

BHARTIYA VIDYA BHAVAN’S

M.M.COLLEGEOFARTS, N.M. INSTITUTE OF SCIENCE

H.R.J.COLLEGEOFCOMMERCE BHAVAN’SCOLLEGE

MUNSHINAGAR, ANDHERI WEST,

MUMBAI-400058

BHARATIYA VIDYA BHAVAN

BHAVAN’S COLLEGE, ANDHERI (W)

**CERTIFICATE**

This is to certify that \_**Mr. Kabira The** Seat no**: SYIT00** of S.Y.B.SC Information Technology has satisfactorily completed the practical course in Software Engineering And Management Practices Practical’s as prescribed by the University of Mumbai during the academic year 2022-2023.

Signature Signature

|  |  |
| --- | --- |
| Staff- In-Charge  Coordinator |  |
| Examiner Signature | Stamp |

**INDEX**

|  |  |  |
| --- | --- | --- |
| **SR.**  **NO.** | **NAME** | **SIGN** |
| **1** | **An introduction to software engineering** |  |
| **2** | **Development of SRS document, Design document for the selected project** |  |
| **3** | **Development of DFD, data dictionary, E-R diagram, structured chart for the project** |  |
| **4** | **To study and draw various UML daigrams** |  |
| **5** | **To illustrate the use of class diagrams** |  |
| **6** | **To draw an activity diagram and use case diagram for ATM and Library Management System** |  |
| **7** | **Draw Object Diagram for ATM System** |  |
| **8** | **Development of State Transition Diagram** |  |
| **9** | **Draw ER Diagram for Hospital ManagementSystem** |  |

**Experiment No-1**

**Aim: - An introduction to software engineering.**

# Description:-

Software engineering is about teams. The problems to solve are so complex or large, that a single developer cannot solve them anymore. Software engineering is also about communication. Teams do not consist only of developers, but also of testers, architects, system engineers, customer, project managers, etc. Software projects can be so large that we have to do careful planning. Implementation is no longer just writing code, but it is also following guidelines, writing documentation and also writing unit tests. But unit tests alone are not enough. The different pieces have to fit together. And we have to be able to spot problematic areas using metrics. They tell us if our code follows certain standards. Once we are finished coding, that does not mean that we are finished with the project: for large projects maintaining software can keep many people busy for a long time. Since there are so many factors influencing the success or failure of a project, we also need to learn a little about project management and its pitfalls, but especially what makes projects successful. And last but not least, a good software engineer, like any engineer, needs tools, and you need to know about them.

# Developers Work in Teams

In your beginning semesters you were coding as individuals. The problems you were solving were small enough so one person could master them. In the real world this is different:- the problem sizes and time constraints are such that only teams can solve those problems.

For teams to work effectively they need a language to communicate (UML). Also teams do not consist only of developers, but also of testers, architects, system engineers and most importantly the customer. So we need to learn about what makes good teams, how to communicate with the customer, and how to document not only the source code, but everything related to the software project.

# New Language

In previous courses we learned languages, such as Java or C++, and how to turn ideas into code. But these ideas are independent of the language. With Unified Modeling Language (UML) we will see a way to describe code independently of language, and more importantly, we learn to think in one higher level of abstraction. UML can be an invaluable communication and documentation tool.

We will learn to see the big picture: patterns. This gives us yet one higher level of abstraction. Again this increases our vocabulary to communicate more effectively with our peers. Also, it is a fantastic way to learn from our seniors. This is essential for designing large software systems.

# Measurement

Also just being able to write software, doesn’t mean that the software is any good. Hence, we will discover what makes good software, and how to measure software quality. On one hand we should be able to analyses existing source code through static analysis and measuring metrics, but also how do we guarantee that our code meets certain quality standards? Testing is also important in this context, it guarantees high quality products.

# New Tools

Up to now, you may have come to know about an IDE, a compiler and a debugger. But there are many more tools at the disposal of a software engineer. There are tools that allow us to work in teams, to document our software, to assist and monitor the whole development effort. There are tools for software architects, tools for testing and profiling, automation and re-engineering.

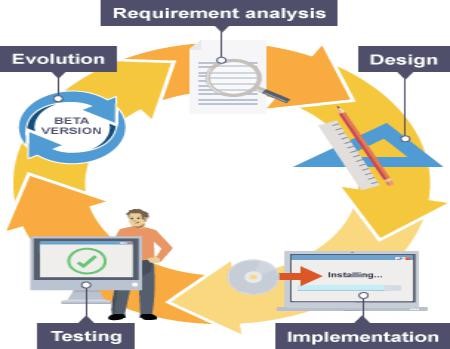
# SDLC (Software Development Life Cycle)

Software Development Life Cycle or SDLC is a process used to develop software. There are different stages or phases within the Software Development Life Cycle and in each phase, different activities take place.

SDLC creates a structure for the development teams to be able to design, create and deliver high quality software by defining various tasks that need to happen The life cycle defines a methodology for improving the quality of software and the overall development process.

The methodology within the SDLC process can vary across organizations, but standards such as ISO/IEC 12207 represent processes that establish a life cycle for software, and provide a standard for building and maintaining software.

The intent of a SDLC process it to help produce a product that is cost-efficient, effective, and of high quality.



# SDLC Phases

* **Requirement Analysis**

Software Development Life Cycle begins with Requirement Analysis phase, where the stakeholders discuss the requirements of the software that needs to be developed to achieve a goal. The aim of the requirement analysis phase is to capture the detail of each requirement and to make sure everyone understands the scope of the work and how each requirement is going to be fulfilled.

It is a normal practice to also discuss how each requirement will be tested and so testers can add great value in participating in requirement analysis meetings.

* **Design**

The next stage of Software Development Life Cycle is the Design phase. During the design phase, developers and technical architects start the high-level design of the software and system to be able to deliver each requirement.

The technical details of the design is discussed with the stakeholders and various parameters such as risks, technologies to be used, capability of the team, project constraints, time and budget are reviewed and then the best design approach is selected for the product.

The selected architectural design, defines all the components that needs to be developed, communications with third party services, user flows and database communications as well as front-end representations and behavior of each components. The design is usually kept in the Design Specification Document (DSD).

* **Implementation**

After the requirements and design activity is completed, the next phase of the Software Development Life Cycle is the implementation or development of the software. In this phase, developers start coding according to the requirements and the design discussed in previous phases.

Database admins create the necessary data in the database, front-end developers create the necessary interfaces and GUI to interact with the back-end all based on guidelines and procedures defined by the company.

Developers also write unit tests for each component to test the new code that they have written, review each other’s code, create builds and deploy software to an environment.

This cycle of development is repeated until the requirements are met.

* **Testing**

Testing is the last phase of the Software Development Life Cycle before the software is delivered to customers. During testing, experienced testers start to test the system against the requirements.

The testers aim to find defects within the system as well as verifying whether the application behaves as expected and according to what was documented in the requirements analysis phase.

Testers can either use a test script to execute each test and verify the results, or use [**exploratory**](http://www.testingexcellence.com/exploratory-testing-tips-best-practices/)[**testing**](http://www.testingexcellence.com/exploratory-testing-tips-best-practices/)which is more of an experience based approach.

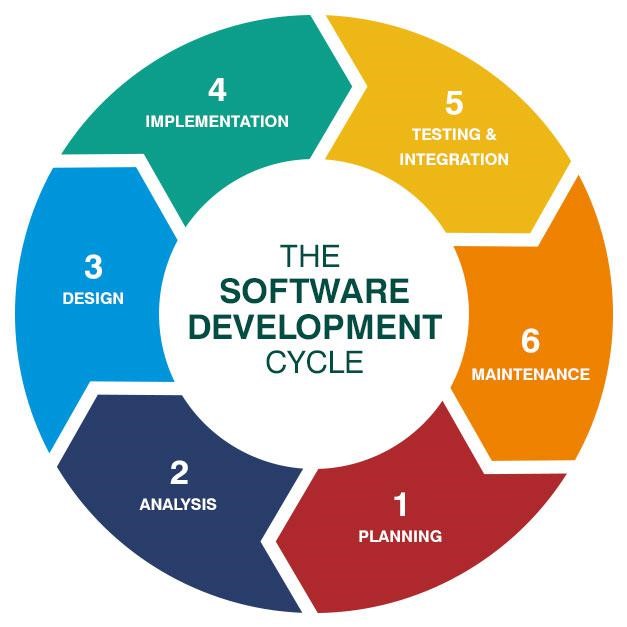
It is possible that defects are identified in the testing phase. Once a defect is found, testers inform the developers about the details of the issue and if it is a valid defect, developers will fix and create a new version of the software which needs to be verified again.

v **Deployment & Maintenance**

Once the software has been fully tested and no [**high**](http://www.testingexcellence.com/severity-and-priority-difference/)[**priority**](http://www.testingexcellence.com/severity-and-priority-difference/)[**issues**](http://www.testingexcellence.com/severity-and-priority-difference/)remain in the software, it is time to deploy to production where customers can use the system.

Once a version of the software is released to production, there is usually a maintenance team that looks after any post-production issues.

If an issue is encountered in the production the development team is informed and depending on how severe the issue is, it might either require a hot-fix which is created and shipped in a short period of time or if not very severe, it can wait until the next version of the software.



# Smart Draw:-

Smart Draw is the easy-to-use Windows software for creating business diagrams, technical drawings, and business documents. Smart Draw requires no artistic talent—users simply drag and drop ready-made graphic elements to create professional-looking diagrams quickly and easily.

Smart Draw competes with products like Microsoft Visio, but Smart Draw is specifically aimed at users who do not have the time or the need to learn such complex programs. Smart Draw is easier to learn, easier to use, and easier to afford than competing products—and it produces better-looking results. Smart Draw offers complete business graphics solutions—integrated sets of symbols, templates, instructions, and tutorials—for many standard business problems, including: Business Process Management, Software Design, Network Design, and more. With Smart Draw, anyone can:

* Illustrate a report.
* Analyze a process.
* Make a presentation.
* Persuade others.
* Document procedures.
* Communicate clearly.
* Help others.

In addition to pre-packaged solutions, Smart Draw offers over 50,000 ready-made symbols and templates for creating all kinds of business charts and diagrams. Smart Draw’s symbols are logically categorized and organized into libraries.

**Experiment No.2**

**Aim: - Development of SRS document, Design document for the selected project.**

**Software Requirements Specification (SRS)**

**Library Management System (LMS)**

# Introduction:-

The purpose of Software Requirements Specification (SRS) document is to describe the external behavior of the Online Library System. Requirements Specification defines and describes the operations, interfaces, performance, and quality assurance requirements of the Online Library System. The document also describes the nonfunctional requirements such as the user interfaces. It also describes the design constraints that are to be considered when the system is to be designed, and other factors necessary to provide complete and comprehensive description of the requirements for the software. The Software Requirements Specification (SRS) captures the complete software requirements for the system, or a portion of the system. Requirements described in this document are derived from the Vision Document prepared for the Online Library System

## Purpose:-

The purpose of this application is as follows:

* The software is for automation of library.
* It provides following facilities to

**Operator:**

* Can enter details related to a particular book.
* Can provide membership to members.

**Admin:**

* Can read and write information about any member.
* Can update, create, and delete the record of membership as per requirement and implementation plants.

**2.) Scope:**

The different areas where we can use this application are :

* Any education institute can make use of it for providing information about author, content of the available books.
* It can be used in offices and modifications can be easily done according to requirements.

**3.) Technology Used:**

Front End: Java, Netbeans.

Back End: MS Access, SQL.

# 4.) Assumptions

* This application is used to convert the manual application to the online application.
* Customized data will be used in this application.
* User does not have right to enter information about books.

1. **Overview:**

Project is related to library management which provides reading services to its members. Any person can become a member of the library by filling a prescribed form.They can get the book issued, so that they cab take home and return them.

1. **Functionality:**

* Online membership.
* Keeps the track of issues and submission of books.

**Functional Requirements:**

* + In this system there is lots of functioning.
  + The user can Search book and see his/her account information.
  + He will be able to get the records in any format.
  + There will also be a reminder and digital diary through this he can go to know aboutits important date like his book submission date.
  + Admin can add new record and saw all record of any student as well as library.

# ModuleDescription

**Type of user:**

* Administrator(librarian)
* HODofallDepartment
* Student
* Faculty
* Staff

**Administrator module:**

* Budget(foralldepartment)
* Stockverification
* Createuser
* Accept/Rejectuser
* Changepassword
* Bookinventory
* Userinformation
* Reportgeneration
* SearchBook

## Complaint

•

Allusercansendtheircomplaint

## User module

* Changepassword
* BookSearch
* Seeaccountinformation

## Fine information

* Alluserchecktheirfine
* Finebydate&Month

## Book inventory

* Purchasingbookinformation
* Maintainbookinventory

## Book search

* Normalsearchbyalphabet
* Quicksearchbytype(Bookid,Author,ISBN,Publication) **Other charges**
* BookBinding,chair,Rack
* Journals
* Newspaper

# External Interface Requirements

## User Interfaces

* The design or layout of every form will be very clear and very interactive to the user.
* When the user open the software the welcome window will appear.
* In the login window the user can easily entered the desired password and login name.
* Then it will give the successfully login message.
* From each and every window the user can easily go to any desired window that is there is will be a absolute and relative linking.
* In every window & Mac there is help and support option is present for the ease of user.
* There will be a proper collection of GUI interface, which will provide better look and feel.

In the screen layout the background color is very light and the graphics and font style will be in proper manner and well organized.

* If the user will print any error statement then it will give the proper error message display.
* In each and every window there will be alert, confirm etc message box for displaying message.
* The user will be able to search any data from the record by using proper guideline shown in the window & Mac.
* In the opening of the software there will be a menu window where the overall table contents of the software will be present through which the user can move to any desired window &Mac.
* This will provide the better security data because the menu window will be displaying according to the login (admin or normal user).
* User can easily save its data in to the database and keep track of the records of purchase, vendor and inventory etc.
* This software will be easily understandable and operable by the user. **HardwareInterfaces**
* The existing Local Area Network (LAN) will be used for collecting data from the users and also for updating the Library Catalogue.

## SoftwareInterfaces

* A firewall will be used with the server to prevent unauthorized access to the system. **CommunicationsInterfaces**
* The Online Library System will be connected to the World Wide Web.

# Other Nonfunctional Requirements

## PerformanceRequirements

* This software is not breakdown suddenly in any disaster like power failure.
* The development of the software will be based on the object oriented model.
* The timeline of this software must be in our mind.

The performance of the functions and every module must be well.

* At every step the output of the one phase is the input of the other phase and it will be reliable and accurate.
* The risk factor must be taken at initial step for better performance of the software.
* For individual function the performance will be well.
* For login to the software password and user name will be matched to the password and name
* Saved in the database and thus only authenticated users are allowed to the login.
* There will be various ways of retrieving data and it takes less time.
* There will be ambiguity in the data and the record.
* This software will be well supported to the other embedded software such as digitaldairy, notepad etc.
* The overall performance of the software will reliable and enable the users to work efficiently.

## SecurityRequirements

* There will be proper security regarding to the accessing of data.
* The external security can be provided by given the login authentication.
* The data that are stored in the database must be private.
* There is also required a user authentication.
* There is also the facility that the admin can lock his private data that will not be accessed by anyone.
* The whole software is secure from the outside accessing.

**SoftwareQualityAttributes**

* Our software has many quality attribute that are given below-

**Adaptability**

* This software is adaptable by any organization.

**Availability-**

* The availability of the software is easy and for everyone.

**Correctness-**

The results of the function are pure and accurate.

### Flexibility-

* The operation may be flexible and reports can be presented in many ways. **Maintainability-**
* After the deployment of the project if any error occurs then it can be easily maintain by the software developer.

**Portability-**

* The software can be deployed at any machine.

### Reliability-

* The performance of the software is better which will increase the reliability of the software. **Reusability-**
* The data and record that are saved in the database can be reused if needed. **Robustness-**
* If there is any error in any window or module then it does not effect the remaining part of the software.

**Testability-**

* The software will be tested at every. Alpha Testing Beta Testing Acceptance Testing

### Usability-

To perform any operations and to understand the functioning of software is very easy.

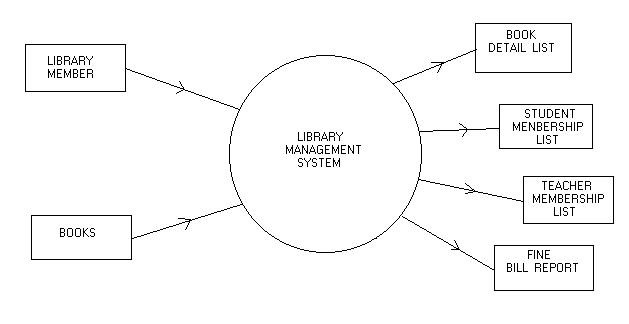
### Productivity-

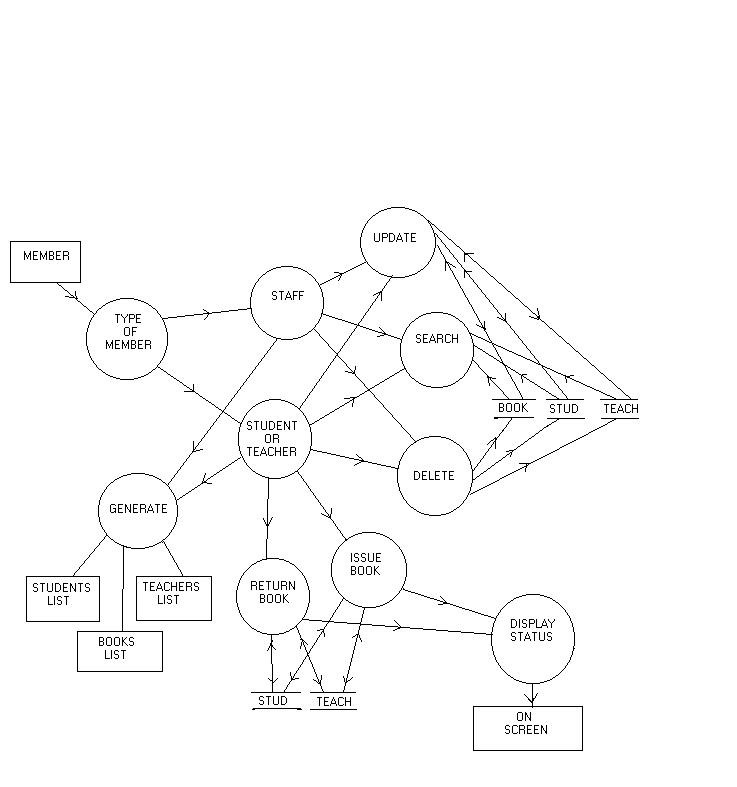
* This software will produce every desired result with accurately. **Timelines-**
* The time limit is very important. It will save much time and provide fast accessing. **Costeffective-**
* This software is less in cost and bearable by any Organization.

**Experiment No-3**

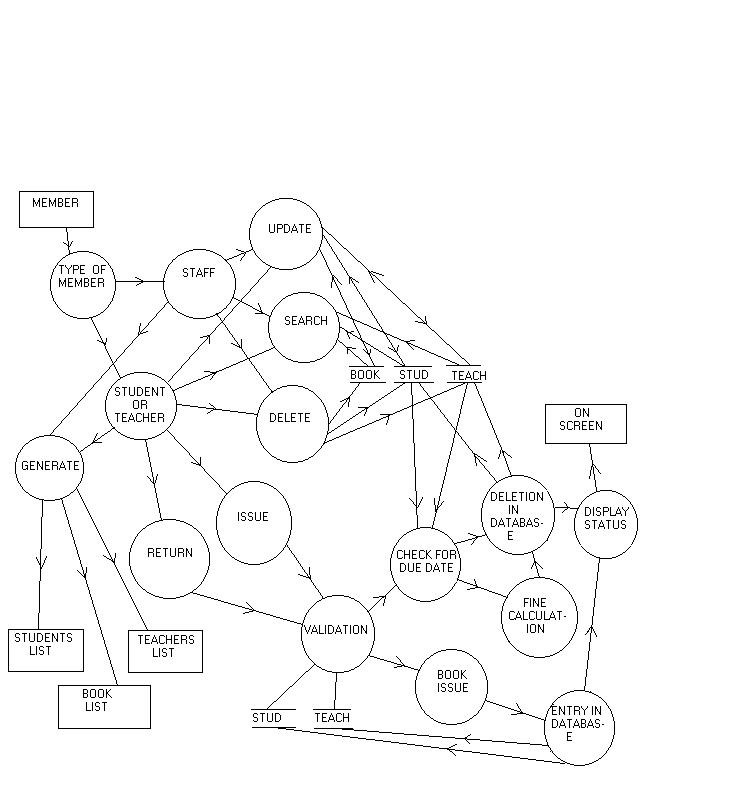
**Aim: -Development of DFD, data dictionary, E-R diagram, structured chart for the project.**



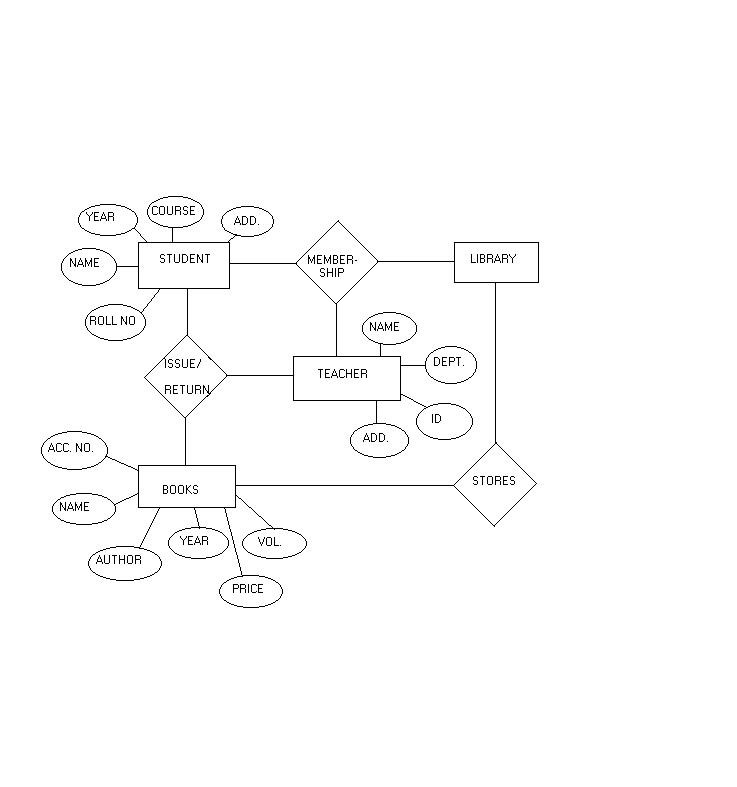












**Experiment**

**No.-4**

**Aim:**

**To**

**study**

**and**

**draw**

**various**

**UML**

**diagrams.**



**UML:-**

UML

stands

for

Unified

Modeling

Language

which

is

used

in

object

oriented

software

engineering.

Although

typically

used

in

software

engineering

it

is

a

rich

language

that

can

be

used

to

model

an

application

structures,

behavior

and

even

business

processes.

There

are

**14**

UML

diagram

types

to

help

you

model

this

behavior.

They

can

be

divided

into

two

main

categories;

structure

diagrams

and

behavioral

diagrams.

All

14

UML

diagram

types

are

listed

below

with

examples

and

a

brief

introduction

to

them

explaining

how

they

are

used

when

modeling

applications.

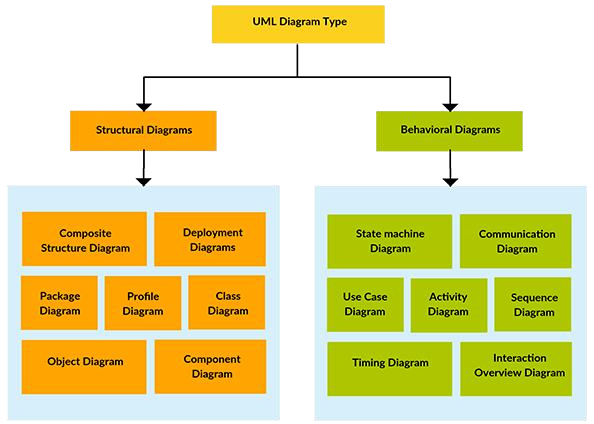


**Types**

**of**

**UML**

**diagram:-**





**Use**

**Case**

**Diagram**

As

the

most

known

diagram

type

of

the

behavioral

UML

diagrams,

Use

case

diagrams

give

a

graphic

overview

of

the

actors

involved

in

a

system,

different

functions

needed

by

those

actors

and

how

these

different

functions

are

interacted.

It

’

s

a

great

starting

point

for

any

project

discussion,

because

you

can

easily

identify

the

main

actors

involved

and

the

main

processes

of

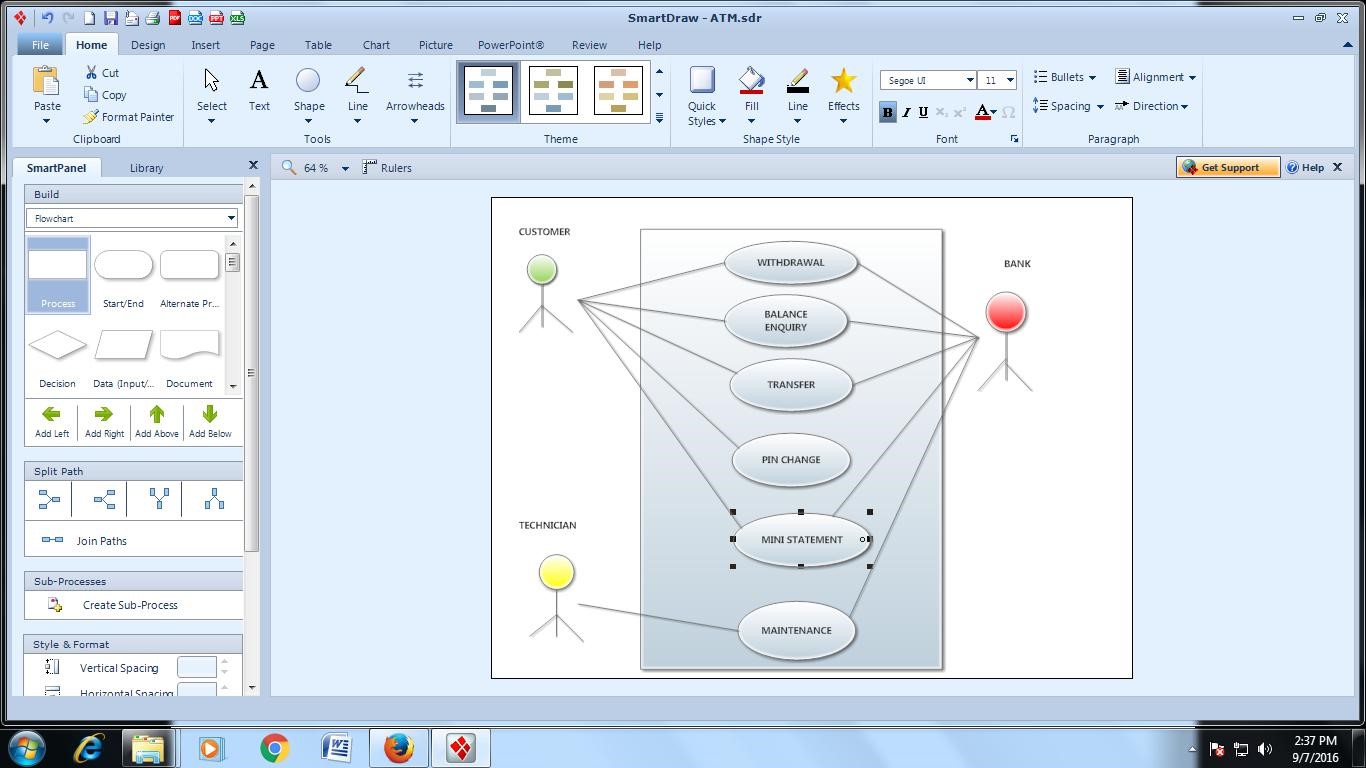
the

system.



**For**

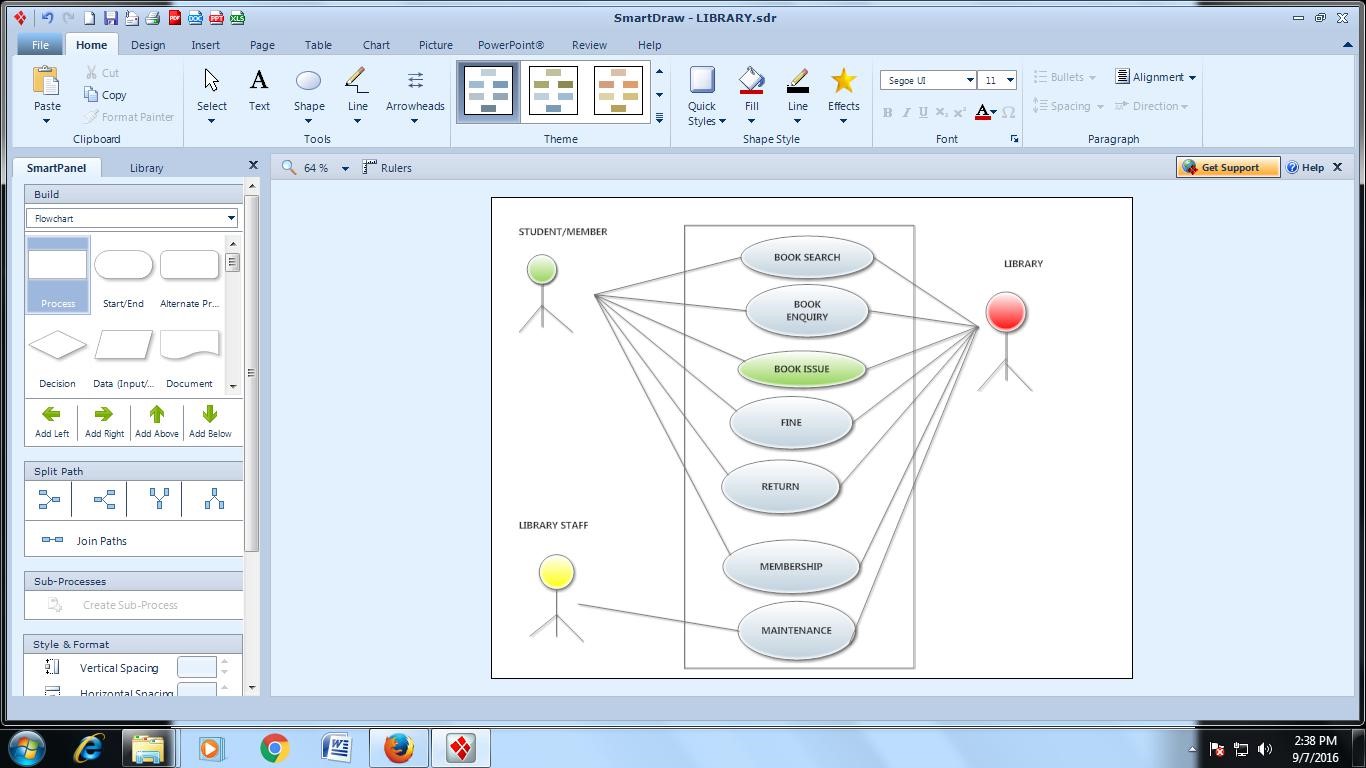
**ATM**





**For**

**Library**



**Experiment:5**

**Aim:**

**To**

**illustrate**

**the**

**use**

**of**

**class**

**diagrams.**

In

[softwar](https://en.wikipedia.org/wiki/Software_engineering)

[e](https://en.wikipedia.org/wiki/Software_engineering)

[engineerin](https://en.wikipedia.org/wiki/Software_engineering)

[g](https://en.wikipedia.org/wiki/Software_engineering)

[,](https://en.wikipedia.org/wiki/Software_engineering)

a

class

diagram

in

the

[Unifie](https://en.wikipedia.org/wiki/Unified_Modeling_Language)

[d](https://en.wikipedia.org/wiki/Unified_Modeling_Language)

[Modelin](https://en.wikipedia.org/wiki/Unified_Modeling_Language)

[g](https://en.wikipedia.org/wiki/Unified_Modeling_Language)

[Languag](https://en.wikipedia.org/wiki/Unified_Modeling_Language)

[e](https://en.wikipedia.org/wiki/Unified_Modeling_Language)

)

(

UML

is

a

type

of

static

structure

diagram

that

describes

the

structure

of

a

system

by

showing

the

system's

[s](https://en.wikipedia.org/wiki/Class_(computer_science))

[classe](https://en.wikipedia.org/wiki/Class_(computer_science))

,

their

attributes,

operations

(

or

methods),

and

the

relationships

among

objects.

The

class

diagram

is

the

main

building

block

of

[object-oriente](https://en.wikipedia.org/wiki/Object-oriented_programming)

[d](https://en.wikipedia.org/wiki/Object-oriented_programming)

modelling.

It

is

used

both

for

general

[conceptua](https://en.wikipedia.org/wiki/Conceptual_model)

[l](https://en.wikipedia.org/wiki/Conceptual_model)

[modellin](https://en.wikipedia.org/wiki/Conceptual_model)

[g](https://en.wikipedia.org/wiki/Conceptual_model)

of

the

systematics

of

the

application,

and

for

detailed

modelling

translating

the

models

into

[g](https://en.wikipedia.org/wiki/Programming_code)

[programmin](https://en.wikipedia.org/wiki/Programming_code)

[e](https://en.wikipedia.org/wiki/Programming_code)

[cod](https://en.wikipedia.org/wiki/Programming_code)

[.](https://en.wikipedia.org/wiki/Programming_code)

Class

diagrams

can

also

be

used

for

[da](https://en.wikipedia.org/wiki/Data_modeling)

[t](https://en.wikipedia.org/wiki/Data_modeling)

[a](https://en.wikipedia.org/wiki/Data_modeling)

modeling

.

The

classes

in

a

class

diagram

represent

both

the

main

elements,

interactions

in

the

application,

and

the

classes

to

be

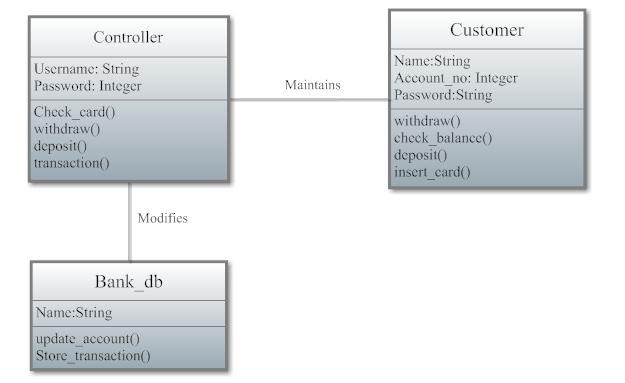
programmed.

**Class**

**Diagram**

**for**

**atm:**



**Class**

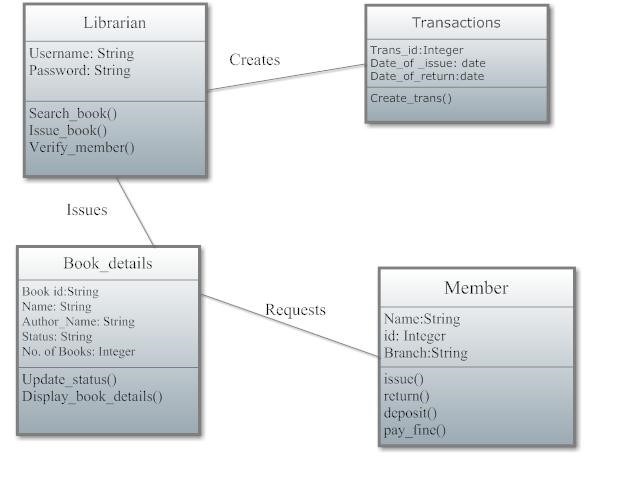
**Diagram**

**for**

**Library**

**management**

**System:**



**Experiment**

**No.-6**

**Aim:**

To

draw

an

activity

diagram

and

use

case

diagram

for

ATM

and

Library

Management

System.

**ACTIVITY**

**DIAGRAM:**

Activity

diagram

is

another

important

diagram

in

UML

to

describe

dynamic

aspects

of

the

system.

Activity

diagram

is

basically

a

flow

chart

to

represent

the

flow

form

one

activity

to

another

activity

.

The

activity

can

be

described

as

an

operation

of

the

system.

So

the

control

flow

is

drawn

from

one

operation

to

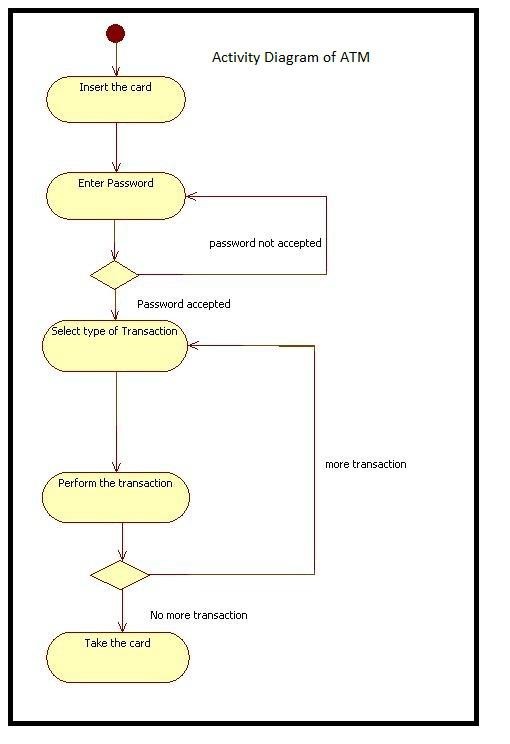
another.

**Activity**

**Diagram**

**for**

**ATM:**



**Activity**

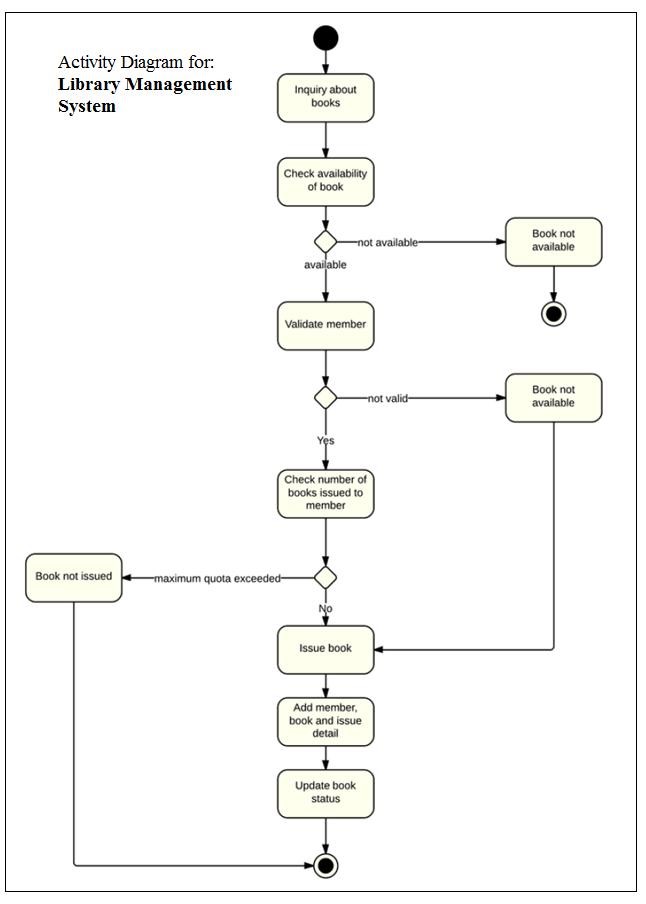
**Diagram**

**for**

**Library**

**Management**

**System:**



**Experiment**

**No**

**–**

**7**

**AIM:**

**-**

**Draw**

**Object**

**Diagram**

**for**

**ATM**

**System.**

**Object**

**Diagram:**

Object

diagrams

are

derived

from

class

diagrams

so

object

diagrams

are

dependent

upon

class

diagrams.

Object

diagrams

represent

an

instance

of

a

class

diagram.

The

basic

concepts

are

similar

for

class

diagrams

and

object

diagrams.

Object

diagrams

also

represent

the

static

view

of

a

system

but

this

static

view

is

a

snapshot

of

the

system

at

a

particular

moment.

Object

diagrams

are

used

to

render

a

set

of

objects

and

their

relationships

as

an

instance

**Purpose:**

The

purpose

of

a

diagram

should

be

understood

clearly

to

implement

it

practically.

The

purposes

of

object

diagrams

are

similar

to

class

diagrams.

The

difference

is

that

a

class

diagram

represents

an

abstract

model

consisting

of

classes

and

their

relationships.

But

an

object

diagram

represents

an

instance

at

a

particular

moment

which

is

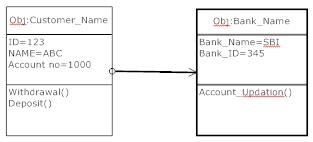
concrete

in

nature.

**ATM**

**System:**



**Experiment**

**No**

**-**

**8**

**Aim:**

**-**

**Development**

**of**

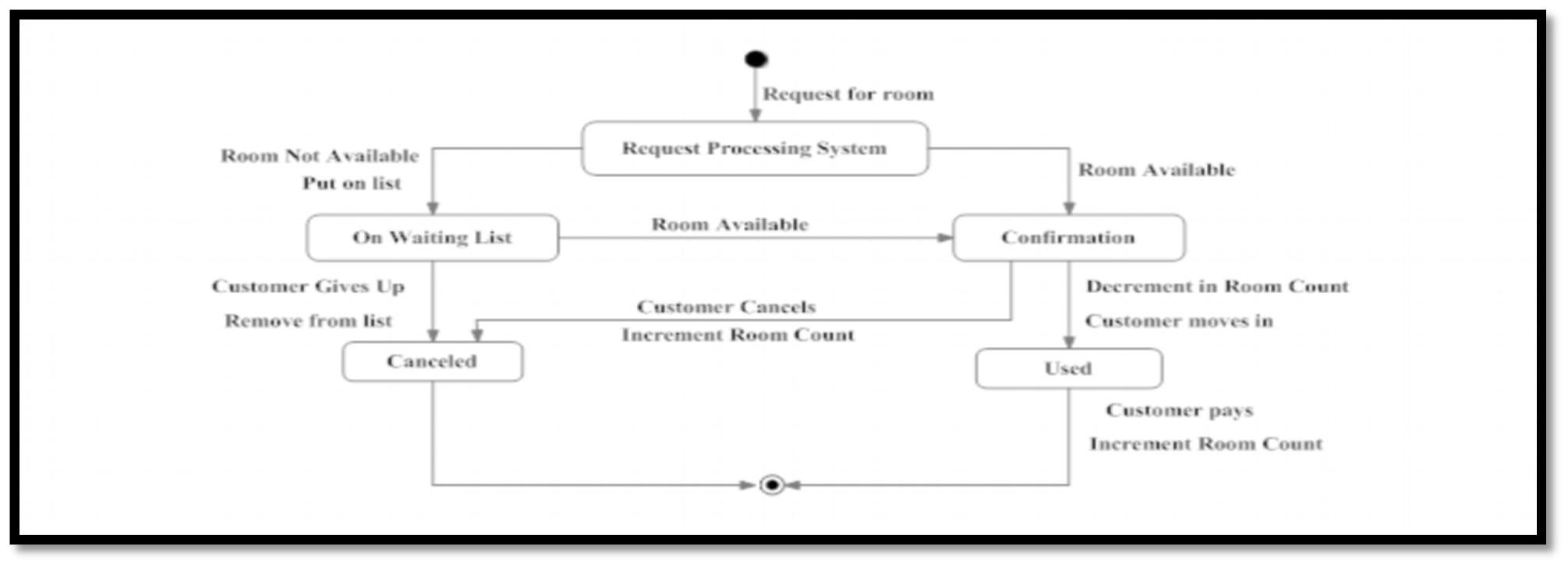
**State**

**Transition**

**Diagram.**

**Hotel**

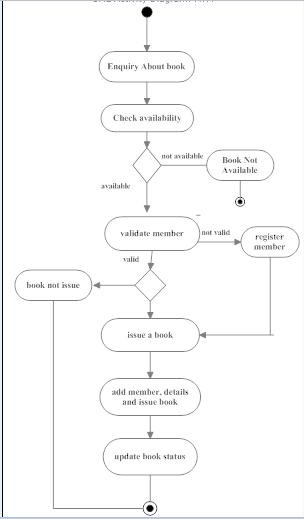
**Reservation**



**Library**

**Management**

**System**



**Experiment**

**No.**

**-**

**9**

**Aim:-Draw**

**ER**

**Diagram**

**for**

**Hospital**

**Management**

**System.**

**ER-Diagram**

Any

object,

for

example,

entities,

attributes

of

an

entity,

relationship

sets,

and

attributes

of

relationship

sets,

can

be

represented

with

the

help

of

an

ER

diagram.

**Entity**

Entities

are

represented

by

means

of

rectangles.

Rectangles

are

named

with

the

entity

set

they

represent.

**Attributes**

Attributes

are

the

properties

of

entities.

Attributes

are

represented

by

means

of

ellipses.

Every

ellipse

represents

one

attribute

and

is

directly

connected

to

its

entity

(

rectangle

).

**Composite**

**Attribute**

If

the

attributes

are

composite,

they

are

further

divided

in

a

tree

like

structure.

Every

node

is

then

connected

to

its

attribute.

That

is,

composite

attributes

are

represented

by

ellipses

that

are

connected

with

an

ellipse.

**Multivalued**

**Attributes**

Multivalued

attributes

are

depicted

by

double

ellipse.

**Derived**

**Attributes**

Derived

attributes

are

depicted

by

dashed

ellipse.

