K. J. Somaiya College of Engineering, Mumbai-77

(A constituent college of Somaiya Vidyavihar University)

Batch: A2 **Roll No.: 16010121045**

Experiment / assignment / tutorial No. 3

TITLE: Study of Umbrello, a Unified Modelling Language tool

AIM: To understand importance of a common language used by various developers and practitioners for planning and coding.

Expected Course outcome of Experiment:

Learn Umbrello, the tool, used to create diagrams of software and other systems in the industry-standard UML format.

Books/ Journals/ Websites referred:

1.
2.
3.

Pre Lab/ Prior Concepts:

The Unified Modelling Language (UML) is a diagramming language or notation to specify, visualize and document models of Object Oriented software systems. UML is not a development method, that means it does not tell you what to do first and what to do next or how to design your system, but it helps you to visualize your design and communicate with others. UML is controlled by the Object Management Group (OMG) and is the industry standard for graphically describing software.

UML is designed for Object Oriented software design and has limited use for other programming paradigms.

UML is composed of many model elements that represent the different parts of a software system. The UML elements are used to create diagrams, which represent a certain part, or a point of view of the system.

The following types of diagrams are supported by Umbrello UML Modeller:

- Use Case Diagrams show actors (people or other users of the system), use cases (the scenarios when they use the system), and their relationships
- Class Diagrams show classes and the relationships between them

- Sequence Diagrams show objects and a sequence of method calls they make to other objects.
- Collaboration Diagrams show objects and their relationship, putting emphasis on the objects that participate in the message exchange
- State Diagrams show states, state changes and events in an object or a part of the system
- Activity Diagrams show activities and the changes from one activity to another with the events occurring in some part of the system
- Component Diagrams show the high level programming components (such as KParts or Java Beans).
- Deployment Diagrams show the instances of the components and their relationships.
- Entity Relationship Diagrams show data and the relationships and constraints between the data.

The above types of diagrams are drawn by different levels of designers/ coders to understand the product to be developed. They model the system and be used by developers, tester, project managers for various purposes.

The Umbrello software can be downloaded from https://umbrello.kde.org/

Installation steps :https://umbrello.kde.org/installation.php

Post Laboratory Activity:

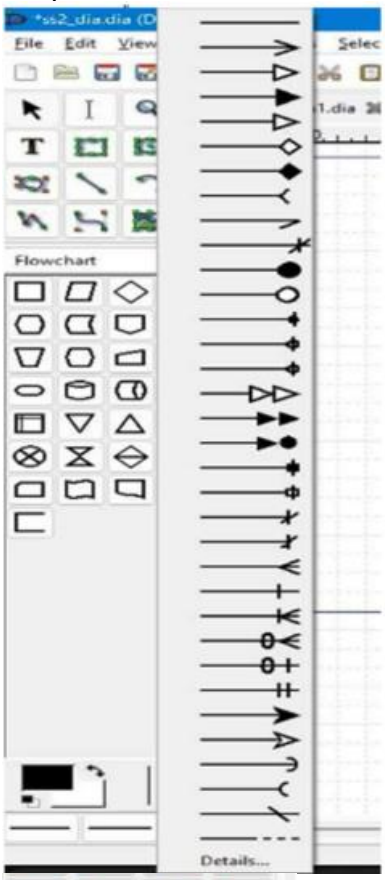
1. Install Umbrello on the desktop
2. Study various options and menus, and get acquainted with the IDE.

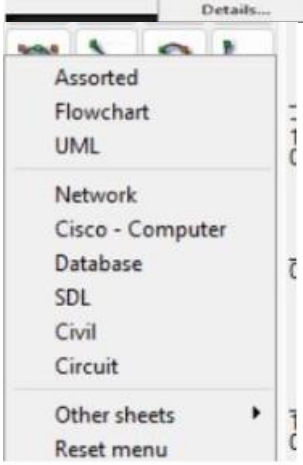
IDE. Using DIA software

Using DIA makes it easy to do many tasks. Several shape packages are available for different needs: flowchart, network diagrams, circuit diagrams , UML sequence diagrams.

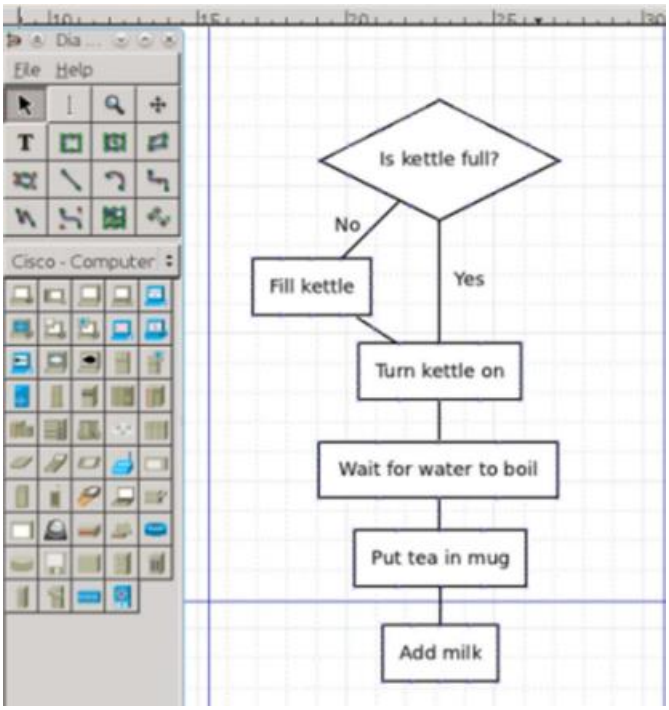
Exploration:

1.Options:

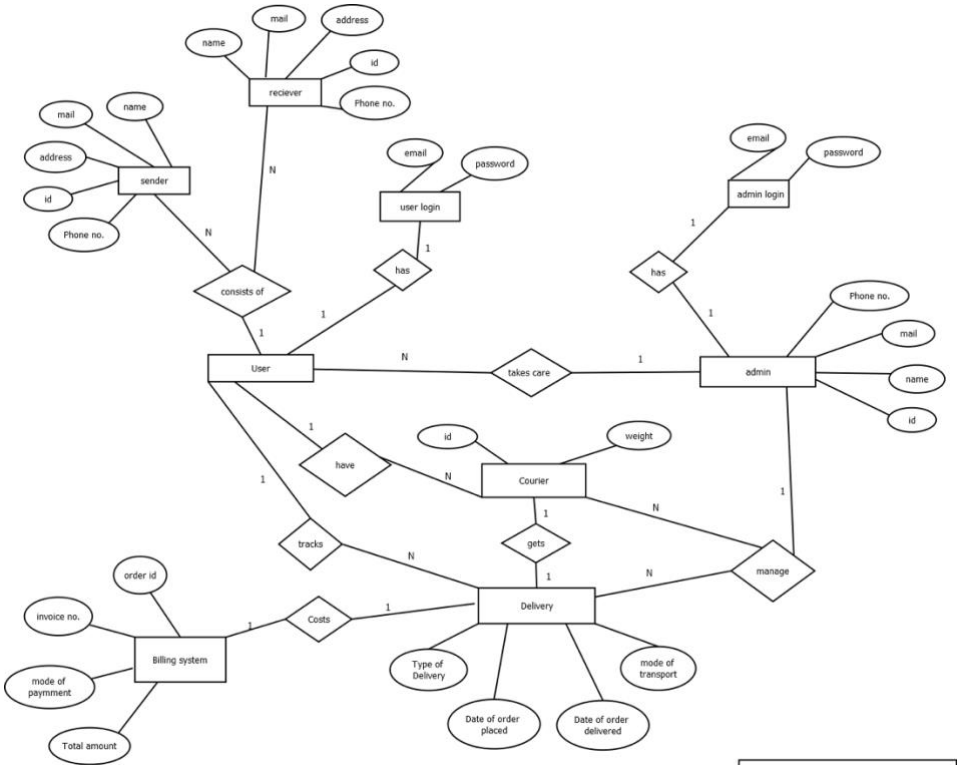




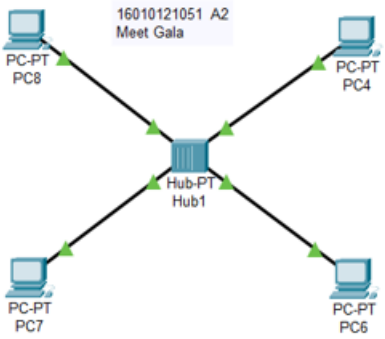
2.We can easily make flowcharts of our algorithms or projects and different types of activity, sequence diagrams , uml class diagrams as it provides all the resources which through drag and drop we can use it easily



3.Entity Relationship models



4.Network diagrams: In terms of understanding networking, instead of downloading different softwares, it can be done and simulated by Dia software at ease.



5.It is also possible to add support for new shapes by writing XML files, using a subset of Scalable Vector Graphics(SVG) to draw the shape. The XML files to save memory , are automatically gzipped and can print large diagram of multiple pages in one shot

Post Lab Descriptive Questions answers must be handwritten and to be submitted BEFORE the next tern.

1. Why do we need to draw different types of diagram during software development process?

Drawing different types of diagrams during the software development process serves several important purposes:

Communication: Diagrams are a visual means of communication that can convey complex ideas and concepts more effectively than text alone. They provide a common language for developers, designers, stakeholders, and clients to discuss and understand the software system.

Visualization: Diagrams help in visualizing the architecture, design, and flow of the software system. They provide a high-level view of the system's structure and behavior, making it easier to spot potential issues or improvements.

Documentation: Diagrams serve as valuable documentation for the software project. They can be used to capture and record design decisions, system requirements, and implementation details, making it easier for future developers to understand and maintain the codebase.

Analysis: Different types of diagrams are used for various purposes, such as UML diagrams for modeling software structure and behavior, flowcharts for illustrating process flows, and data flow diagrams for representing data movement within the system. These diagrams can aid in analyzing the software from different perspectives.

Problem Solving: Diagrams can help in breaking down complex problems into smaller, more manageable parts. For instance, flowcharts can be used to outline the steps involved in a process, helping developers identify bottlenecks or areas for optimization.

Design: Diagrams, such as class diagrams and entity-relationship diagrams, are essential for designing the structure of the software system. They allow developers to define the relationships between different components, classes, or entities.

Testing and Validation: Diagrams can be used to create test cases and scenarios for validating the software's functionality. For example, use case diagrams can help identify and validate user interactions with the system.

Project Management: Diagrams, like Gantt charts and network diagrams, are used for project planning and management. They help in scheduling tasks, allocating resources, and tracking progress.

Architecture: Architectural diagrams provide an overview of the system's components, their interactions, and dependencies. These diagrams are crucial for making architectural decisions and ensuring that the system meets its performance and scalability requirements.

Training and Onboarding: Diagrams can be valuable tools for training new team members or introducing clients to the software system. They provide a visual representation of the system's structure and functionality.

2. List various types of diagrams used in static and dynamic model of a software system.

In software engineering, static and dynamic models are used to represent different aspects of a software system. Various types of diagrams are employed within these models to help analyze, design, and document the system. Here are some common types of diagrams used in both static and dynamic models:

Static Model Diagrams:

Class Diagram: Represents the static structure of the system, showing classes, their attributes, methods, and relationships.

Object Diagram: A specific instance of a class diagram that shows objects and their relationships at a particular point in time.

Package Diagram: Organizes classes and other elements into packages, helping to manage and modularize the system's structure.

Component Diagram: Illustrates the physical components (e.g., libraries, modules) and their relationships within the system.

Deployment Diagram: Describes the physical deployment of software components to hardware nodes, showing how the system is distributed across servers or devices.

Composite Structure Diagram: Focuses on the internal structure of a class, including its parts, ports, and connectors.

Profile Diagram: Extends UML to define custom stereotypes, tagged values, and constraints for modeling domain-specific concepts.

Dynamic Model Diagrams:

Use Case Diagram: Captures interactions between actors (users or external systems) and the system, highlighting the system's functionality from a user's perspective.

Sequence Diagram: Illustrates the time-ordered interactions between objects or components, showing how messages flow between them.

Communication Diagram (formerly Collaboration Diagram): Represents interactions among objects or components, emphasizing the relationships between them and the messages exchanged.

Statechart Diagram: Models the behavior of individual objects or the system as a whole by depicting states, transitions, and events.

Activity Diagram: Describes the workflow or flow of control within the system, representing activities, actions, decisions, and concurrent processes.

Interaction Overview Diagram: Combines activity and sequence diagrams to provide an overview of interactions between objects or components.

Timing Diagram: Depicts the timing constraints and behavior of objects or components concerning time and events.

State Machine Diagram: Similar to statechart diagrams but focuses on the behavior of a single object or class with a finite set of states and transitions.

Global Sequence Diagram: Shows interactions between objects or components across different use cases or scenarios.

Timing Diagram: Represents timing constraints, such as lifelines and lifeline interactions, useful for real-time systems.

Data Flow Diagram (DFD): Illustrates how data flows through a system and how processes transform input data into output data (commonly used in Structured Analysis and Design)