



Course Code	SOFTWARE ENGINEERING & OOAD LAB		L	T	P	C
21A050411	(Common to CSE, CSE-AI & CSE-IOT)		0	0	3	1.5
Pre-requisite	NIL	Semester	IV			

### COURSE OBJECTIVES:

- To Learn and implement the fundamental concepts of software Engineering.
- To explore functional and non-functional requirements through SRS.
- To practice the various design diagrams through appropriate tool.
- To learn to implement various software testing strategies.

### COURSE OUTCOMES:

After completion of the course, student will be able to

- CO1: Demonstrate the basic concepts of Software Engineering.
- CO2: Identify basic issues in software requirements analysis and specification
- CO3: Apply the structured, object-oriented analysis and design (SA/SD) technique.
- CO4: Design test cases for black box and white box testing.
- CO5: Learn and practice project planning activities.

### CO-PO MAPPING:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
CO1	3	2	1	-	-	-	-	-	-	-	-	-	1	-
CO2	3	1	2	-	-	-	-	-	-	-	-	-	1	-
CO3	3	1	2	-	-	-	-	-	-	-	-	-	1	-
CO4	3	2	2	-	-	-	-	-	-	-	-	-	-	2
CO5	3	2	2	-	-	-	-	-	-	-	-	-	-	2

### SE LAB Experiments List

#### **Week-1**

Draw the Work Breakdown Structure for the system to be automated

#### **Week-2**

Using COCOMO model estimate effort.

#### **Week-3**

- a) Calculate effort using FP oriented estimation model.
- b) Analyze the Risk related to the project and prepare RMMM pla

#### **Week-4**

Develop Time-line chart and project table using PERT or CPM project scheduling methods.



#### Week-5

Draw E-R diagrams, and DFD for the project.  
Design of Test cases based on requirements and design.

#### Week-6

Test a piece of code which executes a specific functionality in the code to be tested and asserts a certain behavior or state using Junit.

#### Week-7

- Test the percentage of code to be tested by unit test using any code coverage tools
- Write C/C++/Java/Python program for classifying the various types of coupling.

#### Week-8

- Write a C/C++/Java/Python program for classifying the various types of cohesion.
- Write a C/C++/Java/Python program for object-oriented metrics for design proposed Chidamber and kremer. (Popularly called as CK metrics)

### **OOAD LAB Experiments List**

#### **Take three case studies:**

- Customer Support System (in the Object-Oriented Analysis & Design with the Unified Process by Satzinger, Jackson & Burd Cengage Learning)
- Point-Of-Sale Terminal (in Larman textbook)
- Library Management System (in the reference book no. 2 i.e., UML toolkit)

#### Week-9

✓ Familiarization with Rational Rose or \*UML

#### Week-10

For each case study:  
a) Identify and analyse events  
b) Identify Use cases

#### Week-11

For each case study:  
a) Develop event table  
b) Identify & analyse domain classes



**Week-12**

For each case study:

- a) Represent use cases and a domain class diagram using Rational Rose
- b) Develop CRUD matrix to represent relationships between use cases and problem domain classes

PBR VISVODAYA

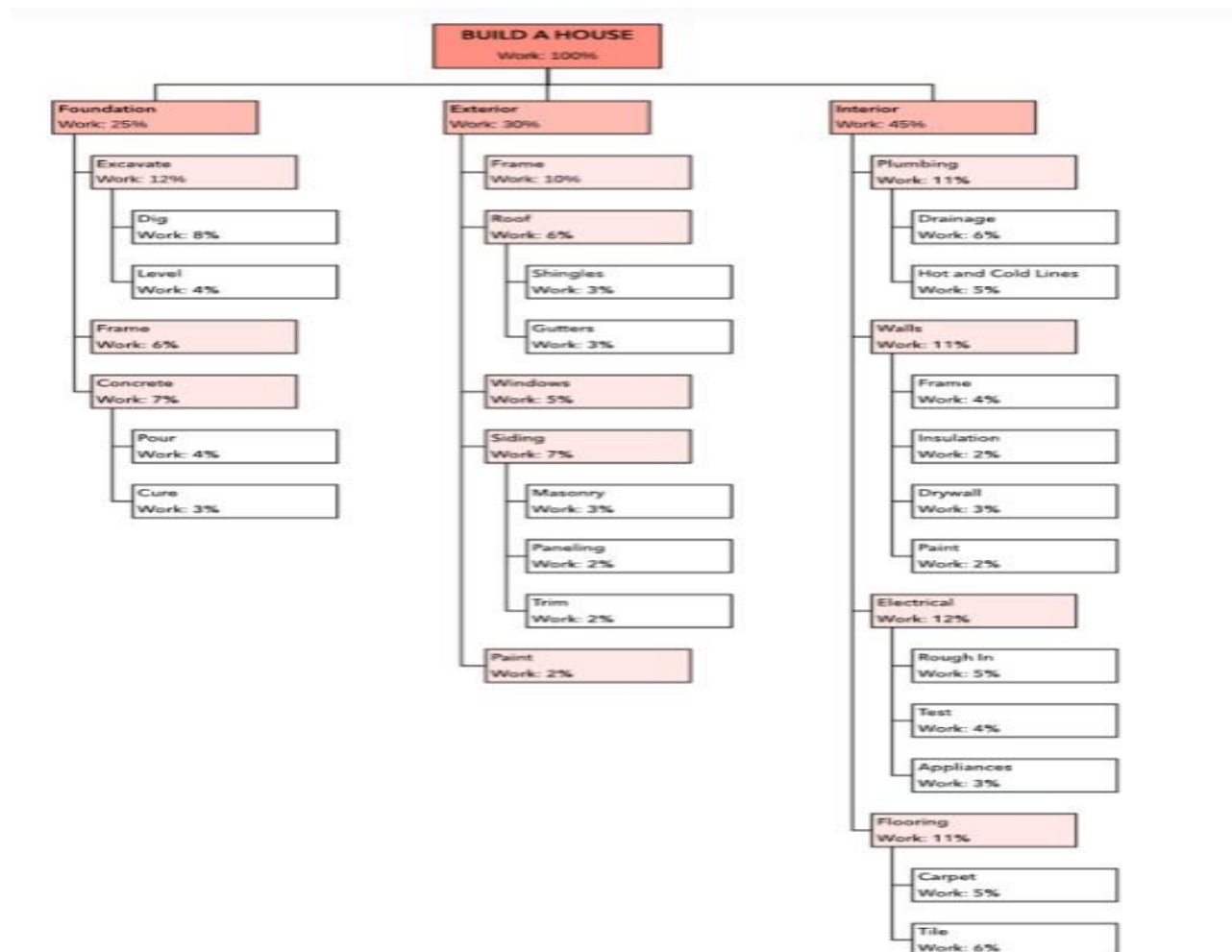
## 1. Draw the Work Breakdown Structure for the system to be automated

### What is a work breakdown structure?

The name is rather self-explanatory. A work breakdown structure starts with a large project or objective and breaks it down into smaller, more manageable pieces that you can reasonably evaluate and assign to teams. Rather than focusing on individual actions that must be taken to accomplish a project, a WBS generally focuses on deliverables or concrete, measurable milestones. These deliverables may also be called work packages, tasks, sub-tasks, or terminal elements. A work breakdown structure looks something like this:

#### Work breakdown structure:

As you are thinking about how to make a work breakdown structure, let's look at an example. This is a work breakdown structure for building a house.



Notice how the rules of building a WBS are applied in this example. First, the house building project is subdivided into three large sections that seem to make sense: foundation, exterior, interior. Those sections are further subdivided to one or two more levels for a maximum of three levels. The effort needed to build a house has been allocated across all of the work packages for a total of 100% effort. There is no duplication of work represented in this diagram. To further enhance this diagram, it would be possible to add the budget for each work package and assign a team.

## **Work breakdown structure formats**

You can choose from several different format options when creating a work breakdown structure. The example above uses a tree format, which is the most visual option. It structures the WBS like an org chart and shows the hierarchy of tasks in addition to providing space for additional information about each work package.

## **Outline structure**

A text outline is the simplest WBS format. It is easy to put together and shows the hierarchy of tasks. However, it is difficult to add additional information about budget, duration, and assignment using this format.

### **Build a House**

- 1** Foundation
  - 1.1** Excavate
    - 1.1.1** Dig
    - 1.1.2** Level
  - 1.2** Frame
  - 1.3** Concrete
    - 1.3.1** Pour
    - 1.3.2** Cure
- 2** Exterior
- 3** Interior

## Hierarchical structure

This format is less visually intuitive but shows the hierarchy of tasks. Because it is a table, this format fits easily onto a page.

```
Build a House
  1 Foundation
    1.1 Excavate
      1.1.1 Dig
      1.1.2 Level
    1.2 Frame
    1.3 Concrete
      1.3.1 Pour
      1.3.2 Cure
  2 Exterior
  3 Interior
```

## Tabular view

A tabular view is a more visually intuitive way to show hierarchy using a table.

Level	WBS Code	Element Name
1	1	Foundations
2	1.1	Excavate
3	1.1.1	Dig
3	1.1.2	Level
2	1.2	Frame
2	1.3	Concrete
3	1.3.1	Pour
3	1.3.2	Cure
1	2	Exterior
1	3	Interior

Level 1	Level 2	Level 3
1 Foundation	1.1 Excavate	1.1.1 Dig 1.1.2 Level
	1.2 Frame	
	1.3 Concrete	1.3.1 Pour 1.3.2 Cure
2 Exterior		
3 Interior		

## Work breakdown structure template

To get you started, here are a number of work breakdown structure templates you can use. Simply click to open the template, and then customize the information, layout, and design.

Level	WBS Code	Element Name	Element Name
1	1	Foundations	All of the work necessary to build a foundation
2	1.1	Excavate	Create a hole ready for the foundation to be framed and poured
3	1.1.1	Dig	Dig a hole of the right shape and size in the correct location
3	1.1.2	Level	Level the hole so that is packed, even, and ready to receive the foundation
2	1.2	Frame	Frame the foundation including steel supports
2	1.3	Concrete	Acquire, transport, pour, and cure the concrete foundation
3	1.3.1	Pour	Pour, pack and level the foundation
3	1.3.2	Cure	All procedures necessary for the foundation to cure successfully
1	2	Exterior	All of the work necessary to complete the exterior of the house
1	3	Interior	All of the work necessary to complete the interior of the house

**RESULT:**

## 2:USING COCOMO MODEL ESTIMATE:

- COCOMO :A heuristic estimation technique constructive cost estimation model was proposed by boehm in 1981.Cocomo prescribes a three stage process for project estimation .In the first stage , an initial estimate is refined to arrive at a more accurate estimate cocomo uses both different stages of estimation.

Boehm postulated that any software development project can be classified into the following three categories based on development complexity.

\*organic

\*semi-detached

\*embedded

The basic cocomo model is a single variable heuristic model that gives an approximate estimation of the project parameters.The basic estimation model is given by the expression of the following forms

$$\rightarrow \text{Effort} = a_1 * (\text{kloc})^{a_2} \text{ PM}$$

$$\rightarrow \text{Tdev} = b_1 * (\text{effort})^{b_2} \text{ months}$$

### ESTIMATION OF DEVELOPMENT EFFORT

$$\text{Organic :effort} = 2.4(\text{kloc})^{1.05}$$

$$\text{Semi-detached:effort} = 2.0(\text{kloc})^{1.12}$$

$$\text{Embedded:effort} = 3.6(\text{kloc})^{1.20}$$

### DEVELOPMENT TIME

$$\text{Organic : tdev} = 2.5(\text{effort})^{0.38}$$

$$\text{Semi-detached : tdev} = 2.5(\text{effort})^{0.35}$$

$$\text{Embedded : tdev} = 2.5(\text{effort})^{0.32}$$

PROBLEM: SUPPOSE A PROJECT WITH ESTIMATED TO BE 400KLOC  
CALCULATE THE EFFORT AND TIMEDEV FOR EACH OF THREE MODELS.

Solution:

i) Organic

$$\text{Effort} = 2.4(400)^{1.05} = 1295.31 \quad 5$$

$$\text{Time} = 2.5(1295.31)^{0.38} = 38.75$$



ii) Semi-Detached

$$\text{Effort} = 3.0(400)^{1.12} = 2462.79$$

$$\text{Time} = 2.15(2462.79)^{0.35} = 38.453$$

iii) Embedded

$$\text{Effort} = 3.6(400)^{1.20} = 4772.81$$

$$\text{Time} = 2.4(4772.81)^{0.32} = 37.59$$

## SOURCE CODE FOR CALCULATING COCOMO MODELS:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int fround(float x)

{

    int a;

    x=x+0.5;

    a=x;

    return(a);

}
void main()

{

    float effort,time,staff,productivity;

    float
table[3][4]={2.4,1.05,2.5,0.38,3.0,1.12,2.5,0.35,3.6,1.20,2.5,0.32};

    int size,model;

    char mode[][15]={ "Organic","Semi-Detached","Embedded"};

    clrscr();

    printf("\nEnter size of project (in KLOC) : ");

    scanf("%d",&size);
```

```

if(size>=2 && size<=50)

    model=0;    //organic

else if(size>50 && size<=300)

    model=1;    //semi-detached

else if(size>300)

    model=2;    //embedded

printf("\nThe mode is %s\n",mode[model]);

effort=table[model][0]*pow(size,table[model][1]);

time=table[model][2]*pow(effort,table[model][3]);

staff=effort/time;

productivity=size/effort;

printf("\nEffort = %f Person-Month",effort);

printf("\n\nDevelopment Time = %f Months",time);

printf("\n\nAverage Staff Required = %d Persons",fround(staff));

printf("\n\nProductivity = %f KLOC/Person-Month",productivity);

getch();

}

O/P:

```

RESULT:

### 3.A) CALCULATE EFFORT USING FP ORIENTED ESTIMATION MODEL

#### FUNCTIONAL POINT[FP]:

- ➔ This was proposed by Albernert in 1983.
- ➔ It is used to overcome the drawbacks of LOC.
- ➔ It can easily be computed from the problem specification itself compared to LOC.

#### FUNCTIONAL POINT METRIC COMPUTATION:

**STEP-1:** Compute the unadjusted functional point(UFP) using a heuristic expression.

$$UFP = (\text{no. inputs}) * 4 + (\text{no. of outputs}) * 5 + (\text{no. of inquiries}) * 4 + (\text{no. of files}) * 10 + (\text{no. of interfaces}) * 10$$

**STEP-2:** Refine UFP to reflect actual complexities of the different parameters use in UFP

TYPE	SIMPLE	AVERAGE	COMPLEX
INPUT(I)	3	4	6
OUTPUT(O)	4	5	7
INQUIRIES(F)	3	4	6
NO.OF.FILES(F)	7	10	15
NO.OF.INTERFACES	5	7	10

**STEP-3:** Compare FP in further refining UFP to account for the specific characteristics of the project that can influence the entire development effort.

$$TCF = (0.65 + 0.01 * DI)$$

$$FP = UFP * TCP$$

**EG:** Compute the functional point, productivity, documentation cost per functional for the following data.

No. of inputs=24,

No. of outputs=46

No. of inquiries=8

No. of files=4

No. of interfaces=2

Effort=36.9 per month

Technical documentation=265 pages

user documentation=122 pages

Cost= \$=4444 per month various processing complexing factors are 4,1,0,3,3,5,4,4,3,3,2,2,4,5.

Sol:- Measure of parameters=(24\*4)=96

External outputs=(46\*5)=230

External files=(4\*10)=40

External inquiries=(8\*4)=32

External interfaces=(2\*7)=14

UFP total=412

Sum of all factors(Fi)=43

Functional point(FP)=(UFP\*TCF)

$$= 412(0.65+0.01*43)$$

$$=412(0.65+0.43)$$

$$=412*1.08$$

$$=444.96$$

Productivity= functional point/effort

$$= 444.96/36.9$$

$$=12.058$$

Total page documentation= Tdev+ user document

$$= 265+122$$

$$=387$$

Documentation= page of documentation/FP

$$= 387/444.96$$

$$=0.869$$

Total cost per function= cost/productivity

$$= 7744/12.058$$

$$=643$$

## B) ANALYZE THE RISK RELATED TO THE PROJECT AND PREPARE RMMM PLAN

A risk management technique is usually seen in the software Project plan. This can be divided into Risk Mitigation, Monitoring, and Management Plan (RMMM). In this plan, all works are done as part of risk analysis. As part of the overall project plan project manager generally uses this RMMM plan.

In some software teams, risk is documented with the help of a Risk Information Sheet (RIS). This RIS is controlled by using a database system for easier management of information i.e creation, priority ordering, searching, and other analysis. After documentation of RMMM and start of a project, risk mitigation and monitoring steps will start.

**Risk Mitigation :**

- It is an activity used to avoid problems (Risk Avoidance).
- Steps for mitigating the risks as follows.

**Finding out the risk.**

1. Removing causes that are the reason for risk creation.
2. Controlling the corresponding documents from time to time.
3. Conducting timely reviews to speed up the work.

**Risk Monitoring :**

- It is an activity used for project tracking.
  - It has the following primary objectives as follows
1. To check if predicted risks occur or not.
  2. To ensure proper application of risk aversion steps defined for risk.
  3. To collect data for future risk analysis.
  4. To allocate what problems are caused by which risks throughout the project.

**Risk Management and planning :**

It assumes that the mitigation activity failed and the risk is a reality. This task is done by Project manager when risk becomes reality and causes severe problems. If the project manager effectively uses project mitigation to remove risks successfully then it is easier to manage the risks. This shows that the response that will be taken for each risk by a manager. The main objective of the risk management plan is the risk register. This risk register describes and focuses on the predicted threats to a software project.

**Example:**

Let us understand RMMM with the help of an example of high staff turnover.

**Risk Mitigation:**

To mitigate this risk, project management must develop a strategy for reducing turnover.

The possible steps to be taken are:

- Meet the current staff to determine causes for turnover (e.g., poor working conditions, low pay, competitive job market).
- Mitigate those causes that are under our control before the project starts.
- Once the project commences, assume turnover will occur and develop techniques to ensure continuity when people leave.

- Organize project teams so that information about each development activity is widely dispersed.
- Define documentation standards and establish mechanisms to ensure that documents are developed in a timely manner.
- Assign a backup staff member for every critical technologist.

#### **Risk Monitoring:**

As the project proceeds, risk monitoring activities commence. The project manager monitors factors that may provide an indication of whether the risk is becoming more or less likely. In the case of high staff turnover, the following factors can be monitored:

General attitude of team members based on project pressures.

1. Interpersonal relationships among team members.
2. Potential problems with compensation and benefits.
3. The availability of jobs within the company and outside it.

#### **Risk Management:**

Risk management and contingency planning assumes that mitigation efforts have failed and that the risk has become a reality. Continuing the example, the project is well underway, and a number of people announce that they will be leaving. If the mitigation strategy has been followed, backup is available, information is documented, and knowledge has been dispersed across the team. In addition, the project manager may temporarily refocus resources (and readjust the project schedule) to those functions that are fully staffed, enabling newcomers who must be added to the team to “get up to the speed“.

**RESULT:**

#### 4. DEVELOP TIME LINE CHART AND PROJECT TABLE USING PERT OR CPM SCHEDULING METHODS:

##### **Critical Path Method:(CPM)**

CPM and PERT are operation research techniques that were developed in the late 1950s. Since then, they have remained extremely popular among project managers. A path in the activity network graph is any set of consecutive nodes and edges in this graph from the starting node to the last node. A critical path consists of a set of dependent tasks that need to be performed in a sequence and which together take the longest time to complete. CPM is an algorithmic approach to determine the critical paths and slack times for tasks not on the critical paths involves calculating the following quantities:

##### **Minimum Time(MT):**

It is the minimum time required to complete the project. It is computed by determining the maximum of all paths from start to finish.

##### **Earliest Start(ES):**

It is the time of a task is the maximum of all paths from the start to this task. The ES for a task is the ES of the previous task plus the duration of the preceding task.

##### **Earliest finish time (EF):**

The EF for a task is the sum of the earliest start time of the task and the duration of the task.

##### **Latest finish (LF):**

LF indicates the latest time by which a task can finish without affecting the final completion time of the project. A task completing beyond its LF would cause project delay. LF of a task can be obtained by subtracting maximum of all paths from this task to finish from MT.

##### **Slack time (ST):**

The slack time (or float time) is the total time that a task may be delayed before it will affect the end time of the project. The slack time indicates the "flexibility" in starting and completion of tasks. ST for a task is  $LS - ES$  and can equivalently be written as  $LF - EF$ .

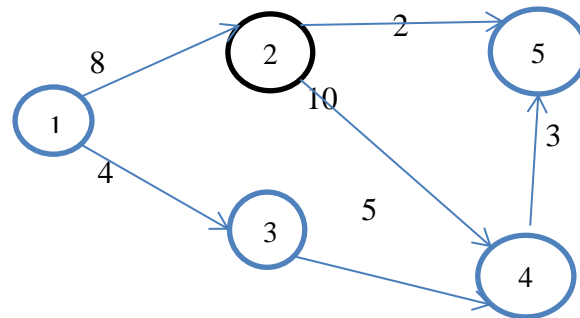
##### **PROBLEM ON CPM :**

Compute the earliest start , earliest finish , latest start and latest finish of each activity of the project given below.

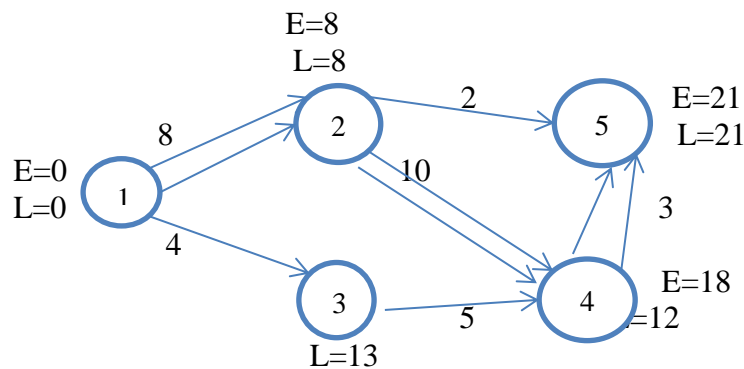
Activity:	1-2	1-3	2-4	2-5	3-4	4-5
Duration	8	4	10	2	5	3

**Solution:**

Now draw the network



Activity	Duration	Earliest start	Earliest Finish	Latest Start	Latest Finish
1-2	8	0	8	0	8
1-3	4	0	4	9	13
2-4	10	8	18	8	18
2-5	2	8	10	10	21
3-4	5	4	9	13	18
4-5	3	18	21	18	21



Therefore, the critical path is 1-2-4-5

## PERT- Project Evaluation and Review Technique:

The activity durations computed using an activity network are only estimated duration. It is therefore not possible to estimate the worst case (pessimistic) and best case (optimistic) estimations using an activity diagram. Since, the actual durations might vary from the estimated durations, the utility of the activity network diagrams are limited.



The CPM can be used to determine the duration of a project but does not provide any indication of the probability of meeting that schedule. Project evaluation and review technique (PERT) charts are a more sophisticated form of activity chart. Project managers know that there is considerable uncertainty about how much time a task would exactly take to complete.

The duration assigned to tasks by the project manager are after all only estimates. In this context, PERT charts can be used to determine the 46 R20 SOFTWARE ENGINEERING probabilistic times for reaching various project mile stones, including the final mile stone. PERT charts like activity networks consist of a network of boxes and arrows. The boxes represent activities and the arrows represent task dependencies.

A PERT chart represents the statistical variations in the project estimates assuming these to be normal distribution. PERT allows for some randomness in task completion times, and therefore provides the capability to determine the probability for achieving project milestones based on the probability of completing each task along the path to that milestone.

Each task is annotated with three estimates:

- Optimistic (O): The best possible case task completion time.
- Most likely estimate (M): Most likely task completion time
- Worst case (W): The worst possible case task completion time

Expected duration of each activity  $T_E = \frac{T_0 + 4T_m + T_p}{6}$

$$\text{Expected variance } \sigma^2 = \frac{(T_p - T_0)^2}{6}$$

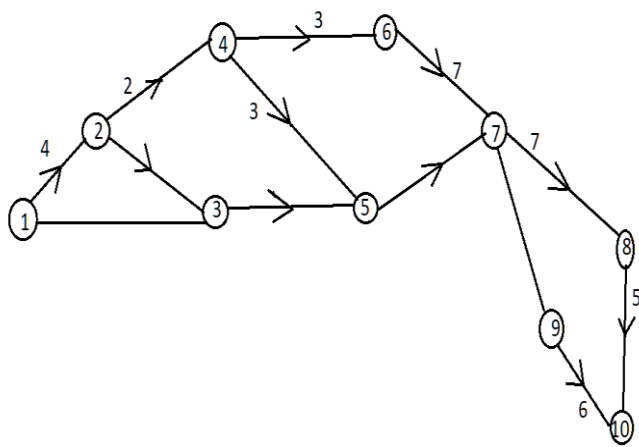
**Expected variance project length = sum of expected variances of all critical activities.**

### Problem on Pert

Construct a network for the project whose activities and three time estimates of these activities are given below

Activity	To	Tm	Tp	$T_E$	$\sigma^2$
1-2	3	4	5	4	0.111
2-3	1	2	3	2	0.111
2-4	2	3	4	3	0.111
3-5	3	4	5	4	0.111
4-5	1	3	5	3	0.444
4-6	3	5	7	5	0.444
5-7	4	5	6	5	0.111
6-7	6	7	8	7	0.111
7-8	2	4	6	4	0.444
7-9	1	2	3	2	0.111
8-10	4	6	8	6	0.444
9-10	3	5	7	5	0.444

Pert chart



RESULT:

## 5. Draw the ER-Diagram and DFD for the following


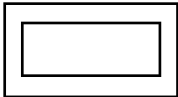
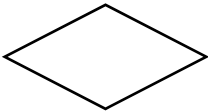
### Entity-Relationship Diagrams


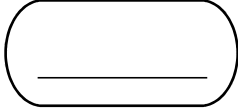
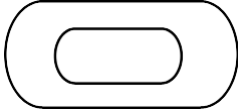

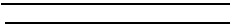
ER-modeling is a data modeling method used in software engineering to produce a conceptual data model of an information system. Diagrams created using this ER- modeling method are called Entity-Relationship Diagrams or ER diagrams or ERDs.

### Purpose of ERD

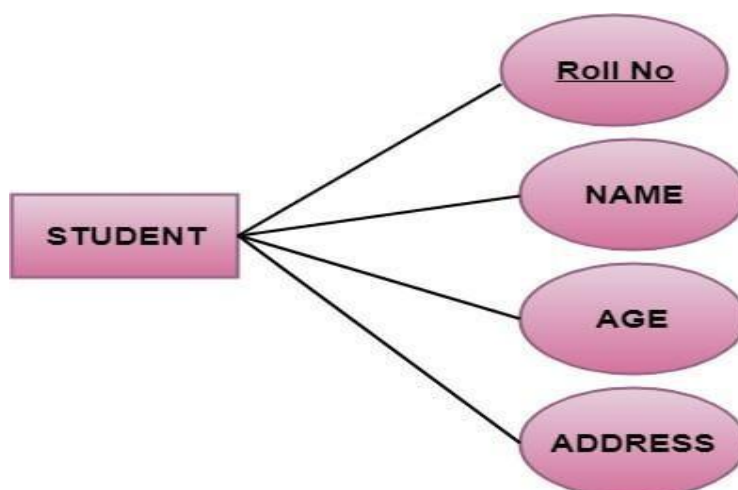
2. The database analyst gains a better understanding of the data to be contained in the database through the step of constructing the ERD.
3. The ERD serves as a documentation tool.
4. Finally, the ERD is used to connect the logical structure of the database to users. In particular, the ERD effectively communicates the logic of the database to users.

### Components of an ER Diagrams:

symbol	name	description
	entity	This is a basic entity that is represented by a rectangle with its name inside
	Weak entity	It inherits the identifiers of its parent entity and often integrated it with partial key
	Strong relationship	A strong relationship is depicted by a single rhombus with its name

	<b>attribute</b>	A basic attribute is represented by a single oval with its name written inside
	<b>Key attribute</b>	This is a special attribute that is used to uniquely identify it is represented as an oval with its name underline
	<b>Multi valued attribute</b>	These are the attributes that can have multiple value and are represented by a double oval
	<b>Partial participation</b>	This depicts that not all the entities in set area part of relationship and its depicted by a single line
	<b>Total participation</b>	This means all the entities in set are in a relationship and are depicted by a double line

Example for ER-diagram



**Draw the data flow diagram for library management system and railway reservation system**

### **Data Flow Diagrams**

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both.


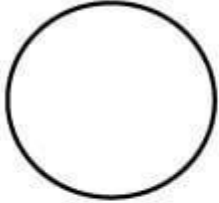
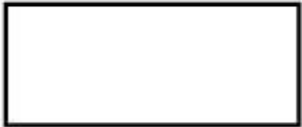
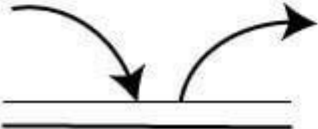
It shows how data enters and leaves the system, what changes the information, and where data is stored.

The objective of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communication tool between a system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called as a data flow graph or bubble chart.

### **The following observations about DFD**

1. Remember that DFD is not a flow chart. Arrows in a flow chart that represents the order of events; arrows in DFD represents flowing data. A DFD does not involve any order of events.
2. Suppress logical decisions. If we ever have the urge to draw a diamond-shaped box in a DFD, suppress that urge! A diamond-shaped box is used in flow charts to represents decision points with multiple exists paths of which the only one is taken. This implies an ordering of events, which makes no sense in a DFD.
3. Do not become bogged down with details. Defer error conditions and error handling until the end of the analysis.

Standard symbols for DFDs are derived from the electric circuit diagram analysis and are shown in fig:

Symbol	Name	Function
	Data flow	Used to Connect Processes to each other, to sources or Sinks; the arrow head indicates direction of data flow.
	Process	Performs Some transformation of Input data to yield output data.
	Source of Sink (External Entity)	A Source of System inputs or Sink of System outputs.
	Data Store	A repository of data; the arrow heads indicate net inputs and net outputs to store.

### Symbols for Data Flow Diagrams

**Circle:** A circle (bubble) shows a process that transforms data inputs into data outputs.

**Data Flow:** A curved line shows the flow of data into or out of a process or data store.

**Data Store:** A set of parallel lines shows a place for the collection of data items. A data store indicates that the data is stored which can be used at a later stage or by the other processes in a different order. The data store can have an element or group of elements.

**Source or Sink:** Source or Sink is an external entity and acts as a source of system inputs or sink of system outputs.

#### Levels in Data Flow Diagrams (DFD)

The DFD may be used to perform a system or software at any level of abstraction. Infact, DFDs may be partitioned into levels that represent information flow and functional detail.

Levels in DFD are numbered 0, 1, 2 or beyond. Here, we will see primarily three levels in the data flow diagram, which are: 0-level DFD, 1-level DFD, and 2-level DFD.

## **0- level DFD**

It is also known as fundamental system model, or context diagram represents the entire software requirement as a single bubble with input and output data denoted by incoming and outgoing arrows. Then the system is decomposed and described as a DFD with multiple bubbles. Parts of the system represented by each of these bubbles are then decomposed and documented as more and more detailed DFDs. This process may be repeated at as many levels as necessary until the program at hand is well understood. It is essential to preserve the number of inputs and outputs between levels, this concept is called leveling by DeMacro. Thus, if bubble "A" has two inputs x1 and x2 and one output y, then the expanded DFD, that represents "A" should have exactly two external inputs and one external output.

## **1- level DFD**

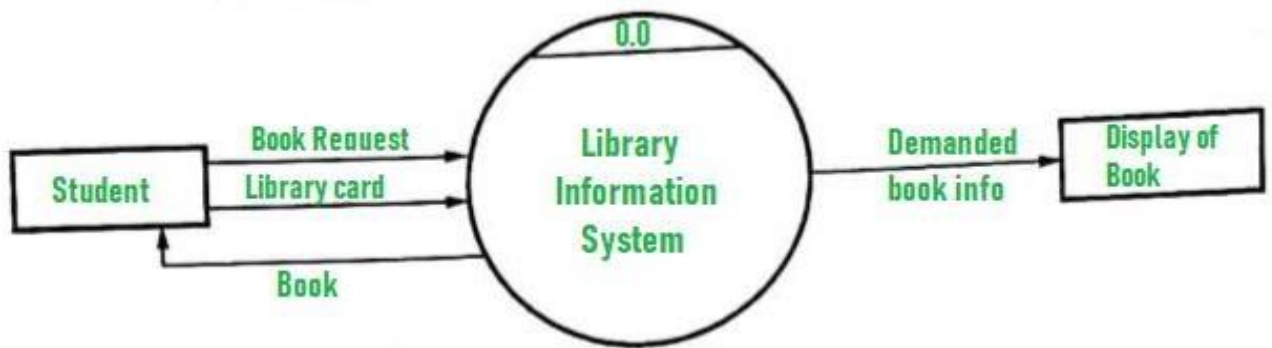
In 1-level DFD, a context diagram is decomposed into multiple bubbles/processes. In this level, we highlight the main objectives of the system and breakdown the high- level process of 0-level DFD into subprocesses.

## **2- Level DFD**

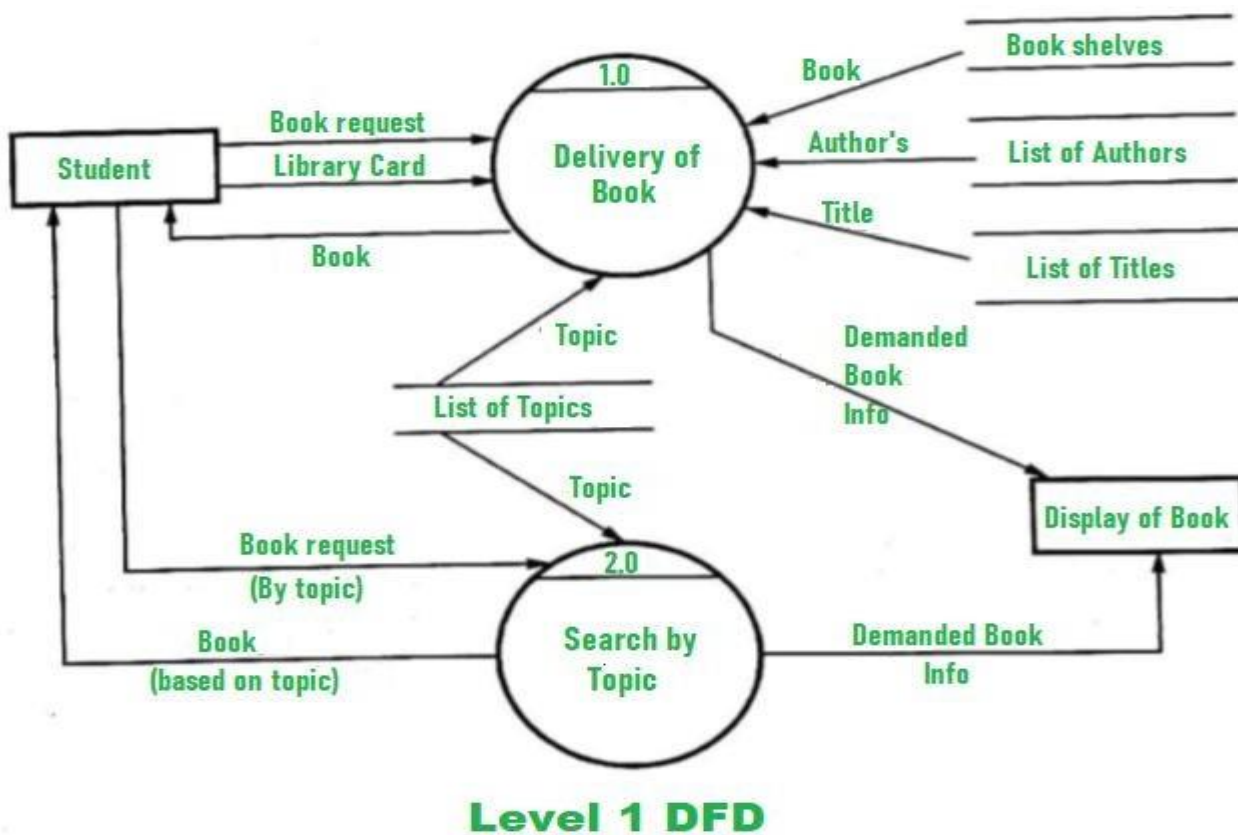
2-level DFD goes one process deeper into parts of 1-level DFD. It can be used to project or record the specific/necessary detail about the system's functioning.

## DFD for library management system

### Level 0 DFD



### level 1 DFD



RESULT:



**6:Test a piece of code which executes a specific functionality in the code to be tested and asserts a certain behavior or state using JUnit.**

JUnit is a Java test framework and an open source project hosted at Github. JUnit 5 (also known as Jupiter) is the latest major release of JUnit. It consists of a number of discrete components:

. JUnit Platform - foundation layer which enables different testing frameworks to be launched on the JVM

- JUnit Jupiter - is the JUnit 5 test framework which is launched by the JUnit Platform
- JUnit Vintage - legacy TestEngine which runs older tests

As the usage of JUnit 5 requires multiple libraries to be present, you typically use it with a build system like Maven or Gradle. JUnit 5 needs at least Java 8 to run.

### **How to define a test in JUnit?**

A JUnit *test* is a method contained in a class which is only used for testing. This is called a *Test class*. To define that a certain method is a test method, annotate it with the `@Test` annotation.

This method executes the code under test. You use an *assert* method, provided by JUnit or another assert framework, to check an expected result versus the actual result. These calls are typically called *asserts* or *assert statements*.

Assert statements typically allow to define messages which are shown if the test fails.

You should provide here meaningful messages to make it easier for the user to identify and fix the problem. This is especially true if someone looks at the problem, who did not write the code under test or the test code.

### **JUnit 5 test :**

The following example demonstrates the usage of a JUnit test. Assume you have the following class which you want to test.

```
package com.vogella.junit5;  
  
public class Calculator {  
63 public int multiply(int a, int b) {  
return a * b;  
}
```

```
}
```

```
}
```

A test class for the above class could look like the following.

```
package com.vogella.junit5;
```

```
import static org.junit.jupiter.api.Assertions.assertEquals;
```

```
import org.junit.jupiter.api.BeforeEach;
```

```
import org.junit.jupiter.api.DisplayName;
```

```
import org.junit.jupiter.api.RepeatedTest;
```

```
import org.junit.jupiter.api.Test;
```

```
public class CalculatorTest {
```

```
private Calculator calculator;
```

```
@BeforeEach
```

```
public void setUp() throws Exception {
```

```
calculator = new Calculator();
```

```
}
```

```
@Test
```

```
@DisplayName("Simple multiplication should work")
```

```
public void testMultiply() {
```

```
assertEquals(20, calculator.multiply(4,5),
```

```
"Regular multiplication should work");
```

```
}
```

```
@RepeatedTest(5)
```

```
@DisplayName("Ensure correct handling of zero")
```

```
public void testMultiplyWithZero() {
```

```
assertEquals(0, calculator.multiply(0,5), "Multiple with zero should be  
zero");
```

```
assertEquals(0, calculator.multiply(5,0), "Multiple with zero should be
```

```
zero");
```

```
}
```

```
}
```

Runs before each test

Defines a test method

Name of the test to be displayed by the test runner

assert statement validating expected and actual value

error message in case the test fails

test which run multiple times, in this example 5 times

### ***5.3. Test class and test methods naming conventions***

There are several potential naming conventions for JUnit tests. A widely-used solution for classes is to use the *Test* suffix at the end of test classes names.

As a general rule, a method name for a name should explain what the test does. If that is done correctly, reading the actual implementation can be avoided.

One possible convention is to use the "should" in the test method name. For example, `ordersShouldBeCreated` or `menu ShouldGetActive`. This gives a hint what should happen if the test method is executed.

Another approach is to use given

`[ExplainYourInput]When[WhatIsDone]Then[ExpectedResult]` for the display name of the test method.

With JUnit5 naming the test method is less important, as you can use the `@DisplayName` annotation to add a description for the test method.

### **5.4. Important JUnit annotations**

JUnit uses annotations to mark methods as test methods and to configure them. The following table gives an overview of the most important annotations in JUnit 5. All these annotations are part of the `org.junit.jupiter.api` package.

Table 1. Annotations	
Annotation	Description
@Test	Identifies a method as a test method.
@Disabled("reason")	Disables a test method with an option reason.
@BeforeEach	Executed before each test. Used to prepare the test environment, e.g., initialize the fields in the test class, configure the environment, etc.
@AfterEach	Executed after each test. Used to cleanup the test environment, e.g., delete temporary data, restore defaults, cleanup expensive memory structures.
@DisplayName("<Name>")	<Name> that will be displayed by the test runner. In contrast to method names the name can contain spaces to improve readability.
@RepeatedTest(<Number>)	Similar to @Test but repeats the test a <Number> of times
@BeforeAll	Annotates a method which is executed once, before the start of all tests. It is used to perform time intensive activities, for example, to connect to a database. Methods marked with this annotation need to be defined as static to work with JUnit.
@AfterAll	Annotates a method which is executed once, after all tests have been finished. It is used to perform clean-up activities, for example, to disconnect from a database. Methods annotated with this

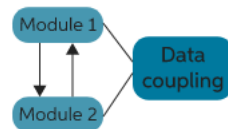
66

Table 1. Annotations	
Annotation	Description
	annotation need to be defined as static to work with JUnit.
@TestFactory	Annotates a method which is a Factory for creating dynamic tests
@Nested	Lets you nest inner test classes to force a certain execution order
@Tag("<TagName>")	Tags a test method, tests in JUnit 5 can be filtered by tag. E.g., run only tests tagged with "fast".
@ExtendWith	Lets you register an Extension class that adds functionality to the tests

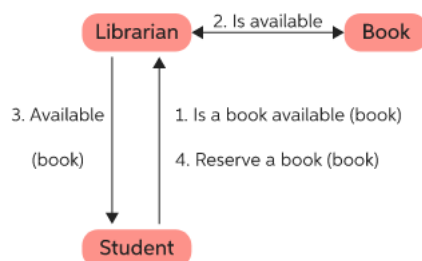
RESULT:

7. Write C/C++/Java/Python program for classifying the various types of coupling.

**Data Coupling:** Data coupling happens when the modules of the program communicate with the other modules by passing or sharing the data. Usually, data is passed through the parameters in the program. It is the best form of coupling.



An example illustrating the object-oriented data coupling is discussed below



Data coupling gets established between the librarian module and the book module. Similarly, the data coupling gets established when the book's availability is informed to the student and when the librarian reserves the book for the student. Let us consider an example program that passes values between functions, thereby establishing a data coupling

DATA COUPLING:

```
main( )
{
    int x, y, z, sum ; //declare variables
    printf ( "\nEnter three numbers: " ) ;
    scanf ( "%d %d %d", &x, &y, &z ) ;
    sum = calculate ( x, y, z ) ; //data is passed through the
    parameters
```

```

printf ( "\nSum = %d", sum ) ;
}
calculate ( d,e,f )
int d,e,f ;49
{
int g ;
g= d +e +f ;
return ( g) ;
}

```

### **Output**

Enter three numbers: 10 20 30  
Sum = 60

### **Control Coupling Definition :**

If two modules are communicating with each other by passing the information about the control flow, then the two modules are known to be in control coupling. Control coupling is one type of coupling method used in software designing system. In control coupling, one module is working on the functions of the modules or takes decision on changing the flow of execution. The second module transfers the control information to the first module and hence forming control coupling.

Overview of Control Coupling:



When the program tries to sort the numbers in ascending or descending order (as per user requirement), then comparison function compares the numbers and raises the flag after sorting is done.

### **#1 Python Program to Sort List in Ascending Order using if statement**

```
NumList = []
Number = int(input("Please enter the Total Number of List Elements: "))
for i in range(1, Number + 1):
    value = int(input("Please enter the Value of %d Element : " %i))
    NumList.append(value)
for i in range (Number):
    for j in range(i + 1, Number):
        if(NumList[i] > NumList[j]):
            temp = NumList[i]
            NumList[i] = NumList[j]
            NumList[j] = temp
print("Element After Sorting List in Ascending Order is : ", NumList)
```

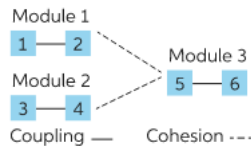
### **#2 Python Program to Sort List in Ascending Order**

```
NumList = []
Number = int(input("Please enter the Total Number of List Elements: "))
for i in range(1, Number + 1):
    value = int(input("Please enter the Value of %d Element : " %i))
    NumList.append(value)
i = 0
while(i < Number):
    j = i + 1
    while(j < Number):
        if(NumList[i] > NumList[j]):
            temp = NumList[i]
            NumList[i] = NumList[j]
            NumList[j] = temp
        j = j + 1
    i = i + 1
print("Element After Sorting List in Ascending Order is : ", NumList)
```

RESULT:

8. Write a C/C++/Java/Python program for classifying the various types of cohesion

Logical Cohesion: A module may be a function, set of functions or a sub-function, which has its own defined functionality, which can be created, modified or removed independently, without impacting the whole system. Logical cohesion is a concept where all modules that are logically doing the same function are put together.



In software engineering, cohesion is the concept of how different modules of software program belong together. It is a type of modular design methodology. A module is said to be logically cohesive when all elements perform similar activities.

Cohesion determines the level of dependency/strength within various functions of a given module. Consider a module 1, which has three set of functions in it- Func 1, Func 2 and Func 3. It is expected that all the functions in a module are strongly related to each other in terms of functionality & dependency.

Hence, for an effective modular design, high cohesion is required.

Consider a real-life example of building/designing a car. Clearly, it is a huge task and should be broken into smaller sub-tasks so that different people may work on different modules. For example, we may break the problem in hand as follows:

- Building the Engine
- Getting Tires
- Buying the vehicle's Body Material (Aluminum etc.)
- Making the doors & windows
- Glasses required for mirrors
  - o Rear-View
  - o Door/Windows
  - o Front & Back
- Seats ...and so on



Now, the task looks easier, and different people can take up each module, and work on it independently, and once all modules are completed, the entire vehicle may be built up by assembling all the body parts and putting them together.

Modules are effective in the sense that the individual modules now exist as a standalone function and may be reused repeatedly. Suppose we write a program which adds two numbers. We may simply put it like this.

```
int main()
{
    int var1, var2;
    cout<< "Enter number1 input";
    cin>>var1;
    cout<< "Enter number2 input";
    cin>>var2;
    cout<<"Sum is :>" <<var1+var2;
}
```

It is a very simple program, and easy to write. Now, consider a scenario where addition of two numbers is to be done 10 times in your program. In such cases, it will be tedious to write the same lines of code ten times. This is where modular design helps. The module will be prepared as shown below:

```
int sumFunc(int var1, int var2)
{
    return var1+var2;
}
int main()
{
    int sum=0;
    cout<< "The program will add 3 seven times!";
    for(int i=0;i<7;i++)
        sum= sumFunc(sum,3); //Reusing the module/function sumFunc
    cout<<"Sum is"<<sum;
}
```

Output:

The program will add 3 seven times!

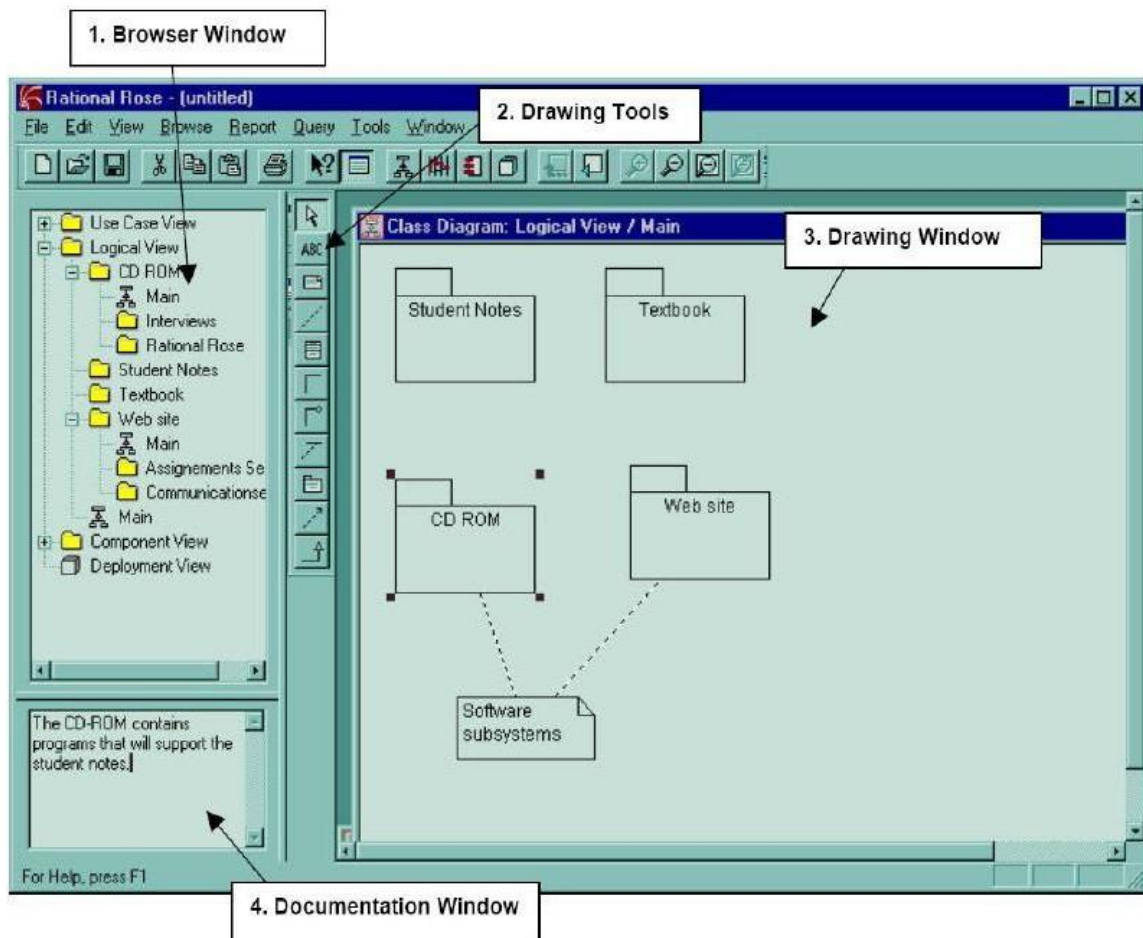
Sum is 21

RESULT:

## OOAD EXPERIMENTS:

### 9. Familiarization with Rational Rose or \*UML

Rational Rose is an object-oriented Unified Modeling Language (UML) software design tool intended for visual modeling and component construction of enterprise-level software applications. Two popular features of Rational Rose are its ability to provide iterative development and round-trip engineering.



#### 1. Browser

Browser contains a list of all the modeling elements in the current model. It contains a tree view of all the elements in the current Rose model. This presents a hierarchical view of the analysis and design model, including all the diagrams and all the individual elements that make up a diagram.

## 2.Documentation Window

Documentation Window may be used to create, review, and modify for any selected modeling element. If nothing is selected, the window displays the string “No selection”. The contents of the Documentation Window will change as different modeling elements are selected. It is strongly recommended that each element added to a diagram have documentation to accompany it. To add documentation, right click on the element, select specification, and fill in the documentation field. The documentation will then be shown in the documentation window each time the mouse is clicked on the element.

## 3.Diagram Windows

Diagram windows allow creating and modifying graphical views of the current model. Each icon on a diagram represents a modeling element. This is the place where the diagram is actually created.

## 4.Drawing Tools

This tool presents a set of icons that indicate the different elements that can be added to a diagram. The elements that can be used will change, depending on the type of diagram being created. Different diagram types have different sets of icons.

Views in Rational Rose There are four views for a model created in Rational Rose, each representing the system from a different point of view.

### 1.Use Case View

The use case view contains the diagrams used in analysis, and all the elements that contain these diagrams. This view looks at actors and use cases along with their interactions. The diagrams in this view are use case diagrams, sequence diagrams and collaboration diagrams. The purpose of the use case view is to visualize what the system must do, without dealing with the specifics of how it will be implemented. More recent versions of Rational Rose also allow for additional documentation in the form of word-processed documents and/or URLs to Web-based materials.

Packages in the use case view can contain actors, use cases, sequence diagrams, and/or collaboration diagrams. To create a package:

- Right-click on the parent modeling element (use case view or another package) to make the shortcut menu visible.

- Select the New: Package menu command. This will add a new package called New Package to the browser.
- While the new package is still selected, enter its name.

Once a package is created, modeling elements may be moved to the package. To move a modeling element to a package:

- Click to select the modeling element to be relocated.
- Drag the modeling element to the package.

## 2. Logical View

The logical view contains the diagrams used in object design. The diagrams in this view are class diagrams and state transition diagrams. It offers a detailed view of how the system visualized in the use case view will be implemented. The basic element in this view is the class, which includes an outline of its attributes and operations. This directly corresponds to a class created in our chosen implementation language. From the logical view, skeletal code can be generated for implementation into a computer language.

More recent versions of Rational Rose not only can generate skeletal code for Visual C++, Visual Java, or Visual BASIC, but also reverse engineer programs created in these languages into Rational Rose models. This allows existing components to be included in documented models, if there is access to the source code. In addition, changes that need to be made during implementation can be reflected in the documentation of the design model.

## 3. Component View

The component view is a step up from the logical view and contains diagrams used in system design. This view contains only component diagram. This includes information about the code libraries, executable programs, runtime libraries, and other software components that comprise the completed systems. Components can be pre-existing; for example, a Windows program in Visual C++ will utilize Microsoft Foundation Class to provide the framework for the Windows interface. Components that do not exist and need to be created by the developers will have to be designed in the logical view.

Rose automatically creates one component diagram called Main. To create an additional component diagram:

- ☐ Right-click on the owning package (either the component view itself or a user created package) to make the shortcut menu visible.
- ☐ Select the New: Component Diagram menu command. This will place a new component diagram called New Diagram in the browser.
- ☐ While the new diagram is still selected, enter its name.

Rose will automatically add the new diagram to the browser.

- ☐ To open a component diagram:

Double-click on the diagram in the browser.

- ☐ To create a component:

- Click to select the package specification icon from the toolbar.
- Click on the component diagram to place the component.
- While the component is still selected, enter its name. Rose will automatically add the new component to the browser.

- ☐ To create a dependency relationship:

- Click to select the dependency icon from the toolbar.
- Click on the package or component representing the client.
- Drag dependency arrow to the package or component representing the supplier.

#### 4. Deployment View

The deployment view illustrates how the completed system will be physically deployed. This view contains only deployment diagram. This view is necessary for complex applications in which a system will have different components located on different machines. For example, interface components may be located on a user machine while other components may be located on a network server.

- ☐ To open the deployment diagram:

Double-click on the Deployment View in the browser.

- ☐ To create a node:

- Click to select the processor icon from the toolbar.
- Click on the deployment diagram to place the node.

- While the node is still selected, enter its name.

☐ To create a connection:

Click to select the connection icon from the toolbar.

Click on the node representing the client.

Drag the connection line to the node representing the supplier.

RESULT:

## **Week : 10 & 11 & 12**

### **CASE STUDY: CUSTOMER SUPPORTING SYSTEM (Online Book store)**

**Aim:** Identify and analyze events

**Procedure:**

Generally event is specification of a significances occurrence that has a location in time & space .The Flow of events of a use case contains the most important information derived from use case modeling work. it should describe the use cases flow of events clearly enough for an outsider to easily understand it. Remember flow of events should present what the system does, not how the system is design to perform the required behavior.

Guidelines for the contents of the flow of events are,

- 1) Describe how the use case starts and ends.
- 2) Describe what data is exchanged between the actor and the use case
- 3) Do not describe the details of the user interface, unless it is necessary to understand the behavior of the system.

**b) Aim: Identify use cases:**

Use case description for online book store: in these we have

**Login:** here we will our credentials and login to the online book store.

**Browser catalog:** here user can browse his required book according to his need such as amount ,author, version etc.....

**Order book:** after selecting the book can order the book on checking the option “buy now”.

**Manage account:** here we will have the option of online payment through our account were it will manage our accountdetails like sending OTPS checking balance ect.

**Ship book:** after the payment of money to the book, the book is shipped to our address in various steps.this is managedby the system.

**Manage catalog:** in this the admin will check and analyze the details of shipping books.



**Search books**

Search books

**Browse catalog**

Browse catalog

**Order book**

Order book

**Manage account**

Manage account

**Ship book**

Ship book

**Manage catalog**

Manage catalog

**c) Aim: Draw Event table****Event table:**

Event	Actor	Description	Object
Login	Customer	Here we will open the web page and by using our credentials we can login to the website of the online book store.	login
Search book	Customer	In search books the users will search books according to their requirements	book
Browser catalog	customer	The user can browser his required book according to their needs such as author, amount ,version	book
Order book	Customer	After selecting the book the customer can order the book on clicking the commands such as “buy new”(or)”add to cart”	book
Manage account	customer	Here we will have the option of cash on delivery (or) online payment .if we choose online payment through our account ,it will manage our account details like sending OTP, checking balance etc.	account
Ship book	system	After the payment of money to the book which the customer had selected, the book is shipped to the customer address in various steps. This is managed by system	books

Manage catalog	Administrator	In managing catalog the administrator will check and analyze the details of shipping books like weather the correct book is being delivered (or)not	books
----------------	---------------	---	-------

**d) Aim:** Identify and analyze domain classes:

#### **Customer**

customer
name:string id:string password:int address:string
Orderbooks()Searchtitle() Fill the orderform()

#### **seller**

Seller
Name: string Address: string Url:int
Check availability()Confirm() Give catalog() Send details() Give order form()

#### **Address**

address
Street: string City:string State:string Country:string Pincode:int

#### **Book details**

Book
Title: string Author: string Publisher : string

#### **Mode of payment**

Mode of payment
Creditcardno:int Checkno:int
Pay by cash() Pay through money order()

### LOGIn

login
Customer:string Ip:address

### LOGOUT

login
Customer:string Ip:address

### ORDER FORM

order form
Customer: string Tax:int Paymentmode:int Deliveryfee:int Totalbill:int Dateoforder:int

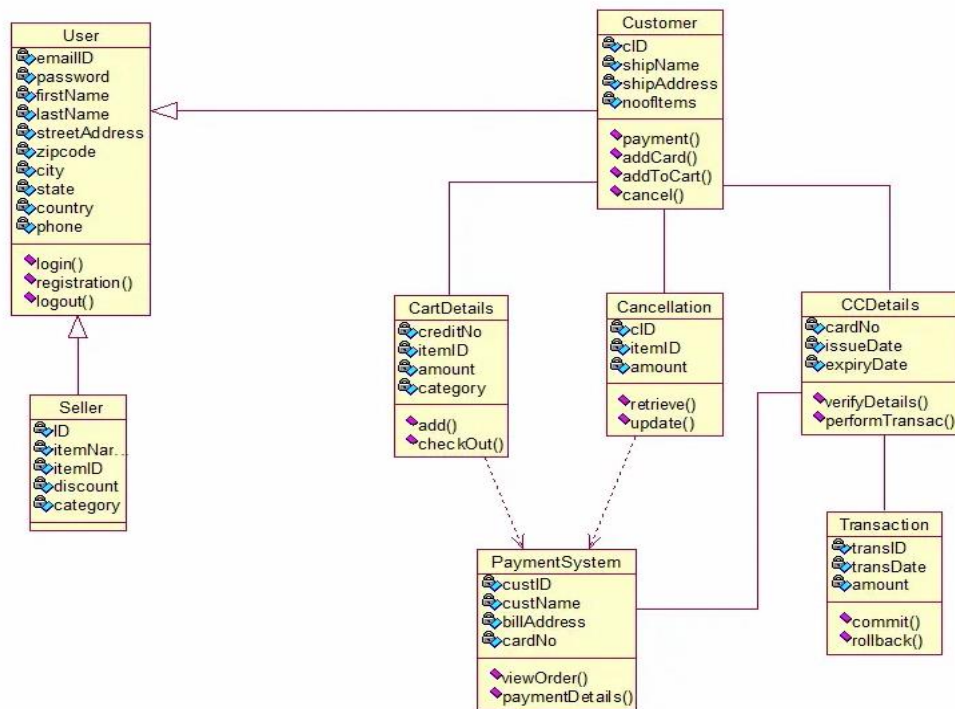
e) Aim: Represent use cases and domain class diagram using rational rose

☐ use case diagram

## USE CASE DIAGRAM



## CLASS DIAGRAM:



f) **Aim:** Develop CRUD matrix to represent relationship between use cases and problem domain classes

Entity/function	customer	login	address	logout	seller	order	book	Mode of Payment
Inventory					C,R			
Create order	C,R,D				C	C		C,U
Bookcatalog	R				C		R	
Login		C,R			C,R			
Managemnt payment	R				C		C	C,R
Order status			R		C	C		

**Result:** The Design was successfully completed.

## **CASE STUDY:POINT OF SALE TERMINAL**

a)Aim:Identify and analyze events

Procedure:

The Flow of events of a use case contains the most important information derived from use case modeling work.it should describe the use cases flow of events clearly enough for an outsider to easily understand it.Remember flow of events should present what the system does,not how the system is design to perform the required behavior.

Guidelines for the contents of the flow of events are, 1)Describe how the use case starts and ends.

2)Describe what data is exchanged between the actor and the use case

3)Donot describe the details of the user interface,unless it is necessary to understand the behavior of the system.

For example,it is often good to use a limited set of web specific terminology when it is known before hand that the application is going to be web based otherwise your run the risk that the use case text is being perceived as too abstract.

words to include in your terminology could be “navigate”,”browser”,”hyperlink”,”page”,”submit”,and ,”browser”.however it is not asvisible to include references to frames or “webpages” in such a way that you are making assumptions about the boundaries between them this is a critical design decision.

- ❖ □Describe the flow of events ,not only the functionality to enforce this,start every action with “when the actor”.
- ❖ □Describe only the events that belong to the usecase and not what happens in other usecases or outside of the system.□ Avoid vague terminology such as “for example”,”information”.
- ❖ □Detail the flow of events all whats should be answered .remember that test designers are to be use this text to identify test cases.
- ❖ If you have used certain terms in other use cases ,be sure to use the exact same terms in this use case,and that their intended meaning is the same.to manage common terms ,put them in a glossary.

**b)Aim: Identify Use cases Procedure:**

1)Buy Product

2)Bar Code scanning

3)Pay Bill

4) Proces State

5)Close SAle

6)Update Inventory

7)Tax Calculator

**1)Buy product:**

Among the multiple products we will select which product we should buy

**2)barcode scanning:**

Here the barcode is scanned for the product which we are selected.the barcode will be ready predefined on each and every product and according to that the price of the product will be displayed.

**3)pay bill:**

Pay bill is a bill in which the price of the product is printed ,according to that price the customers will play

**4)process sale:**

After receiving the bill the products which we have selected are processed to the cachier counter and they will cross check our products according to our bill.

**5)close sale:**

In this close sale the customer will pay the bill and cashier place a stamp on that bill,that is represents the bill is closed.

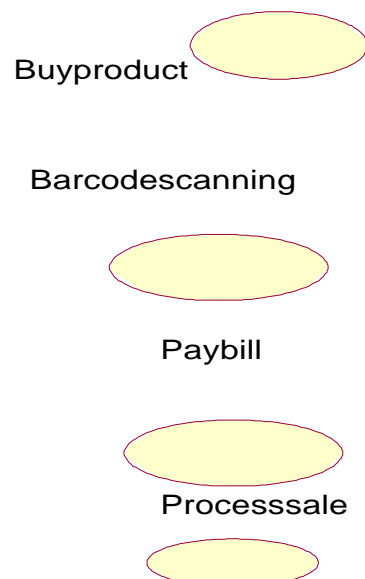
**6)update inventory:**

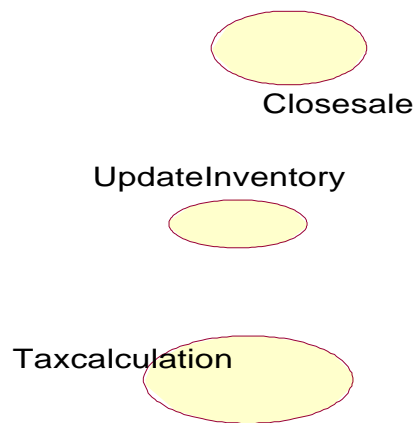
In this the cashier will update the database according to the products in that store.

**7)tax calculation:**

The cashier will calculate the tax in various forms such as transportation cost,maintenance cost.

**Graphical representations:**





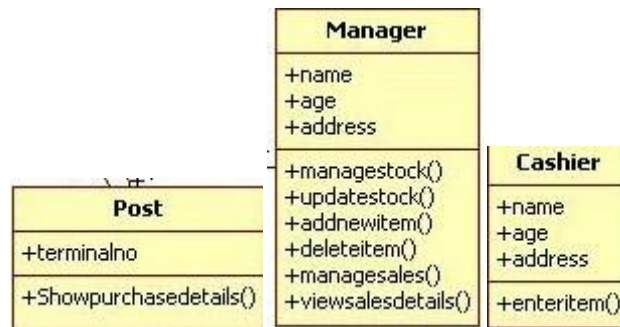
**c) Aim: Develop event table**

USECASE	DESCRIPTION
Buy product	Among the multiple products we will select product we should buy
Barcode scanning	Here the barcode is scanned for the product which we are selected
Pay bill	In this process of the product is printed
Process sale	The bill, products are processed to the cashier counter
Close sale	The customer will pay the bill and cashier place a stamp on that bill
Update inventory	The cashier will update the database according to the products in that store
Tax calculation	The cashier will calculate the tax in various forms such as transportation cost, maintenance cost etc.

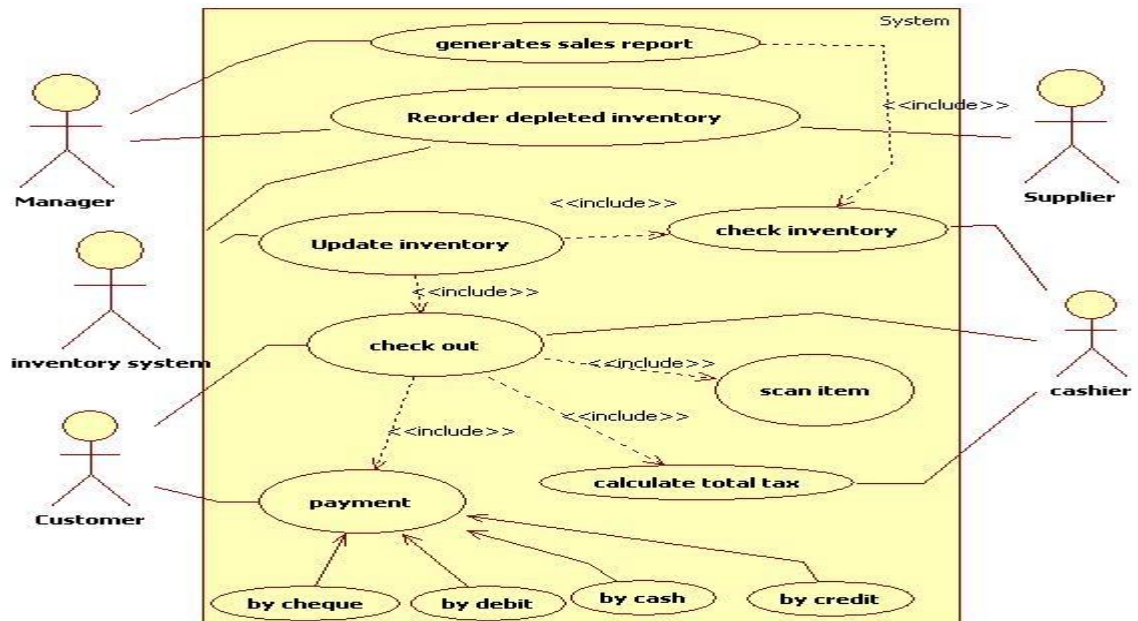
**d) Aim: Identify and analyze domain classes**

Domain classes: A Domain class is a class its description of set of objects that share the same attribute, operations, relationships. A class implements one or more interfaces.

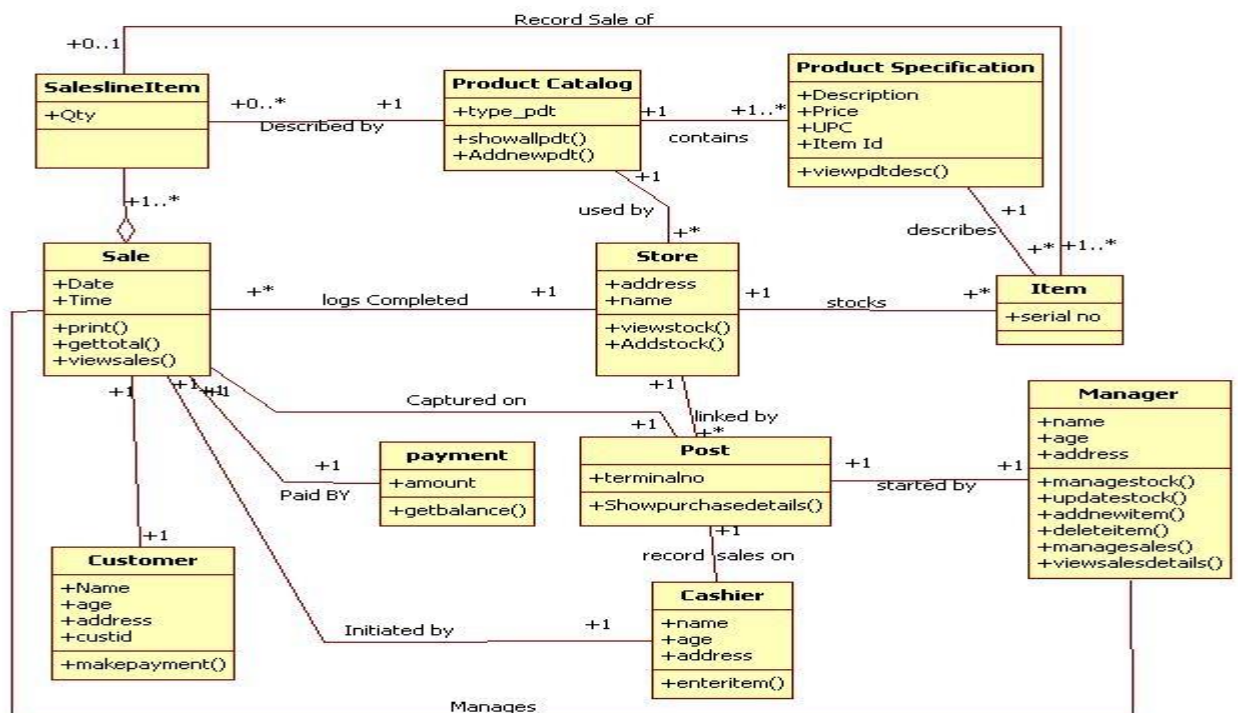




e) Aim: Represent use cases and domain class diagram using rational rose



Class diagram:





**G) Aim: Develop CRUD matrix to represent relationships between use cases and problem domain**

Entity /function	Customer	cust	Product catalog	Psc	sale	Store	payment	post	manager
Buy product	C,D	R	C,R,U	R	U	U		R	R
Barcode scanning		R			R		C		
Paybill	R	C					R		
Process sale			R		C				
Close sale			C,R	R	R	U		R	R
Updated inventory			C,R	U	U	U		R	U
Tax calculations		C					R	R	C,R

**Result:** The Design was successfully completed.

# CASE STUDY: LIBRARY MANAGEMENT SYSTEM

**Aim:** Identify and analyze events

The flow of events of a use case contains the most important information derived from use case modeling work. It should describe the use cases flow of events clearly enough for an outsider to easily understand it. Remember flow of events should present what the system does, not how the system is design to perform the required behavior.

Guidelines for the contents the flow of events are,

- 1) Describe how the use case starts and ends.
- 2) Describe what data is exchanged between the actor and the use case
- 3) Donot describe the details of the user interface, unless it is necessary to understand the behavior of the system.

## Identify use cases

### Procedure:

#### a) Register member:

here the members who can access the library are registered based on the id nos.

#### a.1) verify member:

before issuing book the specific member is verified whether he/she is registered or not

#### a.2) check availability or book:

after verifying the person then the availability of book which he/she selected is checked whether the book is available or not

after verifying & checking the book is issued

#### c) Return book:

after the specific time is over the person should return book during returning

#### c.1) Calculate fine:

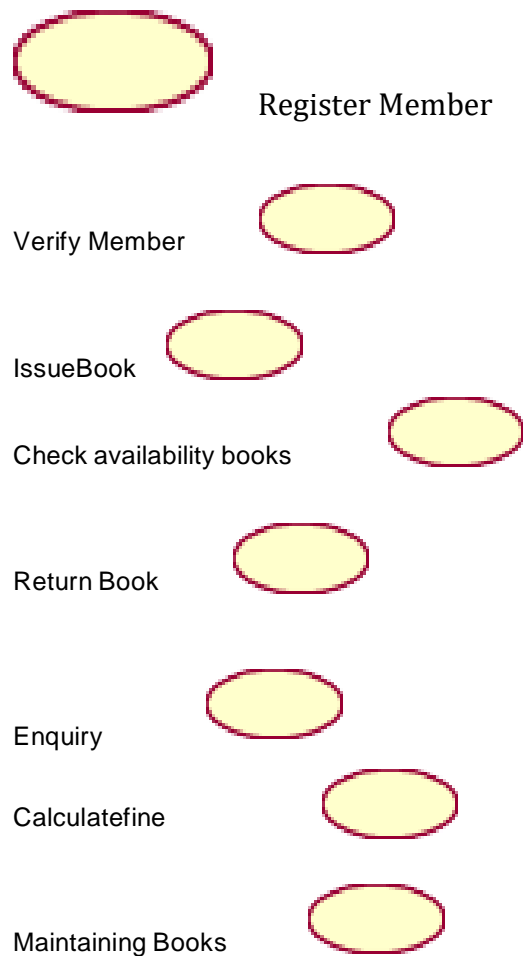
the fine is calculated

#### d) Enquiry:

in this the member is checked if fine is existed or not

### E)Maintaining books:

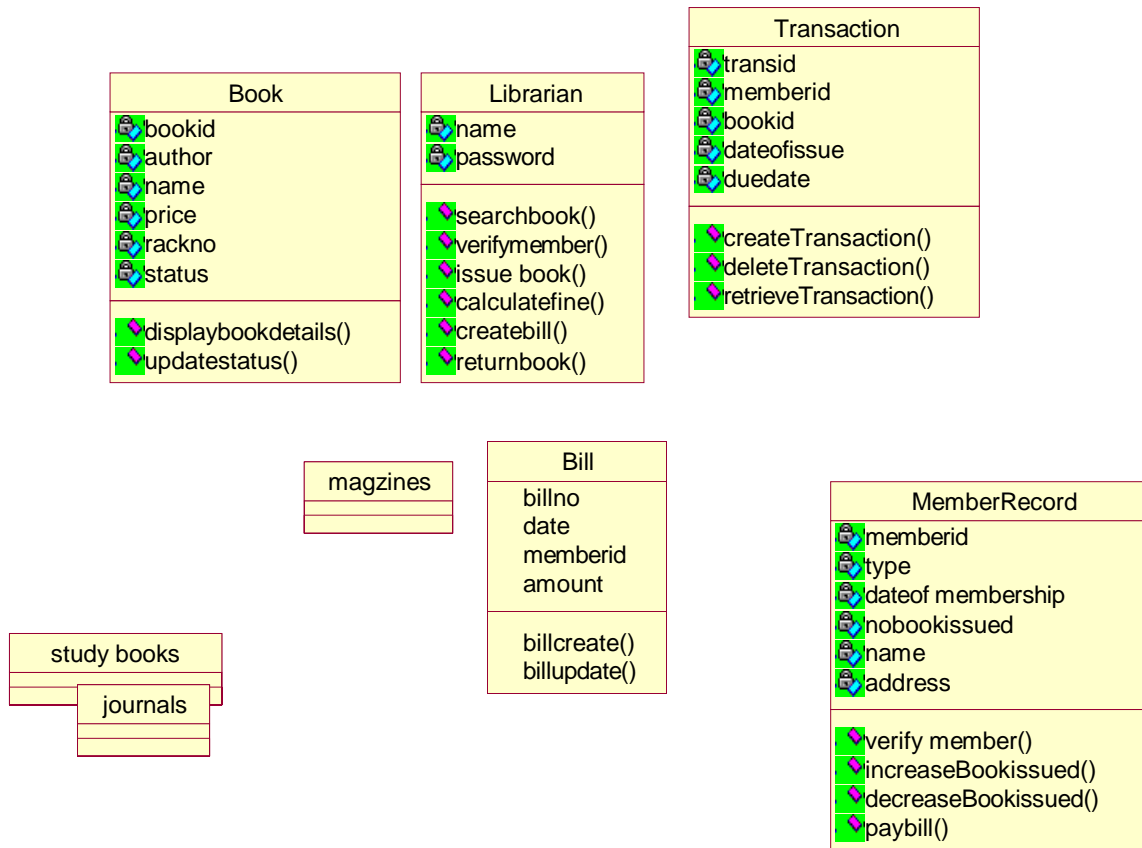
this is done by the librarian that how many books are present, person or members login id and their registrations etc.



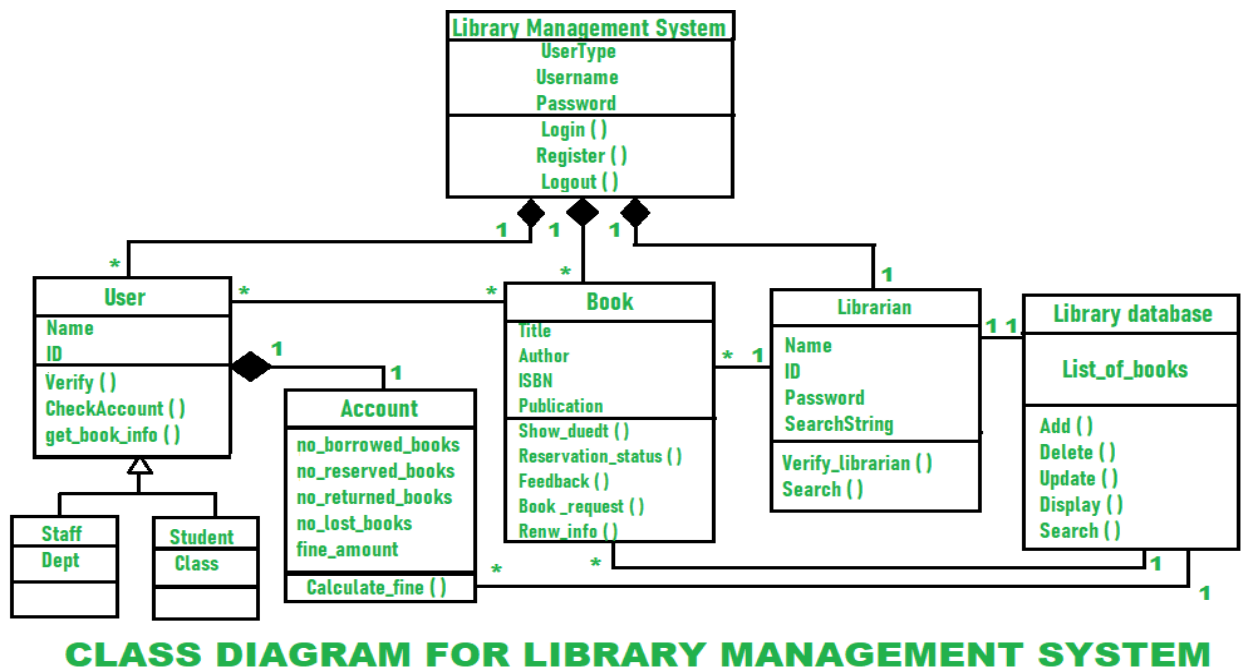
### c)Aim:Develop event table:

Usecase	Description
Register member	The members who can access the library are registered based on the ids
Issue book-verify member	Before issuing the book the specific member is verified whether he/she is registred or not
Check availability of book	Check the book is available or not
Return book	After the specific time is over the person should return
Calculate fine	The fine is calculated if their fine
Enquiry	In this the member is checked if fine is existed or not
Maintaining books	This is done by the library to cross checking the books

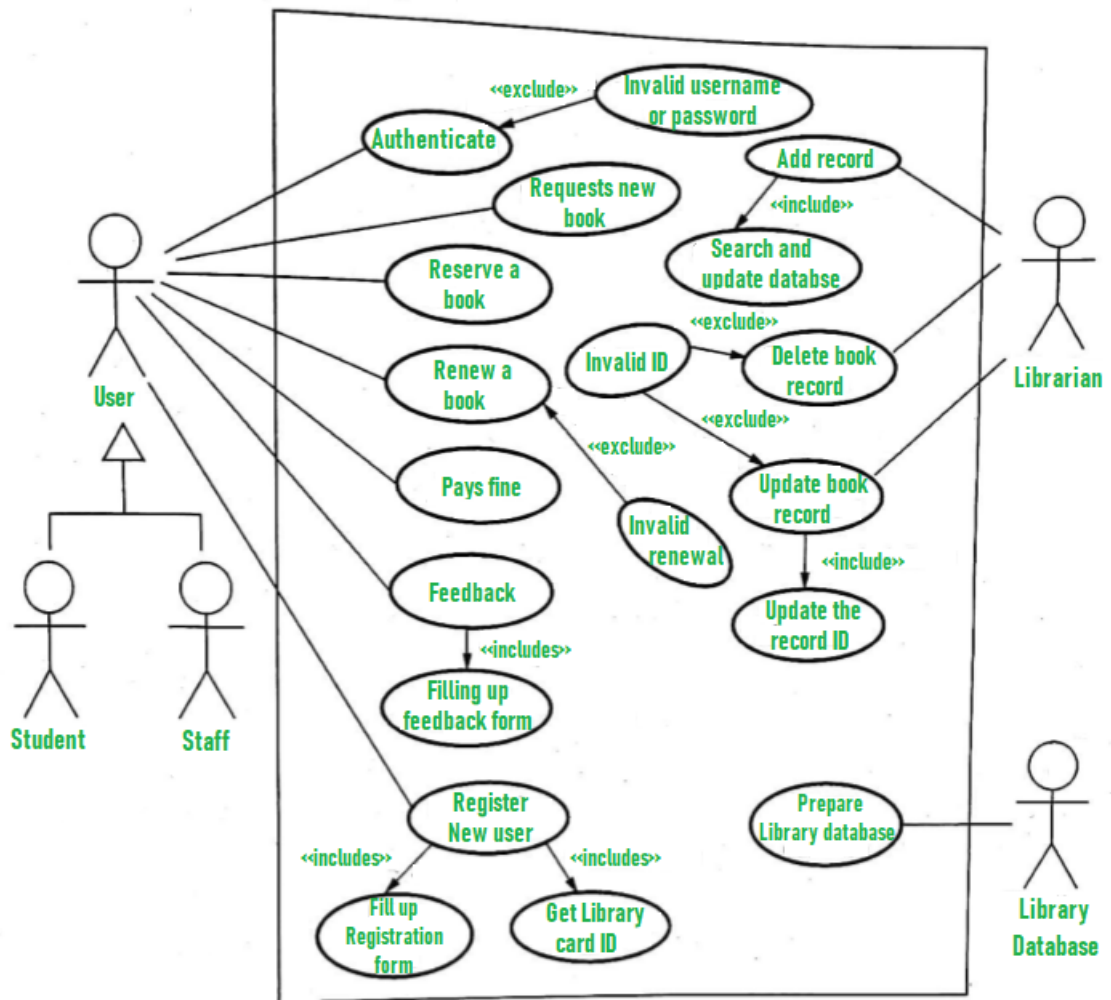
E) **Aim:**Identify and analyze domain classes



**Aim:**Represent class diagram for library management system



Use case diagram:



f) **Aim:** Develop CRUD matrix to represent relationships between usecase and problem domain classes

E n t i t y	Li br ari an	Tra ns a c t i o n	B o o k	j o u r n a l	St u d y b o o k	m a g z i n e s	B i l l	M e m b e r	S t u d e n t	F a c u l t y
Register member	UC ,D									
Enquiry	C							R	R	R
IssueBook	C		R	R	R	R				
VerifyBook	R									
VerifyMember										
Check availability			R	R	R	R		R	R	R

ty										
ReturnB ook	R	R							C	C
Calculat eFine	C	C							R	R
Maintai nBooks	C									

**Result:** The Design was successfully completed.