DEWAN VS INSTITUTE OF ENGINEERING & TECHNOLOGY, MEERUT (311)

Department of Computer Science & Engineering

SOFTWARE ENGINEERING LAB (BCS-651)

B TECH VI SEM CSE

LIST OF PROGRAMS

1. Prepare a SRS Document in line with IEEE recommended standard.

2. Draw the case diagram and also specify the role of each of the actors.

3. Draw the activity diagram.

4. Identify & Classify week and strong classes and draw the class diagram.

5. Draw the sequence diagram for any two scenarios.

6. Draw the collaboration diagram.

7. Draw the state chart diagram.

8. Draw the component diagram.

9. Perform forward Engineering and Reverse Engineering in Java.

10. Draw the deployment diagram.

(Mr Babloo Kumar)

PRACTICAL-1

Experiment Name: Prepare a SRS document in line with the IEEE recommended standards.

What is software requirement specification?

A Software Requirements Specification (SRS) is a document that describes the nature of a project, software or application. In simple words, SRS document is a manual of a project provided it is prepared before you kick-start a project/application. This document is also known by the names SRS report, software document. A software document is primarily prepared for a project, software or any kind of application.

Prepare a SRS document in line with the IEEE recommended standards.

1. Introduction

1.1 Purpose
1.2 Document Conventions
1.3 Intended Audience and Reading Suggestions
1.4 Project Scope 1.5 References

2. Overall Description

2.1 Product Perspective
2.2 Product Features
2.3 User Classes and Characteristics
2.4 Operating Environment
2.5 Design and Implementation Constraints
2.6 Assumptions and Dependencies

3. System Features

3.1 Functional Requirements

4. External Interface Requirements

4.1 User Interfaces
4.2 Hardware Interfaces
4.3 Software Interfaces
4.4 Communications Interfaces

5. Non-functional Requirements

5.1 Performance Requirements
5.2 Safety Requirements
5.3 Security Requirements
5.4 Software Quality Attributes

PRACTICAL-2

Experiment Name: Draw the use case diagram and specify the role of each of the actors Also state the precondition, post condition and function of each use case.

What is Usecase diagram?

A use case describes a sequence of actions that provide a measurable value to an actor. A use case is drawn as a horizontal ellipse on a UML use case diagram.

Use Case Diagram objects.

Use case diagrams consist of 4 objects.

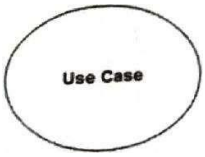
- Actor
- Use case
- System
- Package

1. Actor

An actor is a person, organization, or external system that plays a role in one or more interactions with your system (actors are typically drawn as stick figures on UML Use Case diagrams).

2. Usecase

A use case describes a sequence of actions that provide a measurable value to an actor. A use case is drawn as a horizontal ellipse on a UML use case diagram.

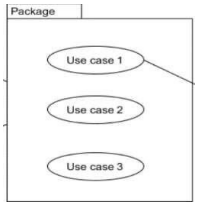


3.System

The system is used to define the scope of the usecase and drawn as a rectangle. This an optional element but useful when you re v1suahzing large systems. For exam le, *you* can create all the use cases and then use the system object to define the scope.

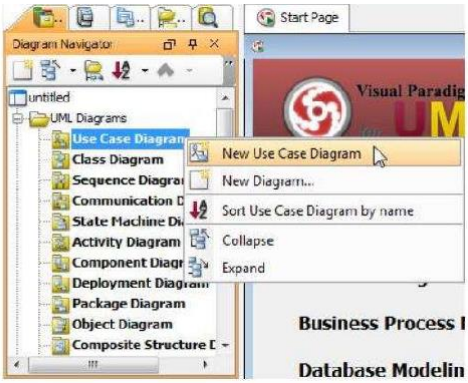
4. Packages

structural diagrams used to show the organization and arrangement of various model elements in the form of packages.

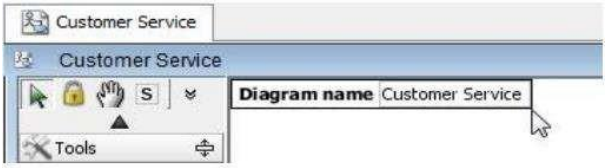


USECASE DIAGRAM:

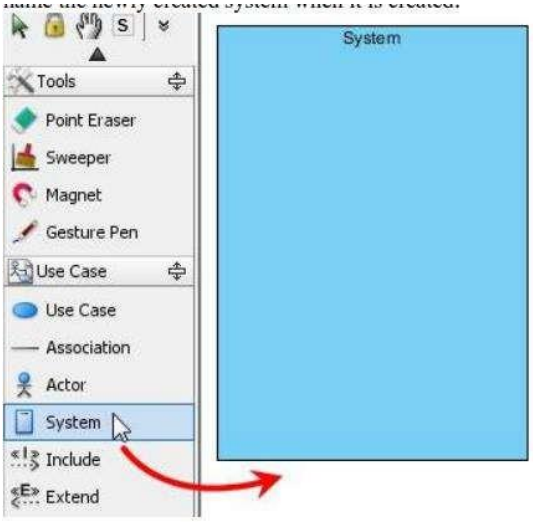
Step 1: Right click Use Case Diagram on Diagram Navigator and select New Use Case Diagram from the pop-up menu.



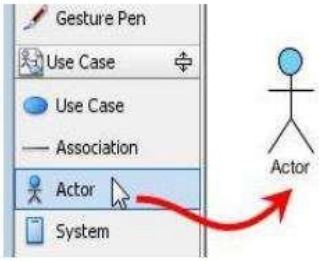
Step 2: Enter name for the newly created use case diagram in the text field of pop-up box on the top left corner.



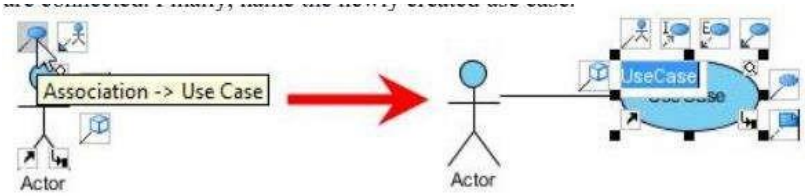
Step 3: Drawing a system To create a system, select System on the diagram toolbar and then click it on the diagram pane. Finally, name the newly created system when it is created.



Step 4: Drawing an actor To draw an actor, select Actor on the diagram toolbar and then click it on the diagram pane. Finally, name the newly created actor when it is created.

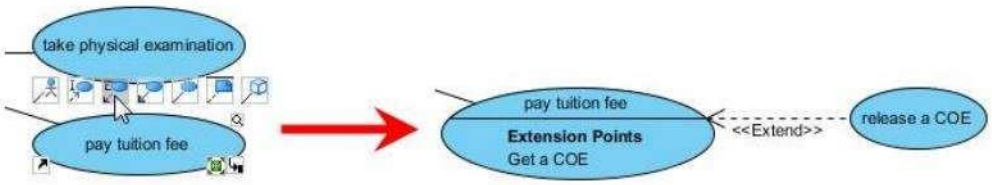


Step 5 :- Drawing a use case Besides creating a use case through diagram toolbar, you can also create it through resource icon. Move the mouse over a shape and press a resource icon that can create use case. Drag it and then release the mouse button until it reaches to your preferred place. The source shape and the newly created use case are connected. Finally, name the newly created use case.

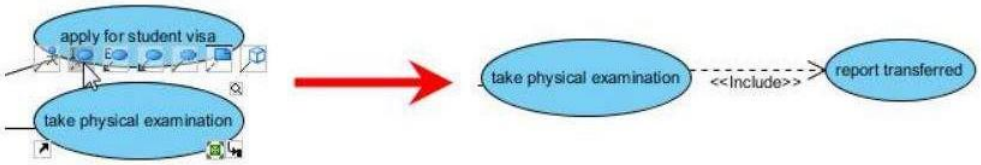


Step 6:- Create a use case through resource icon Line wrapping use case name If a use case is too wide, for a better outlook, you may resize it by dragging the filled selectors. As a result, the name of use case will be line-wrapped automatically.

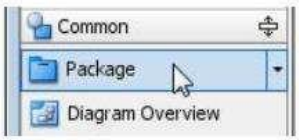
Step 7: Resize a use case To create an extend relationship, move the mouse over a use case and press its resource iconExtend -> Use Case. Drag it to your preferred place and then release the mouse button. The use case with extension points and a newly created use case are connected. After you name the newly created use case, a popup dialog box will ask whether you want the extension point to follow the name of use case. Click Yes if you want it to do so; click NO if you want to enter another name for extension point.



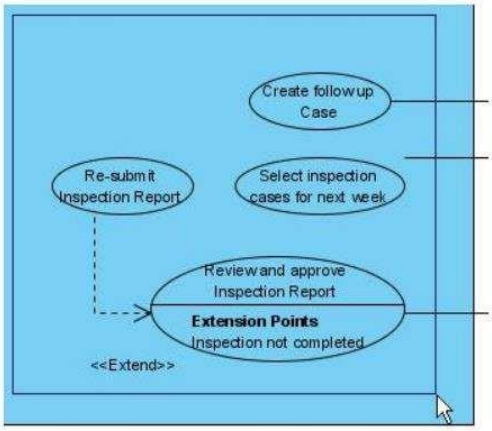
Step 8: Create an extend relationship Drawing <> relationship To create an include relationship, mouse over a use case and press its resource icon Include -> Use Case. Drag it to your preferred place and then release the mouse button. A new use case together with an include relationship is created. Finally, name the newly created use case



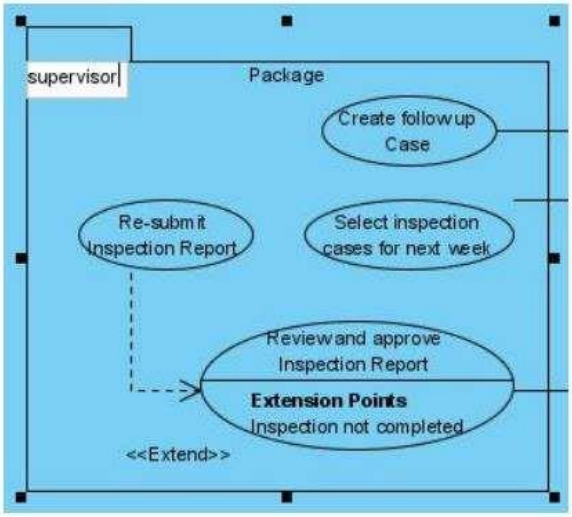
Step 9: Include relationship is created Structuring use cases with package You can organize use cases with package when there are many of them on the diagram. Select Package on the diagram toolbar (under Common category).

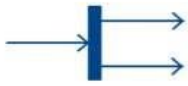


Step 10: Create a package Drag the mouse to create a package surrounding those use cases.



Step 11: Surround use cases with package Finally, name the package.



Symbol	Name	Use
	Start/ Initial Node	Used to represent the starting point or the initial state of an activity
	Activity / Action State	Used to represent the activities of the process
	Action	Used to represent the executable sub-areas of an activity
	Control Flow / Edge	Used to represent the flow of control from one action to the other
	Object Flow / Control Edge	Used to represent the path of objects moving through the activity
	Activity Final Node	Used to mark the end of all control flows within the activity
	Flow Final Node	Used to mark the end of a single control flow
	Decision Node	Used to represent a conditional branch point with one input and multiple outputs
	Merge Node	Used to represent the merging of flows. It has several inputs, but one output.
	Fork	Used to represent a flow that may branch into two or more parallel flows
	Merge	Used to represent a flow that may branch into two or more parallel flows
	Signal Sending	Used to represent the action of sending a signal to an accepting activity
	Signal Receipt	Used to represent that the signal is received
	Note/ Comment	Used to add relevant comments to elements

PRACTICAL-3

Experiment Name: Draw the activity diagram.

Activity diagram:

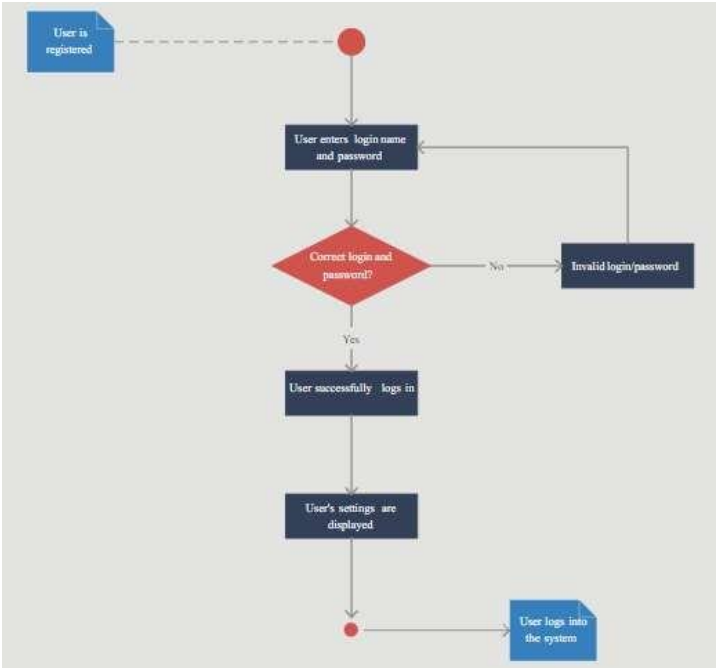
Activity diagrams can be used in all stages of software development and for various purposes. And because they are a lot similar to flowcharts, they are generally more popular than other UML diagram types.

A UML activity diagram helps to visualize a certain use case at a more detailed level. It is a behavioural diagram that illustrates the flow of activities through a system.

Activity Diagram Symbols

UML has specified a set of symbols and rules for drawing activity diagrams. Following are the commonly used activity diagram symbols with explanations

A user login system activity diagram

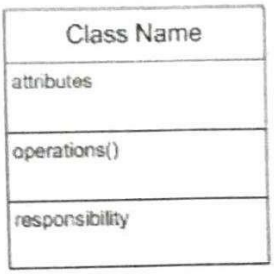


PRACTICAL-4

Experiment Name: Identify the classes. Classify them as weak and strong classes and draw the class diagram.

What is Class diagram?

A class diagram is a visual representation of the classes, their attributes, and their relationships in a software system, providing a static view of its structure.



Class

A description of a group of objects all with similar roles in the system, which consists of:

- **Structural features** (attributes) define what objects of the class "know".
  - Represent the state of an object of the class.
  - Are descriptions of the structural or static features of a class.
- **Behavioural features** (operations) define what objects of the class "can do".
  - Define the way in which objects may interact.
  - Operations are descriptions of behavioural or dynamic features of a class.
  - Operations are descriptions of behavioural or dynamic features of a class.

CLASS DIAGRAM NOTATIONS

1. Class Name: The name of the class, written inside the top section of the box representing the class.

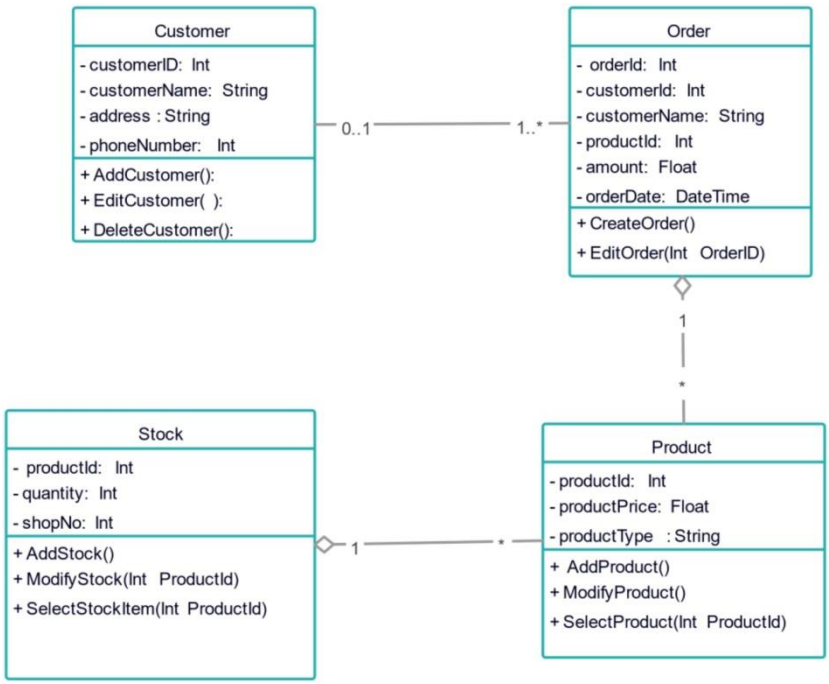
2. Attributes: The attributes or properties of the class, listed in the middle section of the class box, typically in the format "attributeName: attributeType".

3. Methods: The methods or operations of the class, written in the bottom section of the class box, usually in the format "methodName(parameterList): returnType".

4. Visibility Markers: Symbols such as "+" for public, "-" for private, "#" for protected, and "~" for package level, indicating the visibility of attributes and methods.

5. Relationships: Connectors, such as association lines, inheritance arrows, aggregation or composition diamonds, and multiplicity notations, to represent relationships between classes.

Class diagram Ex: Online Order System



PRACTICAL-5

Experiment Name: Draw the sequence diagram for any two scenarios.

What is Sequence diagram?

A sequence diagram in software engineering illustrates the interactions and order of messages between objects or components in a system, representing the dynamic behaviour of a scenario or use case.

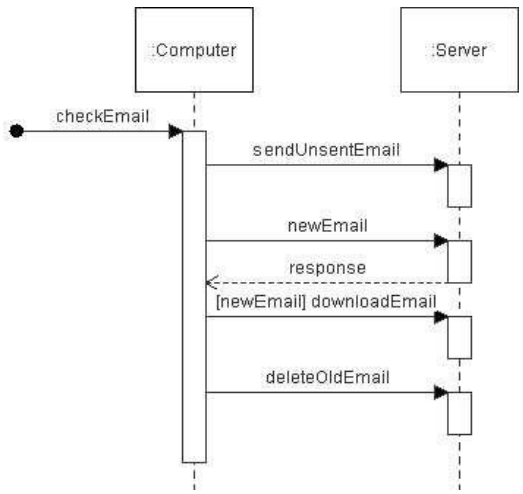
Purpose of Sequence diagram:

Visualizing Dynamic Behaviour: Sequence diagrams allow developers and stakeholders to visualize and understand the dynamic behaviour of a system by illustrating the sequence of interactions and messages exchanged between objects or components.

Analysing System Logic and Communication: Sequence diagrams help in analysing the logic and communication flow within a system. They provide insights into the order of method invocations, the timing of events, and the dependencies between objects, allowing for the identification of potential issues or optimizations.

Designing and Documenting System Architecture: Sequence diagrams are useful for designing and documenting the system architecture. They assist in defining the responsibilities and collaborations between objects, refining the system's design, and communicating the intended behaviour to the development team and other stakeholders.

An example of a UML 2.0 Sequence Diagram showing the sequence of events involved in an email system.



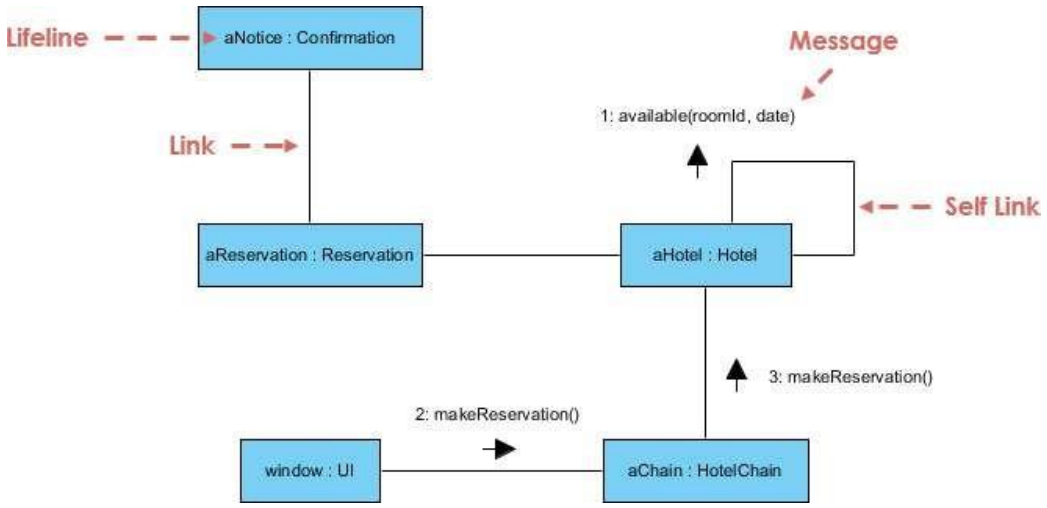
PRACTICAL-6

Experiment Name: Draw the collaboration diagram.

What is Collaboration diagram?

A collaboration diagram, also known as a communication diagram, in software engineering, illustrates the interactions and relationships between objects or components in a system, emphasizing the flow of messages exchanged during a specific scenario or use case. It focuses on the structural organization and communication pathways between objects rather than the chronological order of events like in a sequence diagram.

Diagram example for hotel reservation



PRACTICAL-7

Experiment Name: Draw the State chart diagram.

What is State Chart diagram?

AA State Chart diagram, also known as a State Machine diagram, in software engineering, depicts the various states that an object or system can transition through and the events or conditions that trigger these state changes. It provides a visual representation of the dynamic behavior of a system by illustrating the states, transitions, and actions associated with a particular entity or system component. State Chart diagrams are particularly useful for modeling complex behaviors, such as the lifecycle of an object or the behavior of a software system in response to external stimuli.

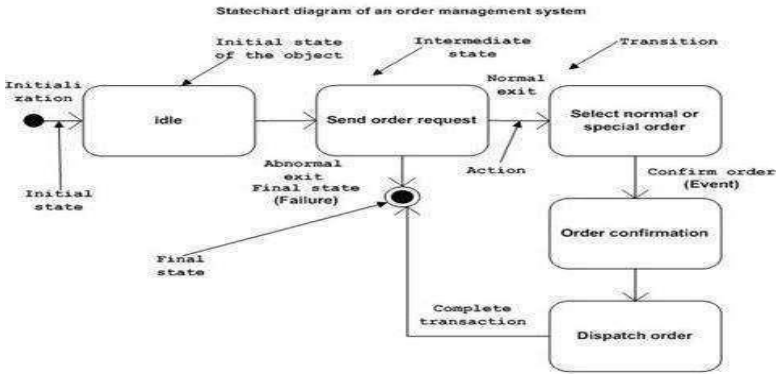
Following are the main purposes of using Statechart diagrams -

- To model the dynamic aspect of a system
- To model the life time of a reactive system.
- To describe different states of an object during its life time.
- Define a state machine to model the states of an object.

The main usage can be described as -

- To model the object states of a system.
- To model the reactive system. Reactive system consists of reactive objects.
- To identify the events responsible for state changes .
- Forward and reverse engineering.

Statechart diagram example:



PRACTICAL-8

Experiment Name: Draw the component diagram.

What is Component Diagram?

Component diagrams are used in modelling the physical aspects of object-oriented systems that are used for visualizing, specifying, and documenting component-based systems and also for constructing executable systems through forward and reverse engineering. Component diagrams are essentially class diagrams that focus on a system's components that often used to model the static implementation view of a system.

Basic Concepts of Component Diagram

A component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. In UML 2, a component is drawn as a rectangle with optional compartments stacked vertically. A high-level, abstracted view of a component in UML 2 can be modelled as:

1. A rectangle with the component's name

2. A rectangle with the component icon

3. A rectangle with the stereotype text and/or icon

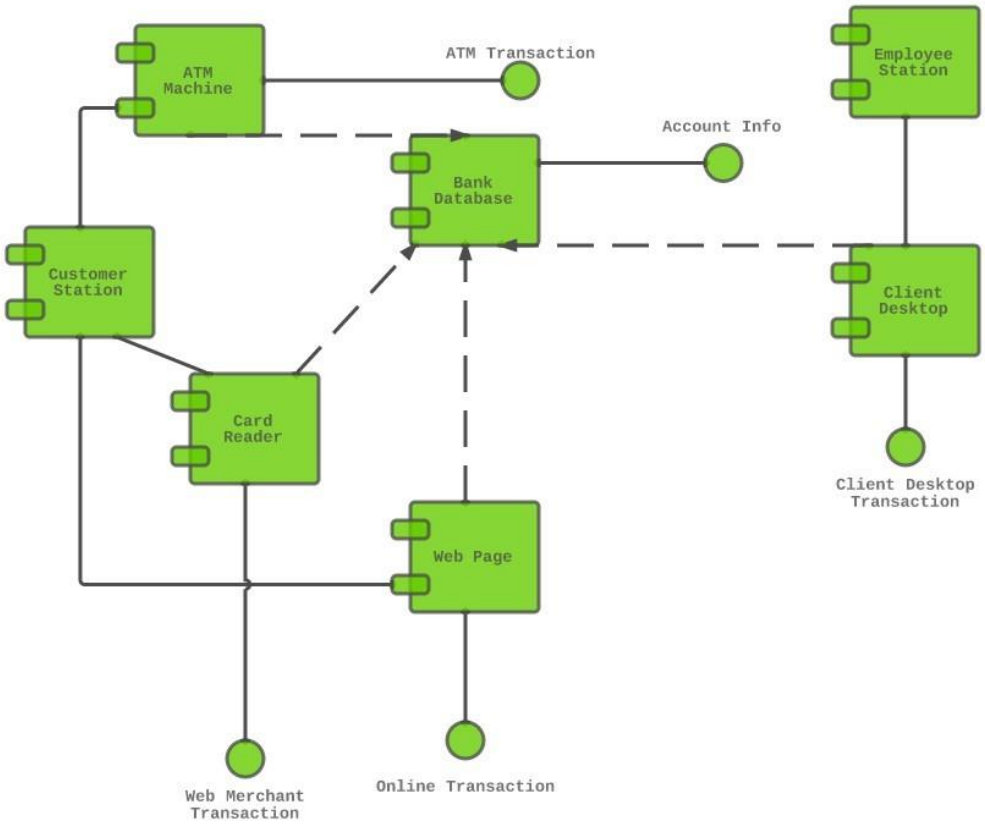
Use of component diagram:

1. System Design and Architecture: Component diagrams help in designing the overall structure of a system or application. They enable architects and designers to identify the major components of the system, their relationships, and how they fit together to form the overall architecture.

2. Component Identification: Component diagrams assist in identifying and defining the individual software components that make up a system. These components represent modular units of the system that can be developed, tested, and maintained independently.

3. Dependency Analysis: Component diagrams illustrate the dependencies and relationships between different components. By visualizing these

Example: Component diagram for an ATM system



PRACTICAL-9

Experiment Name: Perform forward engineering And Reverse Engineering in java.

(Model to code conversion)

Software Forward Engineering:

Forward Engineering is a method of creating or making an application with the help of the given requirements. Forward engineering is the process of building from a high-level model or concept to build in complexities and lower-level details .

Forward engineering is thus related to the term 'reverse engineering, ' where there is an effort to build backward, from a coded set to a model, or to unravel the process of how something was put together.

Forward engineering is the process of building from a high-level model or concept to build in complexities and lower-level details. This type of engineering has different principles in various software and database processes

Goals of Forward engineering:

1. Create new systems or products by designing and implementing them from scratch based on requirements and specifications.

2. Ensure system scalability, reliability, and maintainability through careful architectural planning and design.

3. Improve efficiency and performance by utilizing modern technologies, algorithms, and coding practices.

4. Enable forward compatibility and futureproofing by considering potential changes and advancements in technology.

5. Facilitate effective collaboration and communication among team members by providing clear system design and implementation guidelines.

Software Reverse Engineering:

Software Reverse Engineering is a process of recovering the design, requirement specifications and functions of a product from an analysis of its code. The purpose of reverse engineering is to facilitate the maintenance work by improving the understandability of a system and to produce the necessary documents for a legacy system.

Goals of reverse engineering:

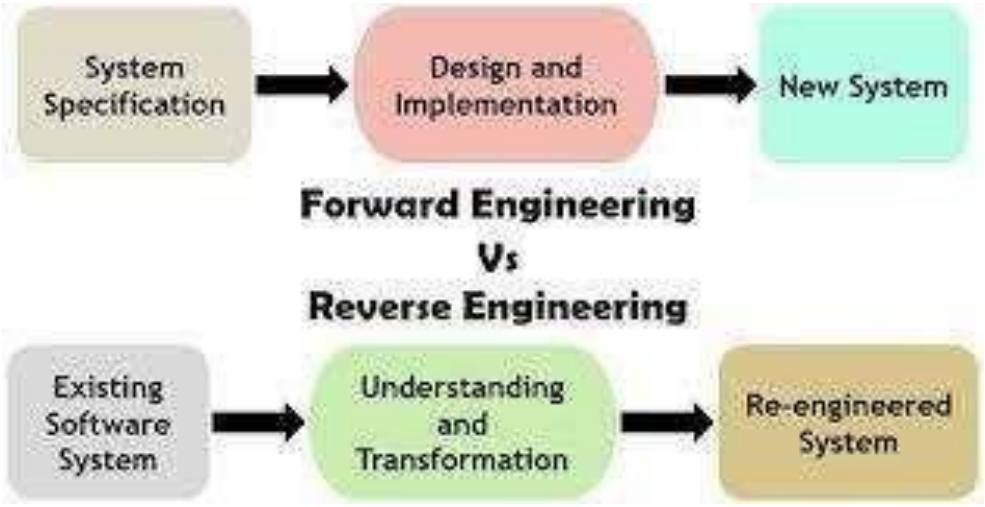
1. Understand the system or product by analysing its structure, behaviour, and functionality.

2. Replicate or recreate a system or product without access to original designs or specifications.

3. Achieve interoperability by reverse engineering closed formats, protocols, or interfaces.

4. Identify weaknesses, inefficiencies, or vulnerabilities for improvement and optimization.

5. Generate accurate documentation for legacy systems or undocumented software to aid in maintenance, troubleshooting, and future development.



PRACTICAL-10

Experiment Name: Draw The deployment Diagram.

(Model to code conversion)

Deployment diagram:

A deployment diagram used to visualize the physical deployment of software artifacts on hardware nodes or environments. It illustrates how software components and artifacts are distributed across different nodes and how they interact with each other.

The main elements of a deployment diagram include:

1. Nodes: Nodes represent physical or virtual computing resources, such as servers, computers, or devices, where software artifacts are deployed. Nodes are depicted as boxes or rectangles.

2. Components: Components represent the software modules, executables, or libraries that are deployed on nodes. They are represented by rectangles with labelled names.

3. Deployment Relationships: Deployment relationships depict the connections and dependencies between components and nodes. They show how components are allocated and executed on specific nodes.

4. Artifacts: Artifacts represent physical files or software packages that are deployed on nodes. They are represented by rectangles with labeled names.

5. Associations: Associations illustrate the communication and interaction between nodes and components. They show the flow of data, requests, or messages between different elements.

Deployment diagrams are commonly used for:

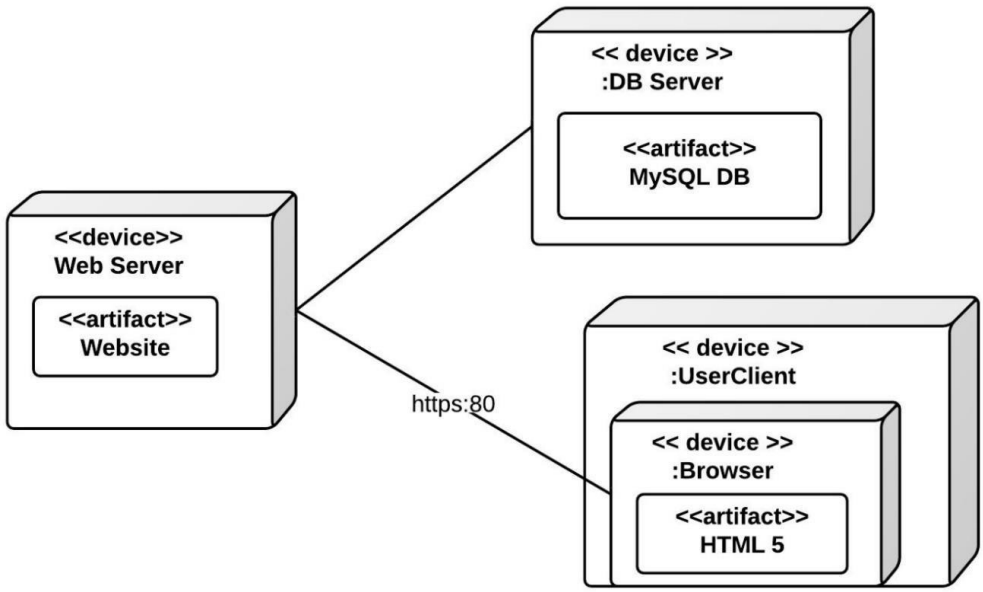
1. System Deployment Planning: They help in planning the deployment and distribution of software components on different nodes or environments.

2. Infrastructure Design: They assist in designing the physical infrastructure required to support the software deployment, such as servers, networks, and storage.

3. System Configuration: They provide guidance for configuring and setting up the software components on the target nodes or environments.

4. System Maintenance: They aid in understanding the deployment configuration, facilitating troubleshooting, and identifying potential performance or scalability issues.

Deployment diagram Example:



This example shows a basic deployment diagram for Lucid chart. There is a web server, a database server, and the machine where the user views the website. You can add more complexity by showing the different parts of the web server and the way Javascript works on the User Client, but this example gives you an idea of how a deployment looks in UML notation.