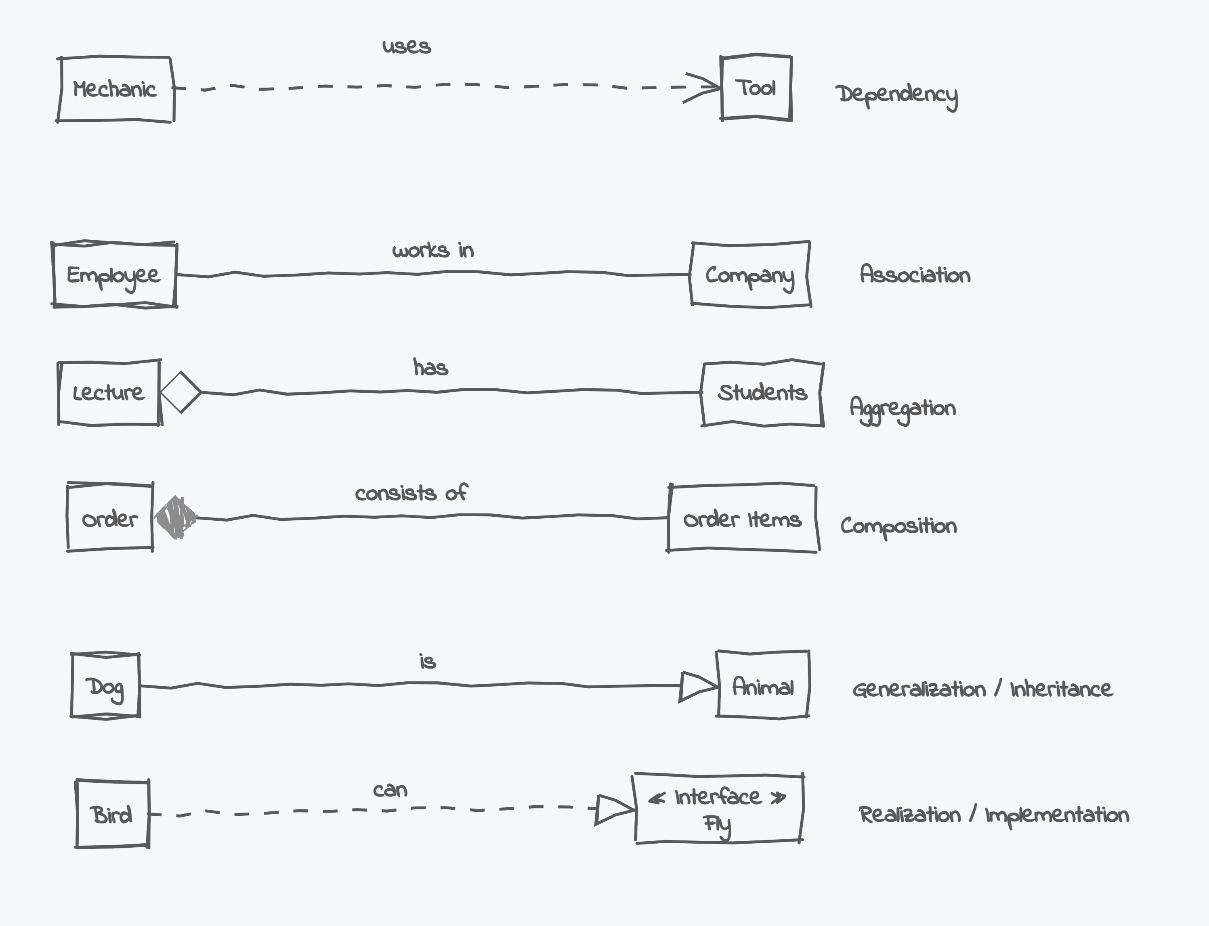
Class members (attributes and methods) are specified with visibility notations:

* **"+" (Public)**: Accessible from any other class.
* **"-" (Private)**: Accessible only within the class.
* **"#" (Protected)**: Accessible within the class and its subclasses.
* **"/" (Derived)**: Calculated from other attributes, not stored.
* **"\_" (Static)**: Belongs to the class rather than instances.

**2.3 Relationships**

UML shows various relationships between classes:

* **Association**: Represents a bidirectional or unidirectional connection between classes.
* **Aggregation**: A special type of association indicating a whole-part relationship.
* **Composition**: A strong form of aggregation where the part cannot exist without the whole.
* **Generalization**: Represents inheritance, where a subclass inherits from a superclass.
* **Dependency**: Indicates that one class depends on another for some function.



**2.4 Multiplicity**

Multiplicity defines how many instances of a class are associated with another class:

* **0..1**: Zero or one instance (optional).
* **1**: Exactly one instance.
* **0..**\* or **\***: Zero or more instances.
* **1..**\*: One or more instances.

#### 3. CASE Tools

* **NetBeans 6.9.1**: Provides tools for creating and managing class diagrams effectively.

#### 4. Example

Refer to the class diagram example for an Online Shopping System to see how classes, attributes, methods, and relationships are represented.

#### 5. Deliverables

* Submit completed class diagrams for the assignment.
* Apply the concepts learned to specify class attributes, methods, and relationships in your term project.

### 6. Software Design: Interaction Diagrams

#### Objectives

* Gain a deeper understanding of interaction diagrams.
* Get familiar with sequence and collaboration diagrams.
* Practice creating interaction diagrams using NetBeans.

#### 1. Outline

* Interaction diagrams: Sequence diagrams and collaboration diagrams.

#### 2. Background

Interaction diagrams model how groups of objects collaborate to perform a behavior. They typically illustrate the behavior of a single use case but do not capture the entire system's behavior, focusing on typical scenarios.

**2.1 Analyzing System Behavior** UML provides two types of interaction diagrams to model system dynamics:

* **Sequence Diagrams**: Show the sequence of message passing between objects over time.
* **Collaboration Diagrams**: Depict the interactions between objects without a time axis.

**2.2 Sequence Diagrams**

* **Purpose**: Illustrate scenarios by showing the sequence of message exchanges.
* **Features**: Indicate object creation and destruction, and whether messages are synchronous or asynchronous.
* **Creation**:
  + Define the scenario to model.
  + Identify classes and objects involved.
  + Arrange objects horizontally with lifelines (dotted vertical lines).
  + Messages are represented by horizontal arrows between lifelines.
  + Use activation lines (narrow rectangles) to show active periods.

**2.3 Collaboration Diagrams**

* **Purpose**: Show interactions between objects with numbered message arrows indicating sequence.
* **Characteristics**: Simpler than sequence diagrams and useful for understanding object communication.

**2.4 Notes**

* Keep diagrams simple.
* For complex "if...then...else" scenarios, draw separate diagrams for each branch and document them accordingly.

#### 3. CASE Tools

* **NetBeans 6.9.1**: Supports the creation of sequence and collaboration diagrams.

#### 4. Exercises

Create a sequence diagram for the process of admitting a patient to a hospital.

#### 5. Deliverables

* Submit completed interaction diagrams for the assignment.
* Use these techniques to create sequence and collaboration diagrams for your term project.

### 

### 7. Software Design: Activity Diagrams

#### Objectives

* Gain a deeper understanding of UML activity diagrams.
* Practice designing flow of events and activity diagrams using NetBeans.

#### 1. Outline

* Overview of activity diagrams.
* Examples and exercises.

#### 2. Activity Diagrams

Activity diagrams are flow charts that represent the workflow of a system. They illustrate the flow of control from one activity to another and show parallel activities and alternate paths.

**2.1 Activity Diagram Notation**

* **Activities**: Represent actions or behaviors in the workflow.
* **Transitions**: Show the flow of control from one activity to another.
* **Decision Points**: Indicate branching based on conditions.
* **Synchronization**: Illustrates concurrent activities and points where activities converge.

#### 3. CASE Tools

* **NetBeans 6.9.1**: Provides tools for creating and managing activity diagrams.

#### 4. Exercises

Design an activity diagram for a Library Management System to issue a book.

#### 5. Deliverables

* Submit the completed activity diagrams for the assignment.
* Apply these techniques to create flow of events and activity diagrams for your term project.

### 8. Software Design: State Transition Diagrams

#### Objectives

* Understand UML state transition diagrams.
* Practice using NetBeans to create state transition diagrams.

#### 1. Outline

* UML state diagrams.
* Notation and details of state diagrams.
* Examples and exercises.

#### 2. Background

State transition diagrams model the behavior of objects with complex states. They show how an object’s state changes in response to events.

**2.1 UML State Diagrams**

* **Purpose**: Analyze how specific objects change states with messages.
* **States**: Represent values describing the object at a given time.
* **Events**: Trigger changes in state.

**2.2 Creating State Transition Diagrams**

* **States**: Represented by rounded rectangles with state names and associated values.
* **Events**: Shown by arrows indicating transitions.
* **State Variables and Operations**: Include entry, do, and exit actions.
* **Special States**: Start (solid black circle) and end states (bull’s eye).

**2.3 State Transition Details**

* **Actions**: Behaviors that occur during transitions.
* **Events**: Messages sent between objects.
* **Guard Conditions**: Boolean expressions that enable transitions.

#### 3. CASE Tools

* **NetBeans 6.9.1**: Supports creation and management of state transition diagrams.

#### 4. Example

Refer to the provided example for a state transition diagram.

#### 5. Deliverables

* Submit completed state transition diagrams for the assignment.
* Use these techniques to create state transition diagrams for your term project.

### 9. System Modeling and Testing Lab Manual

#### 1. System Modeling

**Objective:**

* Gain a deeper understanding of system modeling, focusing on data and functional models.
* Practice drawing Entity-Relationship Diagrams (ERDs) and Data Flow Diagrams (DFDs) using MS Word.

**Outline:**

* System analysis model elements:
  + Data model: Entity-Relationship Diagram (ERD)
  + Functional model: Data Flow Diagram (DFD)

**Background:** Modeling is the process of creating abstractions of reality to simplify complex systems. These abstractions focus on relevant details, ignoring irrelevant ones.

**1.1 Why Model Software?** With the increasing size of software systems, simpler representations are necessary. Modeling helps manage complexity by providing understandable representations of intricate systems.

**1.2 Analysis Model Objectives:**

* Describe customer requirements.
* Provide a basis for software design.
* Define requirements that can be validated post-development.

**1.3 Elements of the Analysis Model:**

* Entity-Relationship Diagram (ERD): Represents objects and their relationships in the data model.
* Data Flow Diagram (DFD): Represents the functional model, focusing on data flow rather than program logic.
* State Transition Diagram: Describes the system's behavior.

**1.3.1 Entity-Relationship Diagram (ERD):** An ERD visually represents data objects and their relationships.

**ERD Notation:**

* **Entity**: An object or concept with data stored about it.
* **Relationship**: Connections between entities showing how they interact.

**Creating an ERD:**

1. Identify objects (nouns) and operations (verbs) from the system's written scope.
2. Define relationships to indicate the connections between objects.

**1.3.2 Data Flow Diagram (DFD):** A DFD represents the functional model, showing how data moves through the system.

**DFD Notation:**

* **External Entity**: Sources or destinations of data.
* **Process**: Functions or operations that transform data.
* **Data Flow**: Movement of data between processes and data stores.
* **Data Store**: Repositories where data is stored.

**Creating a DFD:**

1. Review the ERD to identify data objects and operations.
2. Determine external entities and create a level 0 DFD (Context Diagram) showing a single process.
3. Develop a level 1 DFD to detail processes, maintaining data flow continuity.

**Data Flow Diagram Guidelines:**

* Label all icons with meaningful names.
* Show external entities at level 0.
* Label data flow arrows.
* Avoid procedural logic representation.
* Refine each bubble to perform a single function.

**CASE Tools:**

* Use MS Word 2007 to create ERDs and DFDs if no specialized CASE tools are available.

**Exercises:**

1. Create an ERD for an airline reservation system.
2. Create a DFD for the following:
   * Student Registration System.
   * (a + b) \* (c + a \* d)

**Deliverables:**

* Submit ERD and DFD solutions for the exercises.
* Create ERD and DFD for your term project if applicable.

#### Software Testing

**Objectives:**

* Understand software testing and its documentation.
* Familiarize with JUnit for unit testing.

**Outline:**

* Overview of software testing.
* Unit testing.
* JUnit tutorial.
* Software test specification.

**Background:** Testing aims to execute a program to find errors. A good test case identifies undiscovered errors, while a successful test reveals new errors.

**2.1 Basic Definitions:**

* **Failure**: Unacceptable behavior exhibited by a system.
* **Defect**: Flaw in the system that contributes to failures.
* **Error**: Developer's mistake leading to defects.

**2.2 Good Test Attributes:**

* High probability of finding errors.
* Avoid redundancy.
* Maintain appropriate complexity.

**2.3 Unit Testing:** Focuses on testing individual units (e.g., methods or functions) of a system.

**JUnit:** JUnit is an open-source tool for unit testing in Java. It integrates with Java IDEs and simplifies testing by providing a framework for creating test cases.

**JUnit Terminology:**

* **Unit Test**: Tests a single class.
* **Test Case**: Tests a method with specific inputs.
* **Test Suite**: Collection of test cases.
* **Test Runner**: Runs tests and reports results.
* **Test Fixture**: Sets up necessary data for tests.

**JUnit Basics:**

1. Define a subclass of TestCase.
2. Override setUp() and tearDown() methods.
3. Define public testXXX() methods to exercise the object under test and assert expected results.
4. Optionally define a static suite() method to create a TestSuite.
5. Optionally define main() to run the test case in batch mode.

**In-Class Demo:** Refer to Lab 13 slides for a tutorial on JUnit with examples.

**Exercises:**

* Write a JUnit test class for testing the following method:

public class Exercise {

/\*\* Return the minimum of x and y. \*/

public static int min(int x, int y) { ... }

}

**Deliverables:**

* Submit JUnit test class solutions.
* Utilize JUnit for testing if building a Java project.

#### Appendix

**Sample 1: Use Case Diagram for Library Management System**

**Scenario:** A librarian issues books to students, verifies membership, checks availability, registers new students, and calculates fines upon return. Librarians maintain and update records.

**Use Case Diagram Notations:**

* **Actors**: Users or systems interacting with the system.
* **Use Case**: Major functionalities of the system.
* **Use Case Relationships**: Association, generalization, include, and extend.

**Sample 2: Class Diagram for Online Shopping System**

**Scenario:** An online shopping system allows customers to view products, register, make purchases, and choose payment methods. Administrators manage product descriptions.

**Class Diagram Notations:**

* **Classes**: Represented with boxes containing class name, attributes, and methods.
* **Relationships**: Association, aggregation, composition, generalization, dependency.
* **Multiplicity**: Indicates the number of instances participating in an association.

**Sample 3: Sequence Diagram for Online Exam System**

**Scenario:** Students register and take exams. Examiners contribute questions, check answers, and view profiles. Administrators manage tests, questions, and accounts, and students check results.

**Sequence Diagram Notations:**

* **Objects**: Represented by rectangular boxes.
* **Lifelines**: Vertical dashed lines.
* **Messages**: Horizontal arrows indicating communication between objects.

**Sample 4: Activity Diagram for ATM Machine**

**Scenario:** An ATM allows customers to withdraw cash after card insertion and PIN authentication. Customers can also change their PIN or check balance.

**Activity Diagram Notations:**

* **Actions**: Rounded rectangles.
* **Decisions**: Diamonds.
* **Bars**: Indicate the start or end of concurrent activities.
* **Start/End States**: Black circles represent the start and end of workflows.

**System Diagrams for Each Sample:** Refer to provided diagrams and create visual representations based on the scenarios and notations described.

INDEX

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sl**  **No.** | **Assignment No** | **Name of**  **Assignment** | **Date of**  **Completion** | **Date of**  **Submission** | **Signature** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**N.B. Students should follow this format of the index page in the Lab Note Book.**