Bharath
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Declared as Deemed - to - be - University under section 3 of UGC Act 1956)

173, Agaram Road, Selaiyur, Chennai-600073

BHARATH INSTITUTE OF SCIENCE & TECHNOLOGY

Department of Computer Science &Engineering

SOFTWARE ENGINEERING AND PROGRAM MANAGEMENT PRACTICAL LAB RECORD

SUB CODE: U20CSCJ10

Name: **Section:**

Year: II **Semester: IV** **Batch:2021-2025**

University Register No.

FOR OUTREACH & INCLUSIVITY. 2018

JULY 2023

Bharath

INSTITUTE OF HIGHER EDUCATION AND RESEARCH

(Declared as Deemed - to - be - University under section 3 of UGC Act 1956)

173, Agaram Road, Selaiyur, Chennai-600073

Department of Computer Science & Engineering

SUB CODE : U20CSCJ10

Name: **Section:**

Year: II **Semester: IV** **Batch:2021-2025**

University Register No.

Certified that this is the bonafide record of work done by the above student in the SOFTWARE ENGINEERING AND PROGRAM MANAGEMENT LAB (Sub Code: U20CSCJ10)

Signature of Faculty-In-Charge **Signature of Head of Department**

Submitted for the practical Examination held on.....at Bharath Institute of Higher Education & Research, Chennai-73.

Signature of Internal Examiner **Signature of External Examiner**

INDEX

EX NO. 1 SOFTWARE REQUIREMENT SPECIFICATION
DATE:

Aim: Understanding an SRS.

Requirements:

Hardware Requirements:

- PC with 300 megahertz or higher processor clock speed recommended; 233 MHz minimum required.
- 128 megabytes (MB) of RAM or higher recommended (64 MB minimum supported)
- 1.5 gigabytes (GB) of available hard disk space
- CD ROM or DVD Drive
- Keyboard and Mouse (compatible pointing device).

Software Requirements:

Rational Rose, Windows XP,

Theory:

An SRS is basically an organization's understanding (in writing) of a customer or potential client's system requirements and dependencies at a particular point in time (usually) prior to any actual design or development work. It's a two-way insurance policy that assures that both the client and the organization understand the other's requirements from that perspective at a given point in time.

The SRS document itself states in precise and explicit language those functions and capabilities a software system (i.e., a software application, an e Commerce Web site, and so on) must provide, aswell as states any required constraints by which the system must abide. The SRS also functions as a blueprint for completing a project with as little cost growth as possible. The SRS is often referredto as the "parent" document because all subsequent project management documents, such as design specifications, statements of work, software architecture specifications, testing and validation plans,and documentation plans, are related to it.

It's important to note that an SRS contains functional and nonfunctional requirements only; it doesn't offer design suggestions, possible solutions to technology or business issues, or any other information other than what the development team understands the customer's system requirements to be.

A well-designed, well-written SRS accomplishes four major goals:

- It provides feedback to the customer. An SRS is the customer's assurance that the development organization understands the issues or problems to be solved and the software behavior necessary to address those problems. Therefore, the SRS should be written in natural language (versus a formal language, explained later in this article), in an unambiguous

manner that may also include charts, tables, data flow diagrams, decision tables, and so on.

- It decomposes the problem into component parts. The simple act of writing down software requirements in a well-designed format organizes information, places borders around the problem, solidifies ideas, and helps break down the problem into its component parts in an orderly fashion.
- It serves as an input to the design specification. As mentioned previously, the SRS serves asthe parent document to subsequent documents, such as the software design specification and statement of work. Therefore, the SRS must contain sufficient detail in the functional systemrequirements so that a design solution can be devised.
- It serves as a product validation check. The SRS also serves as the parent document for testing and validation strategies that will be applied to the requirements for verification.

SRSs are typically developed during the first stages of "Requirements Development," which is the initial product development phase in which information is gathered about what requirements are needed--and not. This information-gathering stage can include onsite visits, questionnaires, surveys, interviews, and perhaps a return-on-investment (ROI) analysis or needs analysis of the customer or client's current business environment. The actual specification, then, is written after the requirements have been gathered and analyzed.

SRS should address the following

The basic issues that the SRS shall address are the following:

a. **Functionality**. What is the software supposed to do?
b. **External interfaces**. How does the software interact with people, the system‘s hardware, other hardware, and other software?
c. **Performance**. What is the speed, availability, response time, recovery time of various software functions, etc.?
d. **Attributes**. What are the portability, correctness, maintainability, security, etc. considerations?
e. **Design constraints imposed on an implementation**. Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) etc.?

Characteristics of a good SRS

An SRS should be

1. Correct
2. Unambiguous
3. Complete
4. Consistent
5. Ranked for importance and/or stability
6. Verifiable
7. Modifiable
8. Traceable

Correct - This is like motherhood and apple pie. Of course you want the specification to be correct.

No one writes a specification that they know is incorrect. We like to say - "Correct and Ever Correcting." The discipline is keeping the specification up to date when you find things that are notcorrect.

Unambiguous - An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation. Again, easier said than done. Spending time on this area prior to releasing the SRS can be a waste of time. But as you find ambiguities - fix them.

Complete - A simple judge of this is that it should be all that is needed by the software designers tocreate the software.

Consistent - The SRS should be consistent within itself and consistent to its reference documents. If you call an input "Start and Stop" in one place, don't call it "Start/Stop" in another.

Ranked for Importance - Very often a new system has requirements that are really marketing wishlists. Some may not be achievable. It is useful provide this information in the SRS.

Verifiable - Don't put in requirements like - "It should provide the user a fast response." Another ofmy favorites is - "The system should never crash." Instead, provide a quantitative requirement like:"Every key stroke should provide a user response within 100 milliseconds."

Modifiable - Having the same requirement in more than one place may not be wrong - but tends tomake the document not maintainable.

Traceable - Often, this is not important in a non-politicized environment. However, in most organizations, it is sometimes useful to connect the requirements in the SRS to a higher level document. Why do we need this requirement?

A sample of basic SRS Outline

1. Introduction
1.1 Purpose
1.2 Document conventions
1.3 Intended audience
1.4 Additional information
1.5 Contact information/SRS team members
1.6 References

2. Overall Description
2.1 Product perspective
2.2 Product functions
2.3 User classes and characteristics
2.4 Operating environment
2.5 User environment
2.6 Design/implementation constraints
2.7 Assumptions and dependencies

3. External Interface Requirements

3.1 User interfaces
3.2 Hardware interfaces
3.3 Software interfaces
3.4 Communication protocols and interfaces

4. System Features
4.1 System feature A
4.1.1 Description and priority
4.1.2 Action/result
4.1.3 Functional requirements
4.2 System feature B

5. Other Nonfunctional Requirements
5.1 Performance requirements
5.2 Safety requirements
5.3 Security requirements
5.4 Software quality attributes
5.5 Project documentation
5.6 User documentation

6. Other Requirements
Appendix A:
Terminology/Glossary/Definiti
onons list Appendix B: To be
determined

Conclusion: The SRS was made successfully by following the steps described above.

EX NO. 2 ENTITY RELATIONSHIP DIAGRAM

DATE:

Aim: To draw a sample ENTITY RELATIONSHIP DIAGRAM diagram for real project or system.

Theory:

Entity Relationship Diagrams are a major data modelling tool and will help organize the data in your project into entities and define the relationships between the entities. This process has proved to enable the analyst to produce a good database structure so that the data can be stored and retrieved in a most efficient manner.

Entity:

A data entity is anything real or abstract about which we want to store data. Entity types fall into five classes: roles, events, locations, tangible things or concepts. E.g. employee, payment, campus, book. Specific examples of an entity are called **instances.** E.g. the employee John Jones ,Mary Smith's payment, etc.

Relationship:

A data relationship is a natural association that exists between one or more entities. E.g. Employees process payments. **Cardinality** defines the number of occurrences of one entity for a single occurrence of the related entity. E.g. an employee may process many payments but might not process any payments depending on the nature of her job.

Attribute

A data attribute is a characteristic common to all or most instances of a particular entity. Synonyms include property, data element, field. E.g. Name, address, Employee Number, pay rate are all attributes of the entity employee. An attribute or combination of attributes that uniquely identifies one and only one instance of an entity is called a **primary key** or **identifier**. E.g. Employee Number is a primary key for Employee.

AN ENTITY RELATIONSHIP DIAGRAM METHODOLOGY: (One way of doing it).

1. Identify Entities	Identify the roles, events, locations, tangible things or concepts aboutwhich the end-users want to store data.
2. Find Relationships	Find the natural associations between pairs of entities using a relationship matrix.
3. Draw Rough ERD	Put entities in rectangles and relationships on line segments connecting the entities.
4. Fill in Cardinality	Determine the number of occurrences of one entity for a single occurrence of the related entity.

5. Define Primary Keys	Identify the data attribute(s) that uniquely identify one and only oneoccurrence of each entity.
6. Draw Key-Based ERD	Eliminate Many-to-Many relationships and include primary and foreign keys in each entity.
7. Identify Attributes	Name the information details (fields) which are essential to the system under development.
8. Map Attributes	For each attribute, match it with exactly one entity that it describes.
9. Draw fully attributed ERD	Adjust the ERD from step 6 to account for entities or relationships discovered in step 8.
10. Check Results	Does the final Entity Relationship Diagram accurately depict the system data?

A SIMPLE EXAMPLE

A company has several departments. Each department has a supervisor and at least one employee. Employees must be assigned to at least one, but possibly more departments. At least one employee is assigned to a project, but an employee may be on vacation and not assigned to any projects. The important data fields are the names of the departments, projects, supervisors and employees, as well as the supervisor and employee number and a unique project number.

1. Identify Entities

The entities in this system are Department, Employee, Supervisor and Project. One is tempted to make Company an entity, but it is a false entity because it has only one instance in this problem. True entities must have more than one instance.

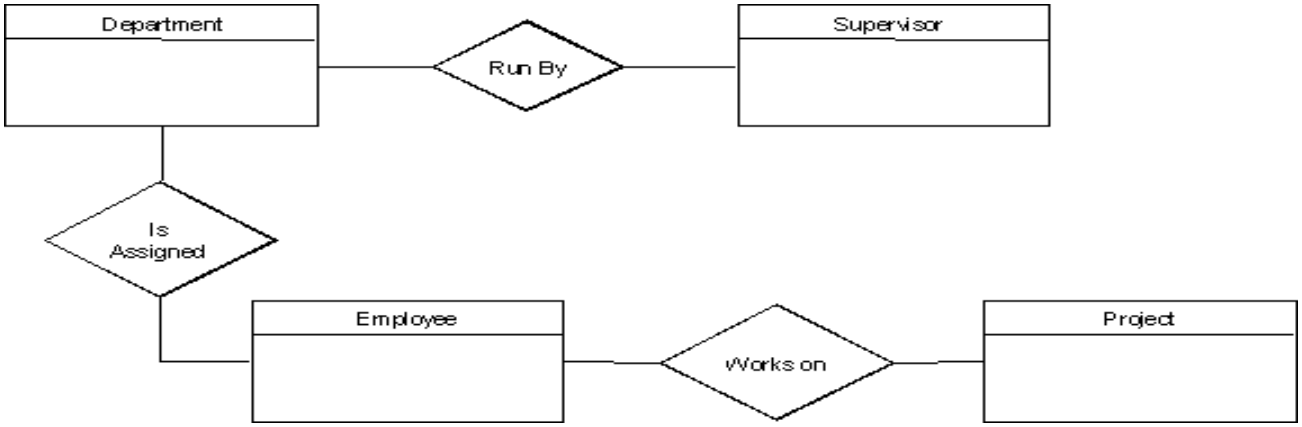
2. Find Relationships

We construct the following Entity Relationship Matrix:

	Department	**Employee**	**Supervisor**	**Project**
Department		is assigned	run by	
Employee	belongs to			works on
Supervisor	runs			
Project		uses		

3. Draw Rough ERD

We connect the entities whenever a relationship is shown in the entity Relationship Matrix.

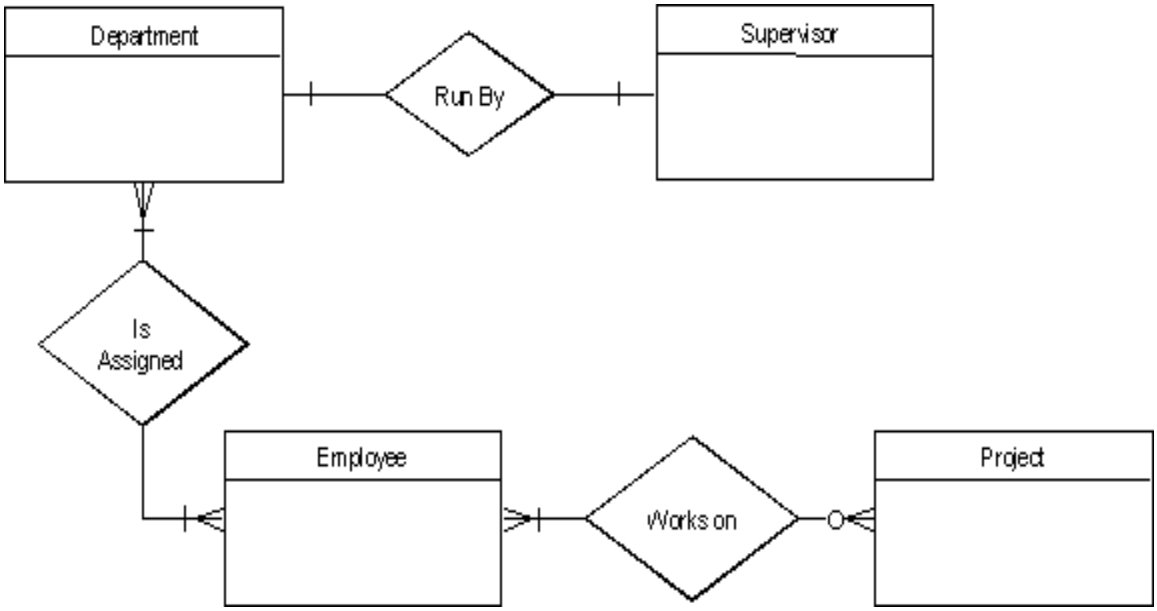


4. Fill in Cardinality

From the description of the problem we see that:

- Each department has exactly one supervisor.
- A supervisor is in charge of one and only one department.
- Each department is assigned at least one employee.
- Each employee works for at least one department.
- Each project has at least one employee working on it.

An employee is assigned to 0 or more projects.



5. Define Primary Keys

The primary keys are Department Name, Supervisor Number, Employee Number, and Project Number.

6. Draw Key-Based ERD

There are too many-to-many relationships in the rough ERD above, between Department and Employee and between Employee and Project. Thus we need the associative entities Department-Employee and Employee-Project. The primary key for Department-Employee is the concatenated

Key Department Name and Employee Number. The primary key for Employee- Project is the concatenated key Employee Number and Project Number.

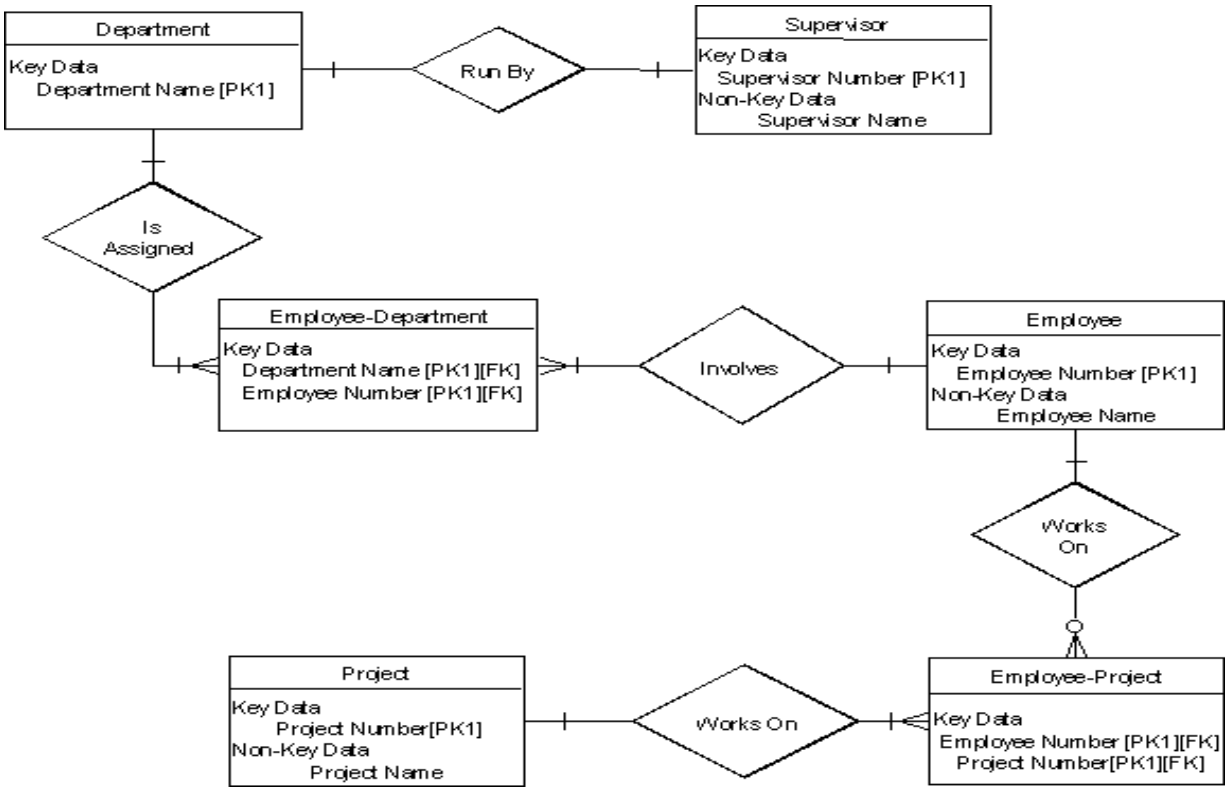
7. Identify Attributes

The only attributes indicated are the names of the departments, projects, supervisors and employees, as well as the supervisor and employee NUMBER and a unique project number.

8. Map Attributes

Attribute	Entity	Attribute	Entity
Department Name	Department	Supervisor Number	Supervisor
Employee Number	Employee	Supervisor Name	Supervisor
Employee Name	Employee	Project Name	Project
		Project Number	Project

9. Draw Fully Attributed ERD



10. Check Results

The final ERD appears to model the data in this system well.

FURTHER DISCUSSION:

Step 1. Identify Entities

A data entity is anything real or abstract about which we want to store data. Entity types fall into five classes: roles, events, locations, tangible things, or concepts. The best way to identify entities is to ask the system owners and users to identify things about which they would like to capture, store and produce information. Another source for identifying entities is to study the forms, files, and reports generated by the current system. E.g. a student registration form would refer to Student (a role), but also Course (an event), Instructor (a role), Advisor (a role), Room (allocation), etc.

Step 2. Find Relationships

There are natural associations between pairs of entities. Listing the entities down the left column and across the top of a table, we can form a relationship matrix by filling in an active verb at the intersection of two entities which are related. Each row and column should have at least one relationship listed or else the entity associated with that row or column does not interact with the rest of the system. In this case, you should question whether it makes sense to include that entity in the system.

Step 3. Draw Rough ERD

Using rectangles for entities and lines for relationships, we can draw an Entity Relationship Diagram (ERD).

Step 4. Fill in Cardinality

At each end of each connector joining rectangles, we need to place a symbol indicating the minimum and maximum number of instances of the adjacent rectangle there are for one instance of the rectangle at the other end of the relationship line. The placement of these numbers is often confusing. The first symbol is either 0 to indicate that it is possible for no instances of the entity joining the connector to be related to a given instance of the entity on the other side of the relationship, 1 if at least one instance is necessary or it is omitted if more than one instance is required. For example, more than one student must be enrolled in a course for it to run, but it is possible for no students to have a particular instructor (if they are on leave).

The second symbol gives the maximum number of instances of the entity joining the connector for each instance of the entity on the other side of the relationship. If there is only one such instance, this symbol is 1. If more than 1, the symbol is a crow's foot opening towards the rectangle.

If you read it like a sentence, the first entity is the subject, the relationship is the verb, the cardinalityafter the relationship tells how many direct objects (second entity) there are.

Step 5. Define Primary Keys

For each entity we must find a unique primary key so that instances of that entity can be distinguished from one another. Often a single field or property is a primary key (e.g. a Student ID). Other times the identifier is a set of fields or attributes (e.g. a course needs a department identifier, a course number, and often a section number; a Room needs a Building Name and a Room Number). When the entity is written with all its attributes, the primary key is underlined.

Step 6. Draw Key-Based ERD

Looking at the Rough Draft ERD, we may see some relationships which are non-specific or many-to-many. I.e., there are crow's feet on both ends of the relationship line. Such relationships spell trouble later when we try to implement the related entities as data stores or data files, since each record will need an indefinite number of fields to maintain the many-to-many relationship.

Fortunately, by introducing an extra entity, called an associative entity for each many-to-many relationship, we can solve this problem. The new associative entity's name will be the hyphenation of the names of the two originating entities. It will have a concatenated key consisting of the keys of these two entities. It will have a 1-1 relationship with each of its parent entities and each parent will have the same relationship with the associative entity that they had with each other before we introduced the associative entity. The original relationship between the parents will be deleted fromthe diagram.

The key-based ERD has no many-to-many relationships and each entity has its primary and foreignkeys listed below the entity name in its rectangle.

Step 7. Identify Attributes

A data attribute is a characteristic common to all or most instances of a particular entity. In this step we try to identify and name all the attributes essential to the system we are studying without trying to match them to particular entities. The best way to do this is to study the forms, files and reports currently kept by the users of the system and circle each data item on the paper copy.

Cross out those which will not be transferred to the new system, extraneous items such as signatures, and constant information which is the same for all instances of the form (e.g. your company name and address). The remaining circled items should represent the attributes you need. You should always verify these with your system users. (Sometimes forms or reports are out of date.)

Step 8. Map Attributes

For each attribute we need to match it with exactly one entity. Often it seems like an attribute should go with more than one entity (e.g. Name). In this case you need to add a modifier to the attribute name to make it unique (e.g. Customer Name, Employee Name, etc.) or determine which entity an attribute "best' describes. If you have attributes left over without corresponding entities, you may have missed an entity and its corresponding relationships. Identify these missed entities and add them to the relationship matrix now.

Step 9. Draw Fully-Attributed ERD

If you introduced new entities and attributes in step 8, you need to redraw the entity relationship diagram. When you do so, try to rearrange it so no lines cross by putting the entities with the most relationships in the middle. If you use a tool like Systems Architect, redrawing the diagram is relatively easy.

Even if you have no new entities to add to the Key-Based ERD, you still need to add the attributes to the Non-Key Data section of each rectangle. Adding these attributes automatically puts them in the repository, so when we use the entity to design the new system, all its attributes will be available.

Step 10. Check Results

Look at your diagram from the point of view of a system owner or user. Is everything clear? Check through the Cardinality pairs. Also, look over the list of attributes associated with each entity to seeif anything has been omitted.

Conclusion: The entity relationship diagram was made successfully by following the steps described above.

EX NO. 3 STUDENT COURSE REGISTRATION SYSTEM
DATE:

1.0 PROBLEM DEFINITION

To build software that automates the student's course registration.

1.1 This is proposed for automating the student's course registration. While the student joins any educational institution his admission is made on the basis of his previous records.

1.2 The students who wish to the institution must be given with the available course details.

1.3 The student is allotted with a seat in the institution based on the marks that he scored in the institution he studied previously. After the confirmation of his joining the student must be given with new identity and records as per the institution.

2.0 SYSTEM REQUIREMENT SPECIFICATION

2.1 THE OVERALL DESCRIPTION

2.1.1 This is proposed for automating the student's course registration. While the student joins any educational institution his admission is made on the basis of his previous records.

2.1.2 The students who wish to the institution must be given with the available course details.

2.2 PRODUCT PERSPECTIVE

2.2.1 Hardware interfaces

2.2.1.1 Hard disk: The database connectivity requires a hardware configuration that is on-line. This makes it necessary to have a fast database system running on high rpm hard disk permitting complete data redundancy and back-up systems to support the primary goal of reliability.

2.2.1.2 The system must interface with the standard output device, keyboard and mouse to interact with this software.

2.2.2 Software interfaces

2.2.2.1.1 Back End: MS-Access

2.2.2.1.2 Front End: Microsoft Visual Basic 6.0

223 Memory Constraints

2.2.1.3.1 No specific constraints on memory.

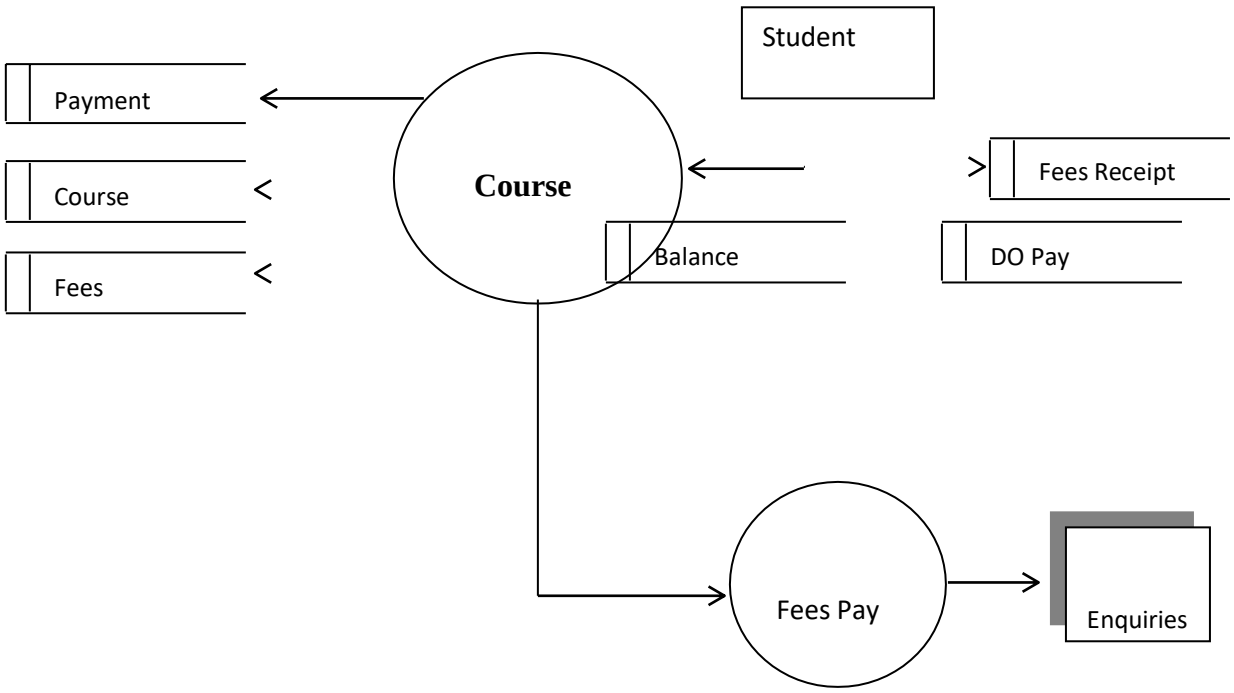
2.3 FRONT – END DESCRIPTION

The Student Course registration system is automated system where the user can register the student for various courses.

2.4 BACK – END DESCRIPTION

The Student Course registration system consists of two tables. One contains the student details such as the name, id number that is the password, address, and date of birth. The Course details consist of the title of the course, code of the course, available seats.

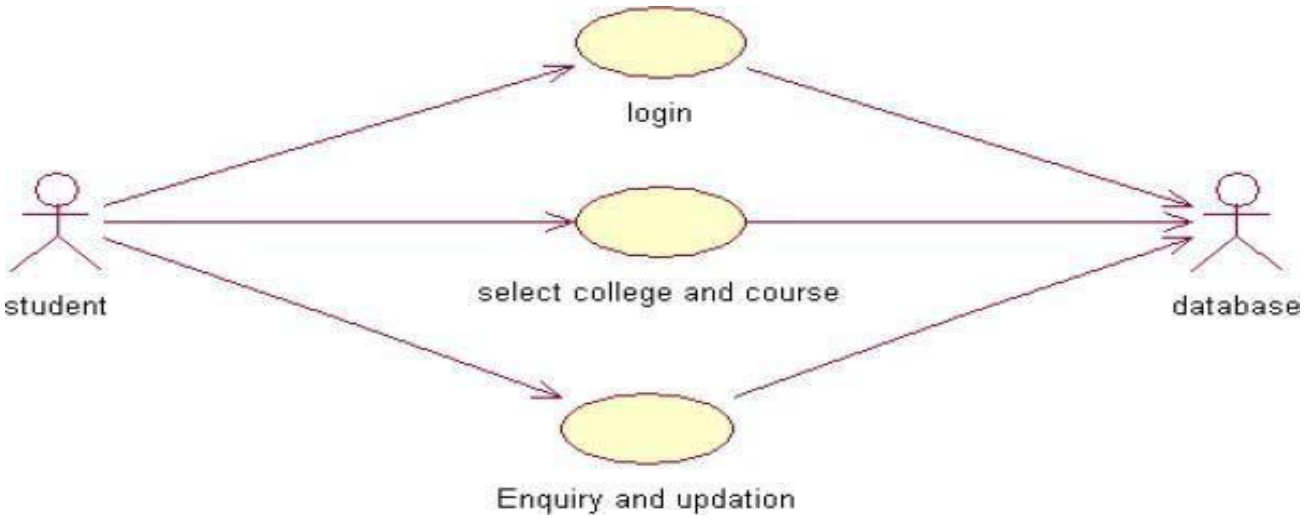
2.5 DATA FLOW DIAGRAM:



UML DIAGRAMS

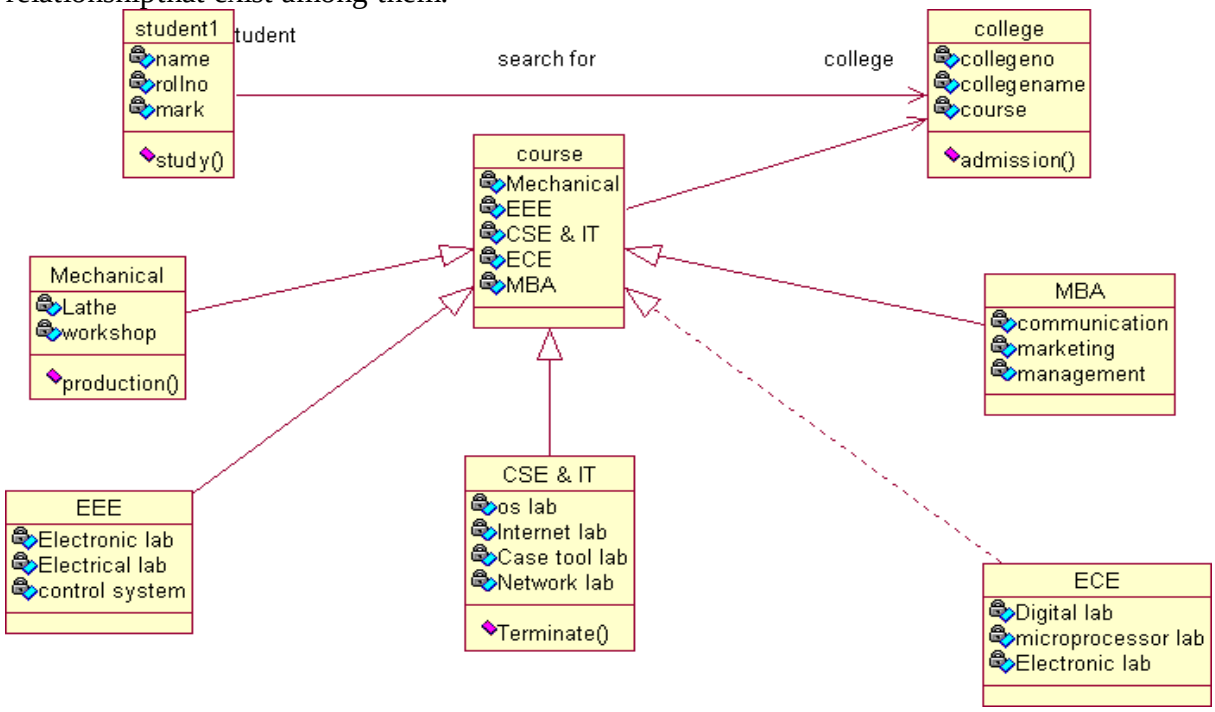
1. USERCASE DIAGRAM

- Use case is a sequence of transaction in a system whose task is to yield result of measurablevalue to individual author of the system
- Use case is a set of scenarios together by a common user goal
- A scenario is a sequence of step describing as interaction between a user and system



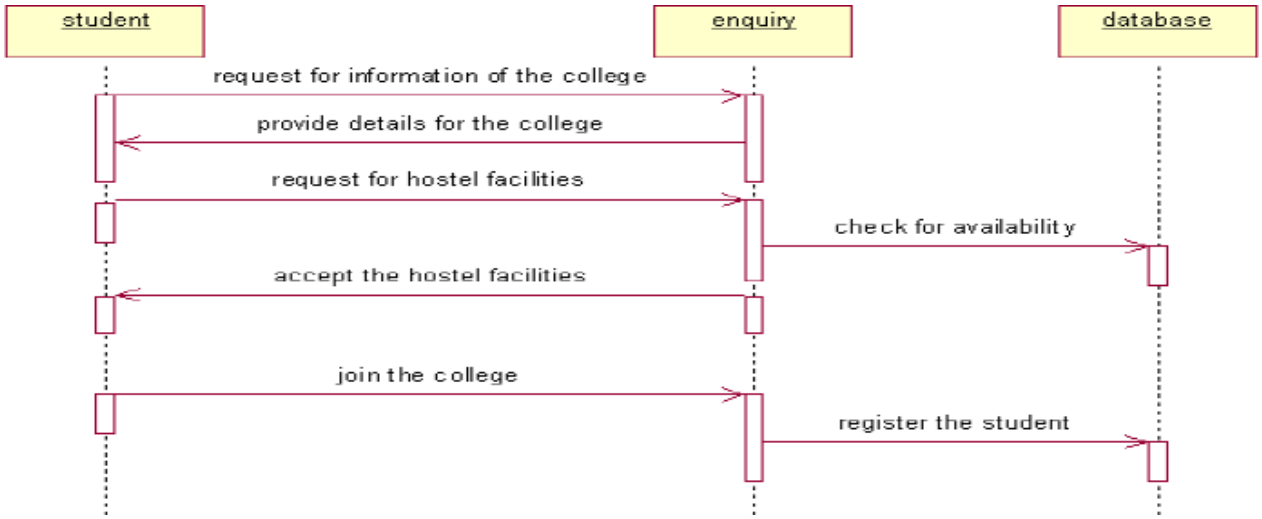
2. CLASS DIAGRAM:

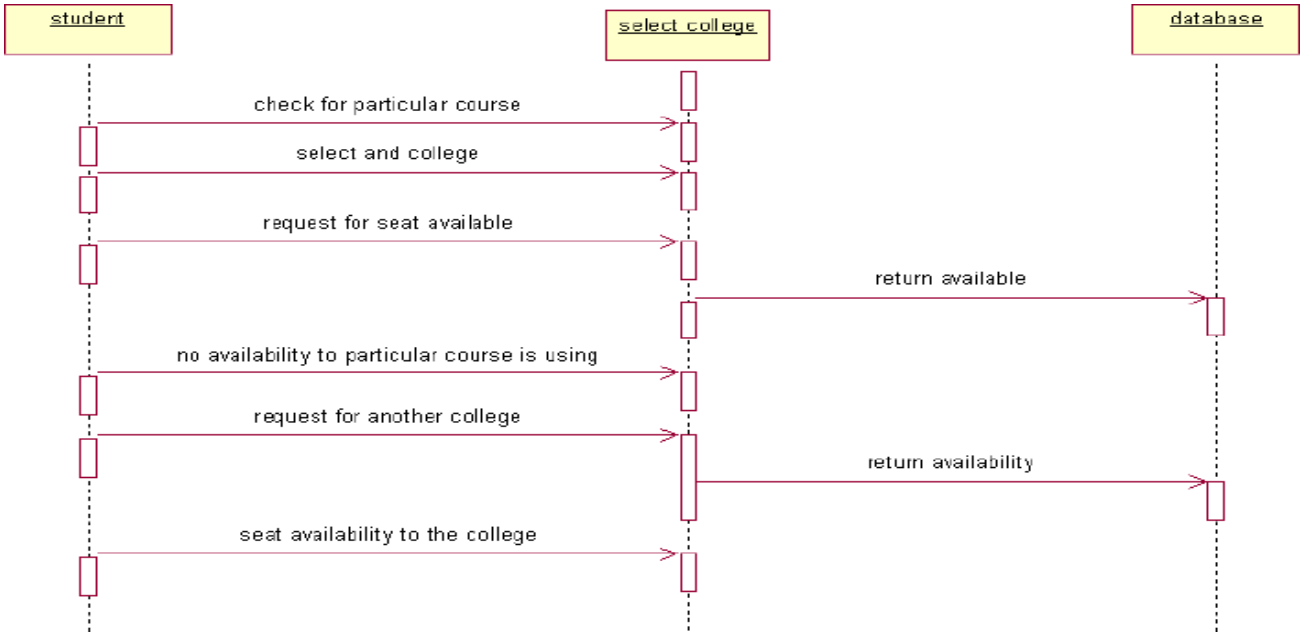
A class diagram describes the type of objectors in the system the various kinds of static relationshipthat exist among them.

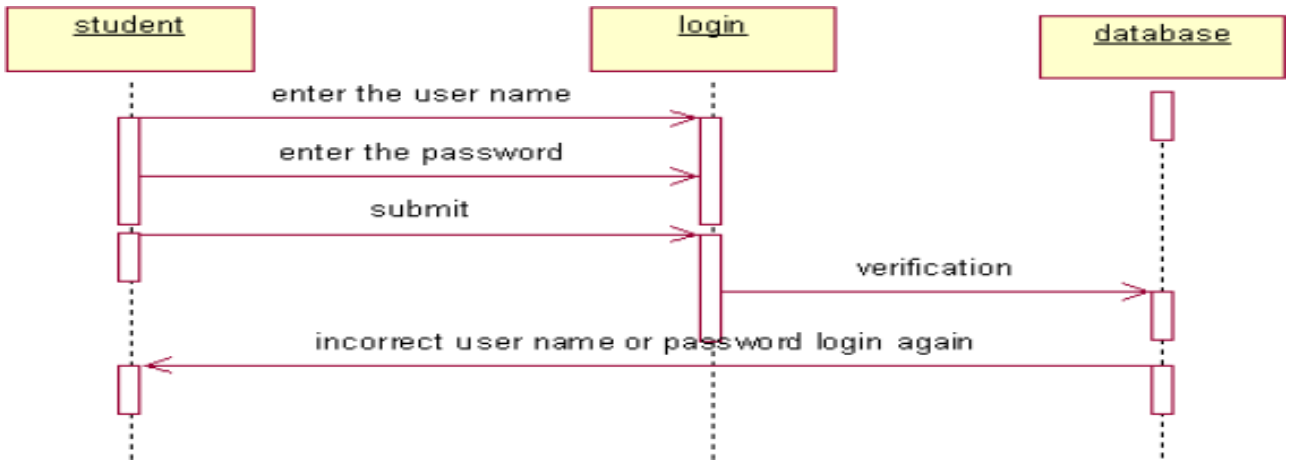


3. SEQUENCE DIAGRAM

A sequence diagram is one that includes the object of the projects and tells the lifetimes and also various action performed between objects.

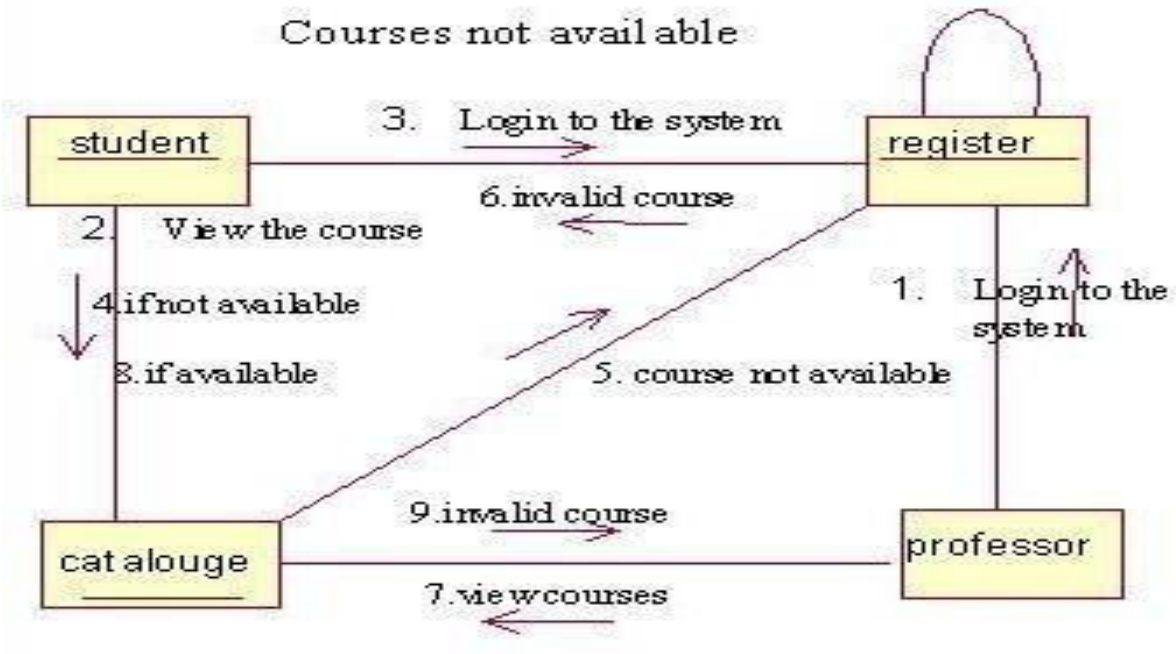

student
enquiry
database
request for information of the college
provide details for the college
request for hostel facilities
check for availability
accept the hostel facilities
join the college
register the student


student
select college
database
check for particular course
select and college
request for seat available
return available
no availability to particular course is using
request for another college
return availability
seat availability to the college


student
login
database
enter the user name
enter the password
submit
verification
incorrect user name or password login again

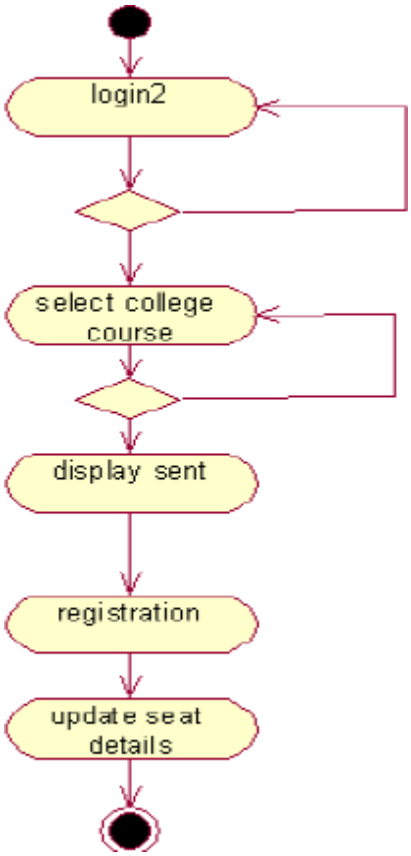
4. COLLOBORATIION DIAGRAM

It is same as the sequence diagram that involved the project with the only difference that we give the project with the only difference that we give sequence number to each process.



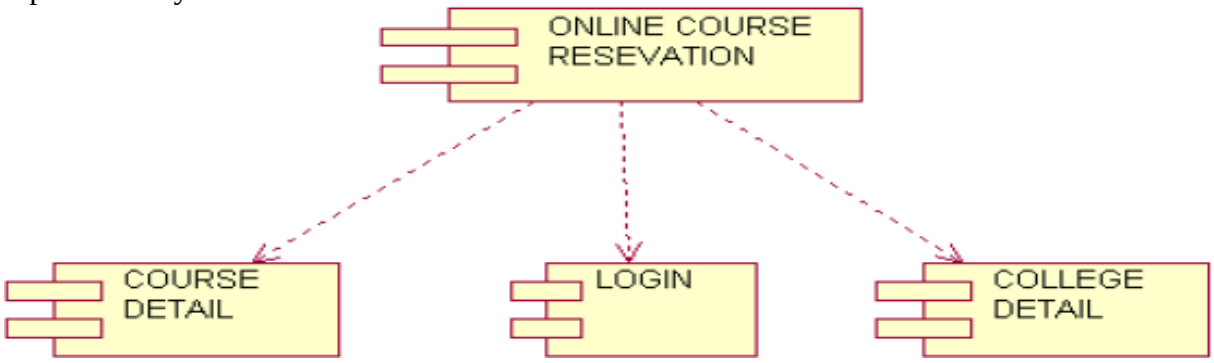
5. ACTIVIY DIAGRAM

It includes all the activities of particular project and various steps using join and forks.



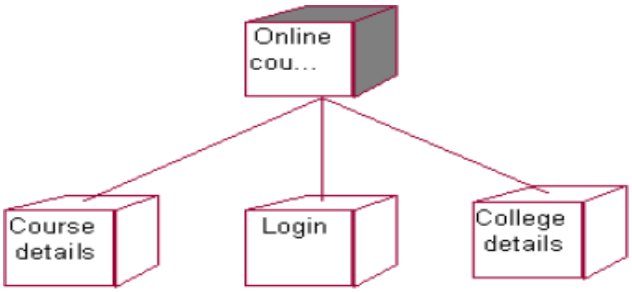
6. COMPONENT DIAGRAM

The component diagram is represented by figure dependency and it is a graph of design of figure dependency. The component diagram's main purpose is to show the structural relationships between the components of a systems. It is represented by boxed figure. Dependencies are represented by communication association



7. DEPLOYMENT DIAGRAM

It is a graph of nodes connected by communication association. It is represented by a three dimensional box. A deployment diagram in the unified modeling language serves to model the physical deployment of artifacts on deployment targets. Deployment diagrams show "the allocation of artifacts to nodes according to the Deployments defined between them. It is represented by 3-dimensional box. Dependencies are represented by communication association. The basic elementof a deployment diagram is a node of two types



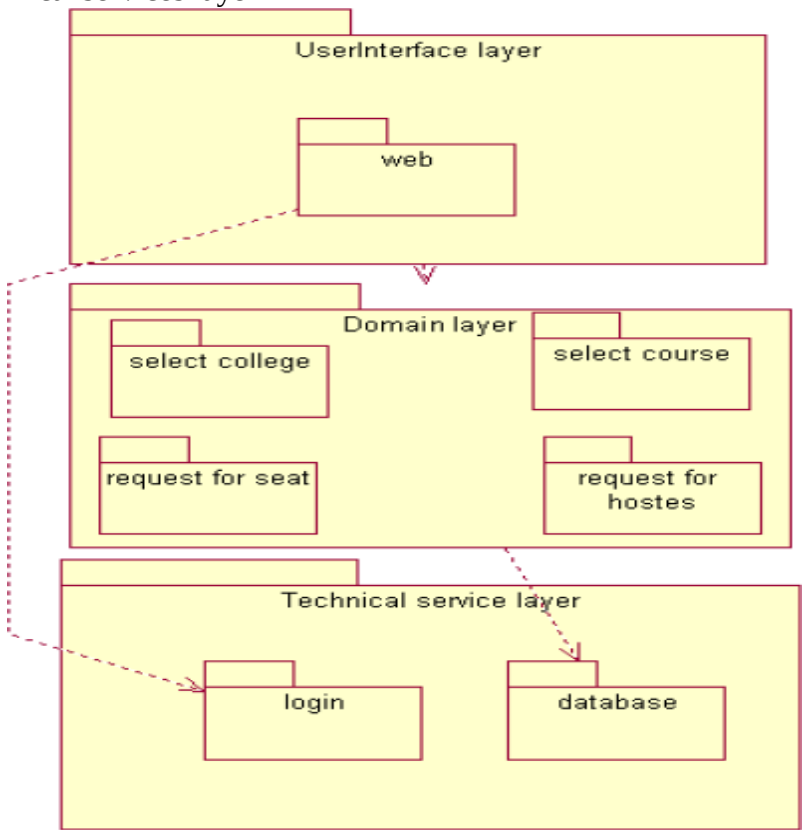
8. PACKAGE DIAGRAM

A package diagram is represented as a folder shown as a large rectangle with a top attached to its upper left corner. A package may contain both sub ordinate package and ordinary model elements. All ump models and Diagrams are organized into package. A package diagram in unified modeling language that depicts the dependencies between the packages that make up model. A Package Diagram (PD) shows a grouping of elements in the OO model, and is a Cradle extension to UML. PDs can be used to show groups of classes in Class Diagrams (CDs), groups of components or processes in Component Diagrams (CPDs), or groups of processors in Deployment Diagrams (DPDs).

There are three types of layer. They are

- User interface layer
- Domain layer

- Technical services layer



Conclusion: The student course registration system was designed successfully by following thesteps described above

EX NO. 4 LIBRARY MANAGEMENT SYSTEM

DATE:

1.0 PROBLEM DEFENITION

The library management system is software, which automates the job of a librarian.

1.1 The user can inquire about the availability of a book in which he can search by entering the author's name or by entering the title of the book.

1.2 The user can borrow a book. He must provide the username and the card number, which is unique and confidential to each user. By confirming the authenticity of a user, the library management system provides information about the number of books already borrowed by the user and by referring to the database whether the user can borrow books or not. The library management system allows the user to enter the title and the author of the book and hence issues the book if it is available.

1.3 By entering the user details and the book details the user can return the borrowed book.

2.0 SYSTEM REQUIREMENT SPECIFICATION

2.1 INTRODUCTION

2.1.1 Purpose

2.1.1.1 The purpose of this SRS is to describe the requirements involved in developing a Library management system.

2.1.1.2 The intended audience is any person, who wants to inquire, borrow and return the books.

2.1.2 Scope

2.1.2.1 The product is titled Library Management System.

2.1.2.2 The product will perform the following tasks

2.1.2.2.1 Enquire about the availability of books.

2.1.2.2.2 Borrow books if available.

2.1.2.2.3 Return the borrowed books.

2.1.3 Definitions, Acronyms and Abbreviations

2.1.3.1 DDBMS – Database Management System.

2.1.4 References

2.1.4.1 IEEE standard 830-1998 recommended practice for Software Requirements Specifications-Description.

2.1.5 Overview

2.1.5.1 The SRS contains an analysis of the requirements necessary to help easy design.

2.1.5.2 The overall description provides interface requirements for the Library Management System, product perspective, hardware interfaces, software interfaces, communication interface, memory constraints, product functions, user characteristics and other constraints.

2.1.5.3 Succeeding pages illustrate the characteristics of typical naïve users accessing the system along with legal and functional constraints enforced that affect Library Management System in any fashion.

2.2 THE OVERALL DESCRIPTION

2.2.1 Product perspective

2.2.1.1 Hardware interfaces

2.2.1.1.1 Hard disk: The database connectivity requires a hardware configuration that is on-line. This makes it necessary to have a fast database system running on high rpm hard disk permitting complete data redundancy and back-up systems to support the primary goal of reliability.

2.2.1.1.2 The system must interface with the standard output devise, keyboard and mouse to interact with this software.

2.2.1.2 Software interfaces

2.2.1.2.1 Back End: MS-Access

2.2.1.2.2 Front End: Microsoft Visual Basic 6.0

2.2.1.3 Memory Constraints

2.2.1.3.1 No specific constraints on memory.

2.2.1.4 Operations

2.2.1.4.1 The software allows three modes of operations

2.2.1.4.2 Enquire about the availability and status of books.

2.2.2 Product Functions

2.2.2.1.1 Enquire about the availability and status of books.

2.2.2.1.2 Search the availability of book by entering the title of the book.

2.2.2.1.3 Search the availability of book by entering the author of the book.

2.2.2.1.4 The software validates the authentic user by extracting their user name and password.

2.2.2.1.5 After the validation of the user software allows the user to borrow a maximum of three books based on the number of books which were already borrowed.

2.2.2.1.6 After the validation of the user software allows the user to return the books, which where borrowed?

2.2.3 User characteristics

2.2.3.1 The intended users of this software need not have specific knowledge as to what is the internal operation of the system. Thus the end user is at a high level of abstraction that allows easier, faster operation and reduces the knowledge requirement of end user

2.2.3.2 The Product is absolutely user friendly, so the intended users can be the naïve users.

2.2.3.3 The product does not expect the user to possess any technical background. Any person who knows to use the mouse and the keyboard can successfully use this product.

2.2.4 Constraints

The user has a unique username and password, there are no options to retrieve a password or username in case it is forgotten or lost hence the useris requited to remember or store the username and password

2.3 SPECIFIC REQIREMENTS

2.3.1 Logical Database Requirements

2.3.1.1 The system should contain databases that include all necessary information for the product to function according to the requirements. These include relations such as user details and book details.

2.3.1.2 The user details refer to the information such as name, card number, no. of books borrowed, the title and the name of the author of the books that were borrowed.

2.3.1.3 The book details refer to the information such as the title of the book, author availability status and the number of copies that is available.

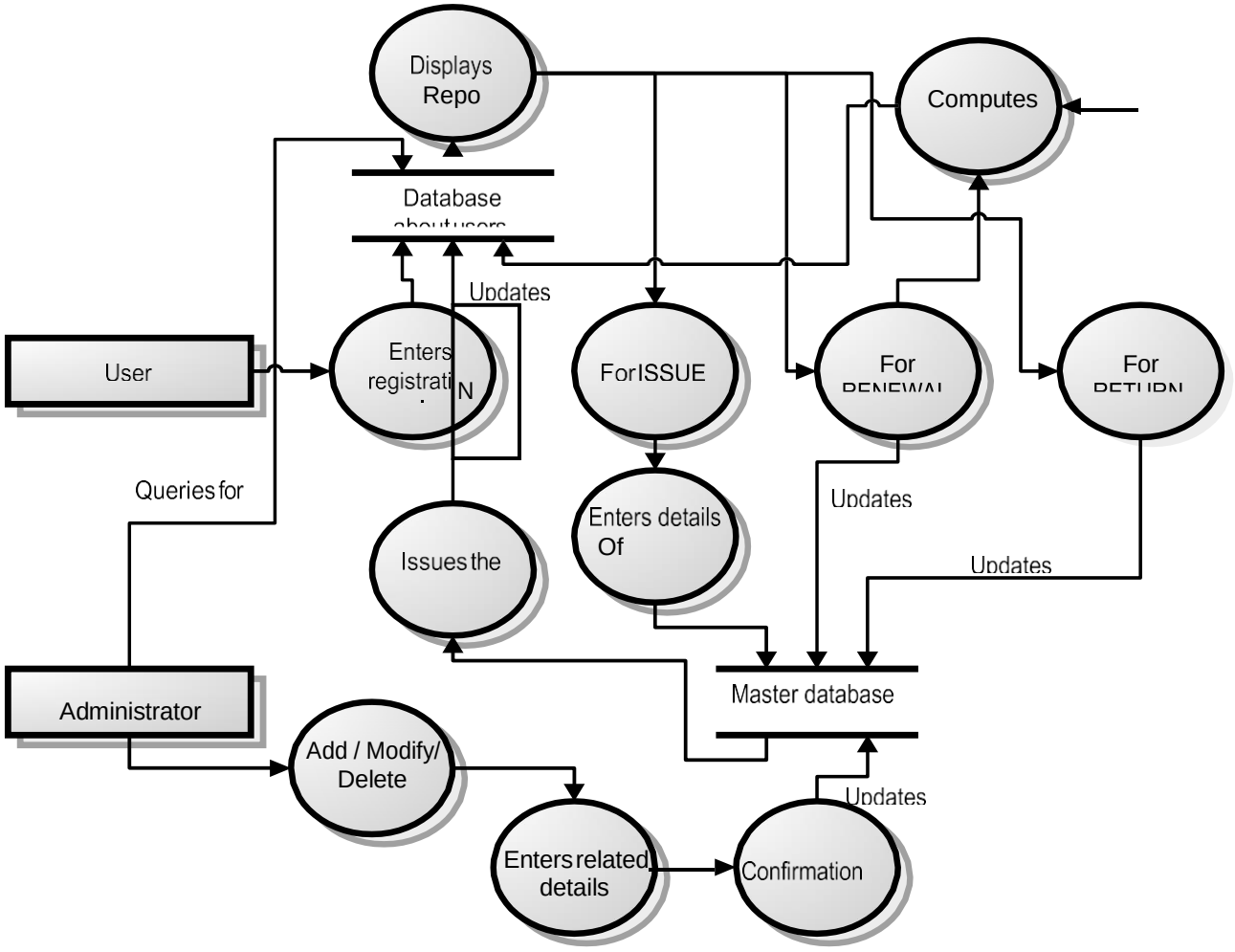
2.4 FRONT – END DESCRIPTION

The library management system is automated library system where the user can search for the book by either entering the details of the book or the author's name. By entering the username and the password the software, by checking the number of books that are already borrowed enables us to borrow a maximum of three books. And by entering the username and password (card number), which is unique, the user can return the books.

2.5 BACK – END DESCRIPTION

The library management system consists of two tables. One contains the student details such as the name, card number that is the password, title and the author of the three books, which could be borrowed. The book details consist of the title of the book, number of copies, author and the availability status.

2.6 DATA FLOW DIAGRAM



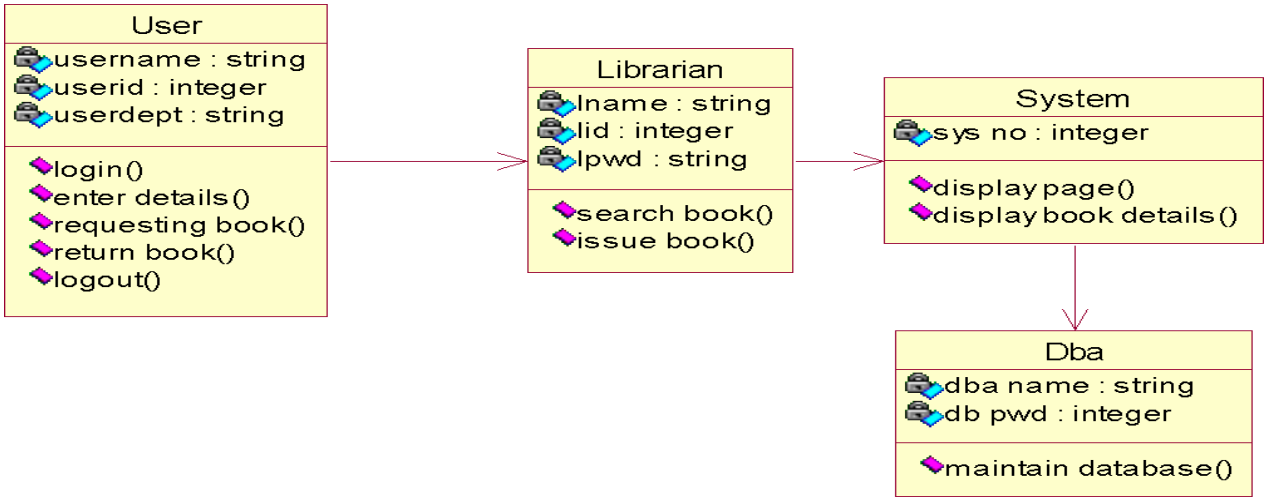
UML DIAGRAMS

1. USERCASE DIAGRAM

Use case is a list of actions or events. Steps typically defining the interactions between a role and asystem to achieve a goal. The use case diagram consists of various functionality performed by actors like user, librarian, system and DBA

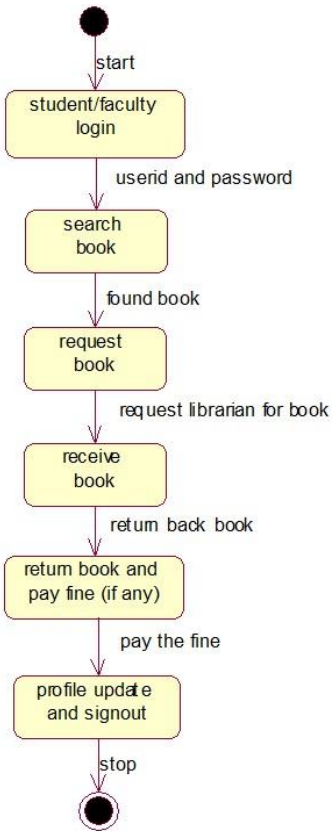
2. CLASS DIAGRAM

A class diagram in the unified modeling language is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations and the relationships among objects. The library management system makes use of the following class user, Librarian, system and DBA.



3. ACTIVITY DIAGRAM

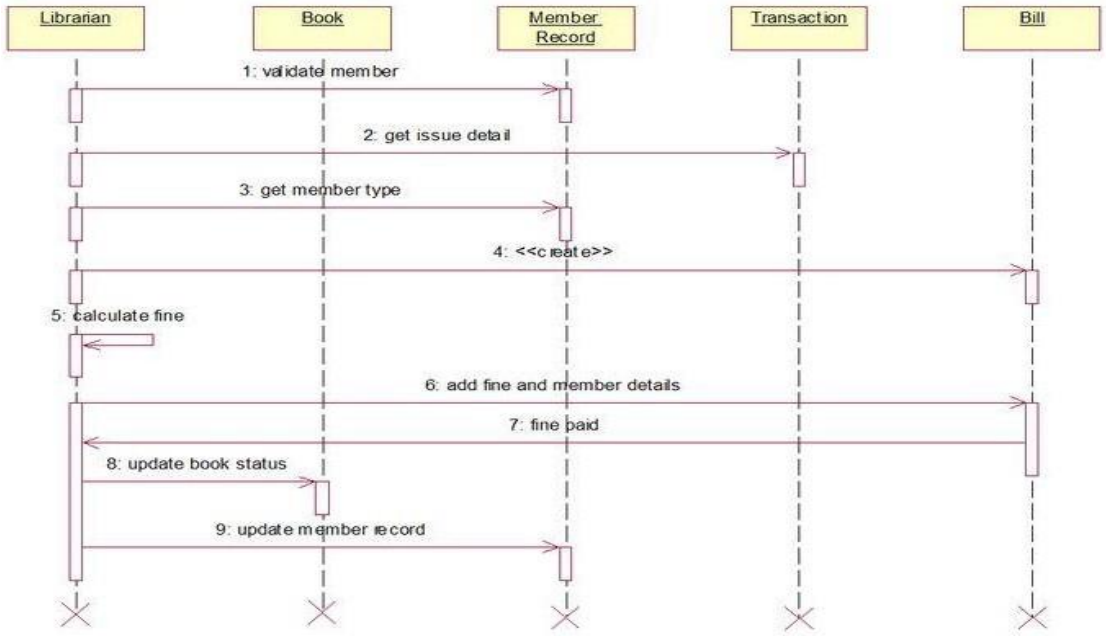
Activity diagram are graphical representation of workflows of stepwise activities and actions with support for choice, iteration and concurrency. Herein the activity diagrams the user login to the system and perform some main activity which is the main key element to the system.



4. SEQUENCE DIAGRAM

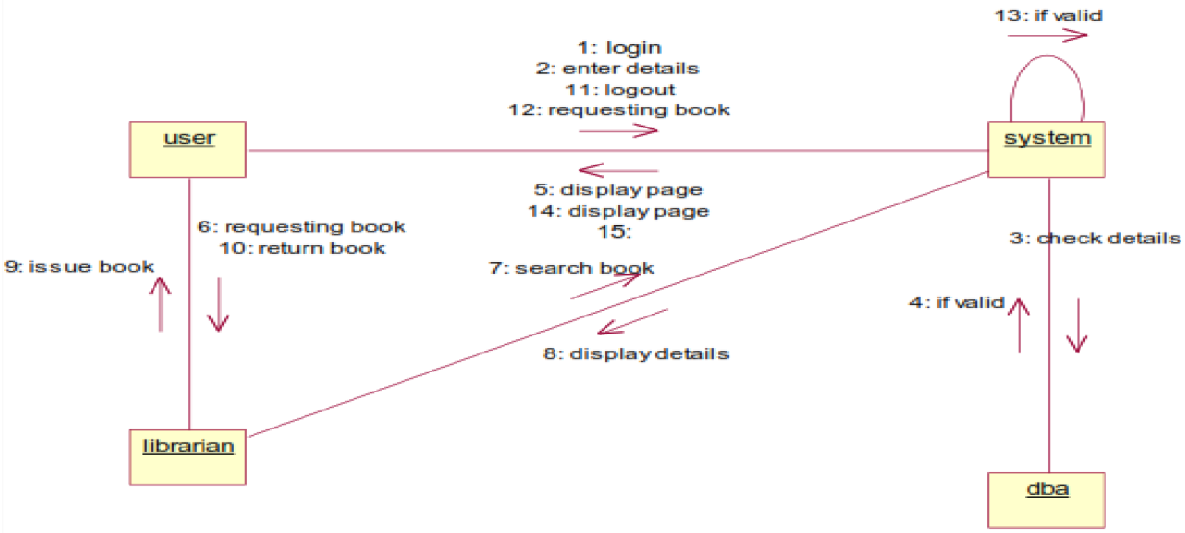
A sequence diagram represents the sequence and interactions of a given use case or scenario. Sequence diagram capture most of the information about the system. It is also representing in order by

which they occur and have the object in the system send message to one another. Here the sequence starts with interaction between user and the system followed by database. Once the book have beenselected the next half of sequence starts between librarian and user followed by database.



5. COLLABORATIO DIAGRAM

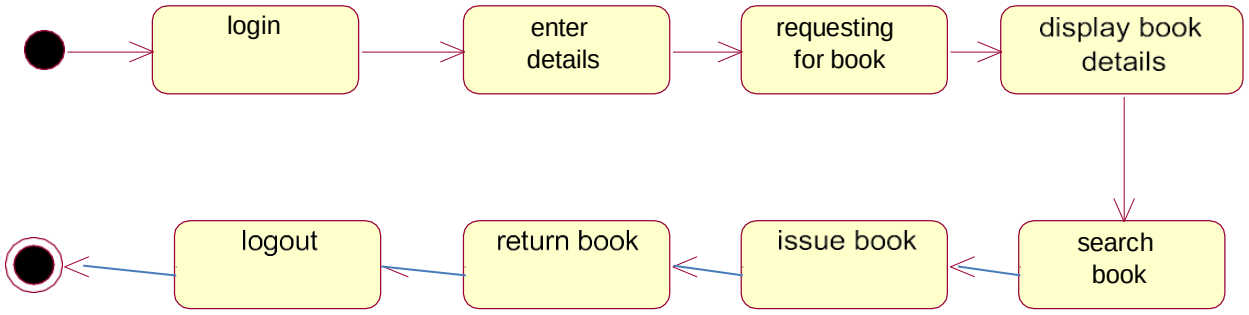
Like sequence diagram collaboration diagrams are also called as interaction diagram. Collaboration diagram convey the same information's as sequence diagram but focus on the objectroles instead of the times that messages are sent. Here the actions between various classes are represented by number format for the case of identification.



6. STATECHART DIAGRAM

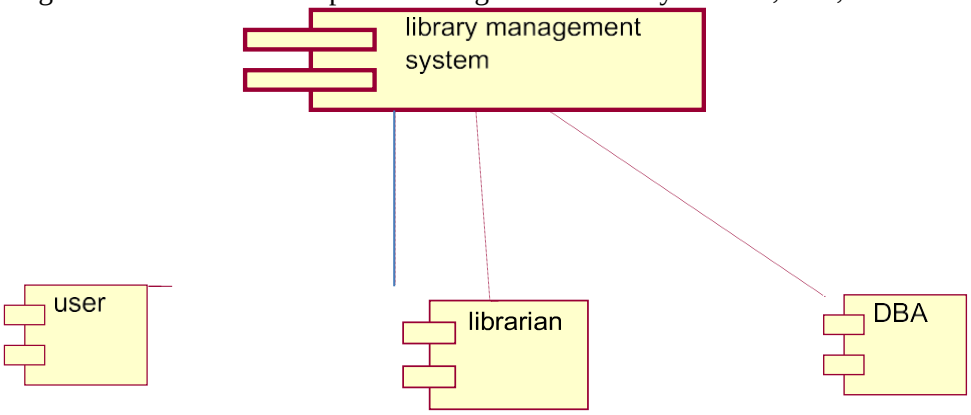
State chart diagram is also called as state machine diagram. The state chart diagram contains the states in the rectangular boxes and the states are indicated by the dot enclosed. The state chart diagram describes the behavior of the system. The state chart diagram involves eight stages such

as login, enter details, requesting for book, display book details, search book, issue book, return book and logout.



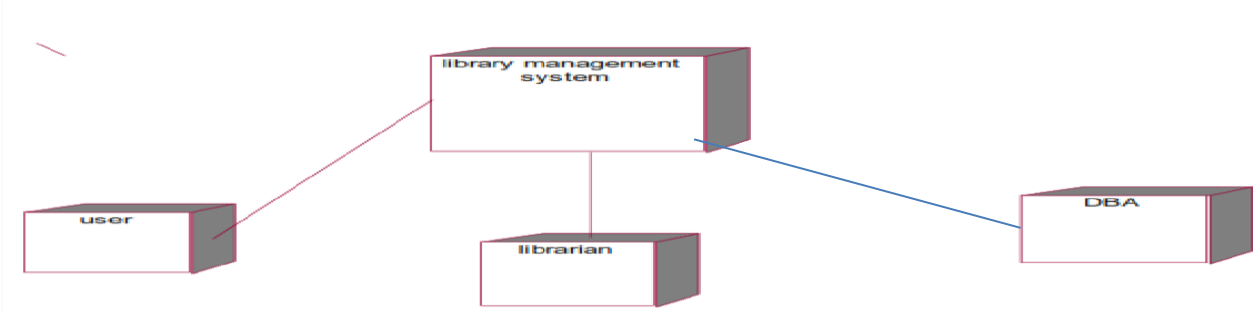
7. COMPONENT DIAGRAM

Component diagram shows the dependencies and interactions between software components. Component diagram carries the most important living actors of the system i.e., user, librarian sand dab.



8. DEPLOYMENT DIAGRAM

Deployment diagram is a structure diagram which shows architecture of the system as deployment of software artifacts to deployment target. It is the graph of nodes connected by communication association. It is represented by three dimensional box. The device node is library management system and execution environment nodes are user, librarian, system and DBA.



Conclusion: The library management system was designed successfully by following the steps described above

EX NO. 5 TRADING SYSTEM
DATE:

1.0 PROBLEM DEFENITION

The requirement to be developed is a Trading system which is required to perform the following functions.

1.1 It should allow user to view stock details of each part which must include part number, part name, demand per day, setup cost, holding cost, lead time, cycle time, EOQ, order amount and availability.

1.2 It should allow the user to take parts required for production.

1.3 It should allow the user to order required units of production.

1.3 It should allow the user to add a new part to the stock.

2.0 SRS DOCUMENT FOR INVENTORY SYSTEM

2.1 INTRODUCTION

2.1.1 Purpose

2.1.1.1 The purpose of this SRS is to describe the requirements involved in developing an Inventory system.

2.1.1.2 The intended audience is any person who wants

2.1.1.2.1 To view stock details.

2.1.1.2.2 To take parts from stock production.

2.1.1.2.3 To order required parts of any unit.

2.1.2 Scope

2.1.2.1 The product is titled Inventory System.

2.1.2.2 The product will perform the following tasks

2.1.2.2.1 It should allow the user to view stock details of each part which must include part number, part name, demand per day, set up cost, holding cost, lead time, cycle time, order amount and availability.

2.1.2.2.2 Allow the user to take required parts from production.

2.1.2.2.3 Allow the user to order required parts of production.

2.1.2.2.4 Allow the user to add a new part to the stock.

2.1.3 References

2.1.3.1 IEEE standard 830-1998 recommended practice for Software Requirements Specifications-Description.

2.1.4 Overview

2.1.4.1 The SRS contains an analysis of the requirements necessary to help easy design.

2.1.4.2 The overall description provides interface requirements for the Payroll Inventory System, product perspective, hardware interfaces, software interfaces, communication interface, memory constraints, product functions, user characteristics and other constraints.

2.1.4.3 Succeeding pages illustrate the characteristics of typical naïve users accessing the system along with legal and functional constraints enforced that affect Inventory System in any fashion.

2.2 THE OVERALL DESCRIPTION

2.2.1 Product perspective

2.2.1.1 Hardware interfaces

2.2.1.1.1 Hard disk: The database connectivity requires a hardware configuration that is on-line. This makes it necessary to have a fast database system (such as any RDBMS) running on high rpm hard-disk permitting complete data redundancy and backup systems to support the primary goal of reliability.

2.2.1.1.2 The system must interface with the standard output device, keyboard and mouse to interact with this software.

2.2.1.2 Software interfaces

2.2.1.2.1 Back End: Oracle

2.2.1.2.2 Front End: Microsoft Visual Basic 6.0

2.2.1.3 Memory Constraints

2.2.1.3.1 No specific constraints on memory.

2.2.1.4 Operations

2.2.1.4.1 The software allows two modes of operations

2.2.1.4.1.1 The user mode allows users to view stock details take parts from stock for production, order for required parts and add a new part.

2.2.2 Product Functions

2.2.2.1 Viewing the stock details

The user must have the access to Up-to-date information about the stocks.

2.2.2.1.1 Part number

2.2.2.1.2 Part Name

2.2.2.1.3 Set up cost

2.2.2.1.4 Demand

2.2.2.1.5 Holding cost

2.2.2.1.6 Lead time

2.2.2.1.7 Cycle time

2.2.2.1.8 Order

2.2.2.1.9 Availability.

2.2.2.2 Taking parts from stock

The user must be able to take parts from stocks by supplying the following information about the stock.

2.2.2.2.1 Part number.

2.2.2.2.2 Required units.

2.2.2.3 Ordering parts

The user can add the required number of parts by giving the following information.

2.2.2.3.1 Part number.

2.2.2.3.2 Required units.

2.2.2.4 Adding a new part

The user must be able to add a new part by providing the following information

2.2.2.4.1 Part number

2.2.2.4.2 Set up cost

2.2.2.4.3 Demand

2.2.2.4.4 Lead time

2.2.2.4.5 Holding cost

2.2.2.4.6 Availability.

2.2.3 User characteristics

2.2.3.1 The intended users of this software need not have specific knowledge as to what is the internal operation of the system. Thus the end user is at a high level of abstraction that allows easier, faster operation and reduces the knowledge requirement of end user

2.2.3.2 The Product is absolutely user friendly, so the intended users can be the naïve users.

2.2.3.3 The product does not expect the user to possess any technical background. Any person who knows to use the mouse and the keyboard can successfully use this product.

2.3 SPECIFIC REQIREMENTS

2.3.1 Logical Database Requirements

2.3.1.1 There are two databases, one which contains all the necessary information about the customer which includes customer ID, customer name, address and phone number. The other database contains information about the stock which includes part number, part name and availability.

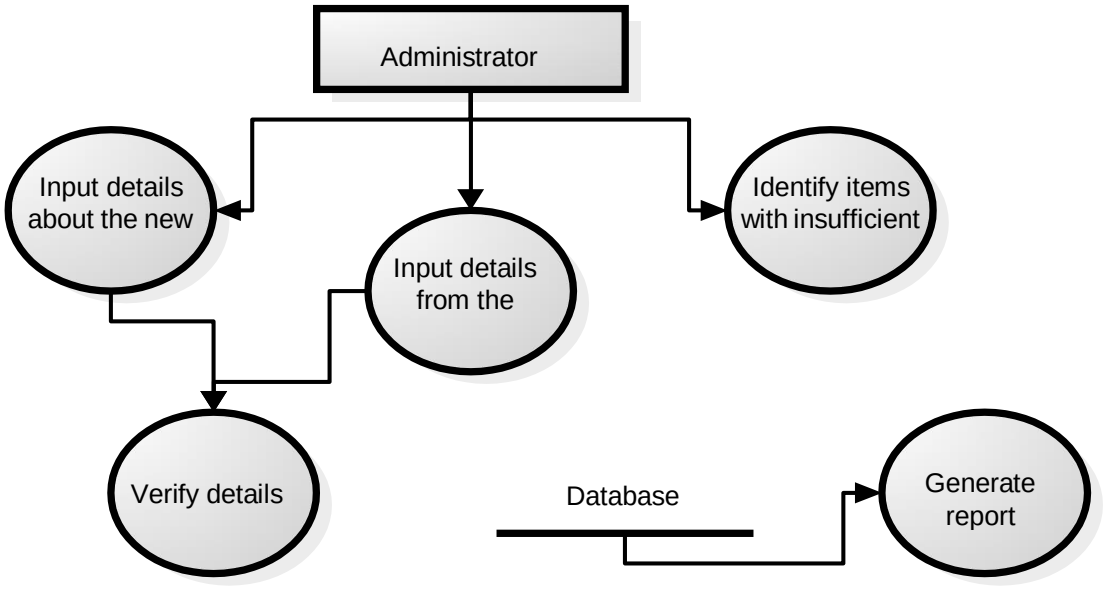
2.4 FRONT – END DESCRIPTION

The front end for the Inventory System is designed using Microsoft Visual Basic 6.0. The front end contains a user friendly interface. The user can view the customer details and stock details; user can order required units of any part or add part to the stock.

2.5 BACK – END DESCRIPTION

There are two tables. The customer table correlates a unique customer ID with his name, address and telephone number. The stock table contains the part number, part name and availability details.

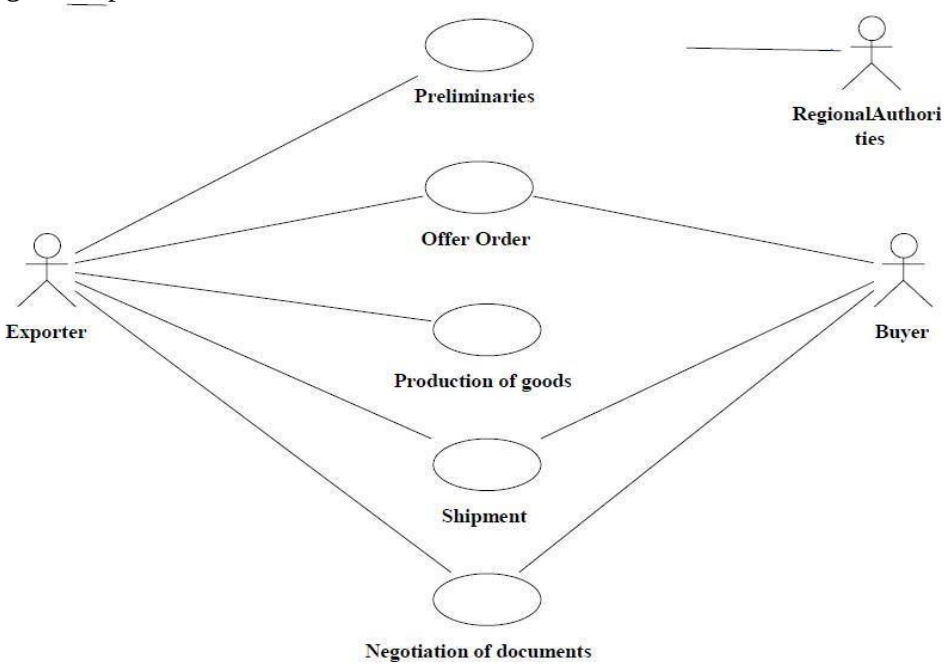
2.6 DATA FLOW DIAGRAM



UML DIAGRAMS

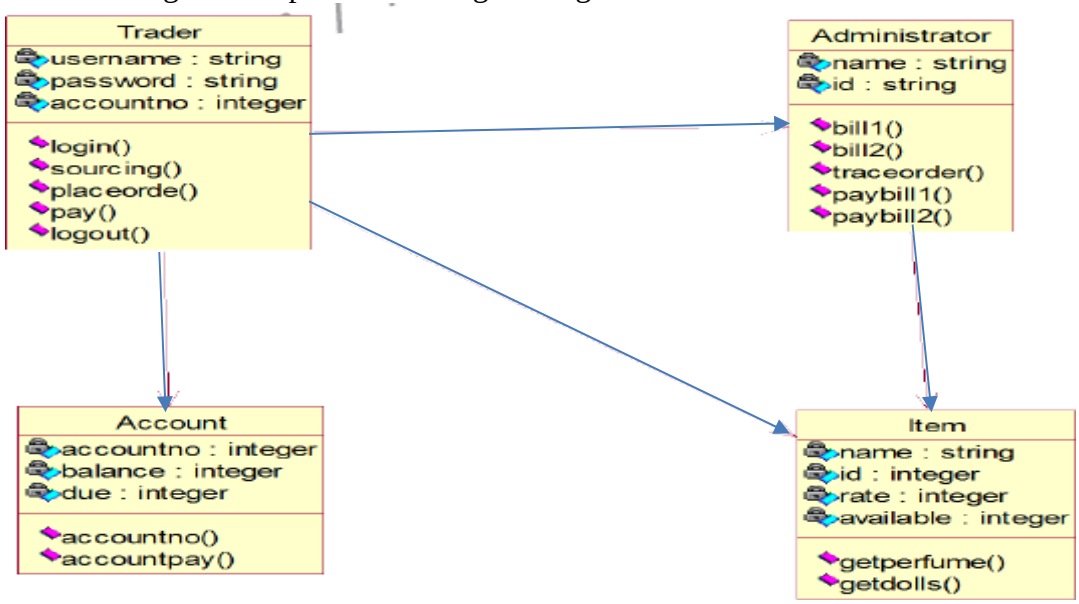
1. USERCASE DIAGRAM

The exporter submits the relevant documents to his buyer (banker) for getting the payment for the goodsexported.



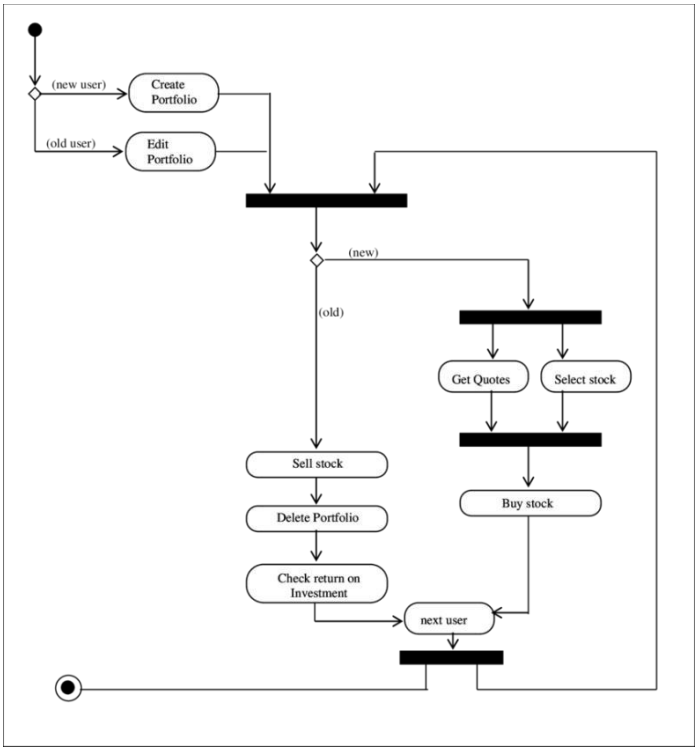
2. CLASSDIAGRAM

A class diagram is a type of static structure diagram that describes the structure of a system. The classes in the class diagram represent both the main objects and or interactions in the application. Theclass diagram is represented using rectangular boxes each of which contains three parts:



3. ACTIVITY DIAGRAM

This diagram represents the graphical representation of workflows of stepwise activities and actionswith support for choice, iteration and concurrency. It shows the overall flow of control.

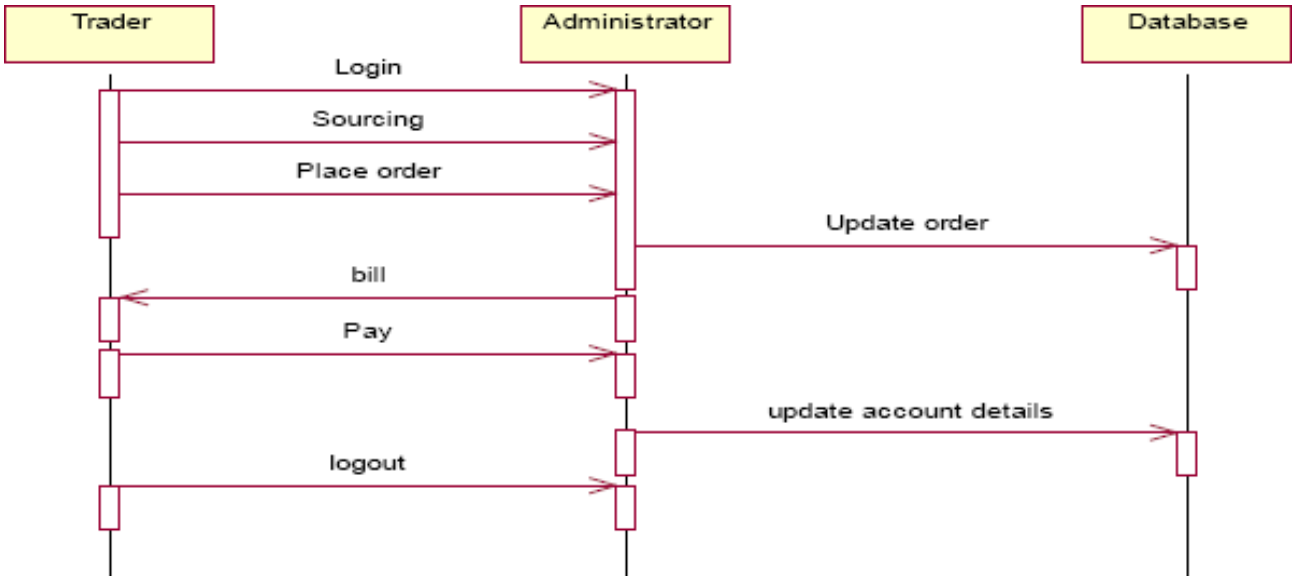


4. SEQUENCEDIAGRAM

A sequence diagram in unified modeling language is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams. This diagram shows a parallel vertical lines called lifelines. There are two dimensions in this diagram

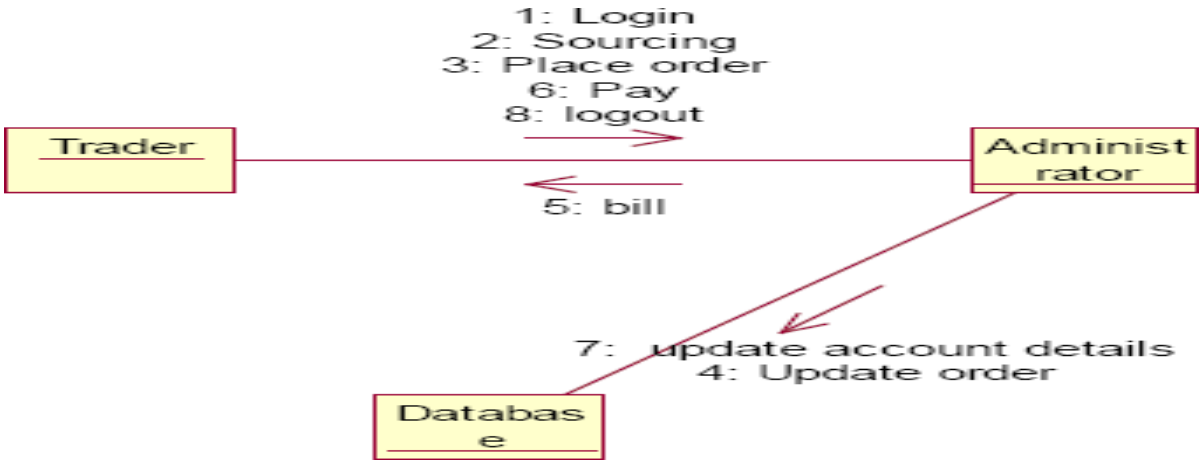
1. Vertical dimension-represents time.

2. Horizontal dimension-represent different object



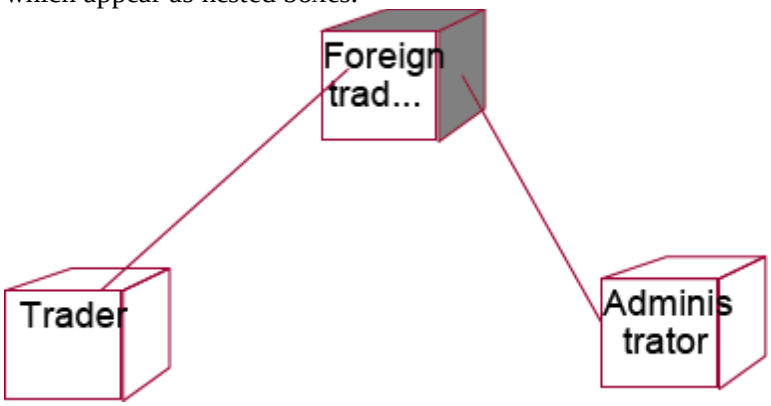
5. COLLABORATION DIAGRAM

A collaboration diagram belongs to a group of UML diagrams called Interaction Diagrams. Collaboration diagrams, like sequence diagrams, show how the objects interact over the course oftime. Collaboration diagrams show the sequence by numbering the messages on the diagram.



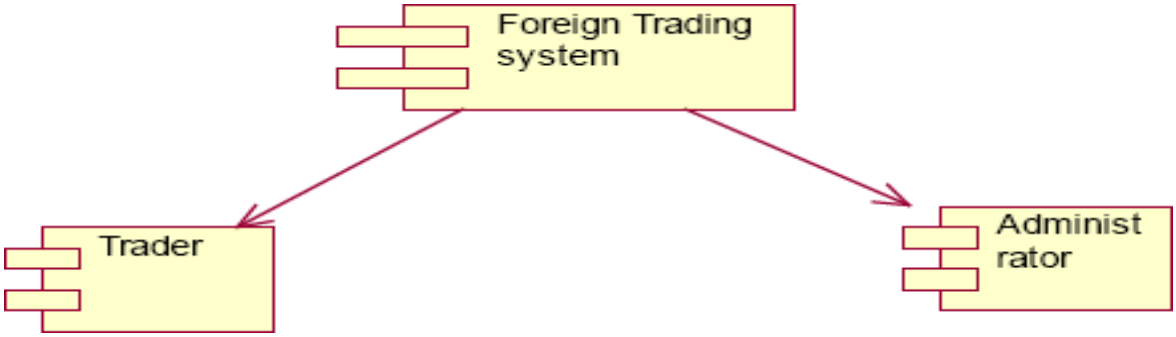
6. DEPLOYMENT DIAGRAM

A deployment diagram models the physical deployment of artifacts on nodes. The nodes appear as boxes, and the artifacts allocated to each node appear as rectangles within the boxes. Nodes may have sub nodes, which appear as nested boxes.



7. COMPONENT DIAGRAM

A component diagram depicts how the components are wired together to form larger components and or software systems. Components are wired together by using an assembly connector to connect the required interface of one component with the provided interface of another component.



8. PACKAGE DIAGRAM

A package diagram in the unified modeling language depicts the dependencies between the packagesthat make up a model. It provides a way to group the elements. There are three types of layers in package diagram. They are

- User interface layer
- Domain layer
- Technical services layer

User interface layer

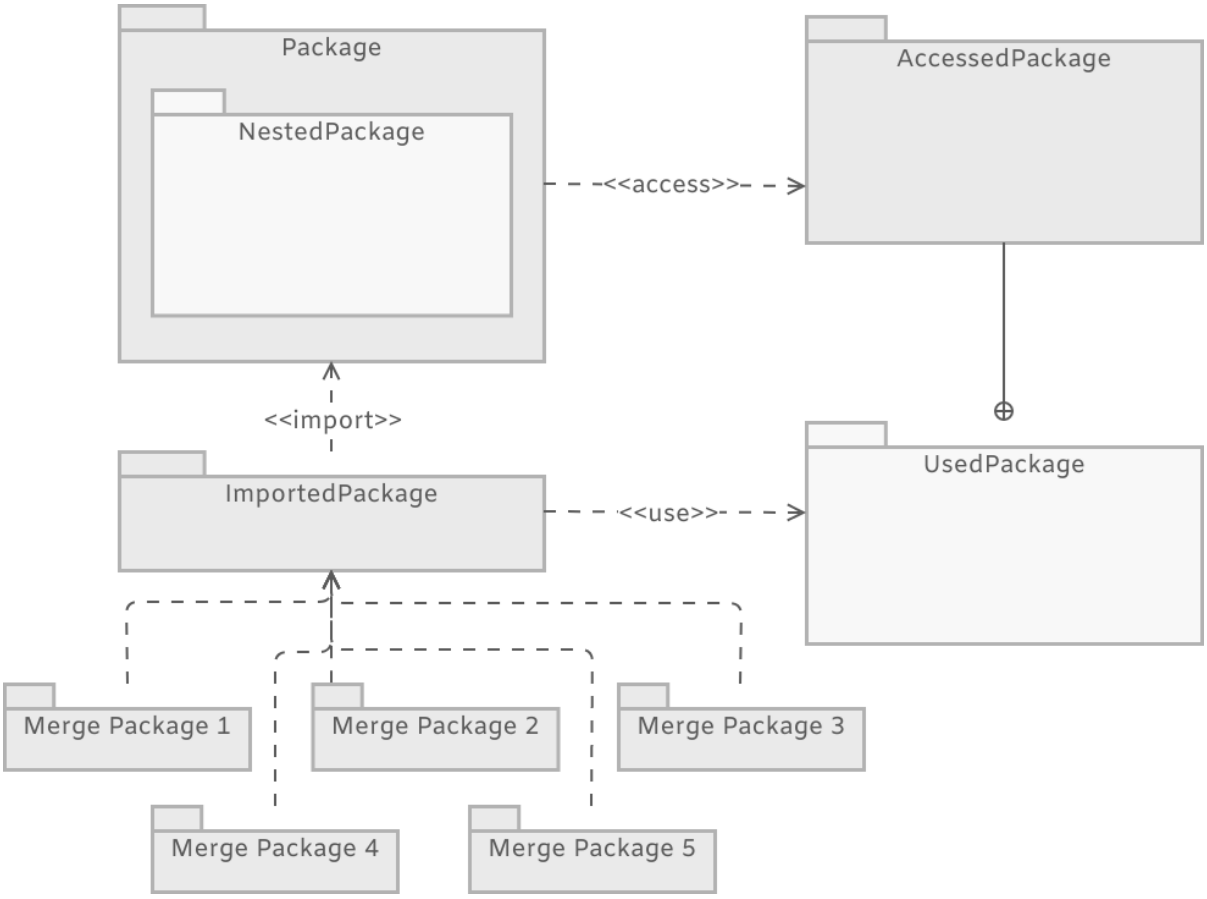
The user interface layer may call upon its directly subordinate application logic layer, and also upon elements of a lower technical service layer, for logging and so forth.

Domain layer

Software objects representing domain concepts (for example, a software class administrator) that fulfill application requirements, such as tracing order and providing the bill.

Technical services layer

General purpose objects and subsystems that provide supporting technical services, such as interfacing with a database or error logging. These services are usually application-independent.



Conclusion: The trading system was designed successfully by following the steps described above

EX NO. 6 E-TICKET RESERVATION SYSTEM
DATE:

1.0 PROBLEM DEFENITION

Ticket reservation system for railway department has to be developed. The system developed should contain the following features

1. The system should contain the following features
2. Search for information about the train by means of train number and destination
3. While displaying information about the train it has to provide availability of seats in different classes (first class, second class, arc)
4. While reserving tickets the system obtains following information from the userPassenger Name, Sex, Age, Address.
   Credit Card Number, Bank Name.
   Class through passenger is going to travel i.e. first class/second class/A.C
   Train number, Train name, Date of Journey and number of tickets to be booked.
5. Based on the availability of tickets, the ticket has to be issued. The ticket issued should contain the following information –PNR number, train no, train name, date of journey, class, number of passengers, sex, age and departure time.
6. Cancellation of booked tickets should be available.

2.0 SRS DOCUMENT FOR RAILWAY RESERVATION SYSTEM

2.1 INTRODUCTION

2.1.1 Purpose

2.1.1.1 The purpose of this SRS is to describe the requirements involved in developing a Railway Reservation system (RRS).

2.1.1.2 The intended audience is any person who wants to reserve or cancel tickets or to check the availability of Railway tickets

2.1.2 Scope

2.1.2.1 The product is titled Railway Reservation system (RRS).

2.1.2.2 The product will perform the following tasks

2.1.2.2.1 The software that is being developed can be used to check the availability of the train tickets for the specified train, destination and date of journey

2.1.2.2.2 If the tickets are available to the users' needs and specification, then the

software provides a facility to book the tickets.

2.1.2.2.3 If the passengers want to cancel the tickets, he can use the cancellation module of the Railway Reservation System.

2.1.3 Definitions, Acronyms and Abbreviations

2.1.3.1 RRS: Railway Reservation System.

2.1.4 References

2.1.4.1 IEEE standard 830-1998 recommended practice for Software Requirements Specifications-Description.

2.1.5 Overview

2.1.5.1 The SRS contains an analysis of the requirements necessary to help easy design.

2.1.5.2 The overall description provides interface requirements for the Railway Reservation system, Reservation System, product perspective, hardware interfaces software interfaces, communication interface, memory constraints, product functions, user characteristics and other constraints.

2.1.5.3 Succeeding pages illustrate the characteristics of typical naïve users accessing the system along with legal and functional constraints enforced that affect Railway Reservation system in any fashion.

2.2 THE OVERALL DESCRIPTION

2.2.1 Product perspective

2.2.1.1 Hardware interfaces

2.2.1.1.1 Hard disk: The database connectivity requires a hardware configuration with a fast database system running on high rpm hard-disk permitting complete data redundancy and back-up systems to support the primary goal of reliability.

2.2.1.1.2 The system must interface with the standard output device, keyboard and mouse to interact with this software.

2.2.1.2 Software interfaces

2.2.1.2.1 Back End: Oracle

2.2.1.2.2 Front End: Microsoft Visual Basic 6.0

2.2.1.3 Operations

2.2.1.3.1 The user mode enables the end-users to do the end user operations like checking the availability, reserving and canceling of train tickets.

2.2.2 Product Functions

2.2.2.1 Viewing Train Details

The user must have the access up-to-date information about the trains including 2.2.2.1.1Train number

2.2.2.1.2 Train Name

2.2.2.1.3 Train route (Start and Destination stations)

2.2.2.1.4 Train timings

2.2.2.1.5 Seat availability in each class.

2.2.2.2 Reserving Tickets

The user must be able to reserve tickets after selecting 2.2.2.2.1Train number

2.2.2.2.2 Train Name

2.2.2.3 Canceling Tickets

The user must be able to cancel tickets that he has earlier reserved by quoting the PNR number, credit card number and bank name.

2.2.3 User characteristics

2.2.3.1 The intended users of this software need not have specific knowledge as to what is the internal operation of the system. Thus the end user is at a high level of abstraction that allows easier, faster operation and reduces the knowledge requirement of end user

2.2.3.2 The Product is absolutely user friendly, so the intended users can be the naïve users.

2.2.3.3 The product does not expect the user to possess any technical background. Any person who knows to use the mouse and the keyboard can successfully use this product.

2.2.4 Constraints

2.2.4.1 At the time of reservation, each user is provided a unique PNR number that must be used for further operation like cancellation. Hence the user is required to remember or store this number carefully.

2.3 SPECIFIC REQIREMENTS

2.3.1 Logical Database Requirements

2.3.1.1 The system should contain databases that include all necessary information for the product to function according to the requirements. These include relations

such as train details, reservation details, and cancellation details.

2.3.1.2 The user details refer to the information such as train number and name, start and destination stations, seat availability.

2.3.1.3 Reservation details refer to personal information that is obtained from the user

2.3.1.4 At the time of reservation, the passenger is provided a unique PNR no that could be used at the time of cancellation.

2.3.1.5 While displaying any information about the train it has to provide the following information

Train no and name

Availability of seats for the

particular train The train

timing

The passenger personal details should be obtained for reserving the tickets.

2.4 FRONT – END DESCRIPTION

The front-end for the Railway Reservation system (RRS) is designed using Microsoft Visual Basic 6.0. The front-end contains a user- friendly interface. The first form contains a welcome screen that provides an option for the user to select one of the following

Enquiry

Reservation

Booking

Details

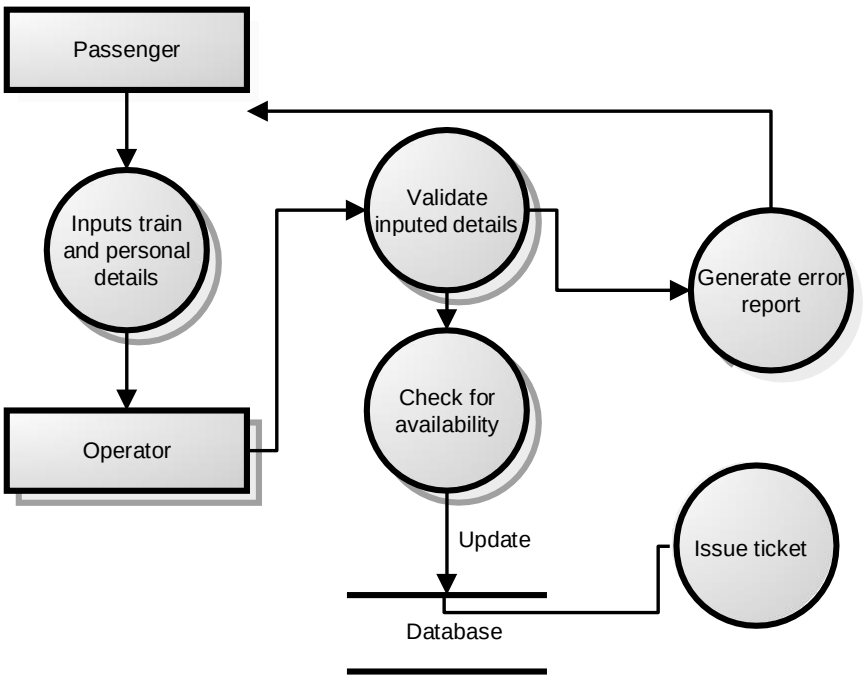
Cancellation

In the Enquiry form the user can get details of the train by means of either train name destination or date 0of journey. In the reservation form, there can book details by entering the personal details. The ticket is displayed with details about the train name and number, number of passengers, PNR number, class, sex and age. The cancellation form helps the user to cancel a ticket, which he had booked earlier.

2.5 BACK – END DESCRIPTION

The Railway Reservation system consists of two tables. One contains the train details such as the train name, train number, destination, date of journey and seats available in each class that is referred to during enquiry. The other table has the passenger details such as name, age, sex, credit card number, and bank name. This table is referred to at the time of reservation or cancellation.

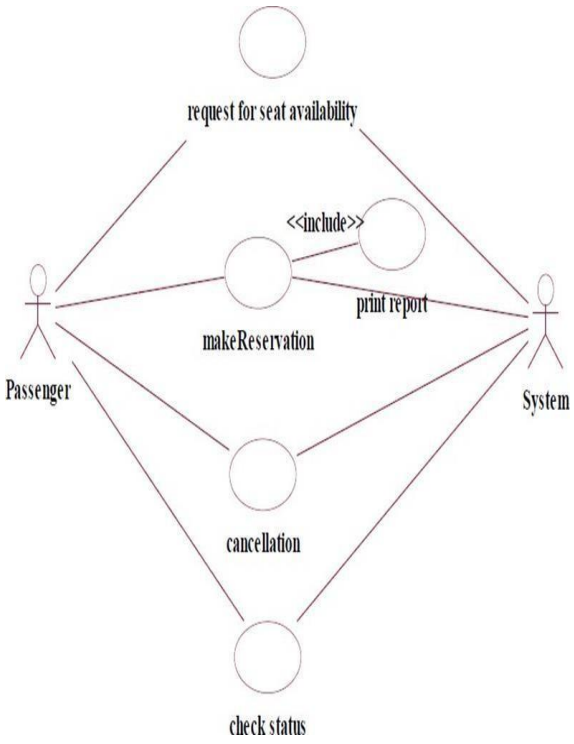
2.6 DATA FLOW DIAGRAM



UML DIAGRAMS

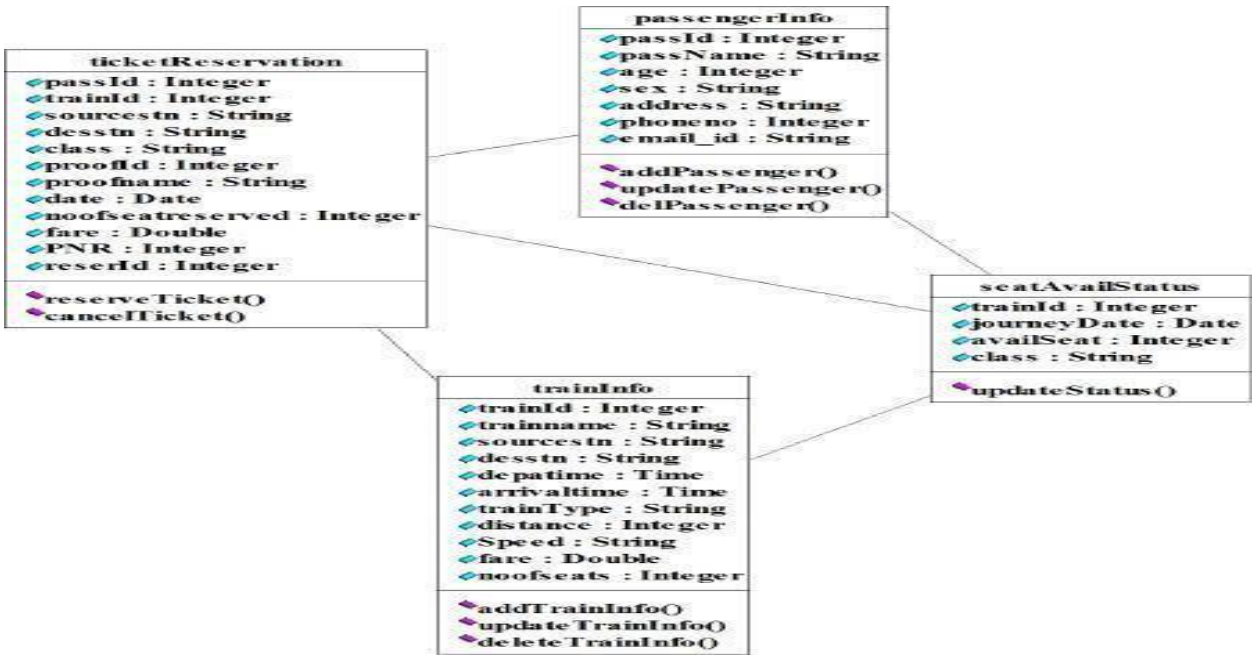
1. **USERCASE DIAGRAM**

The passenger can view the status of the reserved tickets. So the passenger can confirm his/hertravel.



2. CLASS DAIGRAM

A class diagram is a type of static structure diagram that describes the structure of a system. The classes in the class diagram represent both the main objects and or interactions in the application. The class diagram isrepresented using rectangular boxes each of which contains three parts



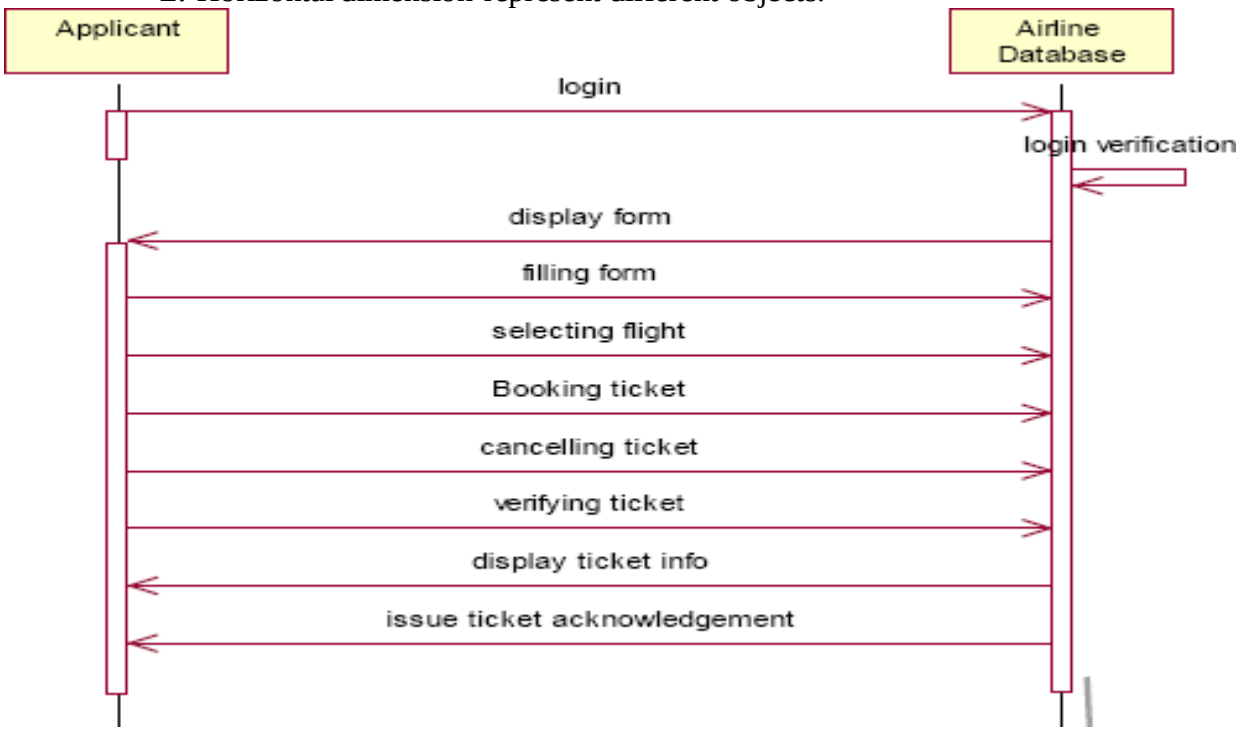
3. ACTIVITY DIAGRAM

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control. An activity is shown as a rounded box containing the name of the operation.

4. SEQUENCE DIAGRAM

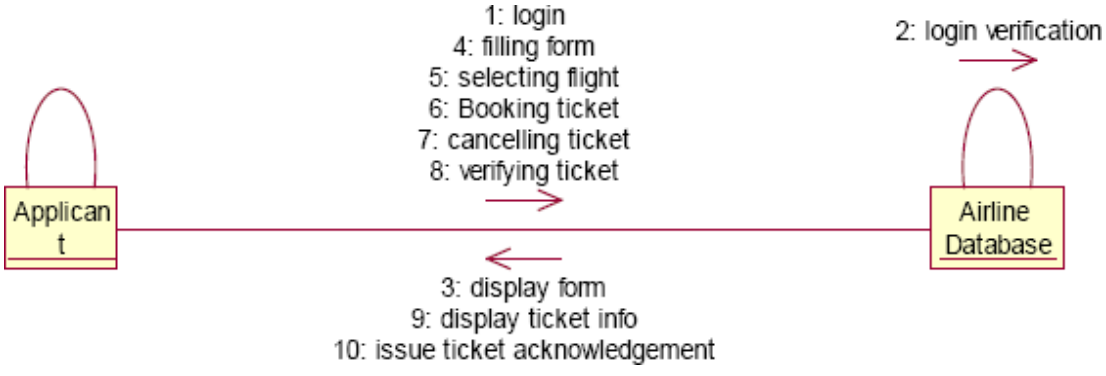
A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. There are two dimensions.

1. Vertical dimension-represent time.
2. Horizontal dimension-represent different objects.



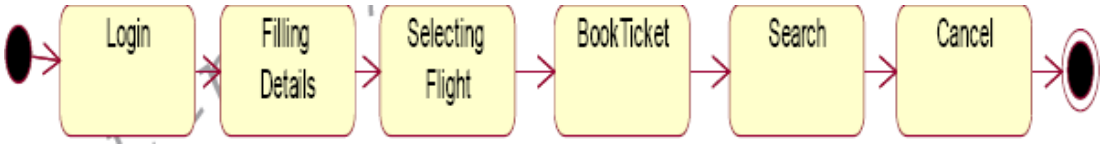
5. COLLABRATION DIAGRAM

A collaboration diagram, also called a communication diagram or interaction diagram, a sophisticated modeling tool can easily convert a collaboration diagram into a sequence diagram and the vice versa. A collaboration diagram resembles a flowchart that portrays the roles, functionality and behavior of individual objects as well as the overall operation of the system in real time.



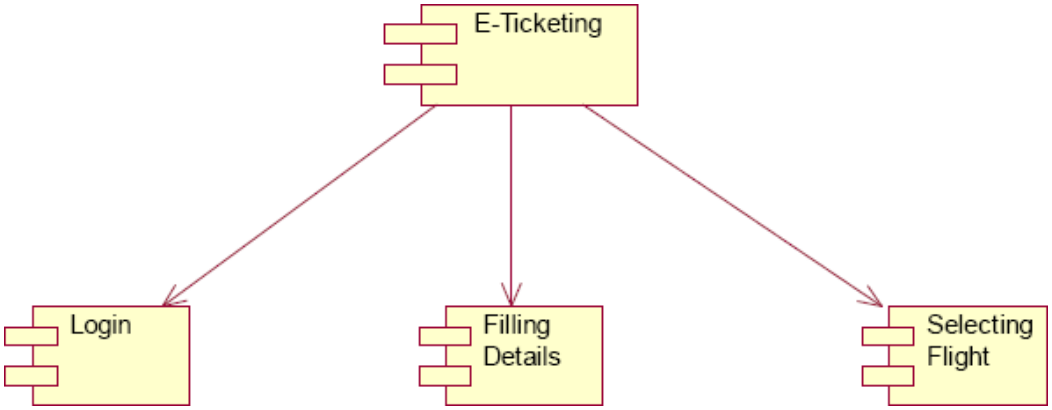
6. STATE CHART DIAGRAM

The purpose of state chart diagram is to understand the algorithm involved in performing a method . It is also called as state diagram. A state is Represented as a round box, which may contain one ormore compartments. An initial state is represented as small dot. A final state is represented as circlesurrounding a small dot.



7. COMPONENT DIAGRAM

The component diagram's main purpose is to show the structural relationships between the components of a system. It is represented by boxed figure. Dependencies are represented by communication association.



Conclusion: The ticket reservation system was designed successfully by following the steps described above.

EX NO. 7 AUTOMATED BANKING SYSTEM

DATE:

1.0 PROBLEM DEFENITION

To develop an automated banking system, which is required to perform the following functions:

1.1 The customer logs into the system using card number and pin number. The system checks for validation.

1.2 The system queries the customer for the type of account either fixed deposit or credit account. After getting the type of account the system shows the balance left.

1.3 The system queries the customer for the transaction type either withdrawal or deposit and the required amount. The user enters the amount and the transaction if carries out

2.0 SRS DOCUMENT FOR AUTOMATED BANKING SYSTEM

2.1 INTRODUCTION

2.1.1 Purpose

2.1.1.1 The purpose of this SRS is to describe the requirements involved in developing an Automated Banding System (ABS).

2.1.1.2 The intended audience is any person who wants

2.1.1.2.1 To create account.

2.1.1.2.2 To withdraw or deposit either in fixed deposit or credit account.

2.1.2 Scope

2.1.2.1 The product is titled Automated Banking System (ABS).

2.1.2.2 The product will perform the following tasks

2.1.2.2.1 Allow a new user to create an account, either fixed or credit account by entering the details and by depositing an initial amount.

2.1.2.2.2 Allow the existing user to enter his account details like card number, pin number and account type to view his balance.

2.1.2.2.3 Allow the existing user to deposit an amount by entering the amount to be deposited after the balance had been viewed.

2.1.2.2.4 Allow the existing user to withdraw an amount by entering the amount to be withdrawn after the balance had been viewed.

2.1.2.2.5 The primary benefits expected of the system are: user friendly, continuous connectivity without failure, fault tolerant and involves lesser manpower.

2.1.3 Definitions, Acronyms and Abbreviations

2.1.3.1 ABS: Automated Banking System.

2.1.4 References

2.1.4.1 IEEE standard 830-1998 recommended practice for Software Requirements Specifications-Description.

2.1.4.2 IEEE Software Requirements Specifications Template http://www.cas.master.ca/~carette/SE3M04/2003/files/ srs_template.doc

2.1.5 Overview

2.1.54.1 The SRS contains an analysis of the requirements necessary to help easy design.

2.1.5.2 The overall description provides interface requirements for the Banking system, product perspective, hardware interfaces, software interfaces, communication interface, memory constraints, product functions, user characteristics and other constraints.

2.1.5.3 Succeeding pages illustrate the characteristics of typical naïve users accessing the system along with legal and functional constraints enforced that affect banking system in any fashion.

2.2 THE OVERALL DESCRIPTION

2.2.1 Product perspective

2.2.1.1 Hardware interfaces

2.2.1.1.1 Hard disk: The database connectivity requires a hardware configuration that is on-line. This makes it necessary to have a fast database system (such as any RDBMS) running on high rpm hard-disk permitting complete data redundancy and backup systems to support the primary goal of reliability.

2.2.1.1.2 The system must interface with the standard output device,

Keyboard and mouse to interact with this software.

2.2.1.2 Software interfaces

2.2.1.2.1 Back End: Oracle

2.2.1.2.2 Front End: Microsoft Visual Basic 6.0

2.2.1.3 Operations

2.2.1.3.1 The user can create a new account.

2.2.1.3.2 The existing user can access his account and view his balance by entering his details.

2.2.1.3.2 The user can deposit and withdraw money from his account.

2.2.2 Product Functions

2.2.2.1 Creating a New Account

The user should provide his personal details to facilitate the bank clerk to create a new account. The user should provide:

2.2.2.1.1 Customer Name.

2.2.2.1.2 Customer address.

2.2.2.1.3 Required account type.

2.2.2.1.4 Pin Number.

2.2.2.1.5 Initial deposit.

2.2.2.2 Operating with created account

The user should be able to operate with his new account after:

2.2.2.2.1 Entering card number.

2.2.2.2.2 Entering pin number.

2.2.2.2.3 Entering the account type, transaction type and amount involved in the transaction.

2.2.3 User characteristics

2.2.3.1 The intended users of this software need not have specific knowledge as to what is the internal operation of the system. Thus the end user is at a high level of abstraction that allows easier, faster operation and reduces the knowledge requirement of end user

2.2.3.2 The Product is absolutely user friendly, so the intended users can be the naïve users.

2.2.3.3 The product does not expect the user to possess any technical background. Any person who knows to use the mouse and the keyboard

Can successfully use this product.

2.2.4 Constraints:

2.2.4.1 At the time of creating the new account, each user gives a pin number and is provided with a unique card number that must be used for further transactions. Hence the user is required to remember or store these numbers carefully.

2.2.4.2 At the time of creating the new account, the initial deposit should not be less than the specified amount.

2.3 SPECIFIC REQIREMENTS

2.3.1 Logical Database Requirements

2.3.1.1 The system should contain databases that include all the necessary information for the product to function according to the requirements. These include relations such as Customer Details and Account Details.

2.3.1.2 Customer details refer to the customer's name and address. Account details of the customer include the card number, account type, transaction type and the pin number given by the user to be used at the time of the transaction at the bank.

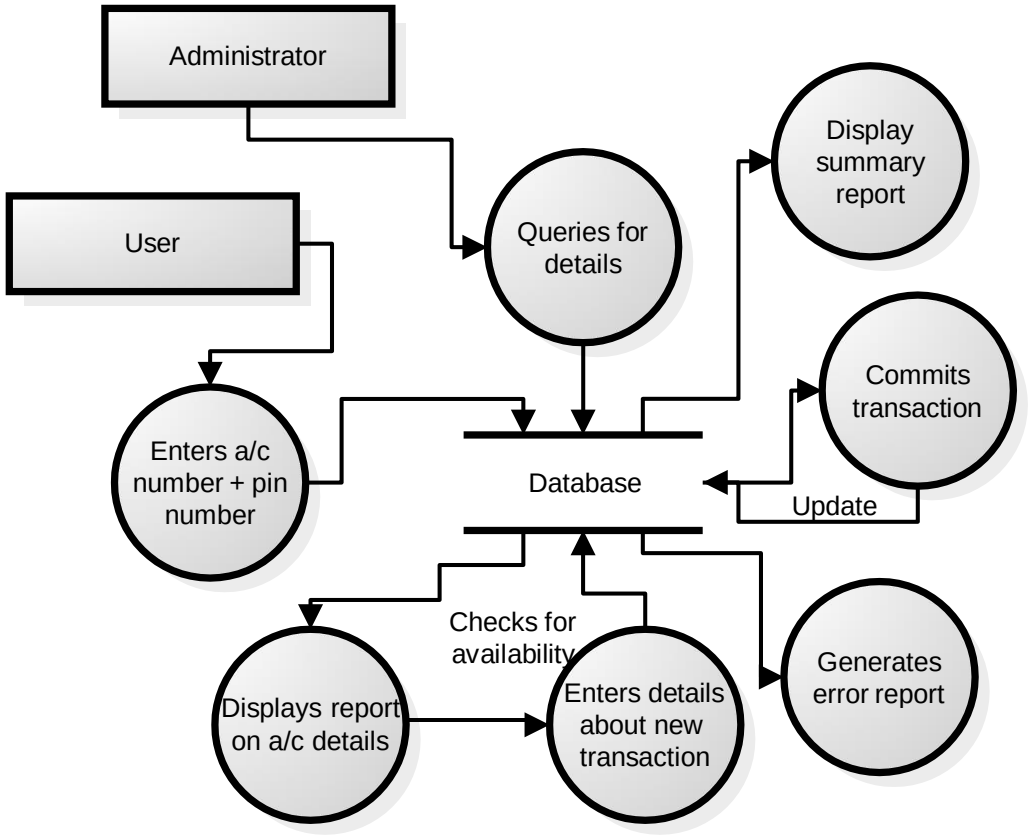
2.4 FRONT – END DESCRIPTION

The front end for the Automated Banking System (ABS) is designed using Microsoft Visual Basic 6.0. The front end contains a user-friendly interface. The first form contains a welcome screen that provides an option for the user to either crate a new account or to operate through an existing account. The "create account" module contains a provision to create a newaccount after collecting the customer name, address and other details. The card number and pin number of the user is obtained every time there is a transaction. The user is requested to select the required type of transaction and the amount involved in the transaction.

2.5 BACK – END DESCRIPTION

The Automated Banking System (ABS) database contains only one table. It correlates a unique card number, customer name, account type, pin number and the balance.

2.0 DATA FLOW DIAGRAM:

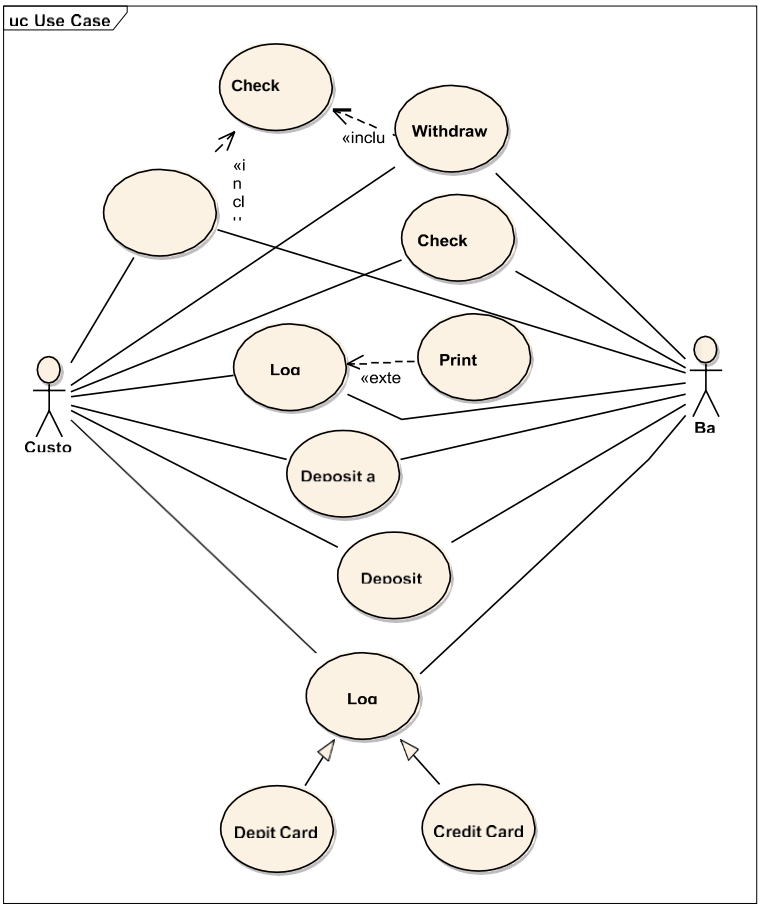


UML DIAGRAMS

1. USERCASE DIAGRAM

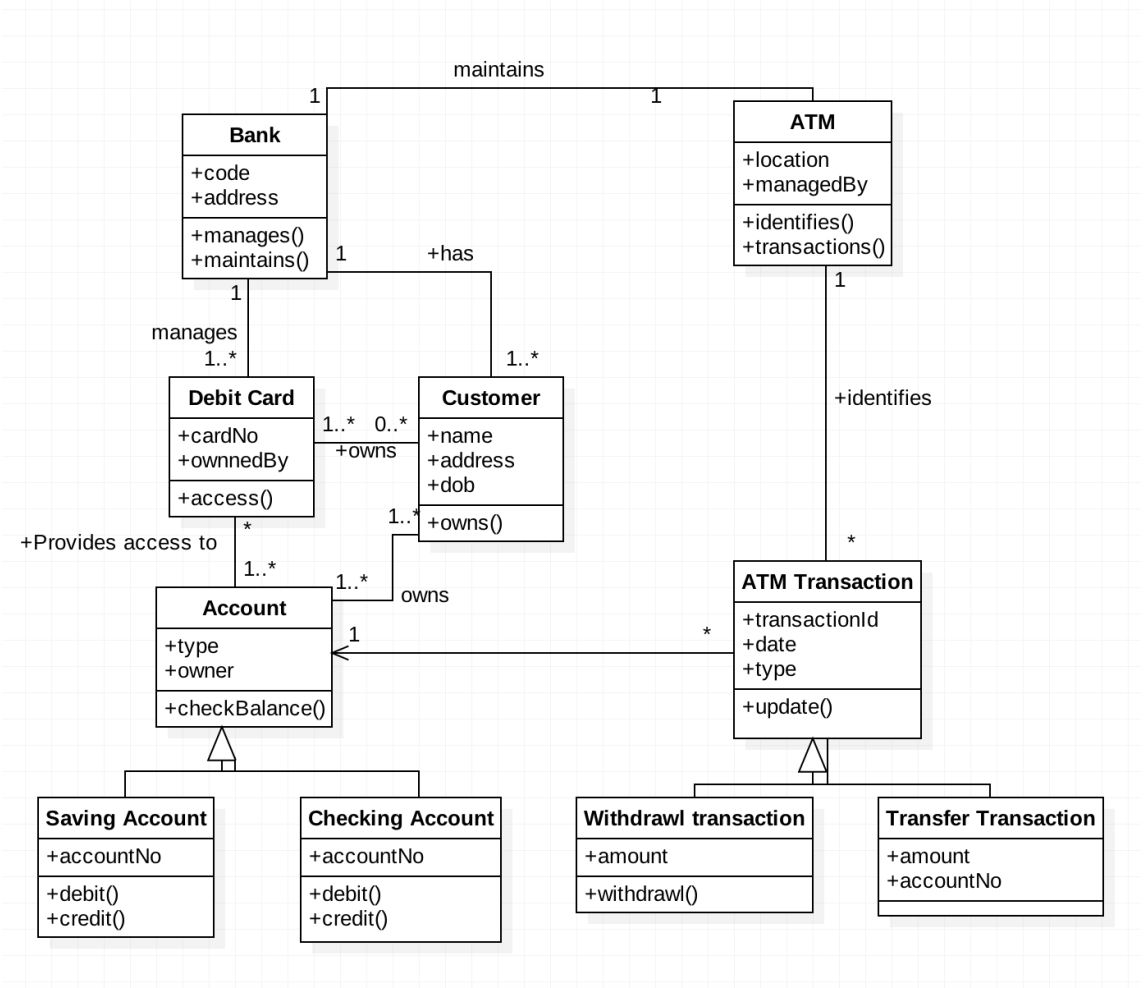
Use case diagrams describe the functionality of a system and users of the system. They containthe
Following elements:
1. Actors, which represent users of a system, including human users and other systems
2. Use cases, which represent functionality or services provided by a system to users.

Here, is a use case diagram for the ATM System.



2. CLASS DIAGRAM

Class diagrams describe the static structure of a system, or how it is structured ratherthan how it behaves.
These diagrams contain the following elements:
1. Classes, which represent entities with common characteristics or features. These features include
Attributes, operations, and associations.
2. Associations, which represent relationships that relate two or more other classes where the relationships have common characteristics or features. These features include attributes and operations

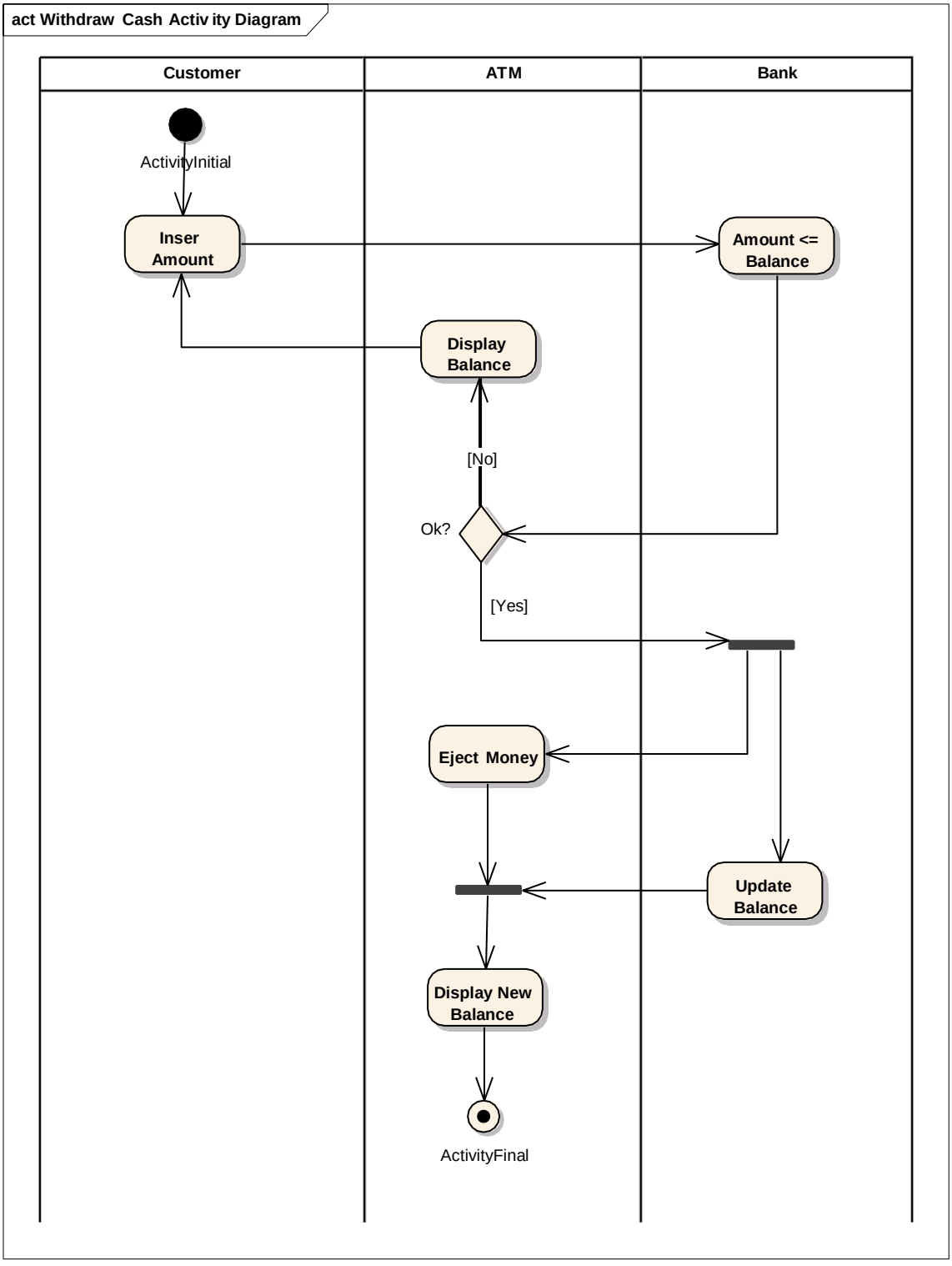


3. ACTIVITY DIAGRAM

Activity diagrams describe the activities of a class. They are similar to state transition diagrams and use similar conventions, but activity diagrams describe the behavior/states of a class in response to internal processing rather than external events. They contain the following elements:

1. Swim lanes, which delegate specific actions to objects within an overall activity
2. Action States, which represent uninterruptible actions of entities, or steps in the execution of an algorithm
3. Action Flows, which represent relationships between the different action states on an entity
4. Object Flows, which represent utilization of objects by action states, or influence of action states on objects.

Following are the examples of Login, Withdraw Activity Diagrams



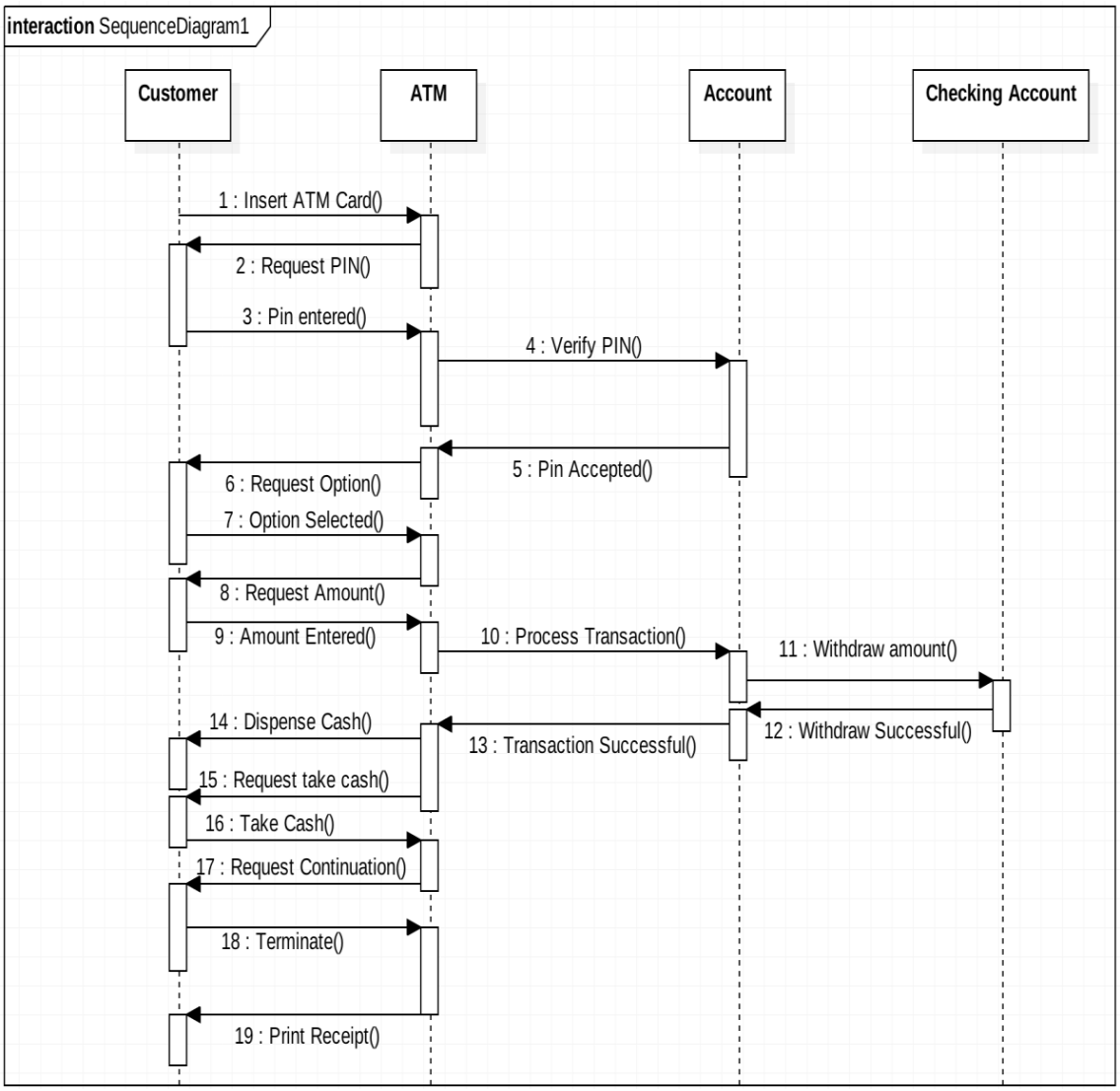
4. SEQUENCE DIAGRAM

Sequence diagrams typically show the flow of functionality through a use case, and consist of the following

Components:

1. Actors, involved in the functionality
2. Objects, that a system needs to provide the functionality
3. Messages, which represent communication between objects

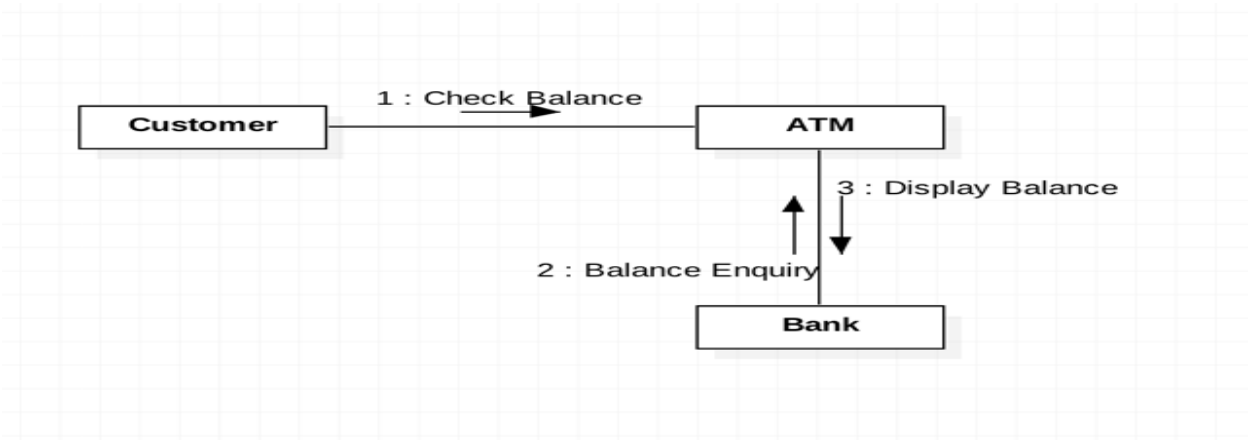
Here, is an example of Sequence diagram for withdrawing amount from ATM.

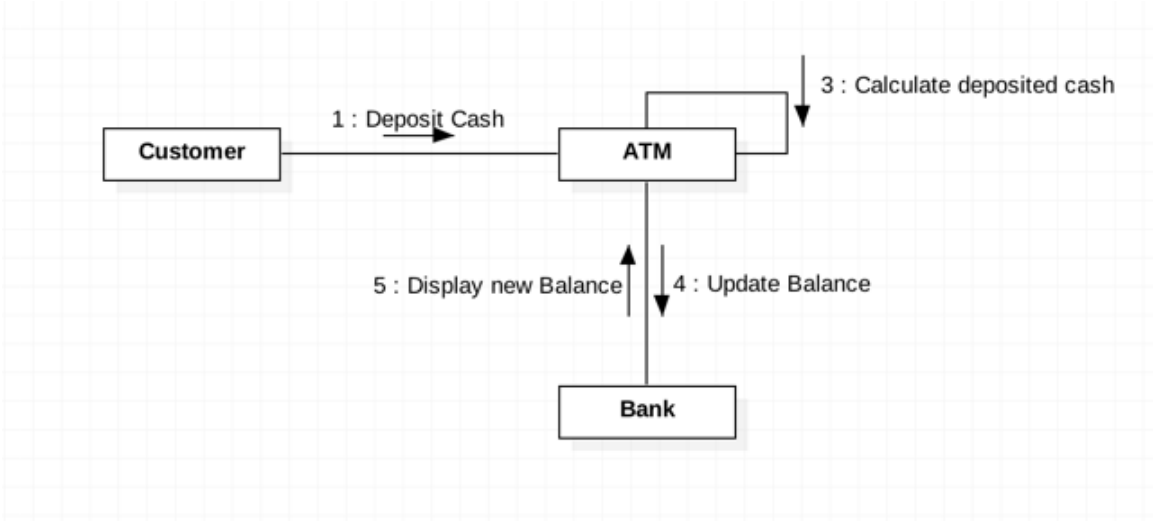


5. COLLABORATION DIAGRAM

A Communication or Collaboration diagram, as shown is a directed graph that uses objects and actors as graph nodes. The focus of the collaboration diagram is on the roles of the objects as they interact to realize a system function. Directional links are used to indicate communication between objects. These links are labeled using appropriate messages. Each message is prefixed with a sequence number indicating the time ordering needed to realize the system function.

Here is an example of the Check Balance communication diagram:





Conclusion: The automated banking system was designed successfully by following the steps described above.

EX NO. 8 BLOOD BANKING SYSTEM

DATE:

PROBLEM DEFENITION

To develop an automated banking system, which is required to perform the following functions:

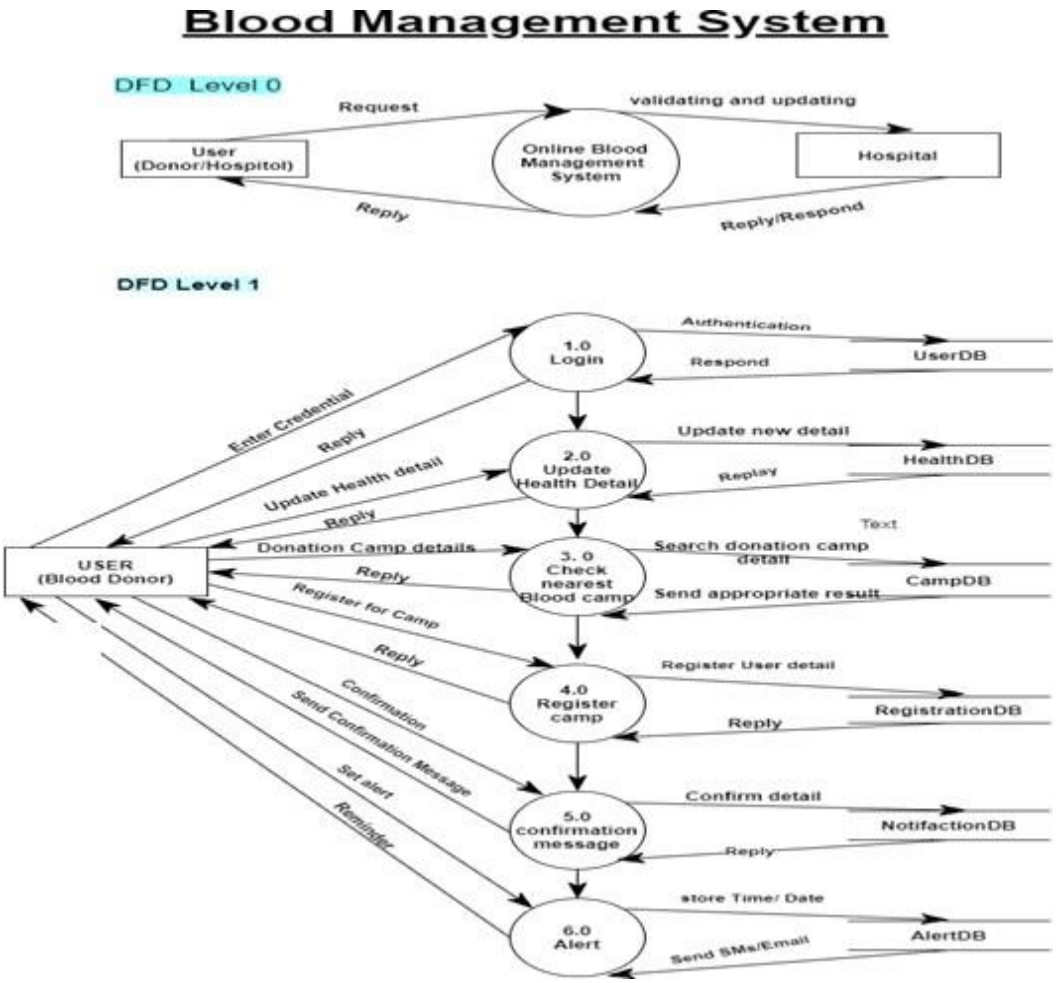
1.0 In this system we are creating a centralized system which is connected to each hospital which tackles the problem of maintain the records of donors.

1.1 This system notifies the donors which are connected to the system regarding blood donation camp.

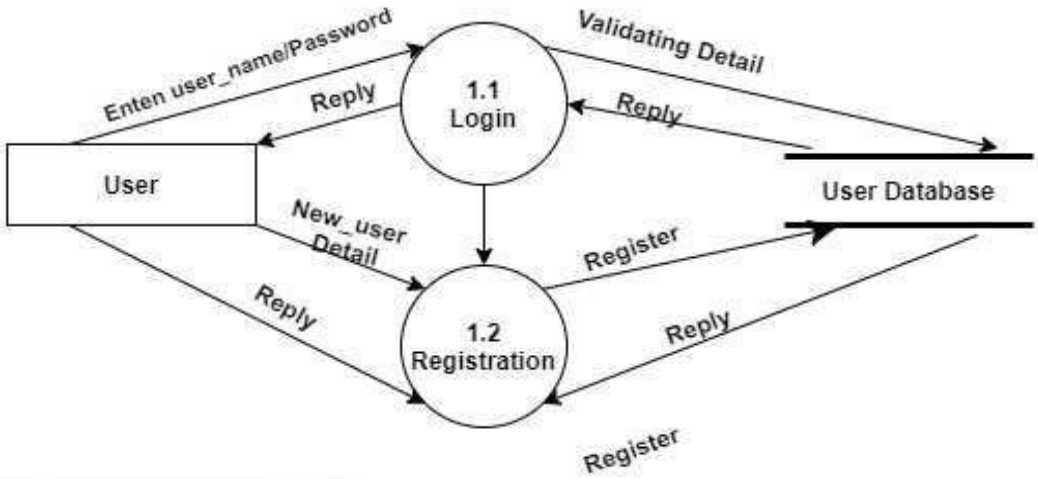
1.2 The biggest advantage of this system is that the hospitals are able to check the blood availability of certain blood groups.

1.3 Since all blood bank are connected through single platform, and all record of blood availability in single UI, there is no problem of non-availability of blood.

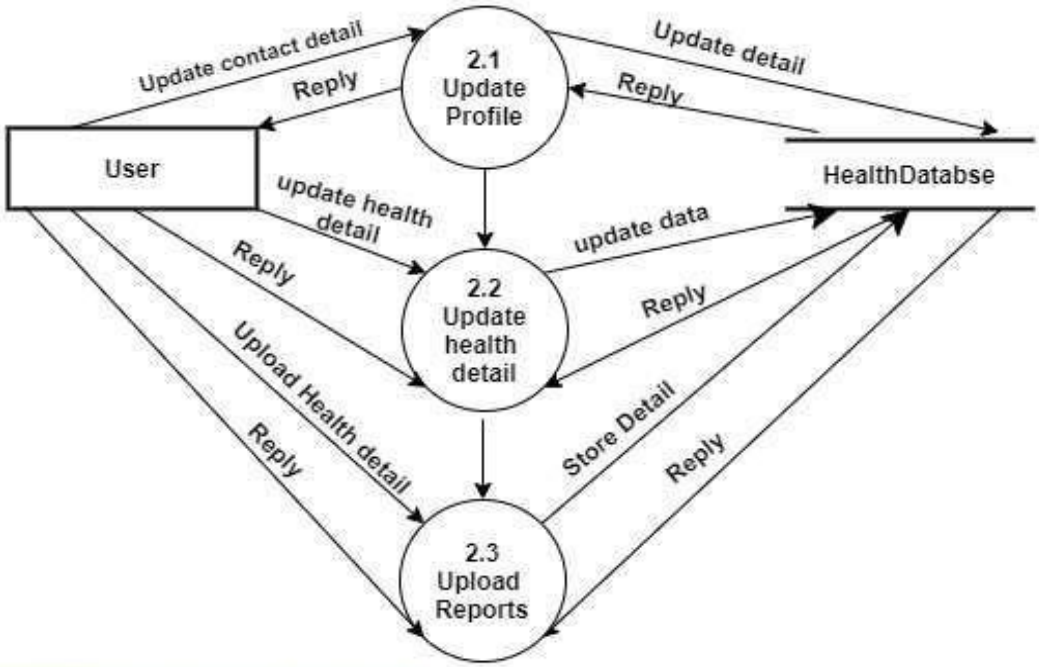
DATA FLOW DIAGRAM:



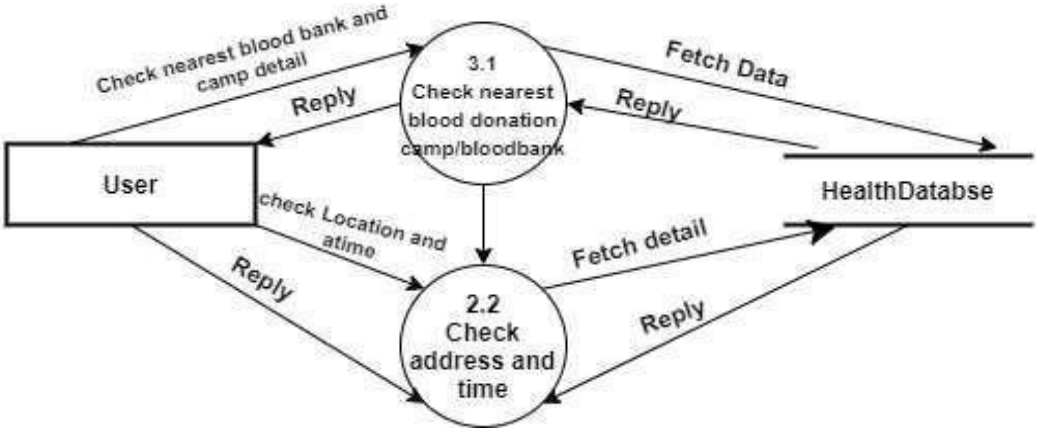
Level 2 for 1.0 (Login/ Registration)

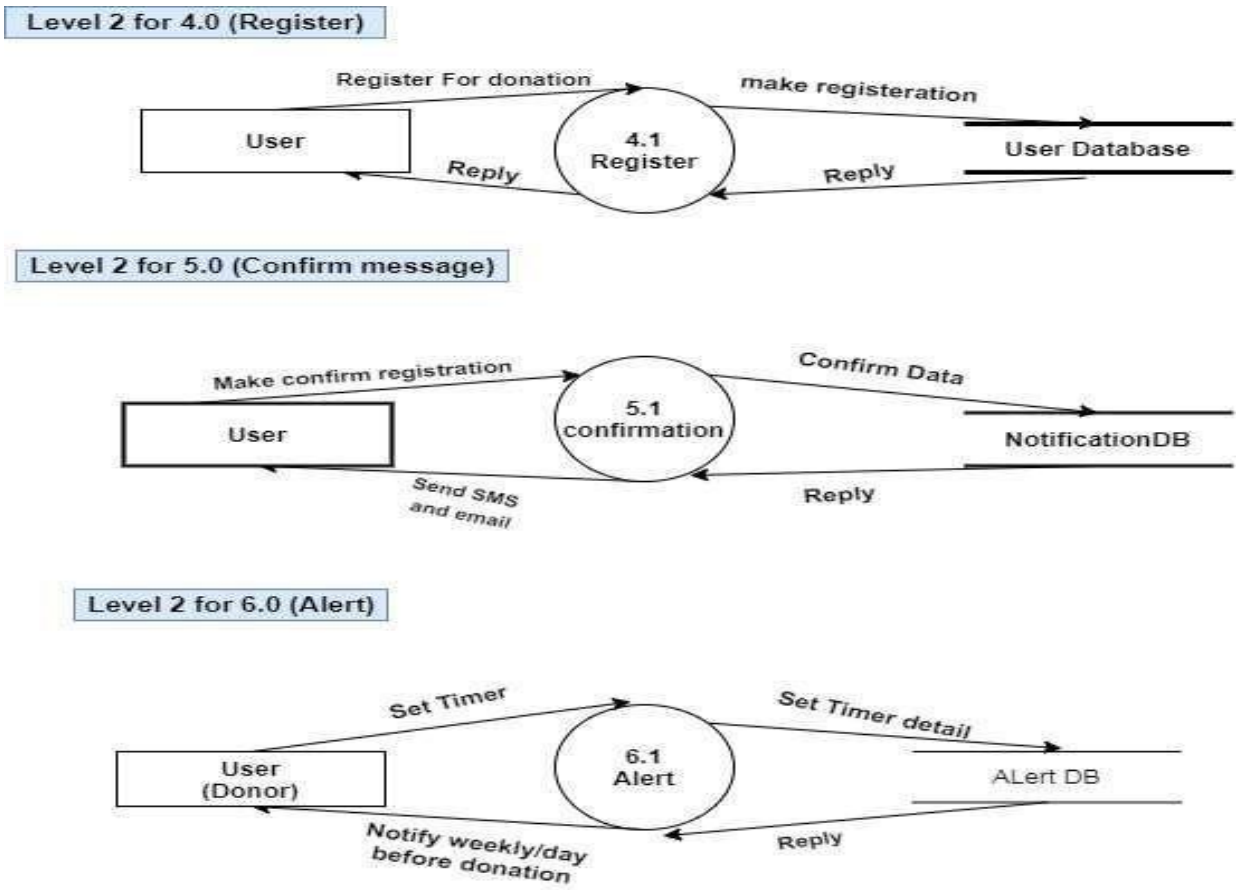


Level 2 for 2.0 (Update/health detail)



Level 2 for 3.0 (Check nearest blood camp





UML DIAGRAMS

1. USECASE DIAGRAM

- **Use Case Diagram** captures the system's functionality and requirements by using actors and use cases. UseCases model the services, tasks, function that a system needs to perform. Use cases represent high-level functionalities and how a user will handle the system. Use-cases are the core concepts of Unified Modellinglanguage modeling.

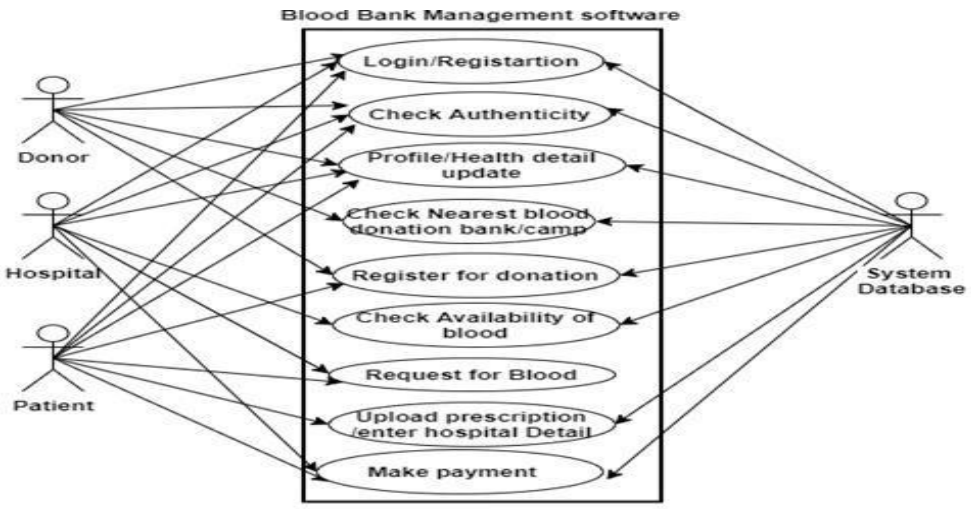


Fig 1. USE-CASE DIAGRAM FOR BLOOD BANK SYSTEM

2. SEQUENCE DIAGRAM

- Sequence diagrams model the flow of logic within the system.
- Sequence diagrams are the most popular UML artifact for dynamic modeling, which focuses on identifying the behavior within your system.

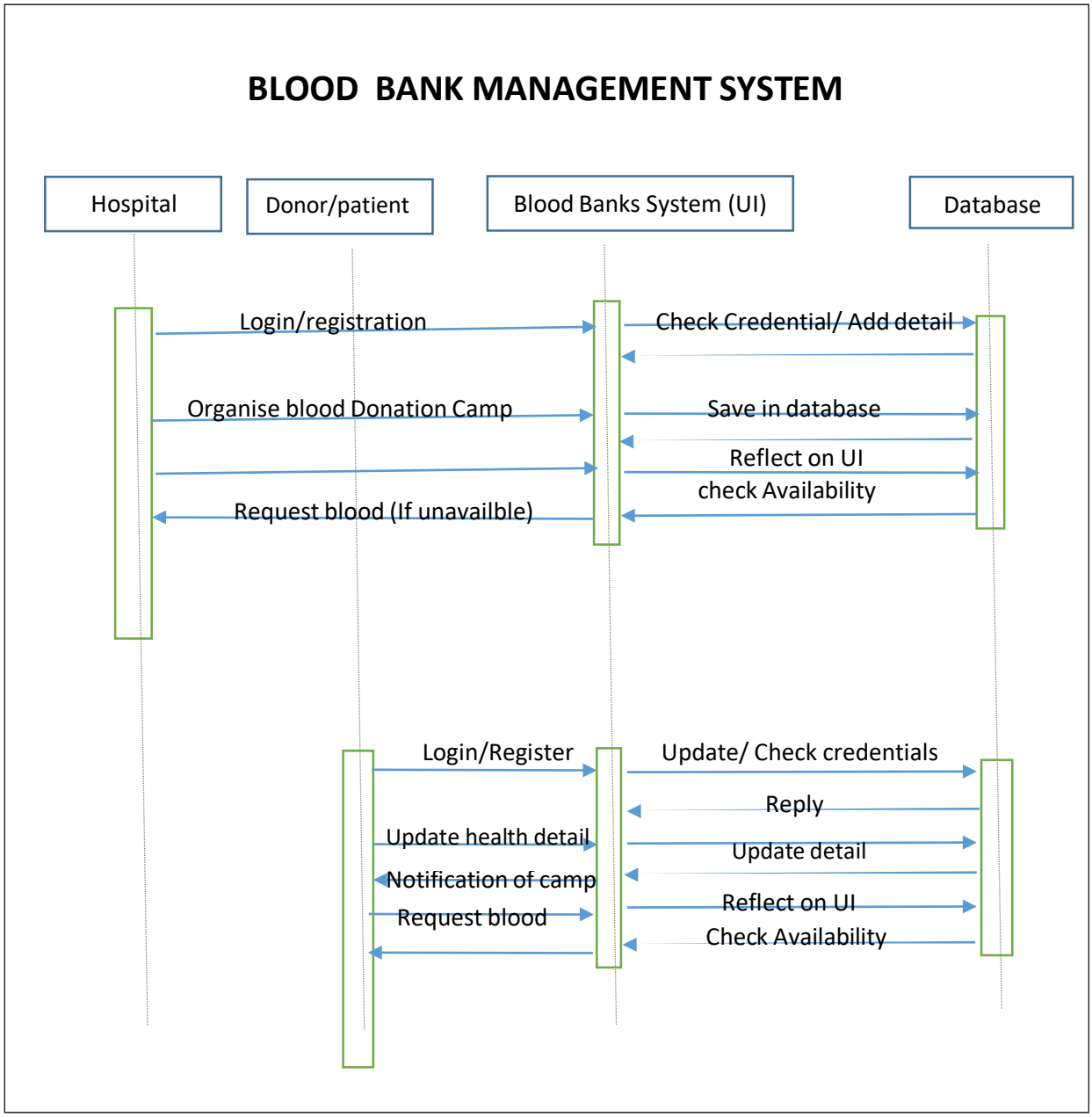


Fig 2. SEQUENCE DIAGRAM FOR BLOOD BANK SYSTEM

3. ACTIVITY DIAGRAM

- Activity diagrams are used for modeling the logic captured by a single use case or usage scenario, orfor modeling the detailed logic of a business rule.
- UML activity diagrams are the object-oriented equivalent of flow charts and data flow diagrams (DFDs)from structured development.

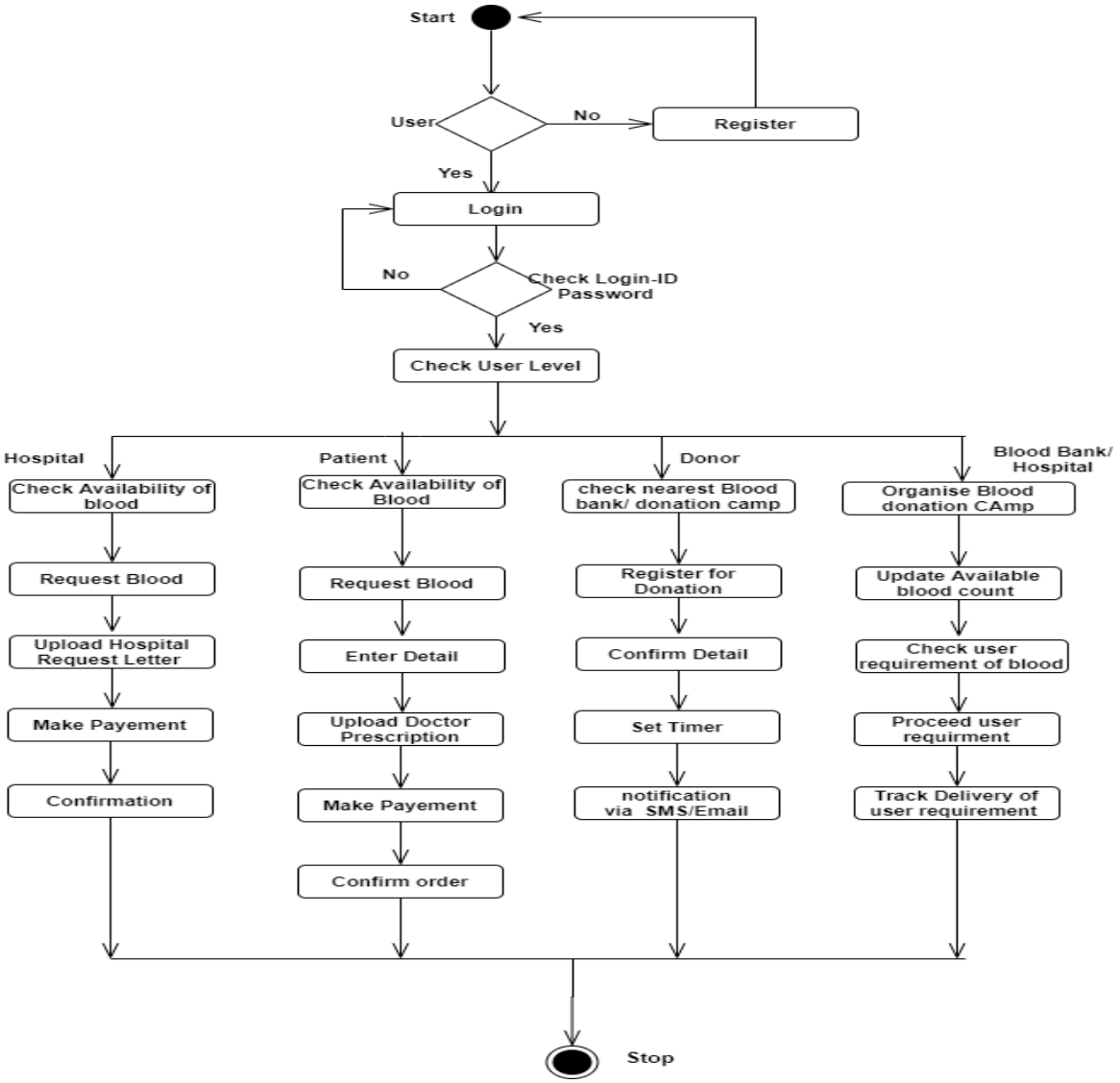


Fig 3. ACTIVITY DIAGRAM FOR BLOOD BANK SYSTEM

4. COMPONENT DIAGRAM

Component diagram shows components, provided and required interfaces, ports, and relationships between them. This type of diagrams is used in Component-Based Development(CBD) to describe systems with Service-Oriented Architecture (SOA).

Component diagrams are different in terms of nature and behavior. Component diagrams are used to model the physical aspects of a system. Now the question is, what are these physical aspects? Physical aspects are the elements such as executable, libraries, files, documents, etc. which reside in a node.

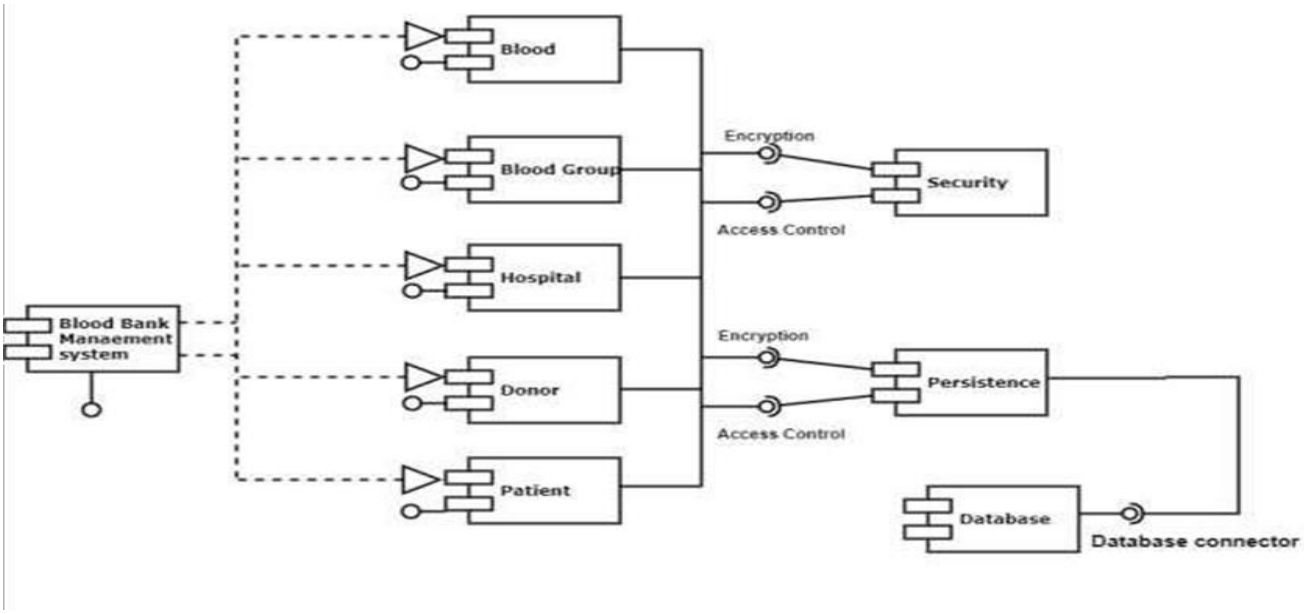


Fig 4. COMPONENT DIAGRAM FOR BLOOD BANK SYSTEM

Conclusion:

The automated blood banking system was designed successfully by following the steps described above

EX NO. 9 **PASSPORT AUTOMATION SYSTEM**

DATE:

AIM:

To develop the Passport Automation System using rational rose tools, visual basic and MS access.

PROBLEM STATEMENT:

Passport Automation System is used in the effective dispatch of passport to all of the applicants. This system adopts a comprehensive approach to minimize the manual work and schedule resources, time in a cogent manner.

The core of the system is to get the online registration form (with details such as name, address etc.,) filled by the applicant whose testament is verified for its genuineness by the Passport Automation System with respect to the already existing information in the database. This forms the first and foremost step in the processing of passport application.

After the first round of verification done by the system, the information is in turn forwarded to the regional administrator's (Ministry of External Affairs) office. The application is then processed manually based on the report given by the system, and any forfeiting identified can make the applicant liable to penalty as per the law. The system also provides the applicant the list of available dates for appointment to 'document verification' in the administrator's office, from which they can select one. The system forwards the necessary details to the police for its separate verification whose report is then presented to the administrator. The administrator will be provided with an option to display the current status of application to the applicant, which they can view in their online interface. After all the necessary criteria have been met, the original information is added to the database and the passport is sent to the applicant.

1. USE CASE DIAGRAM:

The administrator checks or process the application which are submitted by applicant. Process the application means the data which are given by the applicant is processed to create a passport for the applicant and finally dispatches the passport to the applicant.

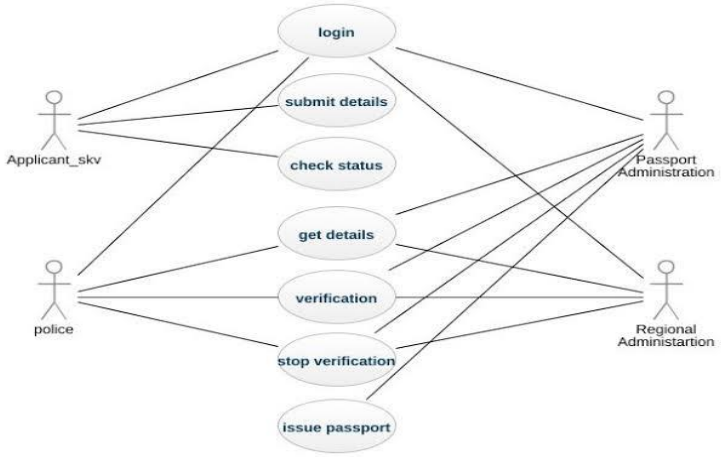


Fig 1. USE-CASE DIAGRAM FOR PASSPORT OFFICE SYSTEM

2. CLASS DIAGRAM:

The class diagram, also referred to as object modelling is the main static analysis diagram. The main task of object modelling is to graphically show what each object will do in the problem domain. The problem domain describes the structure and the relationships among objects. Here the various classes are applicant, passport admin, regional admin, database and police.

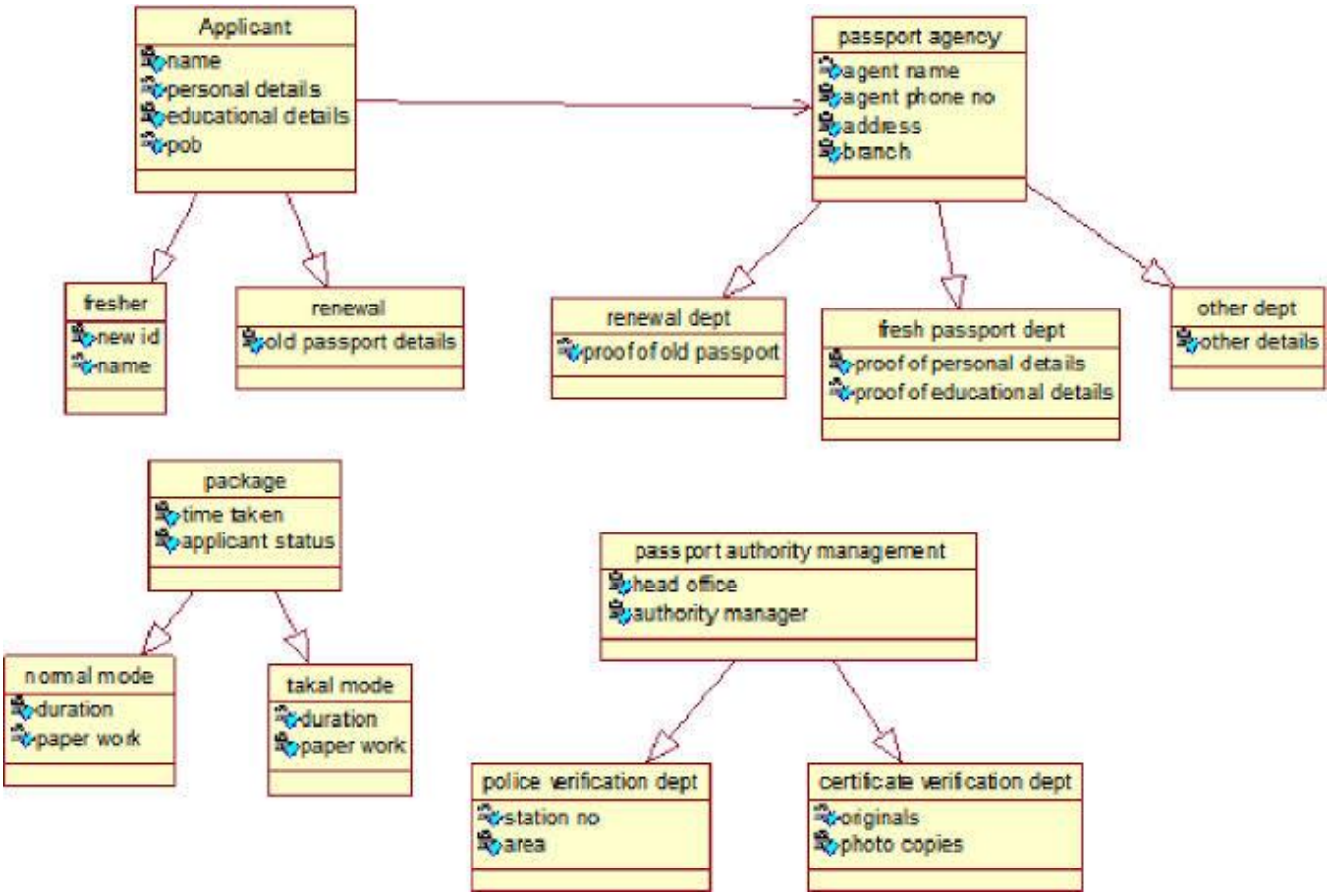


Fig 2. CLASS DIAGRAM FOR PASSPORT OFFICE SYSTEM

3. SEQUENCE DIAGRAM:

A sequence diagram represents the sequence and interactions of a given USE-CASE or scenario. Sequence diagrams can capture most of the information about the system. Most object to object interactions and operations are considered events and events include signals, inputs, decisions, interrupts, transitions and actions to or from users or external devices. The sequence diagram for each USE-CASE that exists when a user become an administrator. Check status and new registration about passport automation system are given.

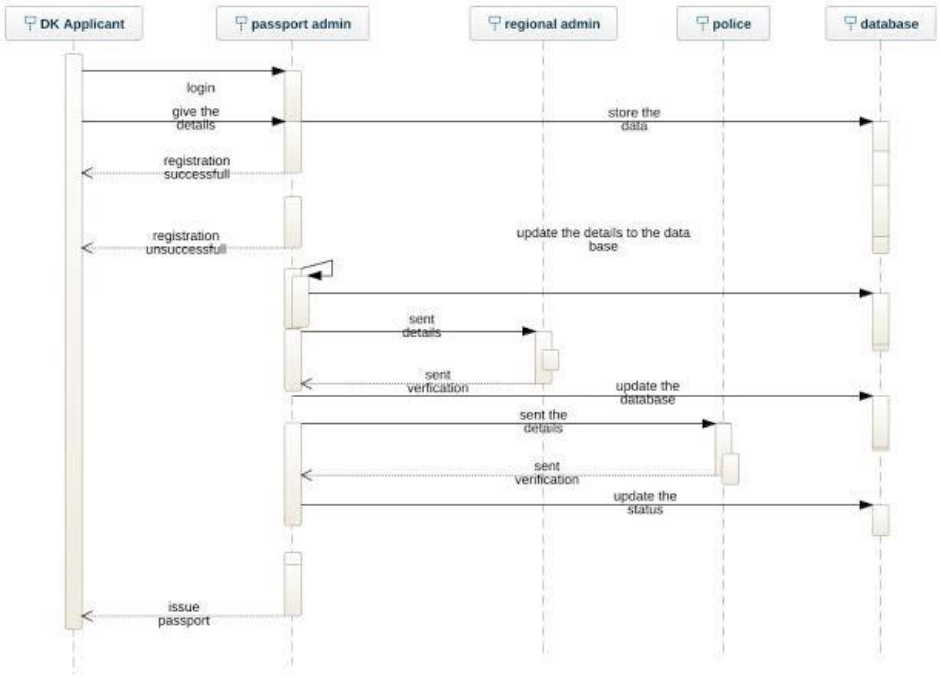


Fig 3. SEQUENCE DIAGRAM

4. COLLABORATION DIAGRAM:

Communication diagram illustrate that object interact on a graph or network format in which object can be placed where in the diagram. In collaboration diagram the object can be placed in anywhere on the diagram. The collaboration comes from sequence diagram.

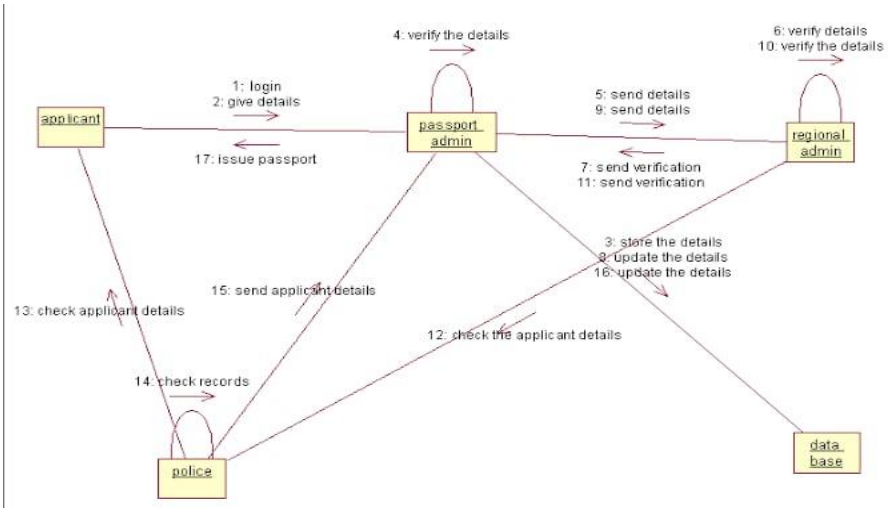


Fig 4. COLLABORATION DIAGRAM

5. **ACTIVITY DIAGRAM**:

The activity diagram shows the activity of the process here first login was done when the user is valid then the welcome page appears. The activity diagram represents the series of activities that are occurring between the objects. Following is activity diagram which represents the Software personnel management system process

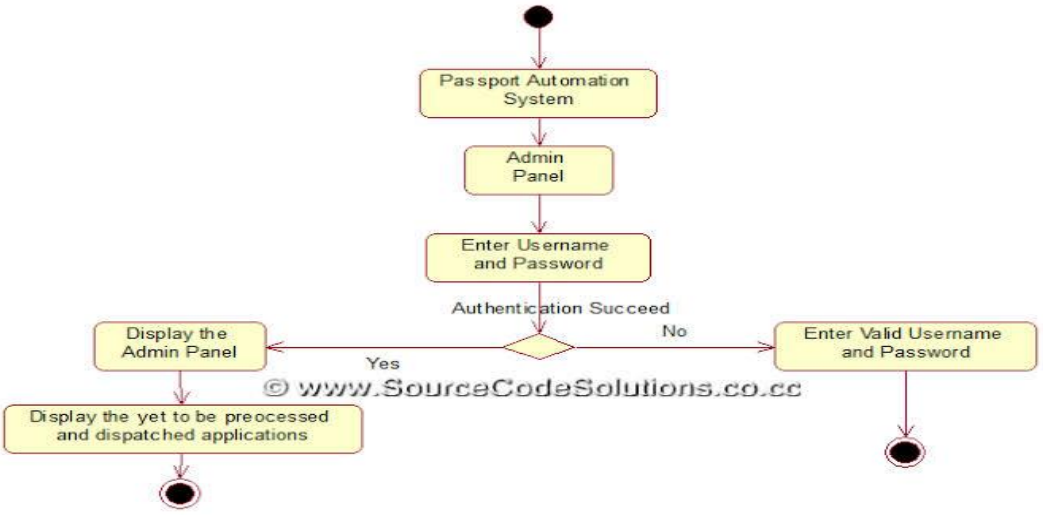


Fig 5. ACTIVITY DIAGRAM

6. DEPLOYMENT DIAGRAM:

Deployment diagram shows the assignment of concrete software artifact to computational nodes. It shows the deployment of software elements to the physical elements. Deployment diagram are useful to communicate or deployment architecture.

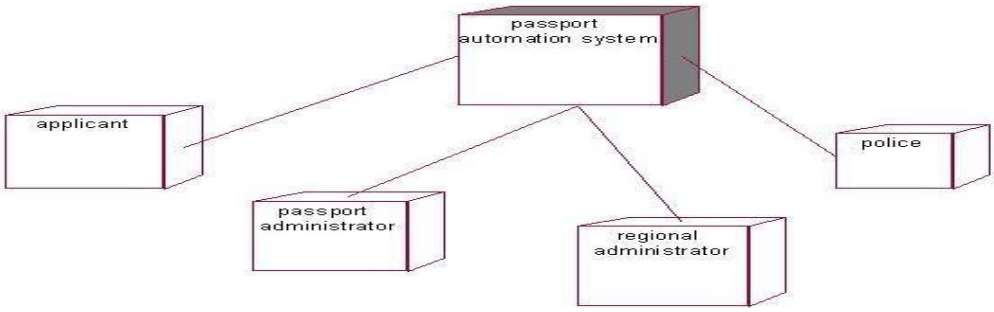


Fig 6. DEPLOYMENT DIAGRAM

7. **COMPONENT DIAGRAM**:

The component diagram is represented by figure dependency and it is a graph of design of figure dependency. The component diagram's main purpose is to show the structural relationships between the components of a systems. It is represented by boxed figure. Dependencies are represented by communication association.

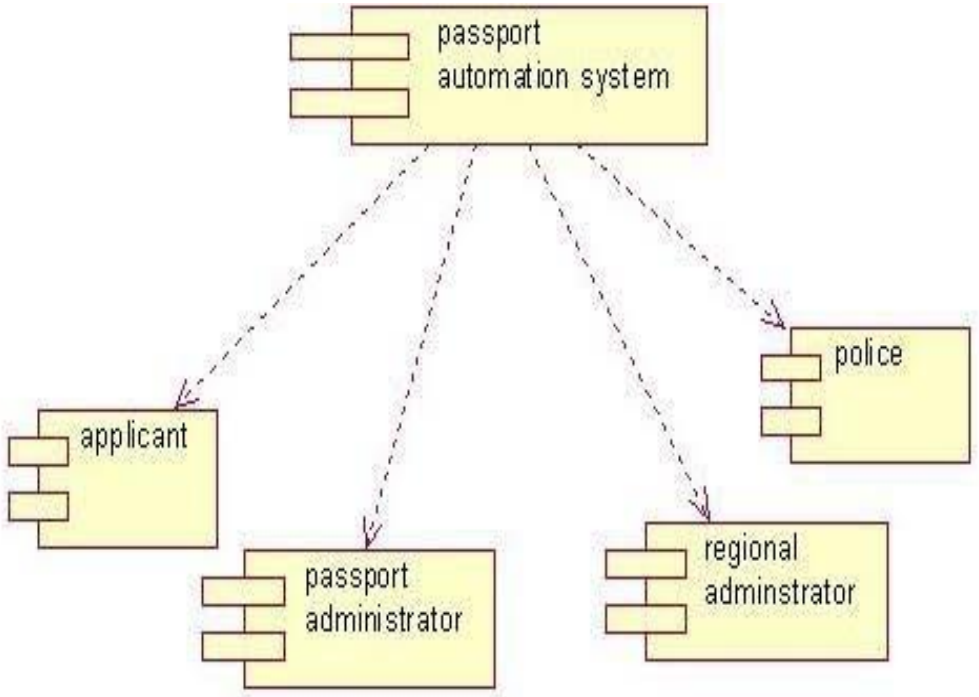


Fig 7. COMPONENT DIAGRAM

RESULT:

Thus the various UML diagrams for passport automation system have been created and the code was generated successfully.

EX NO. 10 CASE STUDY- USE OF TESTING TOOL SUCH AS JUNIT DATE:

AIM- Use of testing tool such as JUnit

Objective: -

To show how unit testing is carried out in java using Junit.

Software Required: -

Eclipse IDE and Junit

OVERALL DESCRIPTION:

JUnit is an open source Java testing framework used to write and run repeatable tests which is wildly used now days**.**

Unit Test

Many different types of tests are implemented and executed. In this report we only talk about JUnit with Unit Test Unit test is a method of testing that verifies the individual units of source code are working properly. A unit is the smallest testable part of an application. In object-oriented programming, the smallest unit is a method, which may belong to a base/super class, abstract class or derived/child class. A unit test examinesthe behavior of a distinct task that is not directly dependent on the completion of any other task.

What is JUnit

JUnit is a simple, open source framework to write and run repeatable tests for Java Programming Language. It is an instance of the unit architecture for unit testing frameworks. JUnit features include:

- Assertions for testing expected results
- Test fixtures for sharing common test data
- Test suites for easily organizing and running tests
- Graphical and textual test runners

Method Design

Here creation of unit test will be described. There are few rules how to write the JUnit method. First of all, we have to create the test class in which all test method will be. It's good to name the test case class after the class under test.

Criteria for test methods

For JUnit to recognize a method as test method, it must meet the following criteria:

- The method must be declared public.
- The method must return nothing (void).
- The name of the method must start with the word test.
- The method can't take any arguments.

Assertion methods

An assertion is a function or macro that verifies the behavior of the unit under test. Failure of an assertion typically throws an exception, aborting the execution of the current test.

The Test Case class extends a utility class name Assert in the JUnit framework. The Assert class are included methods which are used to make assertions about the state of testing object. Some assert methods are as follows.

Assert Equals ([String message], Object expected, Object actual)

This method checks that two values are equal. If they are not, the methodthrows an Assertion Failed Error with the given message (if any).

Assert False ([String message], Boolean condition)

This method checks that a condition is false. If it isn't, the method throws an Assertion Failed Error with the given message (if any).

Assert Not Null ([String message], Object object)

This method checks that the object is not null. If it is, the method throws an Assertion Failed Error with the given message (if any).

Assert Not Same ([String message], Object expected, Object actual)

This method checks that two objects not refer to the same object using == operator. If they do, the method throws an Assertion Failed Error with the given message (if any).

Assert Null ([String message], Object object)

This method checks that the object is null. If it isn't, the method throws an Assertion Failed Error with the given message (if any).

Assert same ([String message], Object expected, Object actual)

This method checks that two objects refer to the same object using == operator. If they donot, the method throws an AssertionFailedError with the given message (if any).

Assert True ([String message], Boolean condition)

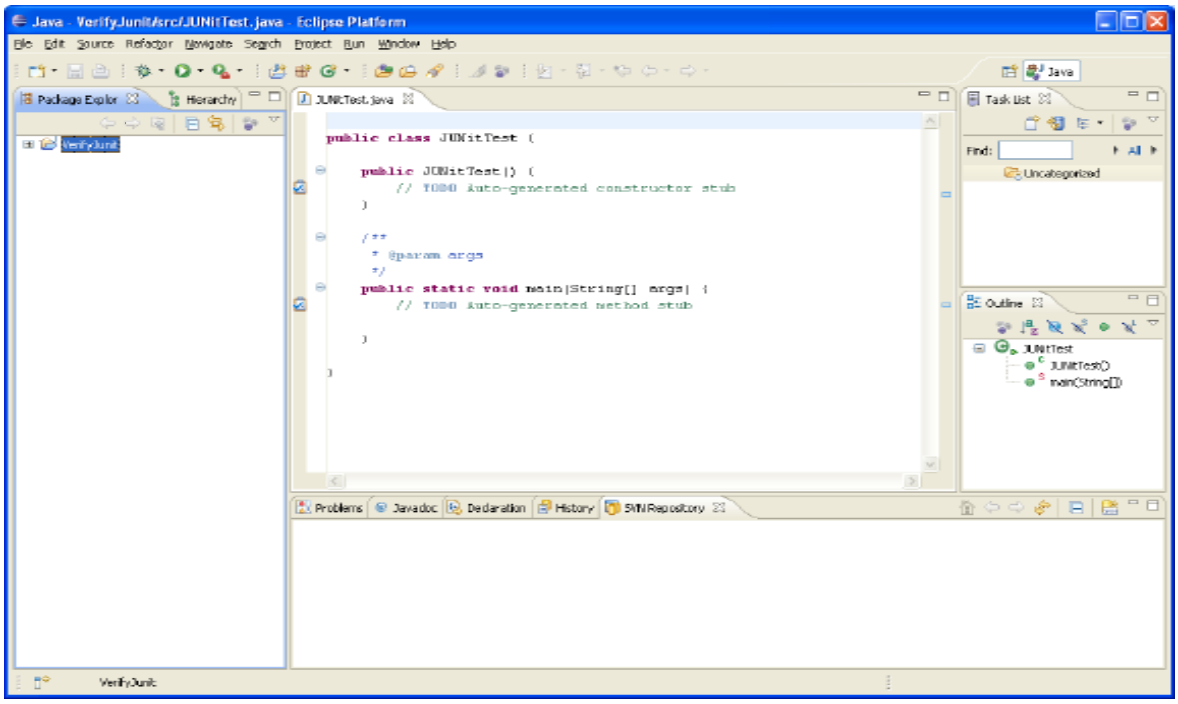
This method checks that a condition is true. If it isn't, the method throws an Assertion Failed Error with the given message (if any).

Fail (String message)

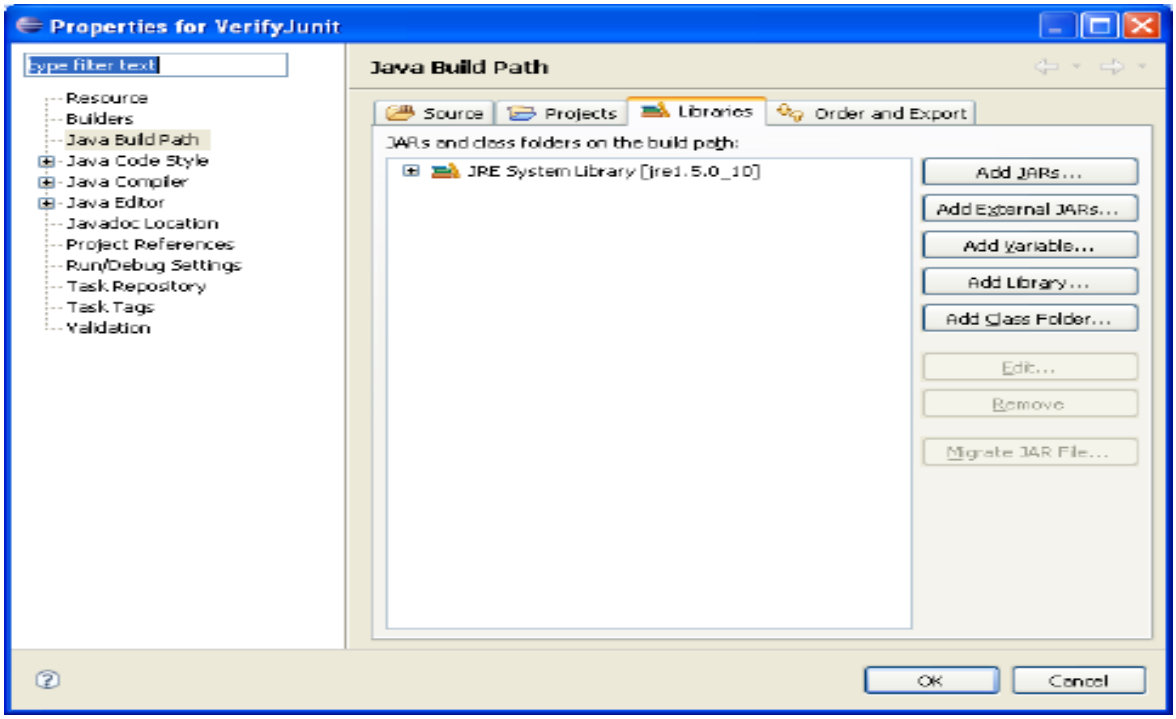
This method fails a test with the given message.

Configure Joint in Eclipse

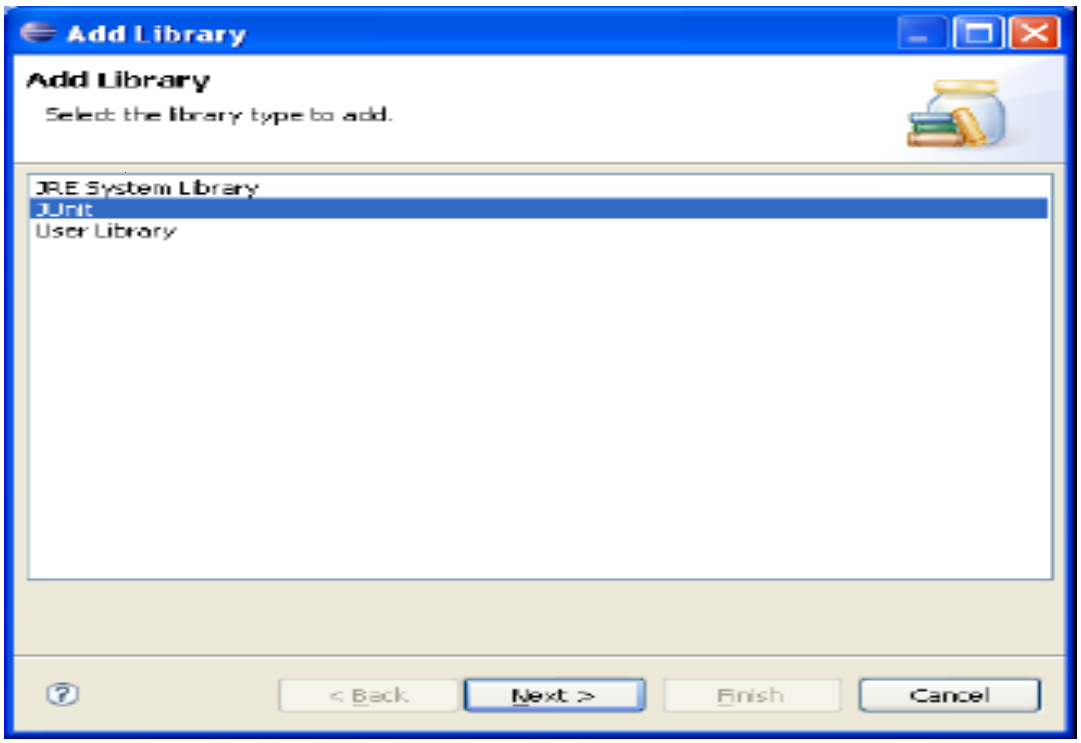
Step 1: In the Package Explorer view, select your project and right click go to properties, the last item in the Menu. Your Project Name will be different.



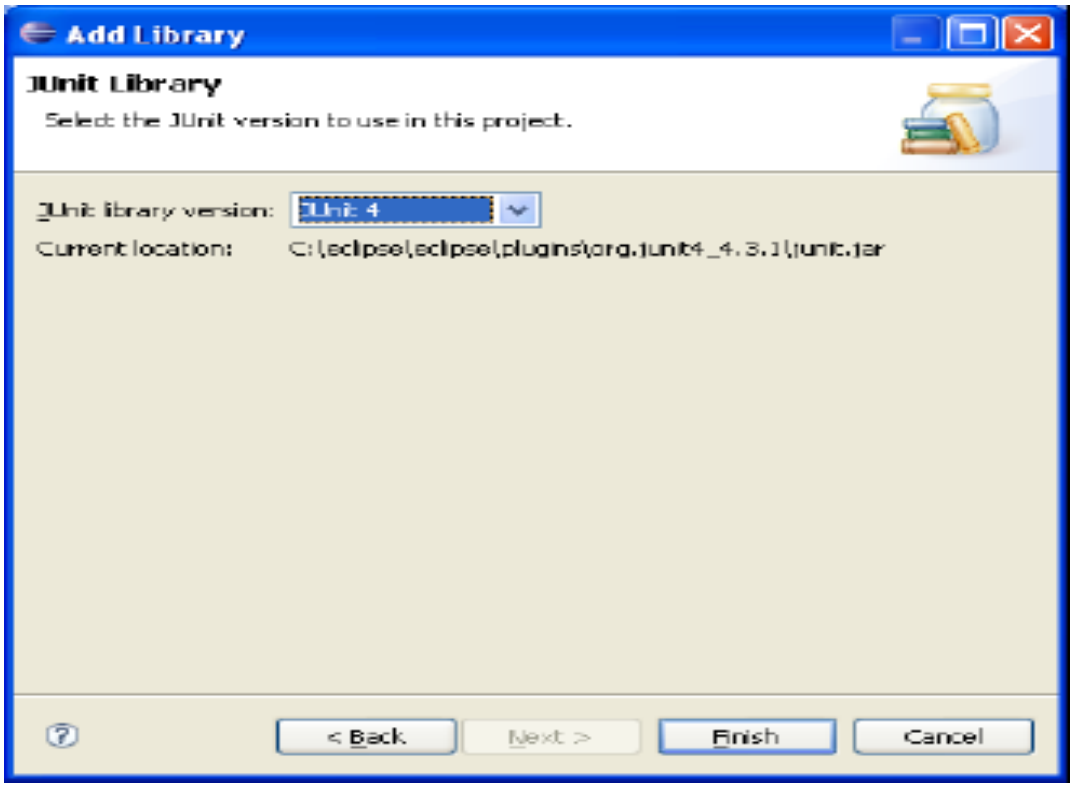
Step 2: Select the Java Build Path and the Libraries tab. The window should look like this.



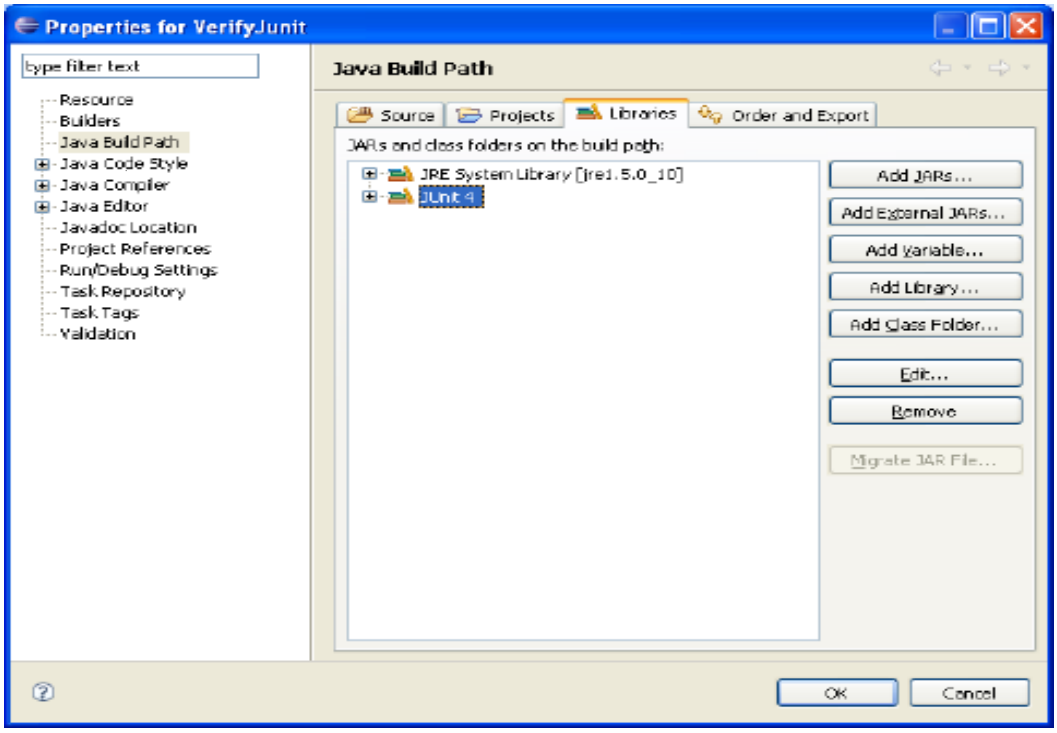
Step 3: Select Add Library and select Junit



Step 4: Select JUnit 4, then Finish.



Step 5: Your Window should look like this.



Conclusion:

The junit testing tool can be implemented successfully by following the steps described above.