ANNAMACHARYA INSTITUTE OF TECHNOLOGY & SCIENCES

VENKATAPURAM, TIRUPATI, CHITTOOR DT.
ANDHRA PRADESH – 517520.

2021 – 2022

LABORATORY MANUAL

OF

SOFTWARE ENGINEERING LAB

(AK-20 REGULATION)

Prepared by

Dr. S. ATHINARAYANAN
PROFESSOR

B.Tech III YEAR – I SEM. (CSE)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ENGINEERING COLLEGE

(AFFILIATED TO JNTUA ANANTHAPURAM- AICITE-INDIA)
AYYALURUMETTA (V), NANDYAL, KURNOOL DT.
ANDHRA PRADESH – 518502

LAB MANUAL CONTENT

SOFTWARE ENGINEERING LAB

Course Objectives:

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

List of Experiments:

1. a) Draw the Control Flow Graph of following using MS-Word:
    i. if-else
    ii. while
    iii. do-while
    iv. for

   b) Draw the Flow chart and CFG for the following Program by using MS Word:

```
if  A = 10 then
if B > C
A = B
else
A = C
endif
endif
print A, B, C.
```

2. Define Functional and Non-Functional Requirements for Hospital Management System.
3. Draw the Deliverable and Phase based Work Breakdown Structure for House construction System using MS Word.
4. Schedule all the Task and sub-Task using the PERT/CPM charts using MS –Excel.

5 Identify and analyze all the possible risks and its risk mitigation plan for the system to be automated

6 Diagnose any risk using Ishikawa Diagram (Can be called as Fish Bone Diagram or Cause & Effect Diagram)

7 Define Complete Project plan for the system to be automated using Microsoft Project Tool

8 Define the Features, Vision, Business objectives, Business rules and stakeholders in the vision document

9 Define the functional and non-functional requirements of the system to be automated by using Usecases and document in SRS document

10 Define the following tracebility matrices:

1. Usecase Vs. Features
2. Functional requirements Vs.Usecases

11. Estimate the effort using the following methods for the system to be automated:

1.Function point metric

2.Usecase point metric

12. Develop a tool which can be used for quantification of all the non-functional requirements
13. Write C/C++/Java/Python program for classifying the various types of coupling.
14. Write a C/C++/Java/Python program for classifying the various types of cohesion.

15. Write a c program to demonstrate the working of the Following constructs:

i) do…while
ii) while…do
ii) if-else
iii)switch
iv)for loop.

16. A program written in c language for matrix multiplication fails —Introspect the causes for its failure and write down the possible reasons for its failure.
17. Take ATM system and study its system specifications and report the various bugs.
18. Write the test cases for Banking application.
19. Create a test plan document for Library Management System.
20. Design and develop a program in a language of your choice to solve the triangle problem defined as follows: Accept three integers which are supposed to be the three sides of a triangle and determine if the three values represent an equilateral triangle, isosceles triangle, scalene triangle, or they do not form a triangle at all. Assume that the upper limit for the size of any side is 10. Derive test cases for your program based on boundary-value analysis, execute the test cases and discuss the results.
21. Design and develop a program in a language of your choice to solve the triangle problem defined as follows: Accept three integers which are supposed to be the three sides of a triangle and determine if the three values represent an equilateral triangle, isosceles triangle, scalene triangle, or they do not form a triangle at all. Derive test cases for your program based on decision table approach, execute the test cases and discuss the results.
22. Design and develop a program in a language of your choice to solve the triangle problem defined as follows: Accept three integers which are supposed to be the three sides of a triangle and determine if the three values represent an equilateral triangle, isosceles triangle, scalene triangle, or they do not form a triangle at all. Assume the upper limit for the size of any side is 10. Derive test cases for your program based on equivalence class partitioning, execute the test cases and discuss the results.
23. Draw standard UML diagrams using an UML modeling tool for a given case study and map design to code and implement a 3 layered architecture. Test the developed code and validate whether the SRS is satisfied.

A. Identify a software system that needs to be developed.
B. Document the Software Requirements Specification (SRS) for the identified system.
C. Identify use cases and develop the Use Case model.
D. Identify the conceptual classes and develop a Domain Model and also derive a Class Diagram from that.
E. Using the identified scenarios, find the interaction between objects and represent themusing UML Sequence and Collaboration Diagrams
F. Draw relevant State Chart and Activity Diagrams for the same system.
G. Implement the system as per the detailed design
H. Test the software system for all the scenarios identified as per the usecase diagram
I. Improve the reusability and maintainability of the software system by applying appropriate design patterns.
J. Implement the modified system and test it for various scenarios

Suggested domain for validate the following system:

a. Passport automation system.
b. Book bank
c. Exam registration
d. Stock maintenance system.
e. Online course reservation system

Unit Outcomes

Student is able to

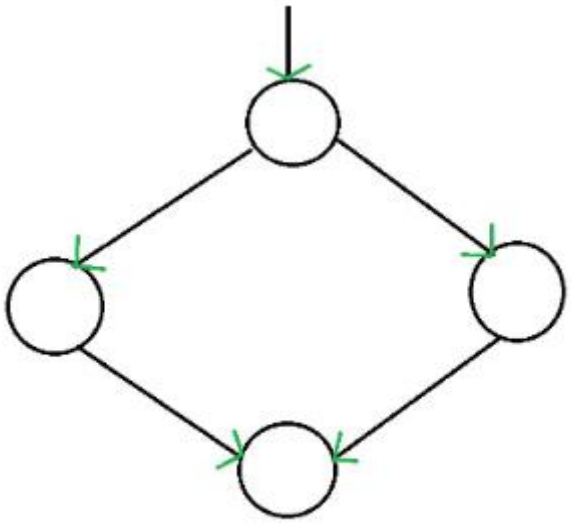
- Acquaint with historical and modern software methodologies
- Understand the phases of software projects and practice the activities of each phase
- Practice clean coding
- Take part in project management
- Adopt skills such as distributed version control, unit testing, integration testing, build management, and deployment

EXPERIMENT - 1

1.a) Draw the Control Flow Graph of following using MS-Word:

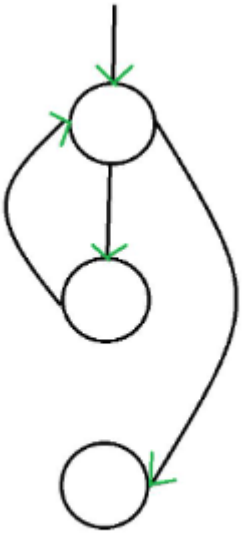
v. if-else
vi. while
vii. do-while
viii. for

1. **If-else:**



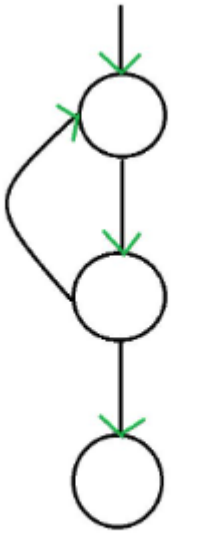
If-then-else

2. **while:**

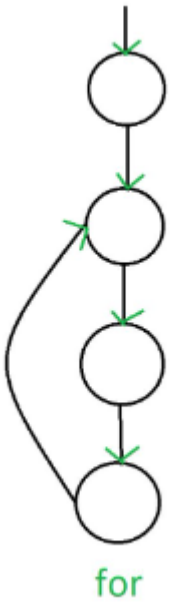


while

3.do-while:



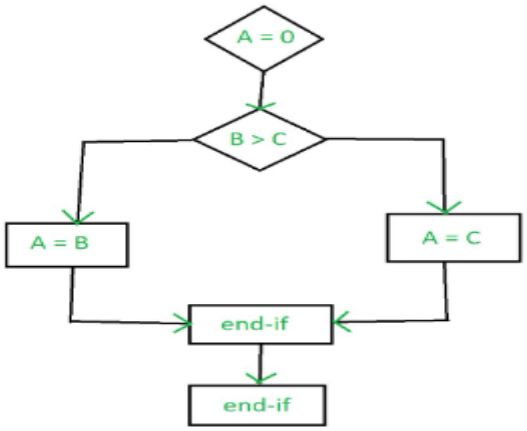
4. **for:**



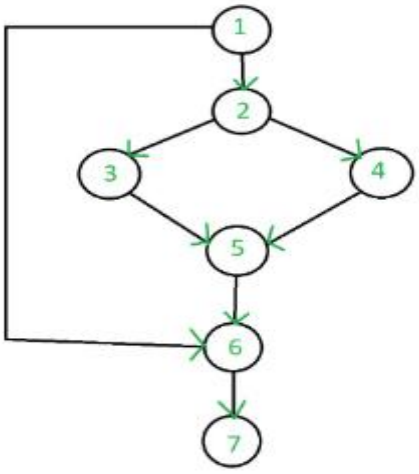
b) Draw the Flow chart and CFG for the following Program by using MS Word:

```
if  A = 10 then
  if B > C
     A = B
  else
     A = C
  endif
  endif
print A, B, C.
```

Flow Chart



Control Flow Graph



Control Flow Graph

Result:

a) Control Flow Graph for the if-else, while, do-while and for loop is successfully drawn.
b) Flow chart and CFG for the given Program has been successfully drawn.

EXPERIMENT - 2

Define Functional and Non-Functional Requirements for Hospital Management System.

Hospital Management System is used to take the data from the patients and then store it for later use. The main goal of the Hospital Management System is to accurately treat as well as decrease overtime pay.

There are various features included in the HMS. Some of the system functions include Registration, Patient check out, Report generation, and more. In this blog, let's check out the functional and non-functional requirements of the Hospital Management System in depth.

Functional Requirements:

There are a lot of software requirements specifications included in the functional requirements of the Hospital Management System, which contains various process, namely Registration, Check out, Report Generation, and Database.

Registration Process of SRS (Software Requirements Specification)

● Adding Patients: The Hospital Management enables the staff in the front desk to include new patients to the system.

● Assigning an ID to the patients: The HMS enables the staff in the front desk to provide a unique ID for each patient and then add them to the record sheet of the patient. The patients can utilize the ID throughout their hospital stay.

Check Out of SRS:

● Deleting Patient ID: The staff in the administration section of the ward can delete the patient ID from the system when the patient's checkout from the hospital.

● Adding to beds available list: The Staff in the administration section of the ward can put the bed empty in the list of beds-available.

Report Generation of SRS:

● Information of the Patient: The Hospital Management System generates a report on every patient regarding various information like patients name, Phone number, bed number, the doctor's name whom its assigns, ward name, and more.

● Availability of the Bed: The Hospital Management system also helps in generating reports on the availability of the bed regarding the information like bed number unoccupied or occupied, ward name, and more.

Database of SRS:

● Mandatory Patient Information: Every patient has some necessary data like phone number, their first and last name, personal health number, postal code, country, address, city, 'patient's ID number, etc.

● Updating information of the Patient: The hospital management system enables users to update the information of the patient as described in the mandatory information included.

Non Functional Requirements

There are a lot of software requirements specifications included in the non-functional requirements of the Hospital Management System, which contains various process, namely Security, Performance, Maintainability, and Reliability.

Security:

● Patient Identification: The system needs the patient to recognize herself or himself using the phone.

● Logon ID: Any users who make use of the system need to hold a Logon ID and password.

● Modifications: Any modifications like insert, delete, update, etc. for the database can be synchronized quickly and executed only by the ward administrator.

● Front Desk Staff Rights: The staff in the front desk can view any data in the Hospital Management system, add new patients record to the HMS but they don't have any rights alter any data in it.

● Administrator rights: The administrator can view as well as alter any information in the Hospital Management System.

Performance:

● Response Time: The system provides acknowledgment in just one second once the 'patient's information is checked.

● Capacity: The system needs to support at least 1000 people at once.

● User-Interface: The user interface acknowledges within five seconds.

● Conformity: The system needs to ensure that the guidelines of the Microsoft accessibilities are followed.

Maintainability:

● Back-Up: The system offers the efficiency for data back up.

● Errors: The system will track every mistake as well as keep a log of it.

Reliability:

● Availability: The system is available all the time.

Hope you got a clear idea on the functional and non-functional requirements and the features required by the hospital. Any other queries on the topic are welcome.

Result:

The Functional and Non Functional Requirements of Hospital Management System is defined successfully.

EXPERIMENT – 3

Draw the Deliverable and Phase based Work Breakdown Structure for House construction System using MS Word.

AIM:

To draw the deliverable and Phase based work breakdown structure for and the House construction system by using MS Word.

Description:

A work breakdown structure starts with a large project or objective and breaks it down into smaller, more manageable pieces that you can reasonably evaluate and assign to teams. Rather than focusing on individual actions that must be taken to accomplish a project, a WBS generally focuses on deliverables or concrete, measurable milestones.

STEPS FOR CREATING WORK BREAKDOWN STRUCTURE:

A good Work Breakdown Structure is created using an iterative process by following these steps and meeting these guidelines:

Gather Critical Documents
Gather critical project documents.
Identify content containing project deliverables, such as the Project Charter, Scope Statement and Project Management Plan (PMP) subsidiary plans.

Identify Key Team Members
Identify the appropriate project team members. Analyze the documents and identify the deliverables.

Define Level 1 Elements
Define the Level 1 Elements. Level 1 Elements are summary deliverable descriptions that must capture 100% of the project scope. Verify 100% of scope is captured. This requirement is commonly referred to as the 100% Rule.

Decompose (Breakdown) Elements
Begin the process of breaking the Level 1 deliverables into unique lower Level deliverables. This "breaking down" technique is called Decomposition. Continue breaking down the work until the work covered in each Element is managed by a single individual or organization. Ensure that all Elements are mutually exclusive.

Create Wbs Dictionary
Define the content of the WBS Dictionary. The WBS Dictionary is a narrative description of the work covered in each Element in the WBS. The lowest Level Elements in the WBS are called Work Packages.

Create Gantt Chart Schedule
Decompose the Work Packages to activities as appropriate. Export or enter the Work Breakdown Structure into a Gantt chart for further scheduling and project tracking.

Deliverable-Based Work Breakdown Structure

A Deliverable-Based Work Breakdown Structure clearly demonstrates the relationship between the

project deliverables (i.e., products, services or results) and the scope (i.e., work to be executed). Figure 1 is an example of a Deliverable-Based WBS for building a house. Figure 2 is an example of a Phase-Based WBS for the same project.

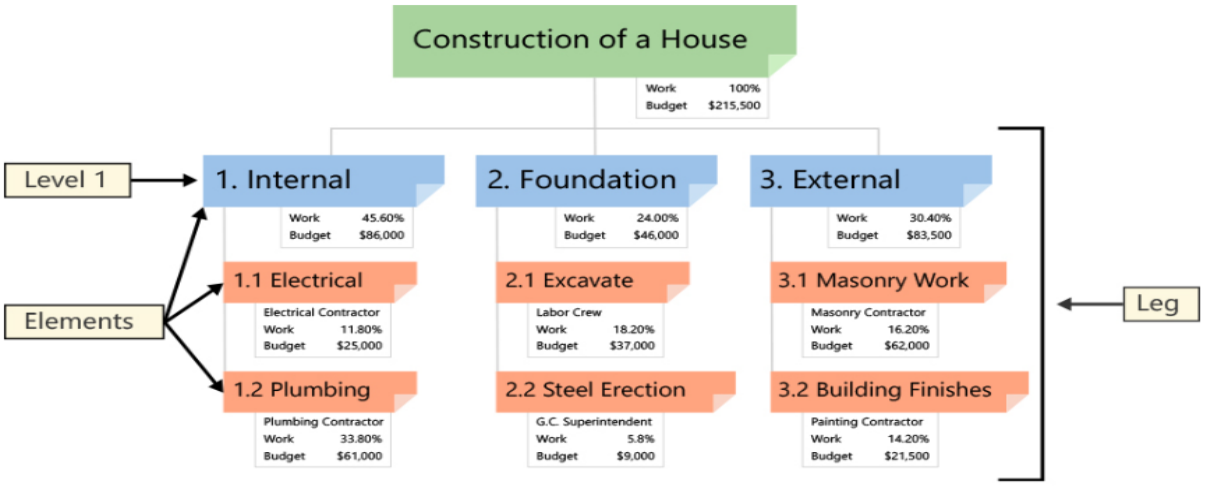


FIGURE 1 – DELIVERABLE BASED WORK BREAKDOWN STRUCTURE

Phase-Based Work Breakdown Structure

In Figure 2, a Phase-Based WBS, the Level 1 has five Elements. Each of these Elements is typical phases of a project. The Level 2 Elements are the unique deliverables in each phase. Regardless of the type of WBS, the lower Level Elements are all deliverables. Notice that Elements in different Legs have the same name. A Phase-Based WBS requires work associated with multiple elements be divided into the work unique to each Level 1 Element.

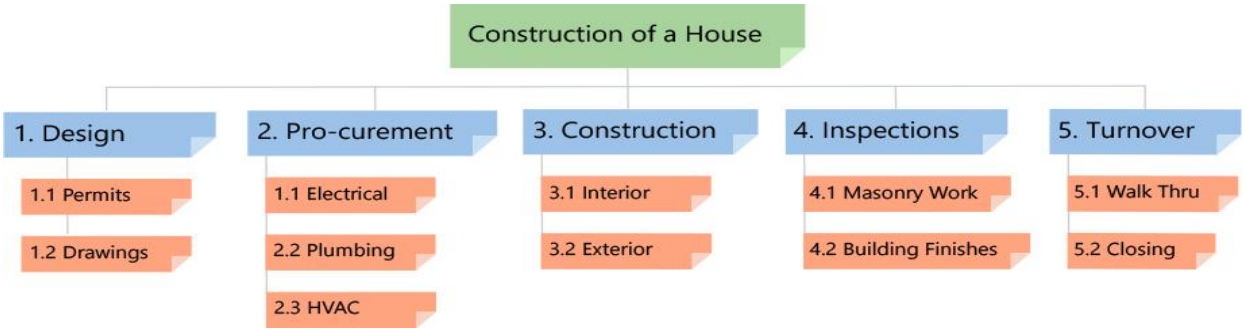


FIGURE 2 - PHASE BASED WORK BREAKDOWN STRUCTURE

RESULT:

Successfully drawn the deliverable and Phase based work breakdown structure for and the House construction system by using MS Word.

EXPERIMENT – 4

Schedule all the Task and sub-Task using the PERT/CPM charts

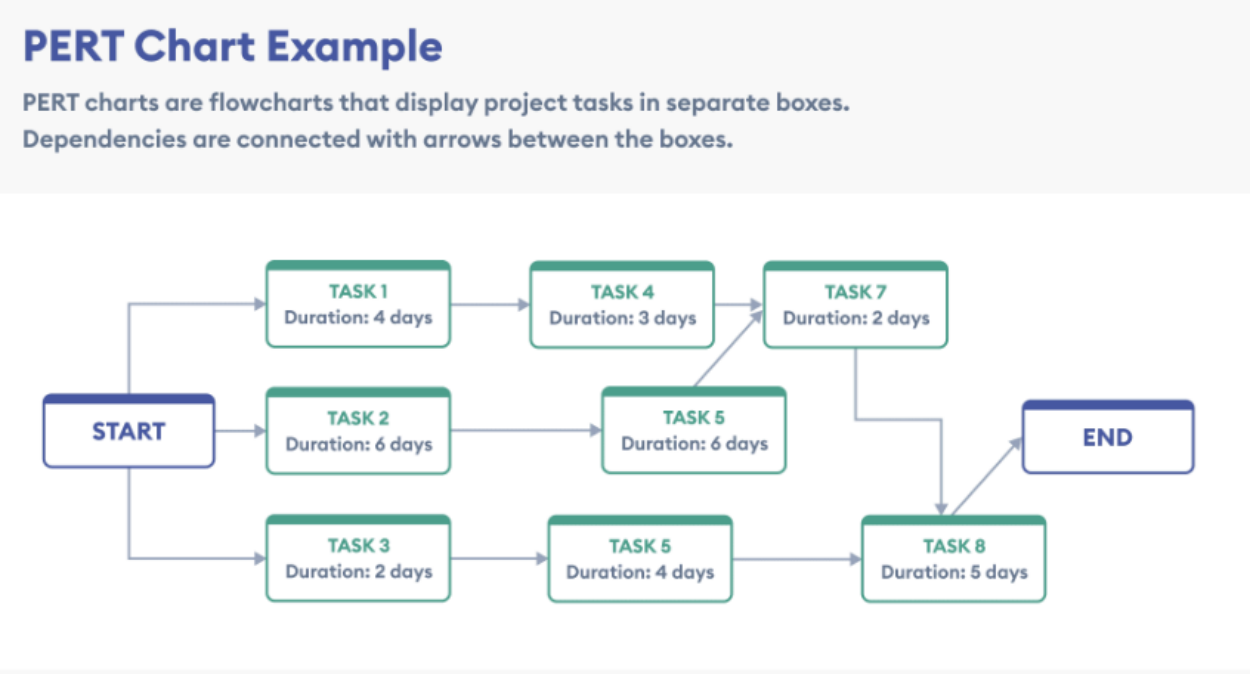
AIM:

To Schedule all the task and sub tasks using the PERT/CPM Charts using Excel.

INPUT:

Program Evaluation Review Technique (PERT)

A PERT chart is a project management tool that provides a graphical representation of a project's timeline. The Program Evaluation Review Technique (PERT) breaks down the individual tasks of a project for analysis. A PERT chart uses circles or rectangles called nodes to represent project events or milestones. These nodes are linked by vectors or lines that represent various tasks.If an arrow is drawn from Task No. 1 to Task No. 2 on a PERT chart, Task No. 1 must be completed before work on Task No. 2 begins.



Critical Path Method (CPM)

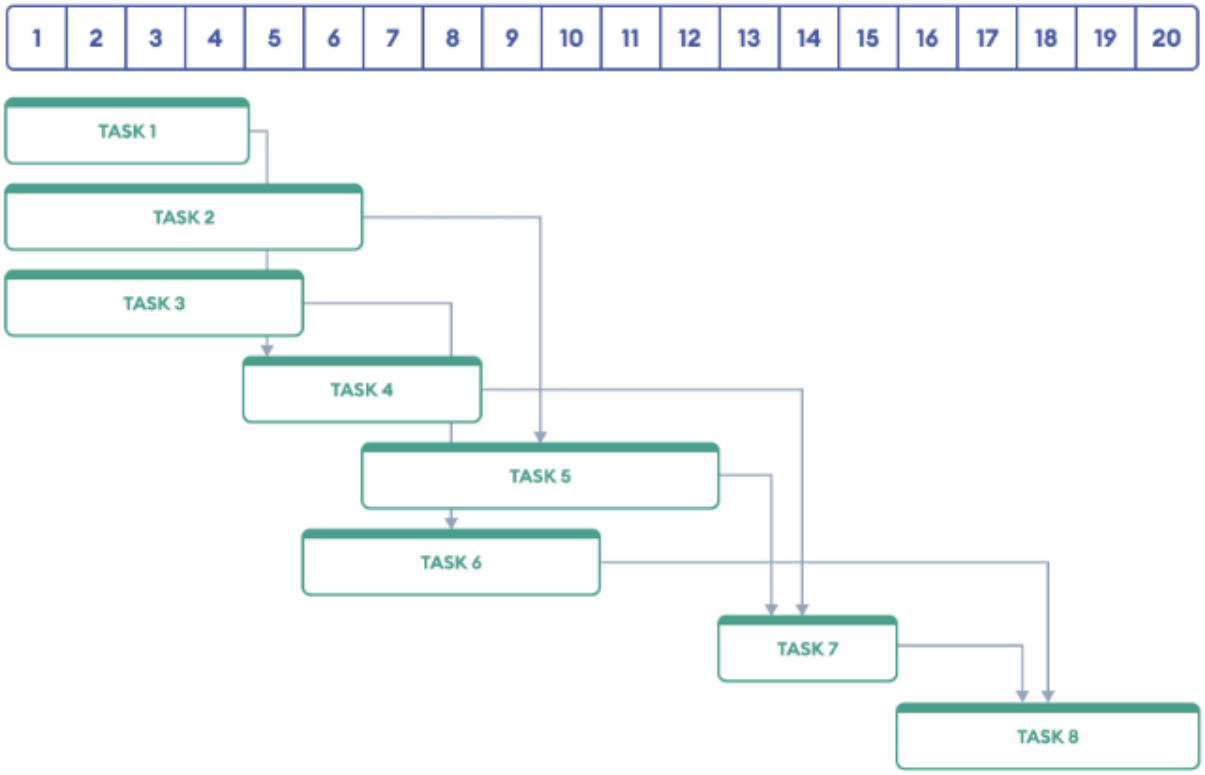
The critical path method (CPM) is a step-by-step project management technique for process planning that defines critical and non-critical tasks with the goal of preventing time-frame problems and process bottleneThe cks. CPM is ideally suited to projects consisting of numerous activities that interact in a complex manner

In applying the CPM, there are several steps that can be summarized as follows:

- Define the required tasks and put them down in an ordered (sequenced) list.
- Create a flowchart or other diagram showing each task in relation to the others.
- Identify the critical and non-critical relationships (paths) among tasks.
- Determine the expected completion or execution time for each task.
- Locate or devise alternatives (backups) for the most critical paths.

Gantt Chart Example

Gantt charts illustrate project tasks in a linear format, along with the time allotted for each & their interdependencies.



RESULT:

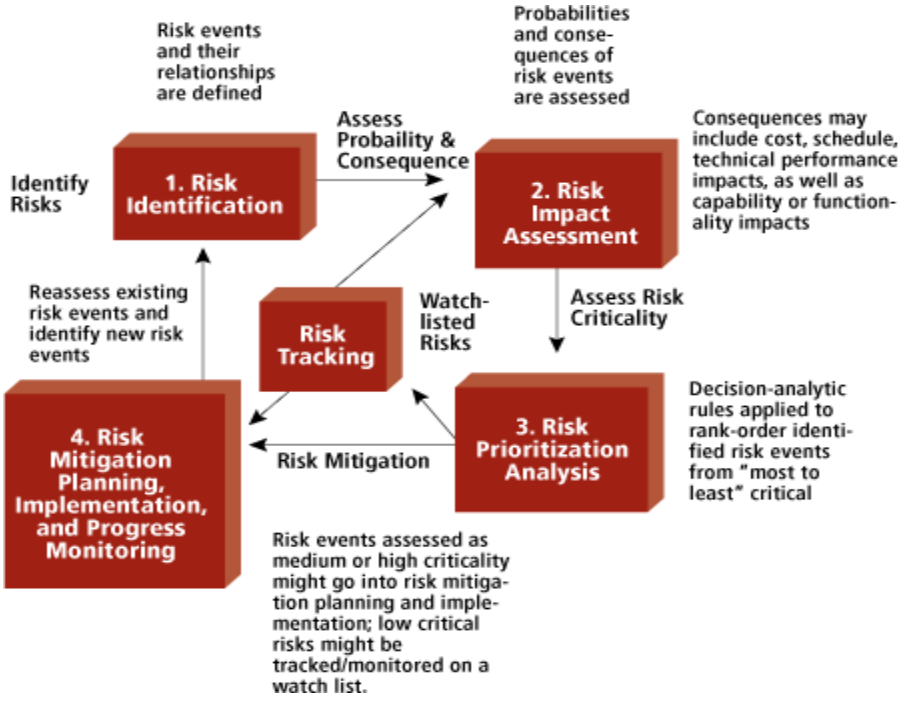
All the task and sub tasks successfully scheduled by using the PERT/CPM Charts

Experiment-5

Problem Statement: Identify and analyze all the possible risks and its risk mitigation plan for the system to be automated

Procedure:

Definition: Risk mitigation planning is the process of developing options and actions to enhance opportunities and reduce threats to project objectives. Risk mitigation implementation is the process of executing risk mitigation actions. Risk mitigation progress monitoring includes tracking identified risks, identifying new risks, and evaluating risk process effectiveness throughout theproject.



Example:

Evolution of Healthcare Enterprise Risk Management (ERM)

ERM encompasses eight risk domains:

1. Operational
2. Clinical & Patient Safety
3. Strategic
4. Financial
5. Human Capital
6. Legal & Regulatory
7. Technological
8. Environmental- and Infrastructure-Based Hazards.

Create a Healthcare Risk Management Plan

There are some fundamental components that belong in all healthcare risk management plans:

Education & Training:

Risk management plans need to detail employee training requirements which should include new employee orientation, ongoing and in-service training, annual review and competency validation, and event-specific training.

Patient & Family Grievances:

To promote patient satisfaction and reduce the likelihood of litigation, procedures for documenting and responding to patient and family complaints should be described in the Risk Management Plan. Response times, staff responsibilities, and prescribed actions need to be articulated and communicated.

Purpose, Goals, &Metrics:

Risk management plans should clearly define the purpose and benefits of the healthcare risk management plan. Specific goals to reduce liability claims, sentinel events, near misses, and the overall cost of the organization's risk should also be well-articulated. Additionally, reporting on quantifiable and actionable data should be detailed and mandated by the plan.

Communication Plan:

While it is critical that the healthcare risk management team promote open and spontaneous dialogue, information about how to communicate about risk and with whom should be provided in the healthcare risk management plan. Next steps and follow-up activities should be documented. It is essential as well that the plan detail reporting requirements to departments and C-Suite personnel. Furthermore, the plan should promote a safe, "no-blame" culture and should include anonymous reporting capabilities.

Contingency Plans:

Risk management plans also need to include contingency preparation for adverse system-wide failures and catastrophic situations such as malfunctioning EHR systems, security breaches, and cyber-attacks. The plan needs to include emergency preparedness for things like disease outbreaks, long-term power loss, and terror attacks or mass shootings.

Reporting Protocols:

Every healthcare organization must have a quick and easy-to-use, system for documenting, classifying, and tracking possible risks and adverse events. These systems must include protocols for mandatory reporting.

Response & Mitigation:

Plans for healthcare risk must also include collaborative systems for responding to reported risks and events including acute response, follow-up, reporting, and repeat failure prevention.

Experiment-6

Problem Statement: Diagnose any risk using Ishikawa Diagram (Can be called as Fish Bone Diagram or Cause & Effect Diagram)

Procedure:

A fishbone diagram is a visualization tool for categorizing the potential causes of a problem. This tool is used in order to identify a problem's root causes. Typically used for root cause analysis, a fishbone diagram combines the practice of brainstorming with a type of mind map template. It should be efficient as a test case technique to determine cause and effect.

A fishbone diagram is useful in product development and troubleshooting processes, typically used to focus a conversation around a problem. After the group has brainstormed all the possible causes for a problem, the facilitator helps the group to rate the potential causes according to their level of importance and diagram a hierarchy.

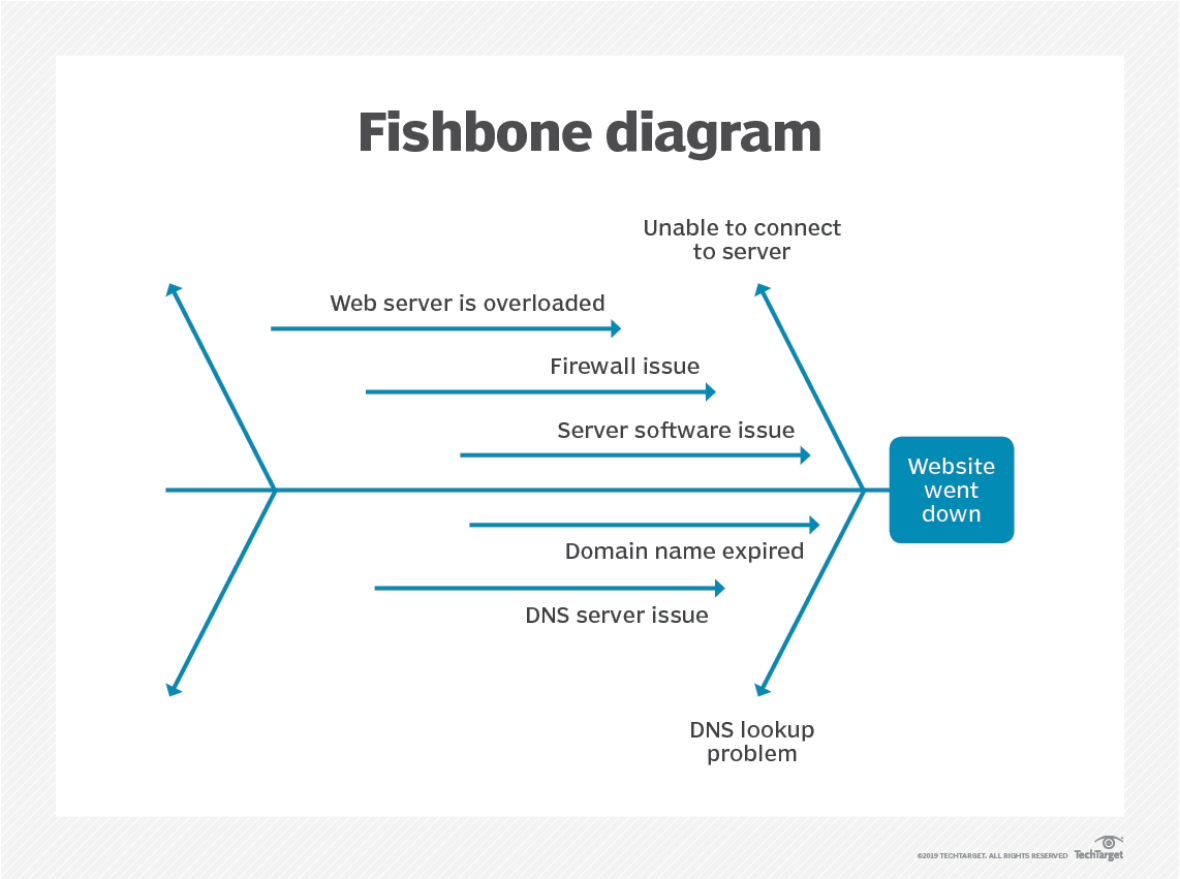
How to create a fishbone diagram

1. The head of the fish is created by listing the problem in a statement format and drawing a box around it. A horizontal arrow is then drawn across the page with an arrow pointing to the head. This acts as the backbone of the fish.
2. Then at least four overarching "causes" are identified that might contribute to the problem. Some generic categories to start with may include methods, skills, equipment, people, materials, environment or measurements. These causes are then drawn to branch off from the spine with arrows, making the first bones of the fish.

3. For each overarching cause, team members should brainstorm any supporting information that may contribute to it. This typically involves some sort of questioning methods, such as the 5 Why's or the 4P's (Policies, Procedures, People and Plant) to keep the conversation focused. These contributing factors are written down to branch off their corresponding cause.
4. This process of breaking down each cause is continued until the root causes of the problem have been identified. The team then analyzes the diagram until an outcome and next steps are agreed upon.

Example of a fishbone diagram

The following graphic is an example of a fishbone diagram with the problem "Website went down." Two of the overarching causes have been identified as "Unable to connect to server" and "DNS lookup problem," with further contributing factors branching off.



When to use a fishbone diagram

A few reasons a team might want to consider using a fishbone diagram are:

- To identify the possible causes of a problem.

- To help develop a product that addresses issues within current market offerings.
- To reveal bottlenecks or areas of weakness in a business process.
- To avoid reoccurring issues or employee burnout.
- To ensure that any corrective actions put into place will resolve the issue.

Experiment-7

Problem Statement:Define Complete Project plan for the system to be automated using Microsoft Project Tool

Procedure:

Microsoft Project Example – Let's create your first real project in a just few steps

Creating a project plan in Microsoft Project isn't difficult at all.

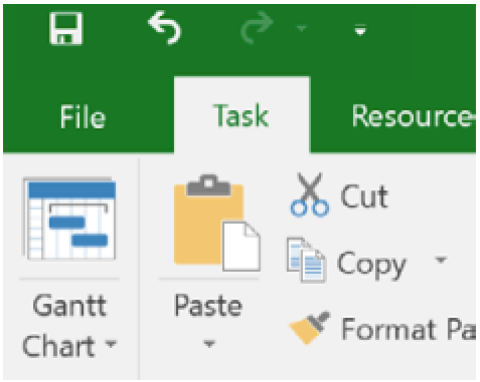
This article will teach you how to create a simple project plan using a real project example.

Some basic configuration before you start

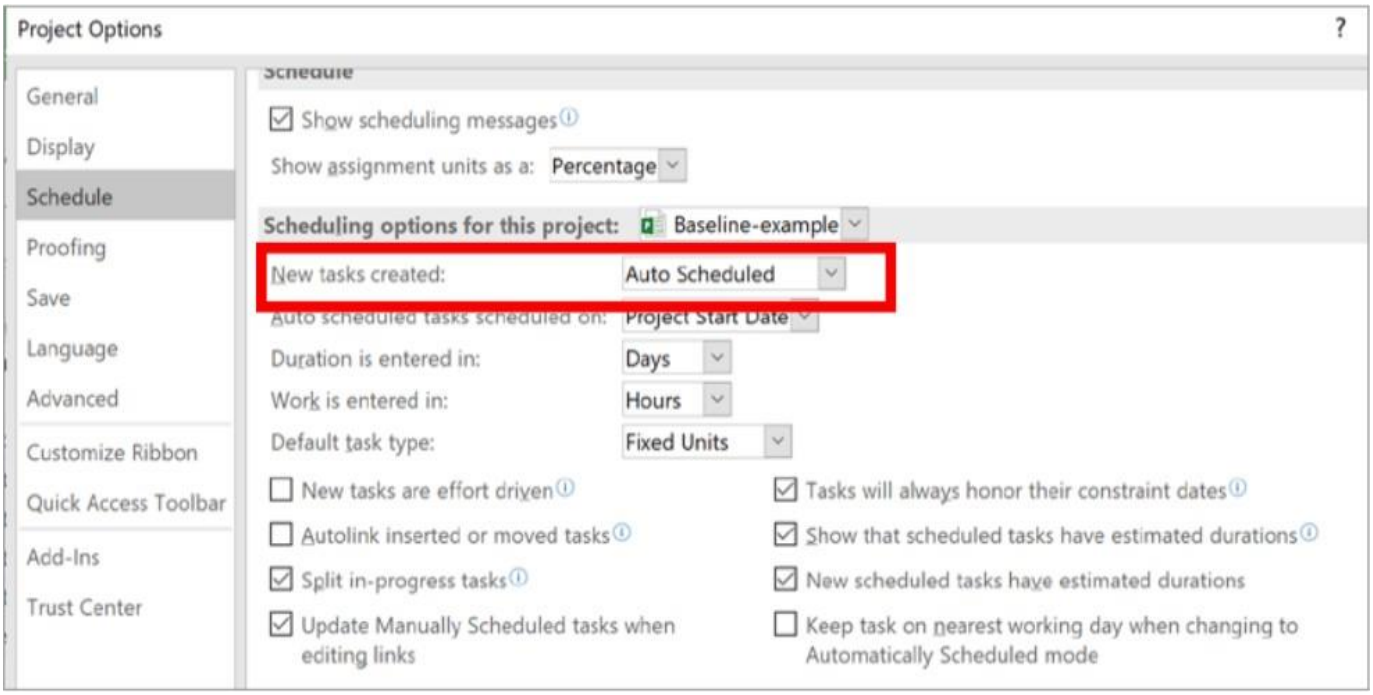
Before you create a schedule, you need to make two important changes in your settings:

Setting change 1: Make Auto Scheduling the default

o to File –> Options



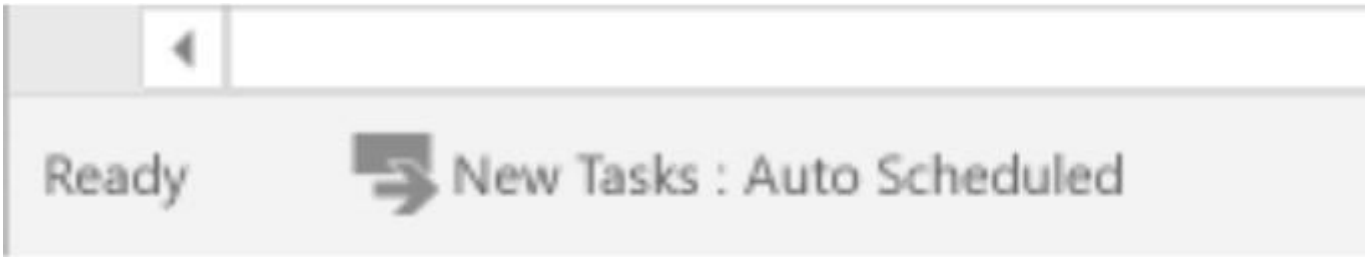
Make sure to set **Auto Schedule** for new tasks:



What does *Auto Scheduled* mean?

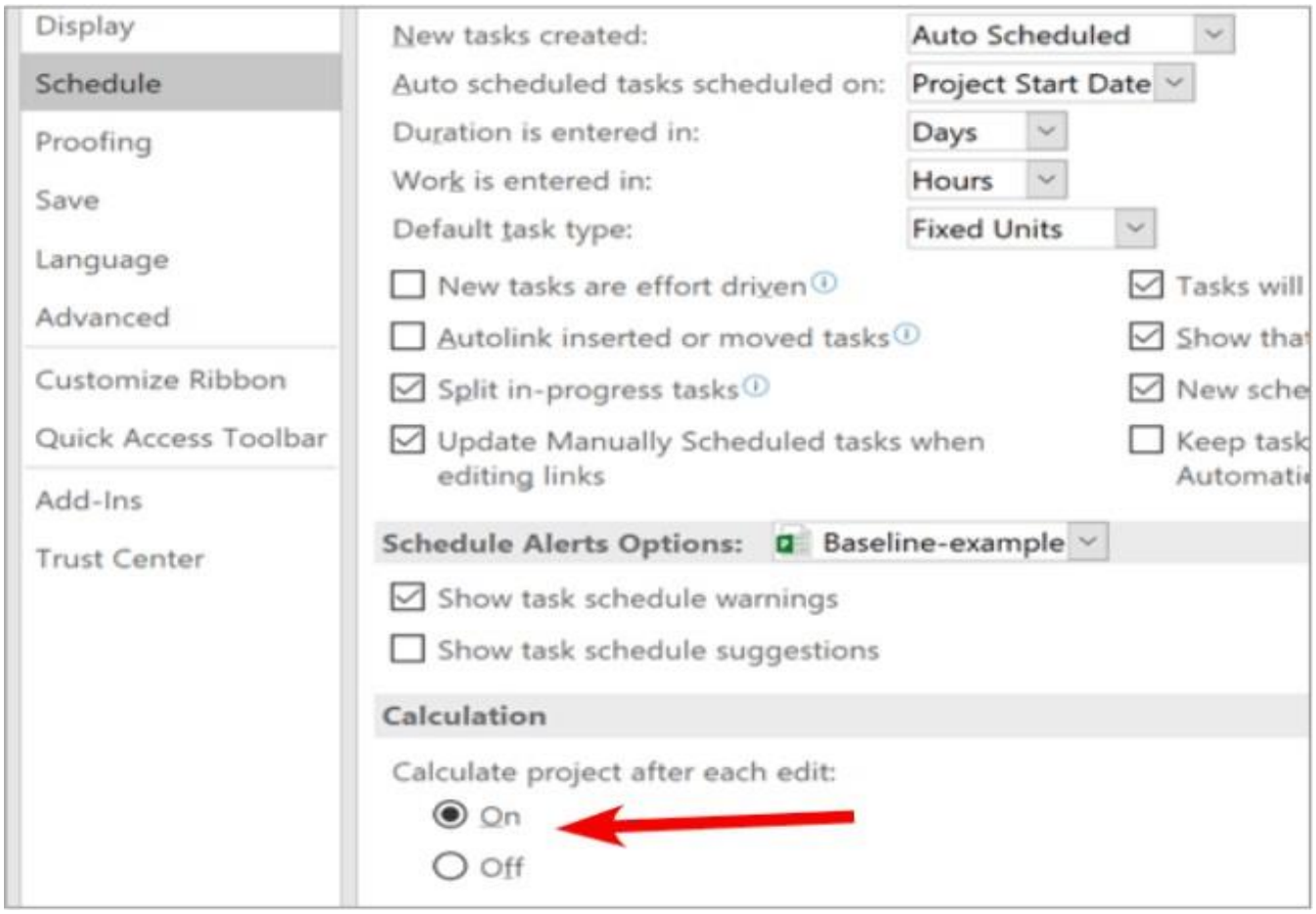
It means that new tasks will be scheduled automatically based on your project start date (or end date). More specifically, Project will determine the optimal start and end date for each task automatically, which is what we want (why would we use a computer-based scheduling tool if we would not want to automate the scheduling? Read more about manual vs. automatic scheduling in Project.)

Close the window. At the bottom left corner of the screen it should look like this:



Setting change 2: Enable immediate calculation

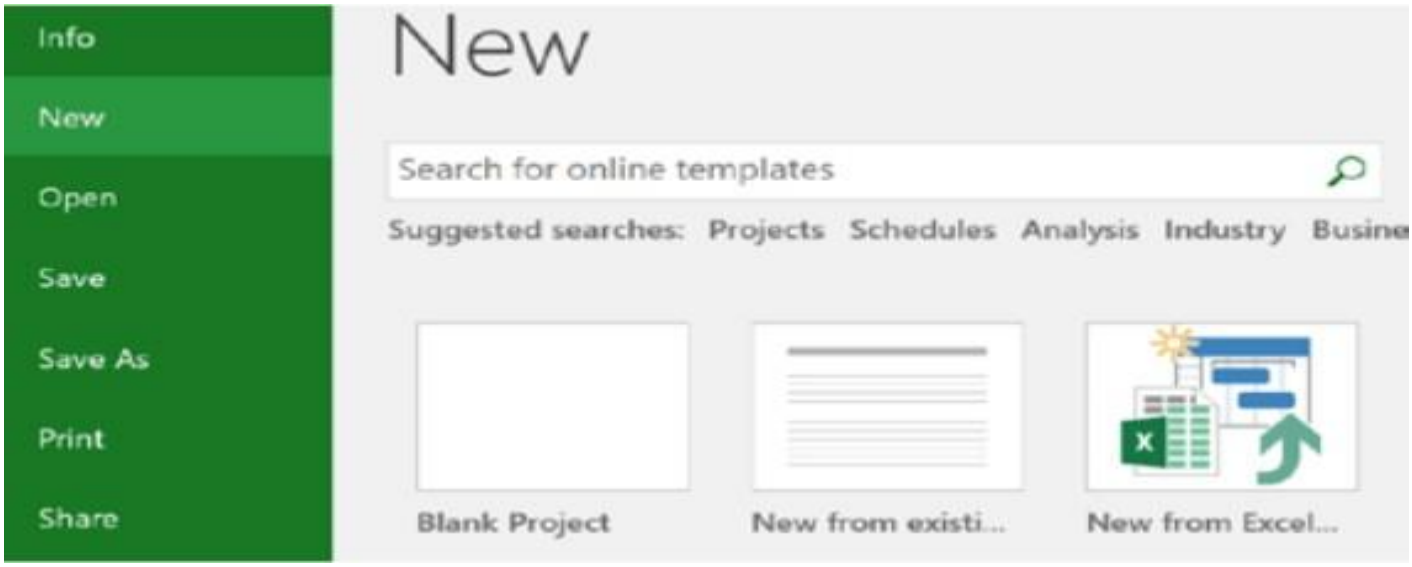
We want Microsoft Project to re-calculate the project schedule immediately after we make a change. This ensures the data you see is always up to date. Unless you run a mega-project, leave the setting enabled.



Now, let's schedule a simple project!

Our sample project: We are setting up our own business. We have picked a business idea and now we need to go from writing a business plan to a fully established business. For these steps, we are going to create a project plan.

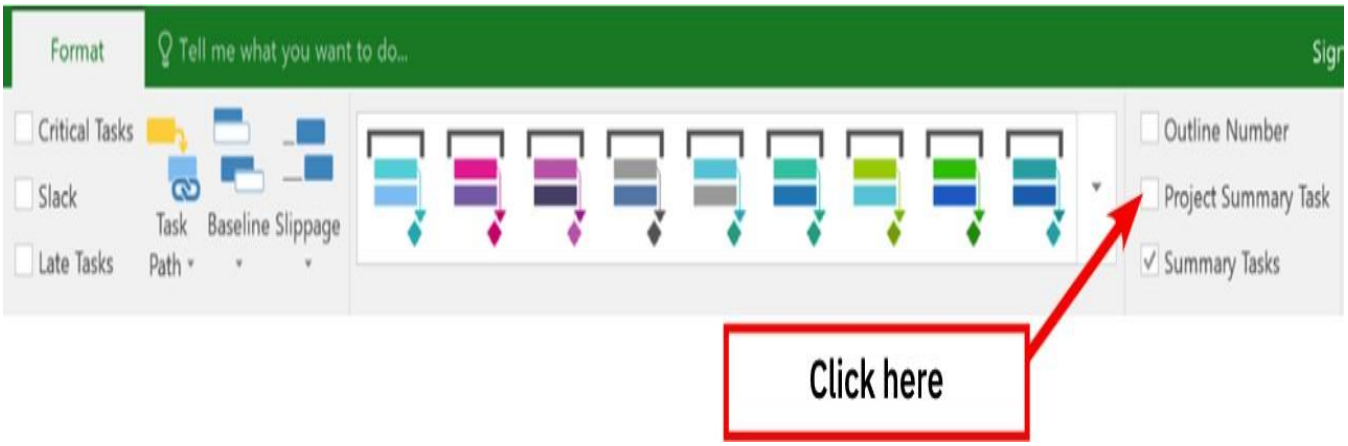
Step 1: Create a new project



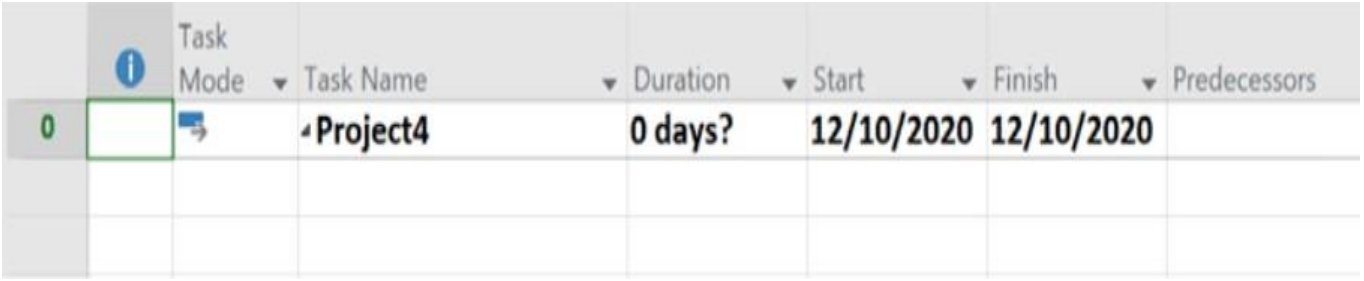
Choose *Blank Project.*

You will see a blank window.

First, let's create a **Project Summary Task**. This is like an overall "wrapper" task that contains our entire project.



Here's what you should see:



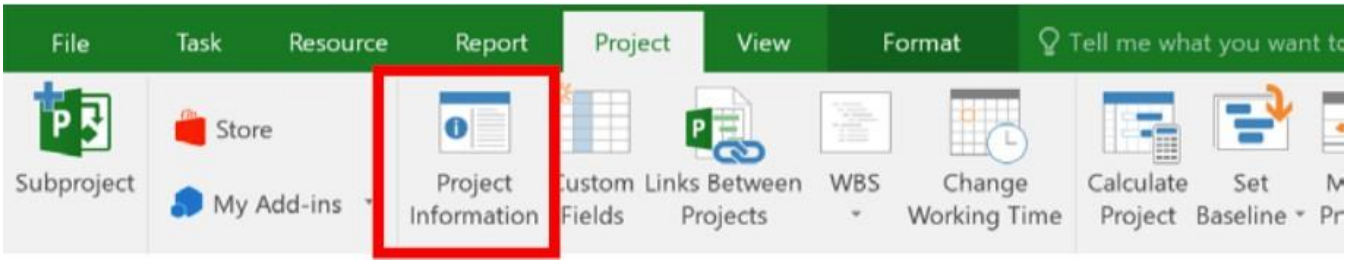
Give your project a suitable name:

Don't worry about the duration and the dates – we'll take care of this later.

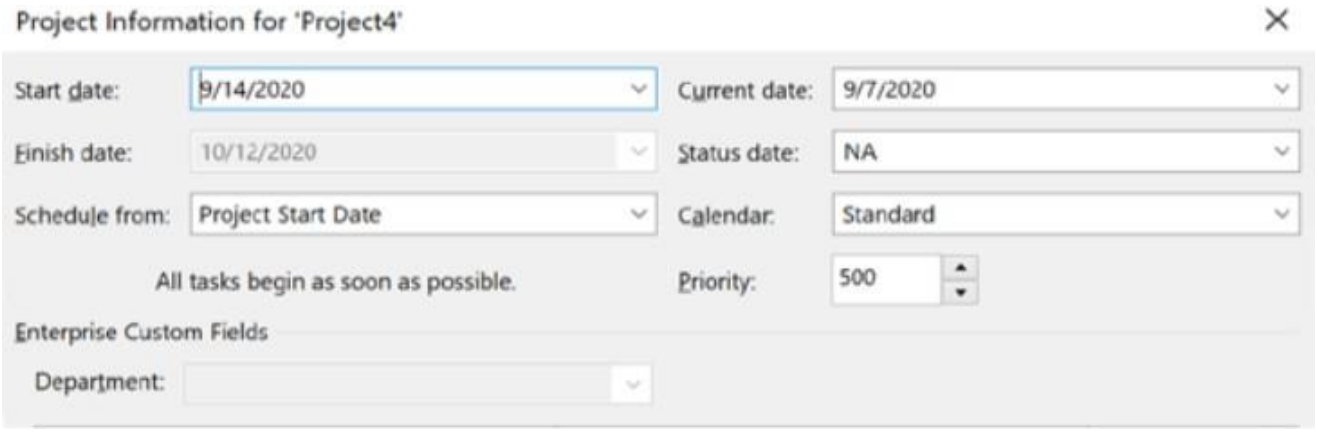
Step 2: Enter a project start date

You need to tell Project the date at which the project officially starts.

To set the project start date, open the Project tab and click *Project Information*:



Enter the project start date (in our example: 14th September 2020):



Note: You can decide whether you want Microsoft Project to schedule your project *forward* from a specific start date or *backward* from a desired end date. If you already have committed to a go live date and want to know by when you need to begin work, then chose Schedule from Project End Date to trigger the backward planning. For this example, we want to base our schedule on a start date of 14th September 2020.

Press OK after you've entered the project start date. You can see now the start date for our tasks is 14th September 2020.

tep 3: Enter the list of tasks

In this project, we need to accomplish the following tasks:

- **Create business plan**
- **Get business license**
- **Set up bank account**
- **Get funding**
- **Pick a business location**
- **Set up office equipment and furniture**
- **Hire team**
- **Run promotion**

If you look at the list, you notice that the tasks must be performed in a specific order. There are also some dependencies between the tasks.

For example, we can't get a bank account without having a business license. We also can't get funding (i.e. a bank loan) without having a business license. And of course we need the money to hire people for our store. So, everything is connected with each other.

Now let's enter those tasks into Microsoft Project.

Enter the tasks into the table next to the *Gantt view*:

	Task Mode	Task Name	Duration	Start	Finish	Prede
1						
2						
3		Create business plan	1 day?	Mon 9/7/20	Mon 9/7/20	
4		Get business license	1 day?	Mon 9/7/20	Mon 9/7/20	
5		Set up bank account	1 day?	Mon 9/7/20	Mon 9/7/20	
6		Get funding	1 day?	Mon 9/7/20	Mon 9/7/20	
7		Pick a business location	1 day?	Mon 9/7/20	Mon 9/7/20	
8		Set up office equipment a	1 day?	Mon 9/7/20	Mon 9/7/20	
9		Hire team	1 day?	Mon 9/7/20	Mon 9/7/20	
10		Run promotion	1 day?	Mon 9/7/20	Mon 9/7/20	

At this stage, Project doesn't have all the information it needs to create a schedule. It doesn't know how long each task is going to take. Therefore the *Duration* column has a question mark and the start and finish dates aren't correct yet.

Let's continue. You now need to "link" all tasks in the right order and **enter the estimated durations**.

Step 4: Enter task durations

Now, tell Project how long each task is going to take. What you enter here is the duration of the task, which is not the same as the effort. Duration is the total timespan until a task is finished. Effort (in Project, effort is called *Work*) is the amount of actual working time.

Enter the estimated duration in the duration column. **Tip:** you can either use the up/down ar- rows to change the values or enter for example "3d" to specify 3 days or "2w for 2 weeks of duration.

		Task Mode	Task Name	Duration	Start	Finish	Predeces
0			**Chocolate Store Project**	**21 days**	**9/14/2020**	**10/12/2020**	
1			Create business plan	5 days	9/14/2020	9/18/2020	
2			Get business license	1 day	9/14/2020	9/14/2020	
3			Set up bank account	1 day	9/14/2020	9/14/2020	
4			Get funding	21 days	9/14/2020	10/12/2020	
5			Pick a business location	5 days	9/14/2020	9/18/2020	
6			Set up office equipment and furniture	2 days	9/14/2020	9/15/2020	
7			Hire team	10 days	9/14/2020	9/25/2020	
8			Run promotion	4 days	9/14/2020	9/17/2020	

CHART

Experiment-8

Problem Statement: Define the Features, Vision, Business objectives, Business rules and stakeholders in the vision document

Procedure:

The Vision Document:

The Vision document is the Rational Unified Process artifact that captures all of the requirements information that we have been discussing in this chapter. As with all requirements documentation, its primary purpose is communication.

You write a Vision document to give the reader an overall understanding of the system to be developed by providing a self-contained overview of the system to be built and the motivations behind building it. To this end, it often contains extracts and summaries of other related artifacts, such as the business case and associated business models. It may also contain extracts from the system use-case model where this helps to provide a succinct and accessible overview of the system to be built.

The purpose of the Vision document is to capture the focus, stakeholder needs, goals and objectives, target markets, user environments, target platforms, and features of the product to be built. It communicates the fundamental "whys and what's" related to the project, and it is a gauge against which all future decisions should be validated.

The Vision document is the primary means of communication between the management, marketing, and project teams. It is read by all of the project stakeholders, including general managers, funding authorities, use-case modelers, and developers. It provides

- A high-level (sometimes contractual) basis for the more detailed technical requirements
- Input to the project-approval process (and therefore it is intimately related to the business case)
- A vehicle for eliciting initial customer feedback
- A means to establish the scope and priority of the product features

It is a document that gets "all parties working from the same book."

Because the Vision document is used and reviewed by a wide variety of involved personnel, the level of detail must be general enough for everyone to understand. However, enough detail must be available to provide the team with the information it needs to create a use-case model and supplementary specification.

The document contains the following sections:

- **Positioning:** This section summarizes the business case for the product and the problem or opportunity that the product is intended to address. Typically, the following areas should be addressed:

    - **The Business Opportunity:** A summary of business opportunity being met by the product
    - **The Problem Statement:** A solution-neutral summary of the problem being solved focusing on the impact of the problem and the benefits required of any successful solution
    - **Market Demographics:** A summary of the market forces that drive the product decisions.
    - **User Environment:** The user environment where the product could be applied.
- **Stakeholders and Users:** This section describes the stakeholders in, and users of, the product. The stakeholder roles and stakeholder types are defined in the project's Vision document—the actual stakeholder representatives are identified as part of the project plan just like any other resources involved in the project.
- **Key Stakeholder and User Needs:** This section describes the key needs that the stakeholders and users perceive the product should address. It does not describe their specific requests or their specific requirements, because these are captured in a separate stakeholder requests artifact. Instead, it provides the background and justification for why the requirements are needed.
- **Product Overview:** This section provides a high-level view of the capabilities, assumptions, dependencies (including interfaces to other applications and system configurations), and alternatives to the development of the product.
- **Features:** This section lists the features of the product. Features are the high-level capabilities (services or qualities) of the system that are necessary to deliver benefits to the users and satisfy the stakeholder and user needs. This is the most important, and consequently usually the longest, section of the Vision document.
- **Other Product Requirements:** This section lists any other high-level requirements that cannot readily be captured as product features. These include any constraints placed on the development of the product and any requirements the planned product places on its operating environment.

In many cases, a lot more work is put into uncovering the business opportunity and understanding the market demographics related to the proposed product than is reflected in the Vision document. This work is usually captured in-depth in business cases, business models, and market research documents. These documents are then summarized in the Vision document to ensure that they are reflected in the ongoing

evolution of the products specification.
We recommend that the Vision document be treated primarily as a report and that the stakeholder types, user types, stakeholder roles, needs, features, and other product requirements be managed using a requirements management tool. If the list of features is to be generated, it is recommended that they be presented in two sections:

- In-Scope features
- Deferred features

Do you really need to do all of this?
You are probably thinking that this all sounds like an awful lot of work, and you probably want to get started on the actual use-case modeling without producing reams and reams of additional documentation.
Well, projects are typically in one of four states when the use-case modeling activities are scheduled to commence:

- A formal Vision document has been produced.
- The information has already been captured but not consolidated into a single Vision document.
- There is a shared vision, but it has not been documented.
- There is no vision.

If your project is in one of the first two states, and the information is available to all the stakeholder representatives, then you are in a position to proceed at full speed with the construction and completion of the use-case model. If your project is in one of the last two states, then you should be very careful not to spend too much effort on the detailed use-case modeling activities. This does not mean that use-case modeling cannot be started—it simply means that any modeling you do must be undertaken in conjunction with other activities aimed at establishing a documented vision for the product. In fact, in many cases, undertaking some initial use-case modeling can act as a driver and facilitation device for the construction of the vision itself.
Our recommendation would be to always produce a Vision document for every project and to relate the information it contains to the use-case model to provide focus, context, and direction to the use-case modeling activities. For-mally relating the two sets of information also provides excellent validation of their contents and quality. If there is sufficient domain knowledge and agreement between the stakeholder representatives, then producing and reviewing the Vision document can be done very quickly. If there isn't, then there is no point in undertaking detailed use-case modeling; the resulting specifications would be ultimately worthless as they would not be a reflection of the product's true requirements.

Summary

Before embarking on any use-case modeling activities it is essential to establish a firm foundation upon which to build. The foundation has two dimen-sions, which must be

evolved in parallel with one another:

1. An understanding of the stakeholder and user community
2. The establishment of a shared vision for the product

Understanding the stakeholder community is essential as the stakeholders are the primary source of requirements. The following are the key to understanding the stakeholder community:

- **Stakeholder Types:** Definitions of all of the different types of stakeholder affected by the project and the product it produces.
- **User Types:** Definitions of characteristics and capabilities of the users of the system. The users are the people and things that will take on the roles defined by the actors in the use-case model.

For the use-case modeling activities to be successful, the stakeholders and users will need to be actively involved in them. The stakeholders and users directly involved in the project are known as stakeholder representatives. To ensure that the stakeholder representatives understand their commitment to the project, it is worthwhile to clearly define the "stakeholder roles" that they will be adopting. The stakeholder roles serve as a contract between the stakeholder representatives and the project, reflecting the responsibilities and expectations of both sides.

To establish a shared vision for the project, the following are essential:

- **The Problem Statement:** A solution-neutral summary of the problem being solved, focusing on the impact of the problem and the benefits required of any successful solution.
- **Stakeholder Needs:** The true "business requirements" of the stakeholders presented in a solution-neutral manner. These are the aspects of the problem that affect the individual stakeholders.
- **Features, Constraints, and Other High-Level Product Requirements:** A high-level definition of the system to be developed. These complement and provide a context for the use-case model and enable effective scope management.
- **Product Overview:** A summary of the other aspects of the product not directly captured by the high-level requirements.

The Vision document can be used to capture all of this information in a form that is accessible to all the stakeholders of the project.

The vision does not have to be complete before use-case modeling activities start; in fact, undertaking some initial use-case modeling can act as a driver and facilitation device for the construction of the vision itself, but if the vision is not established alongside the use-case model, then there is a strong possibility that it will not be a true reflection of the real requirements.

Experiment-9

Problem Statement:Define the functional and non-functional requirements of the system to be automated by using Use cases and document in SRS document

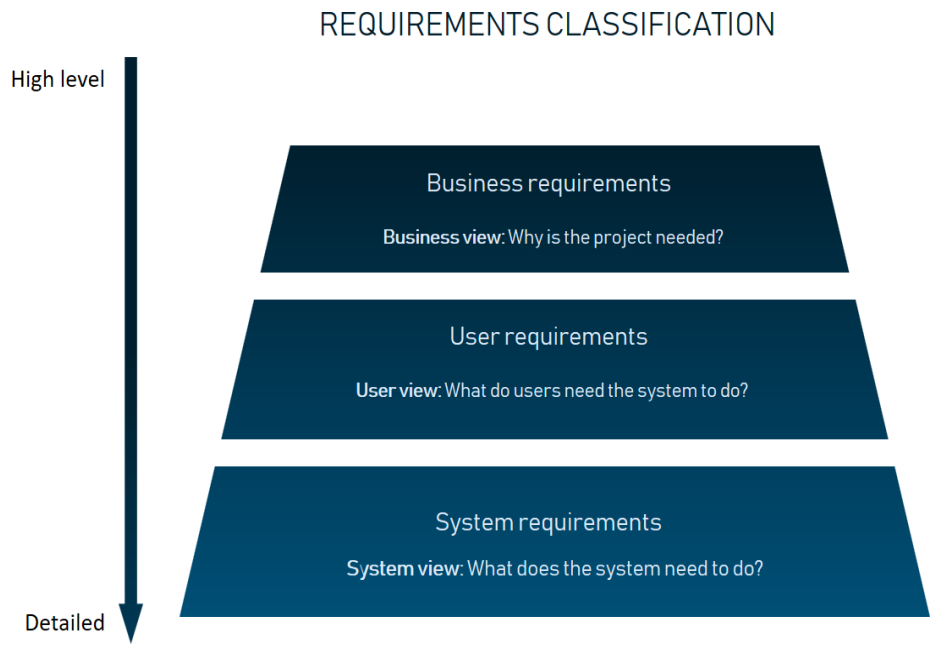
Procedure:

Clearly defined requirements are essential signs on the road that leads to a successful project. They establish a formal agreement between a client and a provider that they are both working to reach the same goal. High-quality, detailed requirements also help mitigate financial risks and keep the project on a schedule. According to the Business Analysis Body of Knowledge definition, requirements are a usable representation of a need.

Creating requirements is a complex task as it includes a set of processes such as elicitation, analysis, specification, validation, and management. In this article, we'll discuss the main types of requirements for software products and provide a number of recommendations for their use.

Classification of requirements

Prior to discussing how requirements are created, let's differentiate their types.



Business requirements. These include high-level statements of goals, objectives, and needs.

Stakeholder requirements. The needs of discrete stakeholder groups are also specified to define what they expect from a particular solution.

Solution requirements. Solution requirements describe the characteristics that a product must have to meet the needs of the stakeholders and the business itself.

- **Nonfunctional** requirements describe the general characteristics of a system. They are also known as *quality attributes*.
- **Functional** requirements describe how a product must behave, what its features and functions.

Transition requirements. An additional group of requirements defines what is needed from an organization to successfully move from its current state to its desired state with the new product.

Let's explore functional and nonfunctional requirements in greater detail.

Functional requirements and their specifications

Functional requirements are product features or functions that developers must implement to enable users to accomplish their tasks. So, it's important to make them clear both for the development team and the stakeholders. Generally, functional requirements describe system behavior under specific conditions. For instance:

A search feature allows a user to hunt among various invoices if they want to credit an issued invoice.

Here's another simple example: *As a guest, I want a sofa that I can sleep on overnight*.

Requirements are usually written in text, especially for Agile-driven projects. However, they may also be visuals. Here are the most common formats and documents:

- Software requirements specification document
- Use cases
- User stories
- Work Breakdown Structure (WBS) (functional decomposition)
- Prototypes
- Models and diagrams

Software requirements specification document

Functional and nonfunctional requirements can be formalized in the requirements specification (SRS) document. (To learn more about software documentation, read our article on that topic.) The SRS contains descriptions of functions and capabilities that the product must provide. The document also defines constraints and assumptions. The SRS can be a single document communicating functional requirements or it may accompany other software documentation like user stories and use cases.

We don't recommend composing SRS for the entire solution before the development kick-off, but you should document the requirements for every single feature before actually building it. Once you receive the initial user feedback, you can update the document.

SRS must include the following sections:

Purpose.Definitions, system overview, and background.

Overall description. Assumptions, constraints, business rules, and product vision.
Specific requirements. System attributes, functional requirements, database requirements.
It's essential to make the SRS readable for all stakeholders. You also should use templates with visual emphasis to structure the information and aid in understanding it. If you have requirements stored in some other document formats, link to them to allow readers to find the needed information.
Example: If you'd like to see an actual document, download this SRS example created at Michigan State University, which includes all points mentioned above in addition to presenting use cases to illustrate parts of the product.
Use cases
Use cases describe the interaction between the system and external users that leads to achieving particular goals.
Each use case includes three main elements:
Actors. These are the users outside the system that interact with the system.
System. The system is described by functional requirements that define an intended behavior of the product.
Goals. The purposes of the interaction between the users and the system are outlined as goals.
There are two formats to represent use cases:

- Use case specification structured in textual format
- Use case diagram

A **use case specification** represents the sequence of events along with other information that relates to this use case. A typical use case specification template includes the following information:

- Description
- Pre- and Post- interaction condition
- Basic interaction path
- Alternative path
- Exception path

Example:

Overview	
Title	[Title of the basic flow use case]
Description	[Short description of the basic flow]
Actors and Interfaces	[Identifies the Actors and Interfaces to components and services that participate in the use case]
Initial Status and Preconditions	[A pre-condition (of a use case) is the state of the system that must be present prior to a use case being performed]
Basic Flow	
STEP 1: … STEP 2: …	
Post Condition	
[A post-condition (of a use case) is a list of possible states the system can be in immediately after a use case has finished]	
Alternative Flow(s)	
[Alternative flows are described here if needed]	

Use case specification template

A **use case diagram** doesn't contain a lot of details. It shows a high-level overview of the relationships between actors, different use cases, and the system.

The use case diagram includes the following main elements:

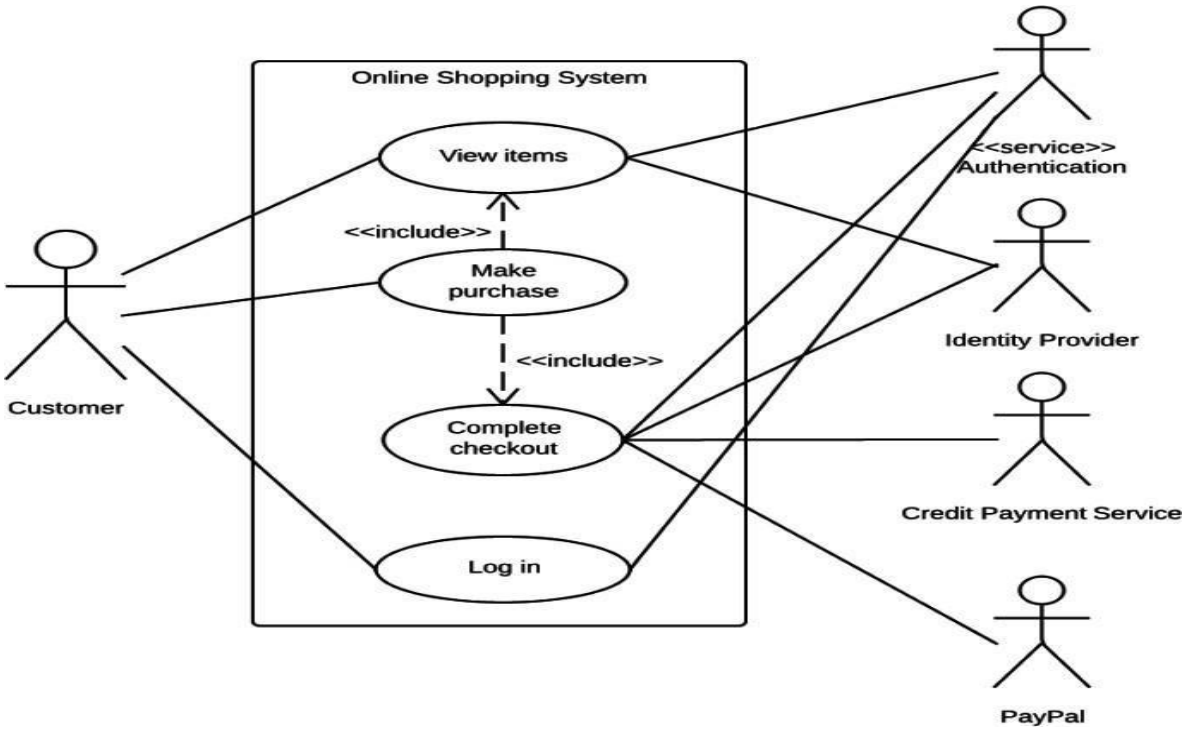
Use cases. Usually drawn with ovals, use cases represent different use scenarios that actors might have with the system (*log in, make a purchase, view items,* etc.)

System boundaries. Boundaries are outlined by the box that groups various use cases in a system.

Actors. These are the figures that depict external users (people or systems) that interact with the system.

Associations. Associations are drawn with lines showing different types of relationships between actors and use cases.

Example:



Use case diagram example

User stories

A user story is a documented description of a software feature seen from the end-user perspective. The user story describes what exactly the user wants the system to do. In Agile projects, user stories are organized in a *backlog*, which is an ordered list of product functions. Currently, user stories are considered to be the best format for backlog items.

A typical user story is written like this:

As a <type of user>, I want <some goal> so that <some reason>.

***Example**:*

As an admin, I want to add descriptions to products so that users can later view these descriptions and compare the products.

User stories must be accompanied by **acceptance criteria**. These are the conditions that the product must satisfy to be accepted by a user, stakeholders, or a product owner. Each user story must have at least one acceptance criterion. Effective acceptance criteria must be testable, concise, and completely understood by all team members and stakeholders. They can be written as checklists, plain text, or by using Given/When/Then format.

Example:

Here's an example of the acceptance criteria checklist for a user story describing a search feature:

- A search field is available on the top-bar.
- A search is started when the user clicks *Submit*.
- The default placeholder is a grey text *Type the name*.

- The placeholder disappears when the user starts typing.
- The search language is English.
- The user can type no more than 200 symbols.
- It doesn't support special symbols. If the user has typed a special symbol in the search input, it displays the warning massage: *Search input cannot contain special symbols*.

Finally, all user stories must fit the **INVEST quality model**:

- **I** – Independent
- **N** – Negotiable
- **V** – Valuable
- **E** – Estimable
- **S** – Small
- **T** – Testable

Independent. This means that you can schedule and implement each user story separately. This is very helpful if you implement continuous integration processes.
Negotiable. This means that all parties agree to prioritize negotiations over specification. This also means that details will be created constantly during development.
Valuable. A story must be valuable to the customer. You should ask yourself from the customer's perspective "why" you need to implement a given feature.
Estimable. A quality user story can be estimated. This will help a team schedule and prioritize the implementation. The bigger the story is, the harder it is to estimate it.
Small. Good user stories tend to be small enough to plan for short production releases. Small stories allow for more specific estimates.
Testable. If a story can be tested, it's clear enough and good enough. Tested stories mean that requirements are done and ready for use.

-

Experiment-10

Problem Statement:Define the following traceability matrices:

1. Use case Vs. Features

2. Functional requirements Vs.Usecases

Procedure:

The concept of Traceability Matrix is to be able to trace from top level requirements to implementation,andfrom top level requirements to test.

A traceability matrix is a table that traces a requirement to the tests that are needed to verify that the requirement is fulfilled. A good traceability matrix will provide backward and forward traceability, i.e. a requirement can be traced to a test and a test to a requirements. The matrix links higher level requirements, design specifications, test requirements, and code files. It acts as a map, providing the links necessary for determining where information is located. This is also known as Requirements Traceability Matrix or RTM.

This is mostly used for QA so that they can ensure that the customer gets what they requested. The Traceability matrix also helps the developer find out why some code was implemented the way it was, by being able to go from code to Requirements. If a test fails, it is possible to use the traceability matrix to see what requirements and code the test case relates to.

The goal of a matrix of this type is -

1. To make sure that the approved requirements are addressed/covered in all phases of development: From SRS to Development to Testing to Delivery.
2. That each document should be traceable: Written test cases should be traceable to its requirement specification. If there is new version then updated test cases should be traceable with that.

Traceability Matrix is used in entire software development life cycle phases:

1. Risk Analysis phase
2. Requirements Analysis and Specification phase
3. Design Analysis and Specification phase
4. Source Code Analysis, Unit Testing & Integration Testing phase
5. Validation – System Testing, Functional Testing phase

In this topic we will discuss:

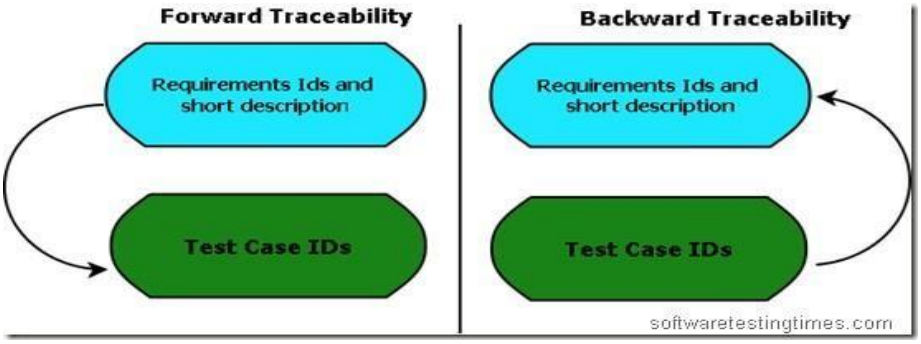
- What is Traceability Matrix from Software Testing perspective? (Point 5)
- Types of Traceability Matrix
- Disadvantages of not using Traceability Matrix
- Benefits of using Traceability Matrix in testing

- Step by step process of creating an effective Traceability Matrix from requirements. Sample formats of Traceability Matrix basic version to advanced version.

In Simple words - A requirements traceability matrix is a document that traces and maps user requirements [requirement Ids from requirement specification document] with the test case ids. Purpose is to make sure that all the requirements are covered in test cases so that while testing no functionality can be missed.
This document is prepared to make the clients satisfy that the coverage done is complete as end to end, this document consists of Requirement/Base line doc Ref No., Test case/Condition, and Defects/Bug id. Using this document the person can track the Requirement based on the Defect id.

Types of Traceability Matrix:

- Forward Traceability – Mapping of Requirements to Test cases
- Backward Traceability – Mapping of Test Cases to Requirements
- Bi-Directional Traceability - A Good Traceability matrix is the References from test cases to basis documentation and vice versa.



Why Bi-Directional Traceability is **required?**
Bi-Directional Traceability contains both Forward & Backward Traceability. Through Backward Traceability Matrix, we can see that test cases are mapped with which requirements.
This will help us in identifying if there are test cases that do not trace to any coverage item— in which case the test case is not required and should be removed (or maybe a specification like a requirement or two should be added!). This “backward” Traceability is also very helpful if you want to identify that a particular test case is covering how many requirements?
Through Forward Traceability – we can check that requirements are covered in which test cases? Whether is the requirements are coved in the test cases or not?
Forward Traceability Matrix ensures – We are building the Right Product.
Backward Traceability Matrix ensures – We the Building the Product Right.

Traceability matrix is the answer of the following questions of any Software Project:

- How is it feasible to ensure, for each phase of the SDLC, that I have correctly accounted for all the customer's needs?

- How can I certify that the final software product meets the customer's needs? Now we can only make sure requirements are captured in the test cases by traceability matrix.

Disadvantages of not using Traceability Matrix [some possible (seen) impact]:

No traceability or Incomplete Traceability Results into:

1. Poor or unknown test coverage, more defects found in production
2. It will lead to miss some bugs in earlier test cycles which may arise in later test cycles. Then a lot of discussions arguments with other teams and managers before release.
3. Difficult project planning and tracking, misunderstandings between different teams over project dependencies, delays, etc

Benefits of using Traceability Matrix

- Make obvious to the client that the software is being developed as per the requirements.
- To make sure that all requirements included in the test cases
- To make sure that developers are not creating features that no one has requested
- Easy to identify the missing functionalities.
- If there is a change request for a requirement, then we can easily find out which test cases need to update.
- The completed system may have "Extra" functionality that may have not been specified in the design specification, resulting in wastage of manpower, time and effort.

Steps to create Traceability Matrix:

1. Make use of excel to create Traceability Matrix:
2. Define following columns:

Base Specification/Requirement ID (If any)
Requirement ID
Requirement description
TC 001
TC 002
TC 003.. So on.

3. Identify all the testable requirements in granular level from requirement document.

Typical requirements you need to capture are as follows:
Used cases (all the flows are captured)
Error Messages

Business rules
Functional rules
SRS
FRS
So on…
4. Identity all the test scenarios and test flows.
5. Map Requirement IDs to the test cases. Assume (as per below table), Test case “TC 001” is your one flow/scenario. Now in this scenario, Requirements SR-1.1 and SR-1.2 are covered. So, mark “x” for these requirements.
Now from below table you can conclude –
Requirement SR-1.1 is covered in TC 001
Requirement SR-1.2 is covered in TC 001
Requirement SR-1.5 is covered in TC 001, TC 003 [Now it is easy to identify, which test cases need to be updated if there is any change request].
TC 001 Covers SR-1.1, SR, 1.2 [we can easily identify that test cases covers which requirements].
TC 002 covers SR-1.3... So on...

Requirement ID	**Requirement description**	**TC 001**	**TC 002**	**TC 003**
SR-1.1	User should be able to do this	x		
SR-1.2	User should be able to do that	x		
SR-1.3	On clicking this, following message should appear		x	
SR-1.4			x	
SR-1.5		x		x
SR-1.6				x
SR-1.7			x	

Use Case
A use case describes the interactions between one of more Actors and the system in

order to provide an observable result of value for the initiating actor.
The functionality of a system is defined by different use cases, each of which represents a specific goal (to obtain the observable result of value) for a particular actor.

In an automated teller machine shown in Figure 1, the Bank Customer can withdraw cash from an account, transfer funds between accounts, or deposit funds to an account. These correspond to specific goals that the actor has in using the system.

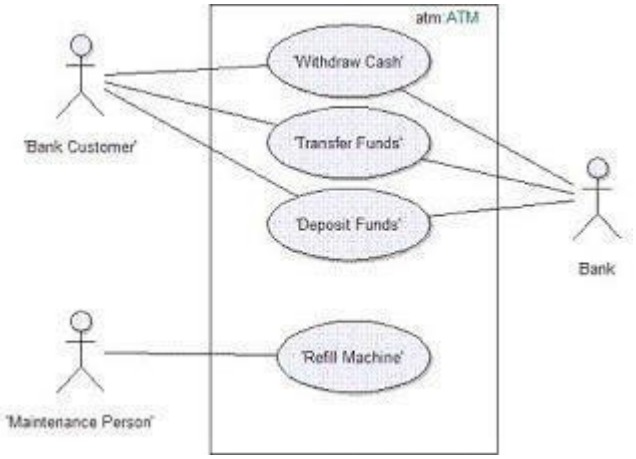


Figure 1: ATM Use-Case Example

Each use case is associated with a goal of one of the actors. The collection of use cases constitutes all the possible ways of using the system. You should be able to determine the goal of a use case simply by observing its name.
A use case describes the interactions between the actor(s) and the system in the form of a dialog between the actor(s) and the system, structured as follows:

1. The actor <<does something>>
2. The system <<does something in response>>
3. The actor <<does something else>>
4. The system ...

Each dialog of this form is called a "Flow of Events".
Because there are many flows of events possible for achieving the goal (for example, the flow may differ depending upon specific input from the actor), and there are situations in which the goal cannot be achieved (for example, a required network connection is currently unavailable), each use case will contain several flows, including one "Basic Flow of Events" and several "Alternative Flows".
The Basic Flow of Events specifies the interactions between the actor(s) and the system for the ideal case, where everything goes as planned, and the actor's goal (the observable result of value) is met. The basic flow represents the main capability provided by the system for this use case.
As the name implies, Alternative Flows specify alternative interactions associated with the same goal.
Closely related to use cases is the concept of a scenario. A scenario is a specific flow of events, for a specific set of inputs to the system, states of the system, and states of the system's environment. Scenarios are closely related to test cases.

Properties of Use Cases

Name

Each use case should have a name that indicates what is achieved by its interaction with the actors. The name may have to be several words to be understood. Note: No two use cases can have the same name.

Brief Description

The brief description of the use case should reflect its purpose.

Flow of Events

Flow of Events - Contents

The flow of events should describe the use case's flow of events clearly enough for an outsider to easily understand. Remember, the flow of events should represent *what* the system does, not *how* the system is design to perform the required behavior.

Follow these guidelines for the contents of the flow of events:

· Describe how the use case starts and ends.

· Describe what data is exchanged between the actor and the use case.

· Do not describe the details of the user interface, unless it is necessary to understand the behavior of the system. Specifying user interface details too early will limit design options.

· Describe the flow of events, not only the functionality. To enforce this, start every action with "When the actor ... ".

· Describe only the events that belong to the use case, and not what happens in other use cases or outside of the system.

· Avoid vague terminology.

· Detail the flow of events. Specify what happens when, for each action.

Remember this text will be used to identify test cases.

If you have used certain terms in other use cases, be sure to use the exact same terms in this use case, and that the meaning of the terms is consistent. To manage common terms, put them in a glossary.

Flow of Events - Structure

The two main parts of the flow of events are **basic flow of events** and **alternative flows of events**. The basic flow of events should cover what "normally" happens when the use case is performed. The alternative flows of events cover behavior of optional or exceptional character in relation to the normal behavior, and also variations of the normal behavior. You can think of the alternative flows of events as detours from the basic flow of events, some of which will return to the basic flow of events and some of which will end the execution of the use case.

The straight arrow in Figure 2 represents the basic flow of events, and the curves represent alternative paths in relation to the normal. Some alternative paths return to the basic flow of events, whereas others end the use case.

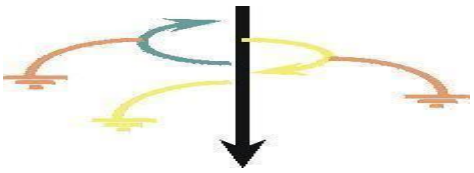


Figure 2: Typical structure of a use case flow of events

Both the basic and alternative flows should be further structured into steps or sub-flows. In doing this, your main goal should be readability of the text. A guideline is that a sub-flow should be a segment of behavior within the use case that has a clear purpose, and is "atomic" in the sense that you do either all or none of the actions described.

Special Requirements

In the Special Requirements of a use case, you describe all the requirements associated with the use case that are not covered by the flow of events. These are non-functional requirements that will influence the design. See also the discussion on non-functional requirements in Concept: Requirements.

Preconditions and Post-conditions

A **precondition** is the state of the system and its environment that is required before the use case can be started. Post-Conditions are the states the system can be in after the use case has ended. It can be helpful to use the concepts of **precondition** and **post-condition** to clarify how the flow of events starts and ends. However, only use them only if the audience for the description of the use case agrees that it is helpful. Figure 3 shows an example.

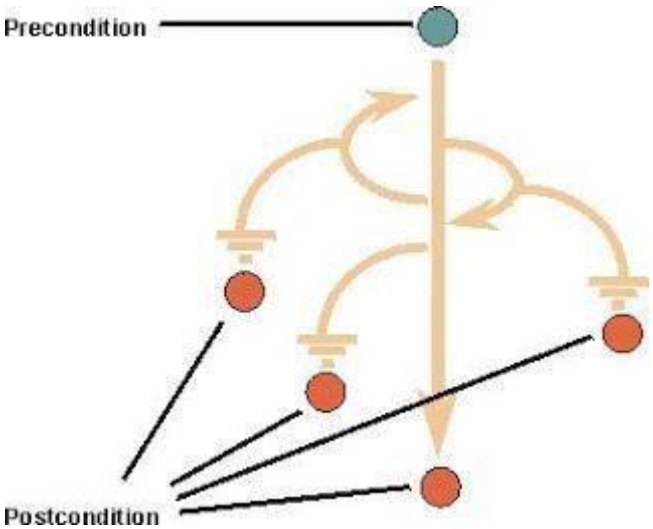


Figure 3: Illustration of preconditions and resulting post-conditions

Examples

A precondition for the use case Cash Withdrawal in the ATM machine: The customer has a personally issued card that fits in the card reader, has been issued a PIN number, and is registered with the banking system.

A post-condition for the use case Cash Withdrawal in the ATM machine: At the end of the use case, all account and transaction logs are balanced, communication with the banking system is reinitialized and the card is returned to the customer.

Experiment-11

Problem Statement:Estimate the effort using the following methods for the system to be automated:

1. Function point metric

2. Use case point metric

Procedure:

For the success of any project test estimation and proper execution is equally important as the development cycle. Sticking to the estimation is very important to build a good reputation with the client.

Experience plays a major role in estimating "Software Testing Efforts". Working on varied projects helps to prepare an accurate estimation of the testing cycle. Obviously one cannot just blindly put some number of days for any testing task. **Test estimation should be realistic and accurate.**

Brief Description of the Test Estimation Process

"Estimation is the process of finding an estimate, or approximation, which is a value that is usable for some purpose even if input data may be incomplete, uncertain, or unstable."

We all come across different tasks and duties and deadlines throughout our lives as professionals, now there are two approaches to find a solution to a problem.

A first approach is a reactive approach whereby we try to find a solution to the problem at hand only after it arrives.

In the second approach which can be called a Proactive Approach whereby we first prepare ourselves well before the problem arrives with our past experiences and then with our past experience, we try to find a solution to the challenge when it arrives.

Estimation can thus be considered as a technique that is applied when we take a proactive approach to the problem.

Thus Estimation can be used to predict how much effort with respect to time and cost would be required to complete a defined task.

Once the testing team is able to make an estimate of the problem at hand then it is easier for them to come up with a solution that would be optimum to the problem at hand.

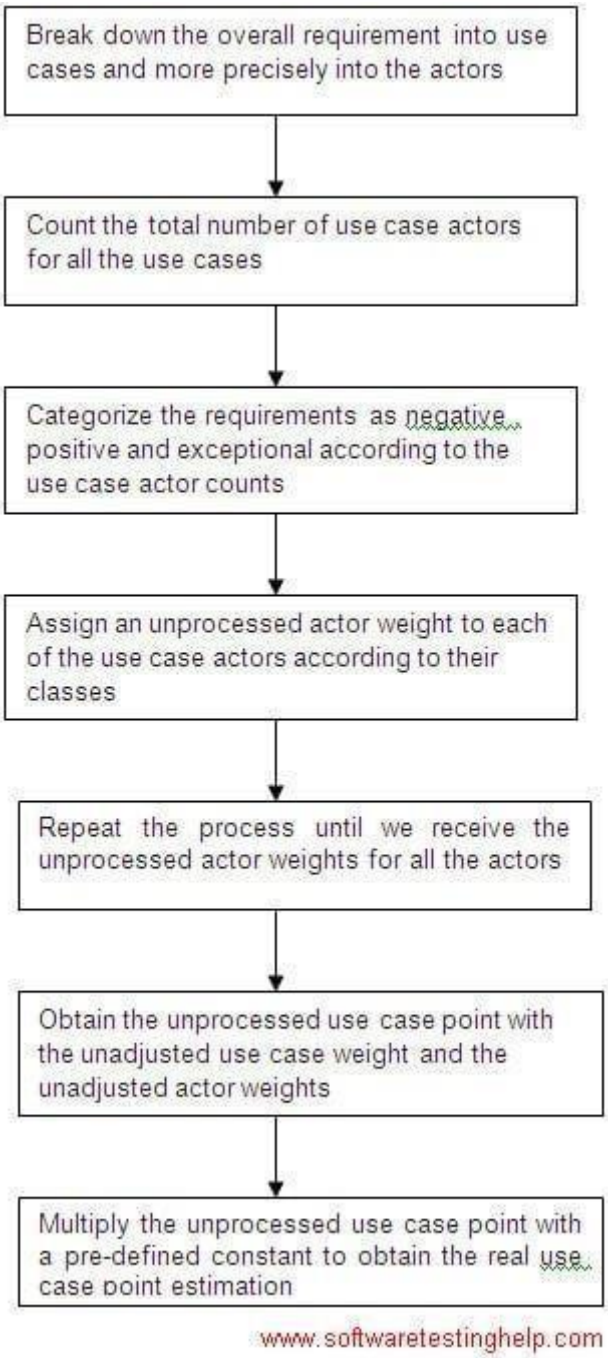
The practice of estimation can be defined then more formally as an approximate computation of the probable cost of a piece of work.

Test Estimation Examples

Use Case Point Estimation Method:

Use-Case Point Method is based on the use cases where we calculate the overall test estimation effort based on the use cases or the requirements.

Here is the detailed process of the Use case point estimation method:



An example of the same is that say in a particular requirement we have 5 use cases, use case 1, and use case 2… use case 5 respectively. Now let us consider that use case 1 consists of 6 actors, use case 2 consists of 15 actors, use cases 3, 4 and 5, 3, 4 and 5 actors respectively.

We consider any use case which involves the total number of actors as less than 5 as negative, any use case with the total number of actors is equal to or more than 5 and less than or equal to 10 as positive and any use case with more than 10 actors as exceptional.

We decide to assign 2 points to the exceptional use cases, 1 to the positive ones and -1 for the negative ones.

Thus we categorize the Use cases 1 and 5 as positive, use case 2 as exceptional and use

case 3, 4 as negative respectively based on our above-stated assumptions.
So the Unprocessed actor weights = Use case 1 = (total number of actors) 5 * 1(the assigned point) = 5. Similarly
Use case 2 = 15 * 2 = 30.
Repeating the process for rest of the use cases we receive the Unprocessed actor weights = 33
Unprocessed use case weight = total no. of use cases = 5
Unprocessed use case point = Unadjusted actor weights + Unadjusted use case weight = 33 + 5 = 38
Processed use case point = 38 * [0.65+ (0.01 * 50] = 26.7 or 28 Person Hours approximately

Function point metric:

A **Function Point** (FP) is a unit of measurement to express the amount of business functionality, an information system (as a product) provides to a user. FPs measure software size. They are widely accepted as an industry standard for functional sizing. For sizing software based on FP, several recognized standards and/or public specifications have come into existence. As of 2013, these are −
ISO Standards

- **COSMIC** − ISO/IEC 19761:2011 Software engineering. A functional size measurement method.
- **FiSMA** − ISO/IEC 29881:2008 Information technology - Software and systems engineering - FiSMA 1.1 functional size measurement method.
- **IFPUG** − ISO/IEC 20926:2009 Software and systems engineering - Software measurement - IFPUG functional size measurement method.
- **Mark-II** − ISO/IEC 20968:2002 Software engineering - Ml II Function Point Analysis - Counting Practices Manual.
- **NESMA** − ISO/IEC 24570:2005 Software engineering - NESMA function size measurement method version 2.1 - Definitions and counting guidelines for the application of Function Point Analysis.

Object Management Group Specification for Automated Function Point
Object Management Group (OMG), an open membership and not-for-profit computer industry standards consortium, has adopted the Automated Function Point (AFP) specification led by the Consortium for IT Software Quality. It provides a standard for automating FP counting according to the guidelines of the International Function Point User Group (IFPUG).
Function Point Analysis (FPA) technique quantifies the functions contained within software in terms that are meaningful to the software users. FPs consider the number of functions being developed based on the requirements specification.
Function Points (FP) Counting is governed by a standard set of rules, processes and

guidelines as defined by the International Function Point Users Group (IFPUG). These are published in Counting Practices Manual (CPM).

History of Function Point Analysis

The concept of Function Points was introduced by Alan Albrecht of IBM in 1979. In 1984, Albrecht refined the method. The first Function Point Guidelines were published in 1984. The International Function Point Users Group (IFPUG) is a US-based worldwide organization of Function Point Analysis metric software users. The **International Function Point Users Group (IFPUG)** is a non-profit, member-governed organization founded in 1986. IFPUG owns Function Point Analysis (FPA) as defined in ISO standard 20296:2009 which specifies the definitions, rules and steps for applying the IFPUG's functional size measurement (FSM) method. IFPUG maintains the Function Point Counting Practices Manual (CPM). CPM 2.0 was released in 1987, and since then there have been several iterations. CPM Release 4.3 was in 2010.

The CPM Release 4.3.1 with incorporated ISO editorial revisions was in 2010. The ISO Standard (IFPUG FSM) - Functional Size Measurement that is a part of CPM 4.3.1 is a technique for measuring software in terms of the functionality it delivers. The CPM is an internationally approved standard under ISO/IEC 14143-1 Information Technology – Software Measurement.

Elementary Process (EP)

Elementary Process is the smallest unit of functional user requirement that −

- Is meaningful to the user.
- Constitutes a complete transaction.
- Is self-contained and leaves the business of the application being counted in a consistent state.

Functions

There are two types of functions −

- Data Functions
- Transaction Functions

Data Functions

There are two types of data functions −

- Internal Logical Files
- External Interface Files

Data Functions are made up of internal and external resources that affect the system.

Internal Logical Files

Internal Logical File (ILF) is a user identifiable group of logically related data or control information that resides entirely within the application boundary. The primary intent of an ILF is to hold data maintained through one or more elementary processes of the application being counted. An ILF has the inherent meaning that it is internally

maintained, it has some logical structure and it is stored in a file. (Refer Figure 1)

External Interface Files

External Interface File (EIF) is a user identifiable group of logically related data or control information that is used by the application for reference purposes only. The data resides entirely outside the application boundary and is maintained in an ILF by another application. An EIF has the inherent meaning that it is externally maintained, an interface has to be developed to get the data from the file. (Refer Figure 1)

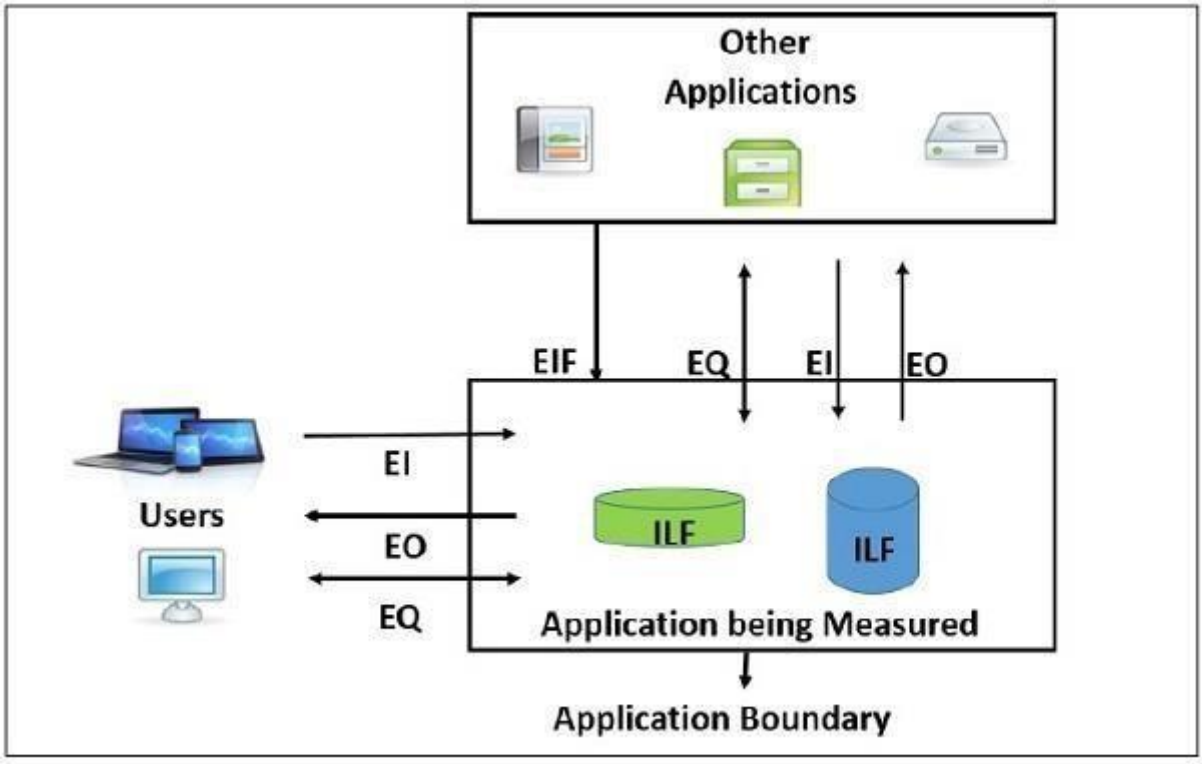


Figure 1: Application Boundary, Data Functions, Transaction Functions

Transaction Functions

There are three types of transaction functions.

- External Inputs
- External Outputs
- External Inquiries

Transaction functions are made up of the processes that are exchanged between the user, the external applications and the application being measured.

External Inputs

External Input (EI) is a transaction function in which Data goes “into” the application from outside the boundary to inside. This data is coming external to the application.

- Data may come from a data input screen or another application.
- An EI is how an application gets information.
- Data can be either control information or business information.
- Data may be used to maintain one or more Internal Logical Files.

- If the data is control information, it does not have to update an Internal Logical File. (Refer Figure 1)

External Outputs

External Output (EO) is a transaction function in which data comes “out” of the system. Additionally, an EO may update an ILF. The data creates reports or output files sent to other applications. (Refer Figure 1).

External Inquiries

External Inquiry (EQ) is a transaction function with both input and output components that result in data retrieval. (Refer Figure 1).

Experiment-12

Problem Statement: Develop a tool which can be used for quantification of all the non-functional requirements

Procedure:

Quantification of desirable properties of a system is an integral part of the software engineering. Most of requirement analysis methods which do not consider nonfunctional requirements lead to serious software development problems. The problems faced because of non-functional requirements omissions are more critical than the problems of functional requirements omissions. Therefore, it is necessary to measure non-functional requirement during software development process. Since software reliability is the major concern for quality system and considered as one of the most desirable quality attributes of the system, hence its quantitative measurement is an essential part of the development of a quality system. Estimation of software reliability becomes more important for those applications where risk is major consideration. Software reliability is the probability to perform failure free operation and produce correct output for a specified time under specified conditions. Once the system is installed, it is not possible to test system reliability in an operational environment because failure data collected in an uncontrolled tested environment may be limited. In this case, faults may not be necessarily removed as failures are identified. Therefore, failures are considered as one of the factors affecting the software. Failure data need to be properly collected and measured during software development process for the assessment of reliability of software systems. Software reliability is measured by collecting the historical data and various distribution curves. As failures are identified, faults are removed from the system to increase reliability. The fault tree uses fuzzy set theory for the quantification of uncertainty in order to make the system reliable. In the software development process, software is designed as an integration of sub-systems instead of a large system. The uncertainty of the overall system is found on the basis of uncertainties in the failure probability of sub-systems. Thus reliability requirements apply to the individual subsystems rather than the whole system. In recent years, various researchers developed different approaches

to analyze software quality attributes such as delta oriented slicing, feature wise measurement, fuzzy logic , scenarios and markov models and scenarios and Bayesian Belief network (BN).

Reliability is a key non-functional requirement, which can be divided into three Sub-NFRs (Availability and Recoverability). A. Availability Every system must be alive for service when it is requested by end-users. Availability of system indicates how reliable system is during operational hours. Therefore Availability is considered as one of the important metric used to assess the performance of a system, accounting for reliability of a system. It is defined as the ability to perform a required function under specified conditions at a given point of time or over a given time interval. It is often expressed as uptime and downtime. Uptime refers to the capability to perform the intended task in a stated environment and downtime refers to not being able to perform the intended task in a stated environment. In order to make system available, uptime of the system must be high and downtime of the system should be low. To reduce downtime, system should be properly maintained by constant monitoring of hardware & software and using anti-malicious software. To increase system availability during peak hours, dynamic Load balancing policies can be used to distribute I/O and CPU requests across all available paths to the storage device and servers respectively.

Recoverability

Recoverability is another important metric used to assess the reliability of a system. It is very important to know how often system fail to deliver expected level of service. This helps to measure the amount of data loss and downtime that a business can endure and decide how to recover from these failures. Good architecture provides recoverability in the time specified in a service-level agreement. To make system recoverable, backup of data should be taken at regular intervals. If mirroring of data is properly maintained, system will be more recoverable in case of any failure or disaster.

FUZZY SETS

Computer programming languages such as C, Java, and COBOL are best suited to develop such software systems whose behavior can be represented by mathematical model or logical reasoning. Since mathematical models do not work to develop software systems which require human judgment and decision making capabilities, Zadeh introduced the framework of Fuzzy sets to deal with a poorly defined concept in a coherent and structured way. Examples of poorly defined concepts suitable for the application of Fuzzy logic are semantic variables, such as high reliability, good performance, low maintenance, etc. The basis for proposing fuzzy logic was that human being relies on imprecise expression such as high, good or excellent. But software based systems work on Boolean logic. In this context, he emphasizes that human beings are desired to achieve the highest possible precision without paying attention to the imprecise character of quality. He proposed the theory of fuzzy sets in a mathematical way to represent vagueness in linguistics and can be considered as a generalization of classical set theory. In a classical set, an object either belongs to the set (member) or does not belong to the set (non-member). It means objects satisfy the

membership of the set precisely. Whereas in Fuzzy sets membership of the objects is approximate which helps to represent the real world system in more refined way. Fuzzy set removes the sharp boundaries that separate members and non-members in a group. In this case, the transition from a member to a non-member is gradual. It means that an object can belong to a fuzzy set either fully or partially. For example, categorizing whether room temperature is hot or cold is decided by considering various degree of membership. This example supports the concept of approximate membership. According to Zadeh, the membership of an object to a fuzzy set is represented by a continuous function defined on closed interval [0, 1] of real numbers. This type of membership incorporates various degrees of membership and also supports the concept of membership/non-membership of classical sets. Thus, classical sets are the collection of objects but Fuzzy sets are the collection of functions which defines the membership of the objects to the interval [0, 1]. Advantage of fuzzy logic is its ability to express the amount of ambiguity in human thinking and subjectivity. It is best suited to the problems which are concerned with continuous variables that are not easily divided into discrete variables. Fuzzy sets are the best choice for managing vague, imprecise, doubtful, contradicting and diverging opinions. IV. BAYESIAN NETWORKS Bayesian belief network represents compact networks of probabilities that capture the probabilistic relationships between variables, as well as historical information on their relationships. From another perspective, Bayesian belief network is a combination of Bayesian probability theory and the concept of conditional independence. It is also known as belief network, causal graph, causal network or probabilistic network. Bayesian belief network allows for clear graphical representation of cause and effect; and are effective for modeling scenarios where some prior information is already known but input data is uncertain, vague, conflicting or partially unavailable. Bayesian network is a way of representing joint probability distribution using Direct Acyclic Graph network structure given a set of variables □. Nodes of graph represent random uncertain variables and the arcs represent the Bayesian probabilistic relationships between these variables. It is a collection of conditional probability distributions where each variable □□ in the directed acyclic graph □ is denoted by a conditional distribution given its parent nodes □□□ሺ□□ሻ [23]. Bayesian network is also known as a network of nodes of influences based on reasoning. It uses Bayes theorem to express conditional probability between each alternative. It is a powerful tool for modeling causes and effects in systems and is sometimes described as a marriage between probability theory and graphical theory [24]. Its advantages are as follows: • There is no need of exact historical data or evidence to produce convincing results. • Despite of uncertainties in the input information, it provides effective output. • It represents the causes and effects relationship between the nodes in the form of arcs connecting nodes. • It helps in diagnosing current situation based on the historical relationship between nodes.

Experiment-13

Problem Statement: Write C/C++/Java/Python program for classifying the various types of coupling.

Procedure:

Coupling refers to the usage of an object by another object. It can also be termed as collaboration. This dependency of one object on another object to get some task done can be classified into the following two types −

- **Tight coupling** - When an object creates the object to be used, then it is a tight coupling situation. As the main object creates the object itself, this object cannot be changed from outside world easily marked it as tightly coupled objects.
- **Loose coupling** - When an object gets the object to be used from the outside, then it is a loose coupling situation. As the main object is merely using the object, this object can be changed from the outside world easily marked it as loosely coupled objects.

Example - Tight Coupling

Tester.java

```
publicclassTester{
  publicstaticvoid main(Stringargs[]){
    A a=newA();

    //a.display() will print A and B
    //this implementation can not be changed dynamically
    //being tight coupling
    a.display();
  }
}

class A {
  B b;
  public A(){
```

```
    //b is tightly coupled to A
    b =newB();
  }

  publicvoid display(){
    System.out.println("A");
    b.display();
  }
}

class B {
  public B(){}
  publicvoid display(){
    System.out.println("B");
  }
}
```

Output

A

B

Example - Loose Coupling

Tester.java

```
importjava.io.IOException;

publicclassTester{
  publicstaticvoid main(Stringargs[])throwsIOException{
    Show b =newB();
    Show c =newC();

    A a=newA(b);
```

```
    //a.display() will print A and B
    a.display();

    A a1 =new A(c);
    //a.display() will print A and C
    a1.display();
  }
}

interfaceShow{
  publicvoid display();
}

class A {
  Show s;
  public A(Show s){
    //s is loosely coupled to A
    this.s= s;
  }

  publicvoid display(){
    System.out.println("A");
    s.display();
  }
}

class B implementsShow{
  public B(){}
  publicvoid display(){
    System.out.println("B");
  }
```

```
}

class C implementsShow{
  public C(){}
  publicvoid display(){
    System.out.println("C");
  }
}
```

Output

A

B

A

C

Experiment-14

Problem Statement: Write C/C++/Java/Python program for classifying the various types of cohesion.

Procedure:

Cohesion in Java is the Object-Oriented principle most closely associated with making sure that a class is designed with a single, well-focused purpose. In object-oriented design, cohesion refers all to how a single class is designed.

The advantage of high cohesion is that such classes are much easier to maintain (and less frequently changed) than classes with low cohesion. Another benefit of high cohesion is that classes with a well-focused purpose tend to be more reusable than other classes.

Example: Suppose we have a class that multiplies two numbers, but the same class creates a pop-up window displaying the result. This is an example of a low cohesive class because the window and the multiplication operation don't have much in common. To make it high cohesive, we would have to create a class Display and a class Multiply. The Display will call Multiply's method to get the result and display it. This way to develop a high cohesive solution.

// Java program to illustrate

// high cohesive behavior

```
class Multiply
{
int a = 5;
int b = 5;
publicintmul(int a, int b)
{
this.a = a;
this.b = b;
return a * b;
}
}

class Display {
                        public static void main(String[] args)
                        {
                            Multiply m = new Multiply();
                            System.out.println(m.mul(5,   5));
                        }
}
```

Output

25

```
// Java program to illustrate
// high cohesive behavior

class Name {
        String name;
        public String getName(String name)
        {
                this.name = name;
                return name;
        }
}
```

```
class Age {
        int age;
        publicintgetAge(int age)
        {
                this.age = age;
                return age;
        }
}

class Number {
        int mobileno; public int getNumber(int
        mobile no)
        {
                this.mobileno = mobileno;
                returnmobileno;
        }
}

class Display {
        public static void main(String[] args)
        {
                Name n = new Name();
                System.out.println(n.getName("Geeksforgeeks"));
                Age a = new Age();
                System.out.println(a.getAge(10));
                Number no = new Number();
                System.out.println(no.getNumber(1234567891));
        }
```

}

Output
Geeksforgeeks

10

1234567891

Difference between high cohesion and low cohesion:

- High cohesion is when you have a class that does a well-defined job. Low cohesion is when a class does a lot of jobs that don’t have much in common.
- High cohesion gives us better-maintaining facility and Low cohesion results in monolithic classes that are difficult to maintain, understand and reduce re-usability

EXPERIMENT 15

Write a c program to demonstrate the working of the fallowing constructs: i) do…while

ii) while…do

iii) if …else

iv)switch

v) for Loops in C language

i) AIM: To demonstrate the working of do...while construct.Objective

To understand the working of do while with different range of values and test cases

```
#include <stdio.h>
void main ()
{
        int i, n=5,j=0;
        clrscr();
        printf(“enter a no”);
        scanf(“%d”,&i);
        do
        {
                if(i%2==0)
                {
                        printf("%d", i);
                        printf("is a even no.");
                        i++;
                        j++;
                }
                else
                {
                        printf("%d", i);
                        printf("is a odd no.\n");

                        i++;
                        j++;
                }
        }
        while(i>0&&j<n);
        getch();
}
```

Input	Actual output
2	2 is even number
	3 is odd number
	4 is even number
	5 is odd number
	6 is even number

Test cases:

Test case no: 1

Test case name: Positive values within range

Input =2	Expected output	Actual output	Remarks
	2 is even number	2 is even number	
	3 is odd number	3 is odd number	success
	4 is even number	4 is even number	
	5 is odd number	5 is odd number	
	6 is even number	6 is even number	

Test case no: 2

Test case name: Negative values within a range

Input = -2	Expected output	Actual output	Remarks
	-2 is even number	-2 is an even number	
	-3 is odd number		fail
	-4 is even number		
	-5 is odd number		
	-6 is even number		

Test case no: 3

Test case name: Out of range values testing

Input	Expected output	Actual output	Remarks
123456789122222222	123456789122222	13 234567891222222215	fail

ii) AIM: To demonstrate the working of while

construct Objective

To understand the working of while with different range of values and test cases

```
#include<stdio.h>
#include <conio.h>
void main ()
{
        int i, n=5,j=1;
        clrscr();
        printf("enter a no");
```

```
        scanf("%d",&i);
        while (i>0 && j<n)
        {
                if(i%2==0)
                {
                        printf("%d",i);
                        printf("is a even number");
                        i++;
                        j++;
                }
                else
                {
                        printf("%d",i);
                        printf("is a odd number");
                         i++;
                         j++;
                 }
        }
        getch();
}
```

Input	**Actual output**
2	2 is even number
	3 is odd number
	4 is even number
	5 is odd number
	6 is even number

Test cases:

Test case no:1

Test case name: Positive values within range

Input =2	**Expected output**	**Actual output**	**Remarks**
	2 is even number	2 is even number	
	3 is odd number	3 is odd number	success
	4 is even number	4 is even number	
	5 is odd number	5 is odd number	
	6 is even number	6 is even number	

Test case no:2

Test case name: Negative values within a range

Input = -2	**Expected output**	**Actual output**	**Remarks**
	-2 is even number	-2 is an even number	
	-3 is odd number		fail
	-4 is even number		
	-5 is odd number		
	-6 is even number		

Test case no: 3

Test case name: Out of range values testing

Input	**Expected output**	**Actual output**	
Remarks			
123456789122222222	1234567891222222222	13 234567891222222215	fail

iii) AIM: To demonstrate the working of if else construct

Objective To understand the working of if else with different range of values and test cases

```
#include<stdio.h>
#include <conio.h>
void main ()
{
        int i;
        clrscr();
        printf("enter a number");
        scanf("%d",&i);
        if(i%2==0)
        {
                printf("%d",i);
                printf("is a even number");
        }
        else
        {
                printf("%d",i);
                printf("is a odd number");
        }
        getch();
}
```

Input	**Actual output**
2	2 is even number
	3 is odd number
	4 is even number

5 is odd number

6 is even number

Test cases:

Test case no: 1

Test case name: Positive values within range

Input =2	**Expected output**	**Actual output**	**Remarks**
	2 is even number	2 is even number	
	3 is odd number	3 is odd number	success
	4 is even number	4 is even number	
	5 is odd number	5 is odd number	
	6 is even number	6 is even number	

Test case no:2

Test case name: Negative values within a range

Input = -2	**Expected output**	**Actual output**	**Remarks**
	-2 is even number	-2 is an even number	
	-3 is odd number		fail
	-4 is even number		
	-5 is odd number		
	-6 is even number		

Test case no: 3

Test case name: Out of range values testing

Input	**Expected output**	**Actual output**	**Remarks**
1234567891222222222	1234567891222222222	13234567891222222215	fail

iv) AIM: To demonstrate the working of switch construct

Objective To understand the working of switch with different range of values and test cases

```
#include<stdio.h>
void main()
{
        int a,b,c;
        clrscr();
        printf("1.Add/n 2.Sub /n 3.Mul /n 4.Div /n Enter Your choice");
        scanf("%d", &i);
        printf("Enter a,b values");
        scanf("%d%d",&a,&b);
        switch(i)
        {
                case 1: c=a+b;
```

```
                printf("The sum of a & b is: %d" ,c);
                break;
                case 2: c=a-b;
                printf("The Diff of a & b is: %d" ,c);
                break;
                case 3: c=a*b;
                printf("The Mul of a & b is: %d" ,c);
                break;
                case 4: c=a/b;
                printf("The Div of a & b is: %d" ,c);
                break;
                default: printf("Enter your choice");
                break;
        }
        getch();
}
```

Output:

Input	**Output**
Enter Ur choice: 1	
Enter a, b Values: 3, 2	The sum of a & b is:5
Enter Ur choice: 2	
Enter a, b Values: 3, 2	The diff of a & b is: 1
Enter Ur choice: 3	
Enter a, b Values: 3, 2	The Mul of a & b is: 6
Enter Ur choice: 4	
Enter a, b Values: 3, 2	The Div of a & b is: 1

Test cases:

Test case no: 1

Test case name: Positive values within range

Input	**Expected output**	**Actual output**	**Remarks**
Enter Ur choice: 1			
Enter a, b Values: 3, 2	The sum of a & b is:5	5	
Enter Ur choice: 2			
Enter a, b Values: 3, 2	The diff of a & b is: 1	1	Success
Enter Ur choice: 3			
Enter a, b Values: 3, 2	The Mul of a & b is: 6	6	
Enter Ur choice: 4			

Enter a, b Values: 3, 2 The Div of a & b is: 1 1

Test case no:2

Test case name: Out of range values testing

Input	**Expected output**	**Actual output**	**Remarks**
Option: 1 a= 22222222222222 b=22222222222222	44444444444444	-2	fail

Test case no: 3

Test case name: Divide by zero

Input	**Expected output**	**Actual output**	**Remarks**
Option: 4 a= 10 & b=0	error		fail

v)AIM: To demonstrate working of for construct

Objective To understand the working of for with different range of values and test cases

```
#include <stdio.h>
#include <conio.h>
void main ()
{
        int i;
        clrscr();
        printf("enter a no");
        scanf("%d",&i);
        for(i=1;i<=5;i++)
        {
                if(i%2==0)
                {
                        printf("%d", i);
                        printf("is a even no");
                        i++;
                }
                else
                {


                        printf("%d", i);
```

```
                printf("is a odd no");
                i++;
            }
        }
        getch();
}
```

Output:

Enter a no: 5

0 is a even no

1 is a odd no

2 is a even no

3 is a odd no

4 is a even no

5 is a odd no

Test cases:

Test case no: 1

Test case name: Positive values within range

Input =2	**Expected output**	**Actual output**	**Remarks**
	0 is even number	0 is even number	
	1 is odd number	1 is odd number	success
	2 is even number	2 is even number	

Test case no: 2

Test case name: Negative values within a range

Input = -2	**Expected output**	**Actual output**	**Remarks**
	0 is even number	0 is an even number	
	-1 is odd number	-1 is even no	fail
	-2 is even number	-2 is odd no	

Test case no: 3

Test case name: Out of range values testing

Input	**Expected output**	**Actual output**	**Remarks**
12345678912222222222	123456789122222222	13 23456789122222215	fail

EXPERIMENT 16

AIM: A program written in c language for matrix multiplication fails —Introspect the causes for its failure and write down the possible reasons for its failure.

Objective: Understand the failures of matrix multiplication

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a[3][3],b[3][3],c[3][3],i,j,k,m,n,p,q;
        clrscr();
        printf(“Enter 1st matrix no.of rows & cols”);
        scanf(“%d%d”,&m,&n);
        printf(“Enter 2nd matrix no.of rows & cols”);
        scanf(“%d%d”,&p,&q);
        printf("\n enter the matrix elements");
        for(i=0;i<m;i++);
        {
                for(j=0;j<n;j++);
                {
                        scanf("%d",&a[i][j]);
                }
        }
        printf("\n a matrix is\n");
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                {
                        printf("%d\t",a[i][j]);
                }
                printf("\n");
        }
        for(i=0;i<p;i++)
        {
                for(j=0;j<q;j++)
```

```
		{
			scanf("%d\t",&b[i][j]);
		}
	}
	printf("\n b matrix is\n");
	for(i=0;i<p;i++)
	{
		for(j=0;j<q;j++)
		{
			printf("%d\t",b[i][j]);
		}
		printf("\n");
	}
	for(i=0;i<m;i++)
	{
		for(j=0;j<q;j++)
		{
			c[i][j]=0;
			for(k=0;k<n;k++)
			{
				c[i][j]=c[i][j]+a[i][k]*b[k][j];
			}
		}
	}
	for(i=0;i<m;i++)
	{
		for(j=0;j<q;j++)
		{
			printf("%d\t",c[i][j]);
		}
		printf("\n");
	}
	getch();
}
```

Output:

Enter Matrix1: 1 1 1
1 1 1
1 1 1

Enter Matrix2: 1 1 1
1 1 1
1 1 1

Actual Output: 3 3 3
3 3 3
3 3 3

Test cases:

Test case no: 1

Test case name: Equal no.of rows & cols

Input	**Expected output**	**Actual output**	**Remarks**
Matrix1 rows & cols= 3 3			
Matrix2 rows & cols= 3 3			
Matrix1: 1 1 1			
1 1 1	3 3 3	3 3 3	
1 1 1	3 3 3	3 3 3	Success
	3 3 3	3 3 3	
Matrix2: 1 1 1			
1 1 1			
1 1 1			

Test case no:2

Test case name: Cols of 1st matrix not equal to rows of 2nd matrix

Input	**Expected output**	**Actual output**	**Remarks**
Matrix1 rows & cols= 2 2	Operation Can_t be Performed		fail
Matrix2 rows & cols= 3 2			

Test case no: 3

Test case name: Out of range values testing

Input	**Expected output**	**Actual output**	**Remarks**

Matrix1 rows & cols= 2 2

Matrix2 rows & cols= 2 2

1234567891	2222222222	fail
2234567891	2222222221	
234567891	22222221533	
213242424	56456475457	

EXPERIMENT 17

AIM: Take ATM system and study its system specifications and report the various bugs.

1. Machine is accepting ATM card.
2. Machine is rejecting expired card.
3. Successful entry of PIN number.
4. Unsuccessful operation due to enter wrong PIN number 3 times.
5. Successful selection of language.
6. Successful selection of account type.
7. Unsuccessful operation due to invalid account type.
8. Successful selection of amount to be withdrawn.
9. Successful withdrawal.
10. Expected message due to amount is greater than day limit.
11. Unsuccessful withdraw operation due to lack of money in ATM.
12. Expected message due to amount to withdraw is greater than possible balance.
13. Unsuccessful withdraw operation due to click cancel after insert card.

EXPERIMENT 18

AIM: Write the test cases for Banking application.

1. Checking mandatory input parameters
2. Checking optional input parameters
3. Check whether able to create account entity.
4. Check whether you are able to deposit an amount in the newly created account (and thus updating the balance)
5. Check whether you are able to withdraw an amount in the newly created account (after deposit) (and thus updating the balance)
6. Check whether company name and its pan number and other details are provided in case of salary account
7. Check whether primary account number is provided in case of secondary account
8. Check whether company details are provided in cases of company's current account
9. Check whether proofs for joint account is provided in case of joint account
10. Check whether you are able deposit an account in the name of either of the person in an joint account.
11. Check whether you are able withdraw an account in the name of either of the person in an joint account.
12. Check whether you are able to maintain zero balance in salary account
13. Check whether you are not able to maintain zero balance (or mini balance) in non-salary account.

EXPERIMENT 19

AIM: Create a test plan document for Library Management System.

The Library Management System is an online application for assisting a librarian managing book library in a University. The system would provide basic set of features to add/update clients, add/update books, search for books, and manage check-in / checkout processes. Our test group tested the system based on the requirement specification**.** This test report is the result for testing in the LMS. It mainly focuses on two problems

1. What we will test

2. How we will test.

3. GUI test

Pass criteria: librarians could use this GUI to interface with the backend library database without any difficulties

4. Database test

Pass criteria: Results of all basic and advanced operations are normal (refer to section 4)

5. Basic function test Add a student

Each customer/student should have following attributes: Student ID/SSN (unique), Name, Address and Phone number.

The retrieved customer information by viewing customer detail should contain the four attributes.

6. Update/delete student

1. The record would be selected using the student ID.
2. Updates can be made on full. Items only: Name, Address, Phone number The record can be deleted if there are no books issued by user. The updated values would be reflected if the same customer's ID/SSN is called for.

7. Check-in book

Librarians can check in a book using its call number

1. The check-in can be initiated from a previous search operation where user has selected a set of books.

2. The return date would automatically reflect the current system date.
3. Any late fees would be computed as difference between due date and return date at rate of 10 cents a day.

EXPERIMENT 20

Design and develop a program in a language of your choice to solve the triangle problem defined as follows: Accept three integers which are supposed to be the three sides of a triangle and determine if the three values represent an equilateral triangle, isosceles triangle, scalene triangle, or they do not form a triangle at all. Assume that the upper limit for the size of any side is 10. Derive test cases for your program based on boundary-value analysis, execute the test cases and discuss the results.

```
#include<stdio.h>
#include<conio.h>
int main( )
{
   int a,b,c,c1,c2,c3;
   do
   {
      printf("enter the sides of triangle\n");
      scanf("%d%d%d",&a,&b,&c);
      c1=((a>=1) && (a<=10));
    c2=((b>=1) && (b<=10));
      c3=((c>=1) && (c<=10));
      if(!c1)
          printf("value of a is out of range");
      if(!c2)
          printf("value of b is out of range");
```

```
    if(!c3)
        printf("value of c  is out of range");
  }while(!c1 || !c2 || !c3);
  if((a+b)>c && (b+c)>a && (c+a)>b)
  {
     if(a==b && b==c)
        printf("Triangle is equilateral\n");
     else if(a!=b && b!=c && c!=a)
        printf("Triangle is scalene\n");
     else
        printf("Triangle is isosceles\n");
  }
  else
     printf("Triangle cannot be formed \n");
  getch( );
  return 0;
}
```

Boundary Value Analysis

Boundary value analysis focuses on the boundary of the input and output space to identify test cases because errors tend to occur near the extreme values of an input variable. The basic idea is to use input variables at their minimum, just above minimum, nominal, just below their maximum and maximum.

Considering Triangle program, we have three variables a, b and c. Each variables value ranges from 1 to 10.

Variables	Min	Min+	Nom	Max-	Max
a	1	2	5	9	10
b	1	2	5	9	10
c	1	2	5	9	10

Boundary Value Analysis = 4n+1 test cases, where n is number of variables

In Triangle program for BVA, we start by taking nominal values for **a** and **b** variables then cross product it with values min, min-, nom, max- and max values of variable **c**. similarly keeping nominal values for variables **a** and **c**, we cross product it with min, min-, nom, max-, max values of variable **b**. Again keeping variable **b** and **c** as nominal combine with 5 values of **a**. By this we get 15 te st cases in which a test case with all nominal values for **a**, **b** and **c** is repeated thrice, so we discard 2 duplicate such cases and finally we get 15-2=**13** test cases which is equal to BVA i.e., 4(3)+1=**13**.

Test cases using Boundary value analysis for Triangle Program

Test cases	Description	Inputs			Output	Comments
		A	B	C		
BVA1	Enter the values of a(nom),b(nom) and c(min)	5	5	**1**	Isosceles	Valid
BVA 2	Enter the values of a(nom),b(nom) and c(min+)	5	5	**2**	Isosceles	Valid
BVA 3	Enter the values of a(nom),b(nom) and c(nom)	5	5	**5**	Equilateral	Valid
BVA 4	Enter the values of a(nom),b(nom) and c(max-)	5	5	**9**	Isosceles	Valid
BVA 5	Enter the values of a(nom),b(nom) and c(max)	5	5	**10**	Triangle cannot be formed	Valid
BVA 6	Enter the values of a(nom),b(min) and c(nom)	5	**1**	5	Isosceles	Valid
BVA 7	Enter the values of a(nom),b(min+) and c(nom)	5	**2**	5	Isosceles	Valid
BVA 8	Enter the values of a(nom),b(max-) and c(nom)	5	**9**	5	Isosceles	Valid
BVA 9	Enter the values of a(nom),b(max) and c(nom)	5	**10**	5	Triangle cannot be formed	Valid
BVA 10	Enter the values of a(min),b(nom) and c(nom)	**1**	5	5	Isosceles	Valid
BVA 11	Enter the values of a(min+),b(nom) and c(nom)	**2**	5	5	Isosceles	Valid
BVA 12	Enter the values of a(max-),b(nom) and c(nom)	**9**	5	5	Isosceles	Valid
BVA 13	Enter the values of a(max),b(nom) and c(nom)	**10**	5	5	Triangle cannot be formed	Valid

EXPERIMENT 21

Design and develop a program in a language of your choice to solve the triangle problem defined as follows: Accept three integers which are supposed to be the three sides of a triangle and determine if the three values represent an equilateral triangle, isosceles triangle, scalene triangle, or they do not form a triangle at all. Derive test cases for your program based on decision table approach, execute the test cases and discuss the results.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int a,b,c;
    printf("enter the sides of triangle\n");
    scanf("%d%d%d",&a,&b,&c);
    if((a+b)>c && (b+c)>a && (c+a)>b)
    {
      if(a==b && b==c)
          printf("triangle is equilateral\n");
      else if (a!=b && b!=c && c!=a)
          printf("triangle is scalene\n");
      else
        printf("triangle is isosceles\n");
    }
```

```
    else
        printf("triangle cannot be formed\n");
return 0;
}
```

Decision Table for Triangle Problem

		R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11
Conditions	**C1: a<b+c**	F	T	T	T	T	T	T	T	T	T	T
	C2: b<c+a	--	F	T	T	T	T	T	T	T	T	T
	C3: c<a+b	--	--	F	T	T	T	T	T	T	T	T
	C4: a=b	--	--	--	T	T	F	F	F	T	T	F
	C5: b=c	--	--	--	T	F	T	F	F	T	F	T
	C6: c=a	--	--	--	T	F	F	T	F	F	T	T
Actions	**A1: Not a triangle**	x	x	x								
	A2: Equilateral				x							
	A3: Isosceles					x	x	x				
	A4: Scalene								x			
	A5: Impossible									x	x	x

Test Cases using Decision Table for Triangle Program

Test cases	Description	Inputs			Output	Comments
		A	B	C		
Case 1	Enter the values of a, b and c such that value of a is greater than sum of b and c.	7	2	3	Not a triangle	Valid
Case 2	Enter the values of a, b and c such that value of b is greater than sum of a and c.	2	8	3	Not a triangle	Valid
Case 3	Enter the values of a, b and c such that value of c is greater than sum of a and b.	2	4	7	Not a triangle	Valid
Case 4	Enter the values of a, b and c such that values of a,b and c are equal.	5	5	5	Equilateral	Valid
Case 5	Enter the values of a, b and c Such that value of a is equal to value of b.	4	4	3	Isosceles	Valid
Case 6	Enter the values of a, b and c such that value of b is equal to value of c.	2	5	5	Isosceles	Valid
Case 7	Enter the values of a, b and c such that value of a is equal to value of c.	6	2	6	Isosceles	Valid
Case 8	Enter the values of a, b and c such that values of a,b and c are different.	2	3	4	Scalene	Valid
Case 9	Enter the values of a, b and c such that value of a is equal to value of b and c but value of b is not equal to c.	?	?	?	Impossible	Valid
Case 10	Enter the values of a, b and c such that value of b is equal to value of c and value of c is equal to a but value of a not equal to b.	?	?	?	Impossible	Valid
Case 11	Enter the values of a, b and c such that value of a is equal to b and value of b is equal to value of c but value of a not equal to c.	?	?	?	Impossible	Valid

EXPERIMENT 22

Design and develop a program in a language of your choice to solve the triangle problem defined as follows: Accept three integers which are supposed to be the three sides of a triangle and determine if the three values represent an equilateral triangle, isosceles triangle, scalene triangle, or they do not form a triangle at all. Assume the upper limit for the size of any side is 10. Derive test cases for your program based on equivalence class partitioning, execute the test cases and discuss the results.

```
#include<stdio.h>
#include<conio.h>
int main()
{
   int a,b,c,c1,c2,c3;
   do
   {
      printf("enter the sides of triangle\n");
      scanf("%d%d%d",&a,&b,&c);
      c1=((a>=1) && (a<=10));
      c2=((b>=1) && (b<=10));
      c3=((c>=1) && (c<=10));
      if(!c1)
         printf("value of a is out of range");
      if(!c2)
        printf("value of b is out of range");
```

```
        if(!c3)
            printf("value of c is out of range");
     }while(!c1 || !c2 || !c3);
 if((a+b)>c && (b+c)>a && (c+a)>b)
 {
     if(a==b && b==c)
        printf("triangle is equilateral\n");
     else if(a!=b && b!=c && c!=a)
       printf("triangle is scalene\n");
     else
       printf("triangle is isosceles\n");
  }
 else
    printf("triangle cannot be formed \n");
getch( );
return 0;
}
```

Equivalence Class Test For The Triangle Program

Output Equivalence Classes are as follows:

R1={<a,b,c>:the triangle with sides a,b and c is Equilateral}

R2={<a,b,c>:the triangle with sides a,b and c is Isosceles}

R3={<a,b,c>:the triangle with sides a,b and c is Scalene}

R4={<a,b,c>:sides a,b and c do not form a Triangle}

Weak Normal /Strong Normal

Test cases	Description	Inputs			Expected output	Comments
		A	B	C		
WN/SN1	Enter the valid values for a, b and c from output equivalence classes.	5	5	5	Equilateral	Valid
WN/SN2	Enter the valid values for a, b and c from output equivalence classes.	5	5	3	Isosceles	Valid
WN/SN3	Enter the valid values for a, b and c from output equivalence classes.	5	3	4	Scalene	Valid
WN/SN4	Enter the valid values for a, b and c from output equivalence classes.	10	1	1	Triangle cannot be formed	Valid

Weak Robust

Test cases	Description	Inputs			Expected output	Comments
		A	B	C		
WR 1	Enter the valid values for b and c from output equivalence classes and invalid value for a.	-1	5	5	Value of a is not in a range	Triangle cannot be formed
WR 2	Enter the valid values for a and c from output equivalence classes and invalid value for b.	3	-1	4	Value of b is not in a range	Triangle cannot be formed
WR 3	Enter the valid values for a and b from output equivalence classes and invalid value for c.	10	10	-1	Value of c is not in a range	Triangle cannot be formed
WR 4	Enter the valid values for b and c from output equivalence classes and invalid value for a.	11	3	3	Value of a is not in a range	Triangle cannot be formed
WR 5	Enter the valid values for a and c from output equivalence classes and invalid value for b.	5	11	6	Value of b is not in a range	Triangle cannot be formed
WR 6	Enter the valid values for a and b from output equivalence classes and invalid value for c.	9	10	11	Value of c is not in a range	Triangle cannot be formed

Strong Robust

Test cases	Description	Inputs			Expected output	Comments
		A	B	C		
SR 1	Enter the valid value for b from output equivalence classes and invalid values for a and c.	-1	3	-1	Values of a and c are not in range	Triangle cannot be formed
SR 2	Enter the valid value for a from output equivalence classes and invalid values for b and c.	5	-1	-1	Values of b and c are not in range	Triangle cannot be formed
SR 3	Enter the valid value for c from output equivalence classes and invalid values for a and b.	-1	-1	10	Values of a and b are not in range	Triangle cannot be formed
SR 4	Enter the valid value for a from output equivalence classes and invalid values for b and c.	7	11	11	Values of b and c are not in range	Triangle cannot be formed
SR 5	Enter the valid value for c from output equivalence classes and invalid values for a and b.	11	11	10	Values of a and b are not in range	Triangle cannot be formed
SR 6	Enter the valid value for b from output equivalence classes and invalid values for a and c.	11	5	11	Values of a and c are not in range	Triangle cannot be formed

SR 7	Enter the valid values for b and c from output equivalence classes and invalid value for a.	-1	5	5	Values of a is not in range	Triangle cannot be formed
SR 8	Enter the valid values for a and c from output equivalence classes and invalid value for b.	10	-1	10	Values of b is not in range	Triangle cannot be formed
SR 9	Enter the valid values for a and b from output equivalence classes and invalid value for c.	7	6	-1	Values of c is not in range	Triangle cannot be formed
SR 10	Enter the valid values for b and c from output equivalence classes and invalid value for a.	11	5	4	Values of a is not in range	Triangle cannot be formed
SR 11	Enter the valid values for a and c from output equivalence classes and invalid value for b.	2	11	3	Values of b is not in range	Triangle cannot be formed
SR 12	Enter the valid values for a and b from output equivalence classes and invalid value for c.	3	4	11	Values of c is not in range	Triangle cannot be formed
SR 13	Enter the invalid value for a, b and c.	11	11	11	Values of a, b and c are not in a range	Triangle cannot be formed
SR 14	Enter the invalid value for a, b and c.	-1	-1	-1	Values of a, b and c are not in a range	Triangle cannot be formed

23. Draw standard UML diagrams using an UML modeling tool for a given case study and map design to code and implement a 3 layered architecture. Test the developed code and validate whether the SRS is satisfied.

Designing Guidelines:

A. Identify a software system that needs to be developed.
B. Document the Software Requirements Specification (SRS) for the identified system.
C. Identify use cases and develop the Use Case model.
D. Identify the conceptual classes and develop a Domain Model and also derive a Class Diagram from that.
E. Using the identified scenarios, find the interaction between objects and represent them using UML Sequence and Collaboration Diagrams
F. Draw relevant State Chart and Activity Diagrams for the same system.
G. Implement the system as per the detailed design
H. Test the software system for all the scenarios identified as per the usecase diagram
I. Improve the reusability and maintainability of the software system by applying appropriate design patterns.
J. Implement the modified system and test it for various scenarios

Suggested domain for validate the following system:

a. Passport automation system.
b. Book bank
c. Exam registration
d. Stock maintenance system.
e. Online course reservation system

EXPERIMENT : 23 (a)

PASSPORT AUTOMATION SYSTEM

AIM:

To create an automated system to perform the Passport Process.

(I) 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.

(II)SOFTWARE REQUIREMENT SPECIFICATION:

2.1SOFTWARE INTERFACE

- **Front End Client** - The applicant and Administrator online interface is built using JSP and HTML. The Administrators's local interface is built using Java.

•**Web Server** - Glassfish application server(Oracle Corporation).

•**Back End** - Oracle database.

2.2HARDWARE INTERFACE

The server is directly connected to the client systems. The client systems have access to the database in the server.

(III) USECASE DIAGRAM :

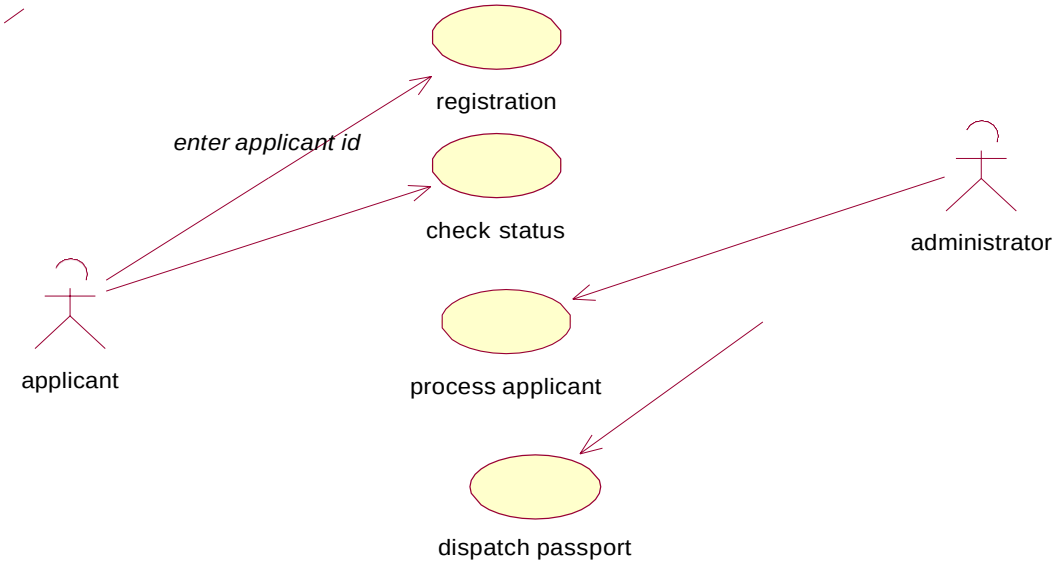


Fig.3. USECASE DIAGRAM FOR PASSPORT AUTOMATION SYSTEM

(IV) ACTIVITY DIAGRAM:

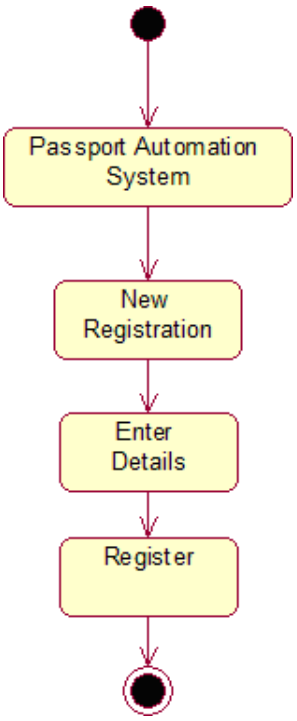


Fig.4.1. ACTIVITY DIAGRAM FOR REGISTER

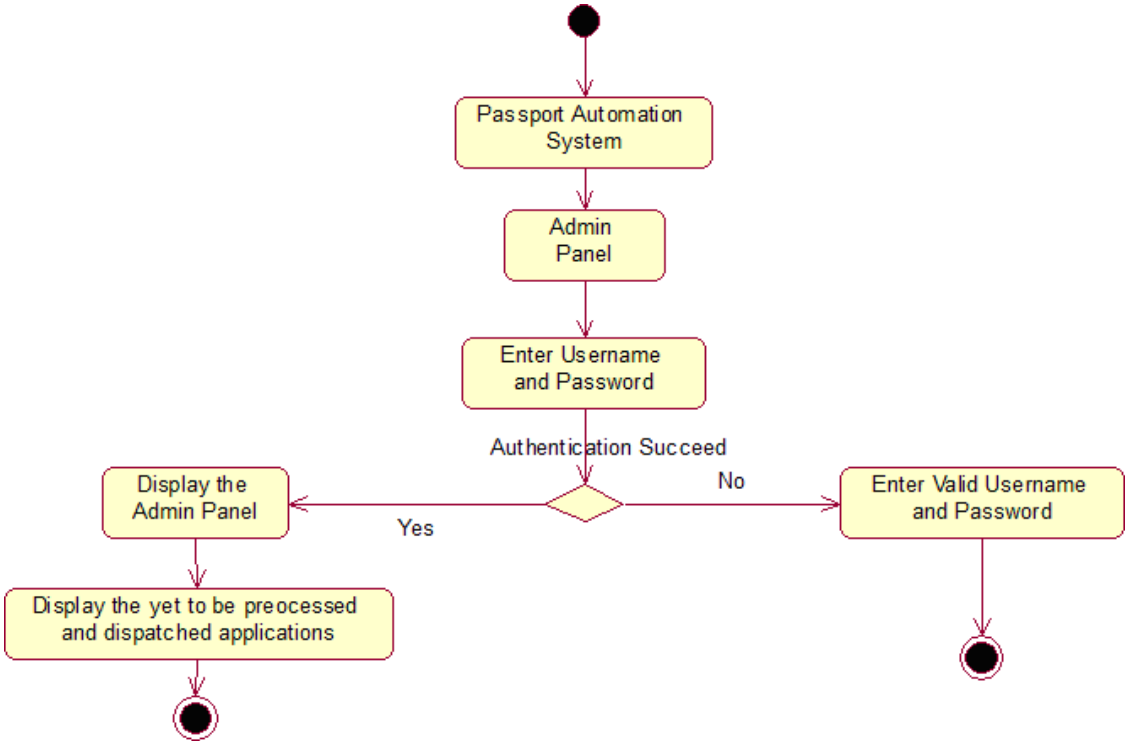


Fig.4.2. ACTIVITY DIAGRAM FOR ADMINISTRATION

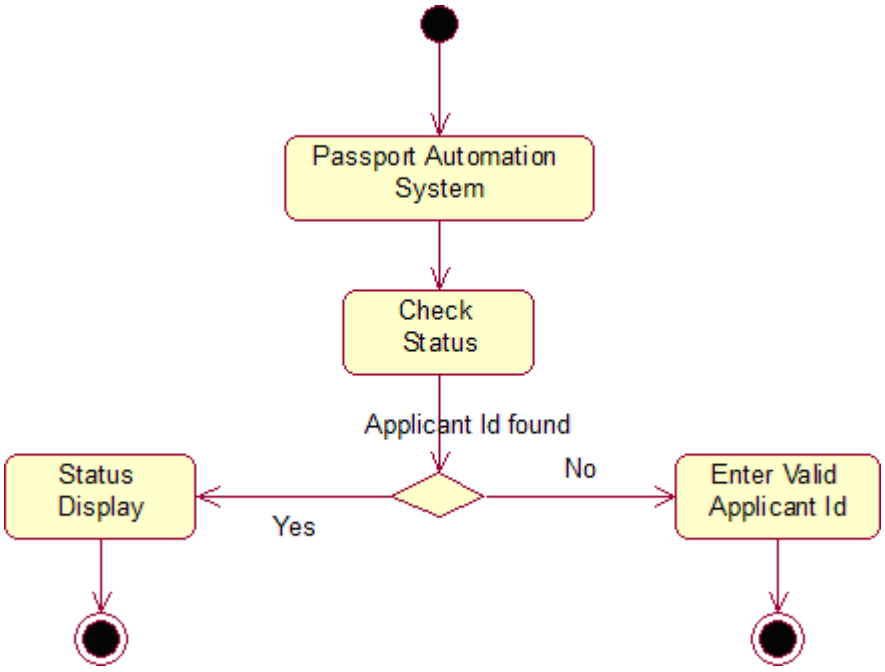


Fig.4.3. ACTIVITY DIAGRAM FOR CHECKING STATUS

(V) CLASS DIAGRAM:

The class diagram, also referred to as object modeling is the main static analysis diagram. The main task of object modeling is to graphically show what each object will do in the problem domain. The problem domain describes the structure and the relationships among objects.

The Passport Automation system class diagram consists of four classesPassport Automation System

1. New registration
2. Gender
3. Application Status
4. Admin authentication
5. Admin Panel

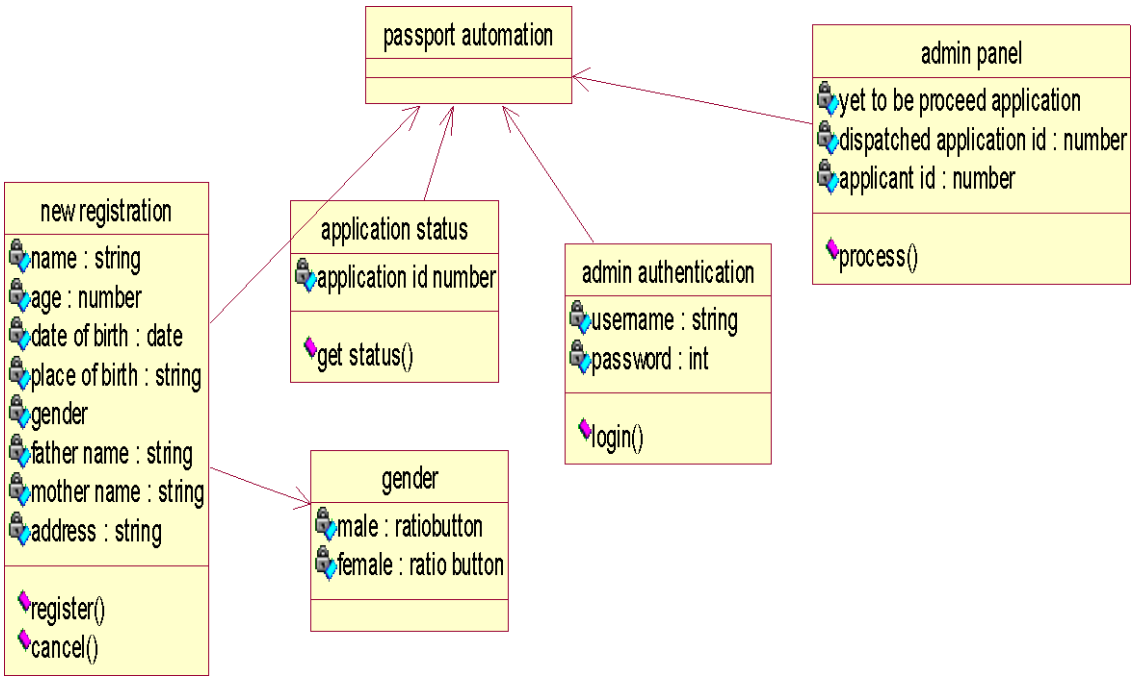


Fig.5. CLASS DIAGRAM FOR PASSPORT AUTOMATION SYSTEM

(VI) INTERACTION 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.

An event also is considered to be any action by an object that sends information. The event line represents a message sent from one object to another, in which the “form” object is requesting an operation be performed by the “to” object. The “to” object performs the operation using a method that the class contains.

It is also represented by the order in which things occur and how the objects in the system send message to one another.

The sequence diagram for each USE-CASE that exists when a user administrator, check status and new registration about passport automation system are given.

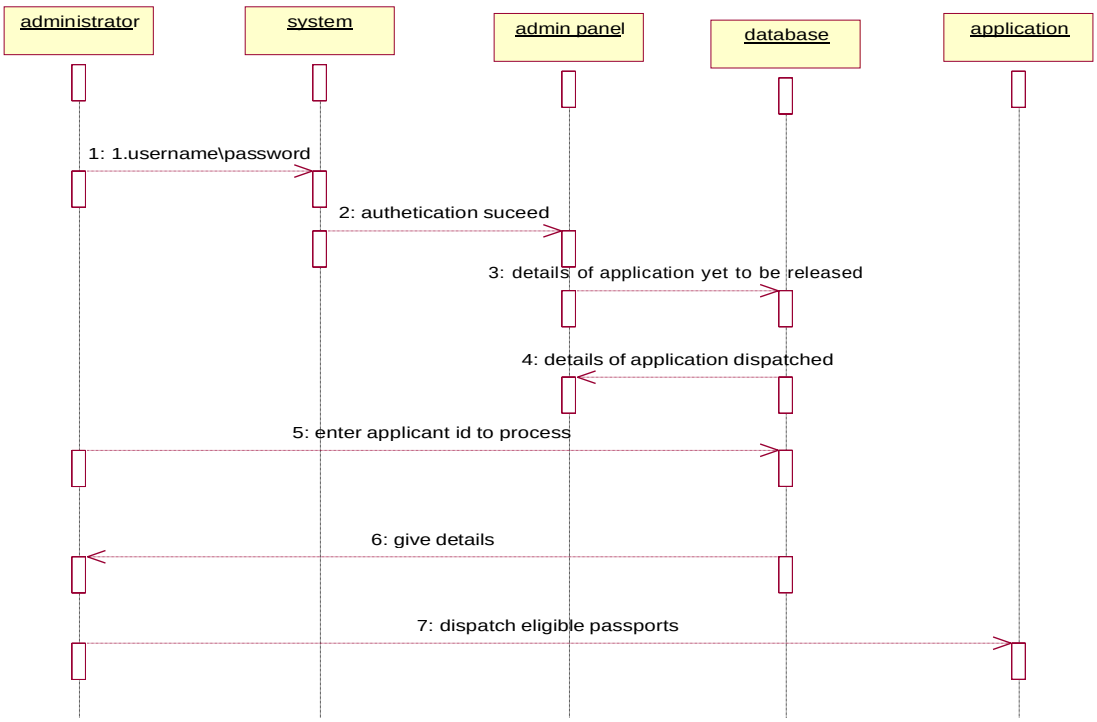


Fig.6.1.SEQUENCE DIAGRAM FOR ADMINISTRATOR

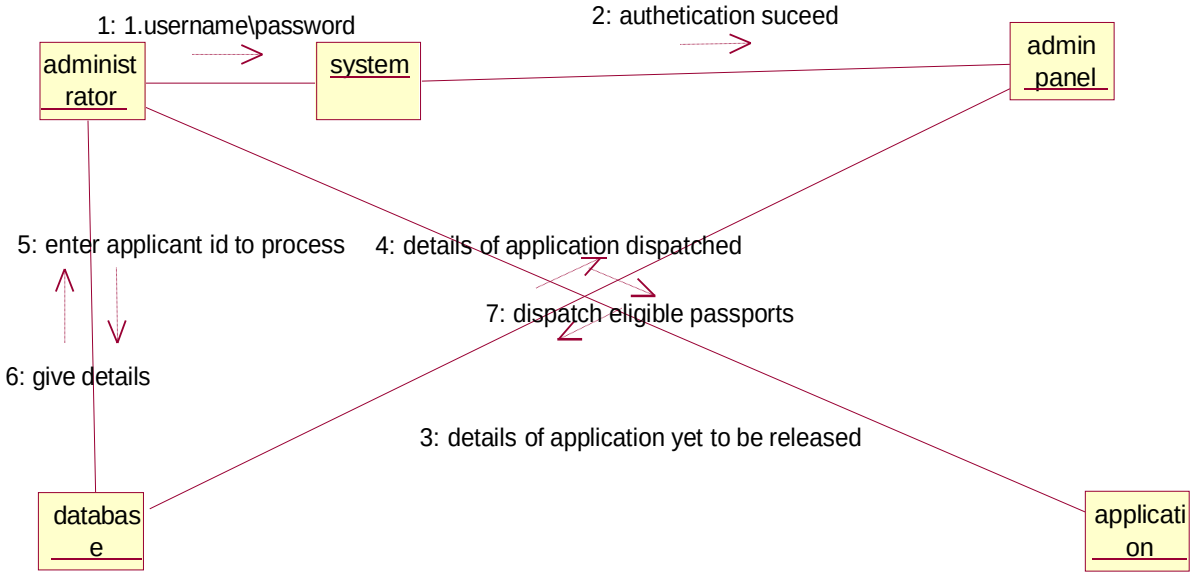


Fig.6.2.COLLABORATION DIAGRAM FOR ADMINISTRATOR

The diagrams show the process done by the administrator to the Passport Automation system. The applicant has to enter his details. The

details entered are verified by the administrator and the applicant is approved if the details match then the passport is dispatch, otherwise an appropriate error message is displayed.

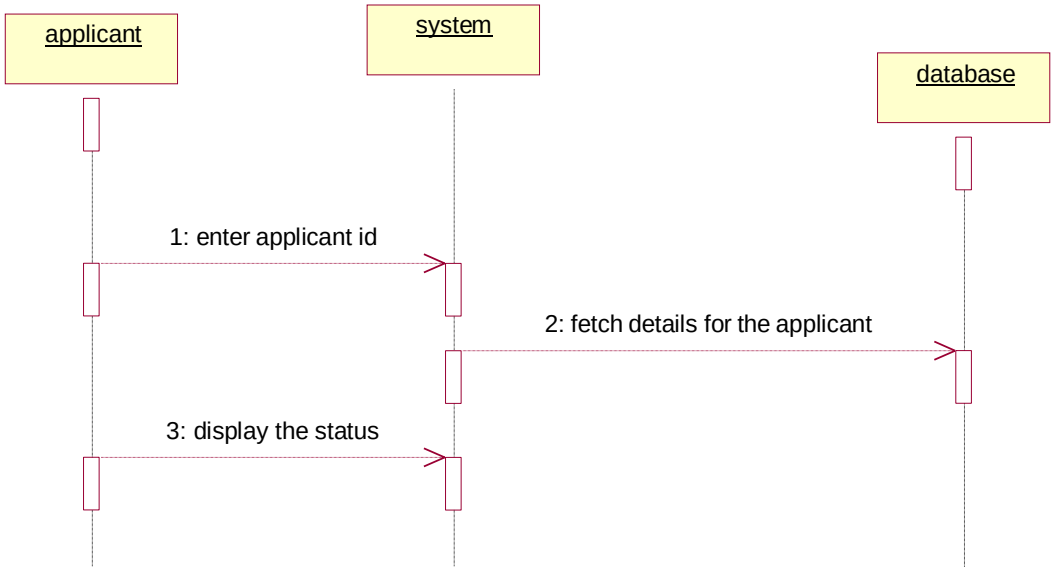


Fig.6.3.SEQUENCE DIAGRAM FOR CHECKING STATUS

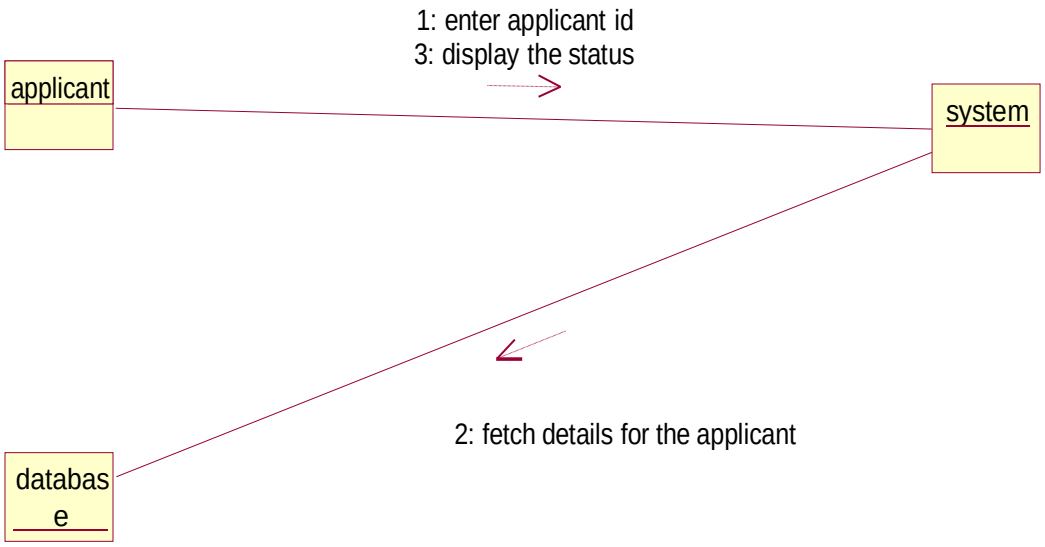


Fig.6.4.COLLABORATION DIAGRAM FOR CHECKING STATUS

The diagrams show the applicant enters his id and the system fetch the details from the database and display the status.

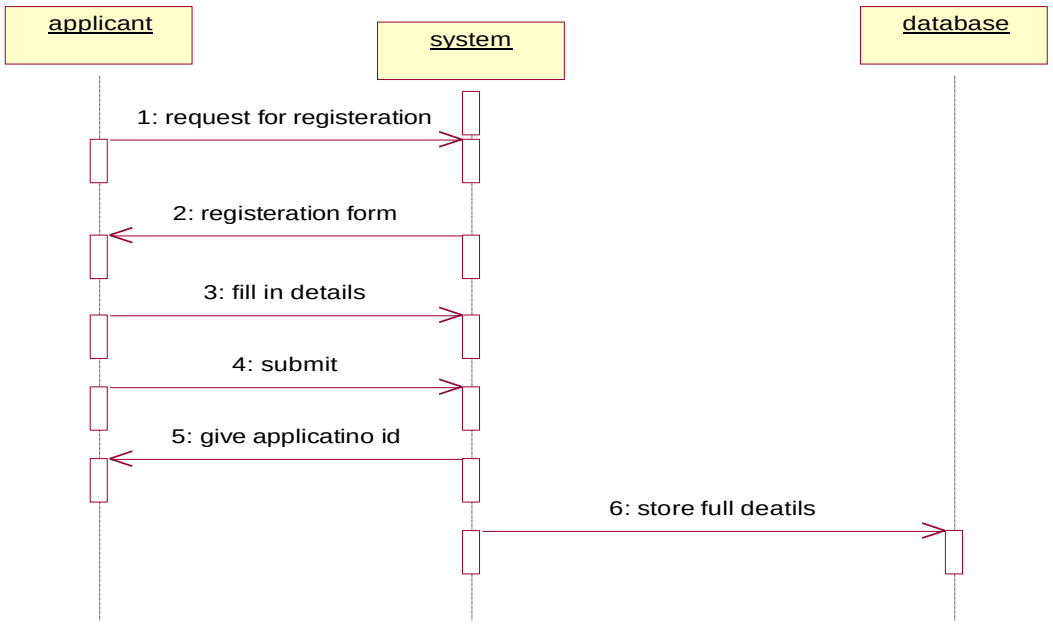


Fig.6.5.SEQUENCE DIAGRAM FOR NEW REGISTRATION

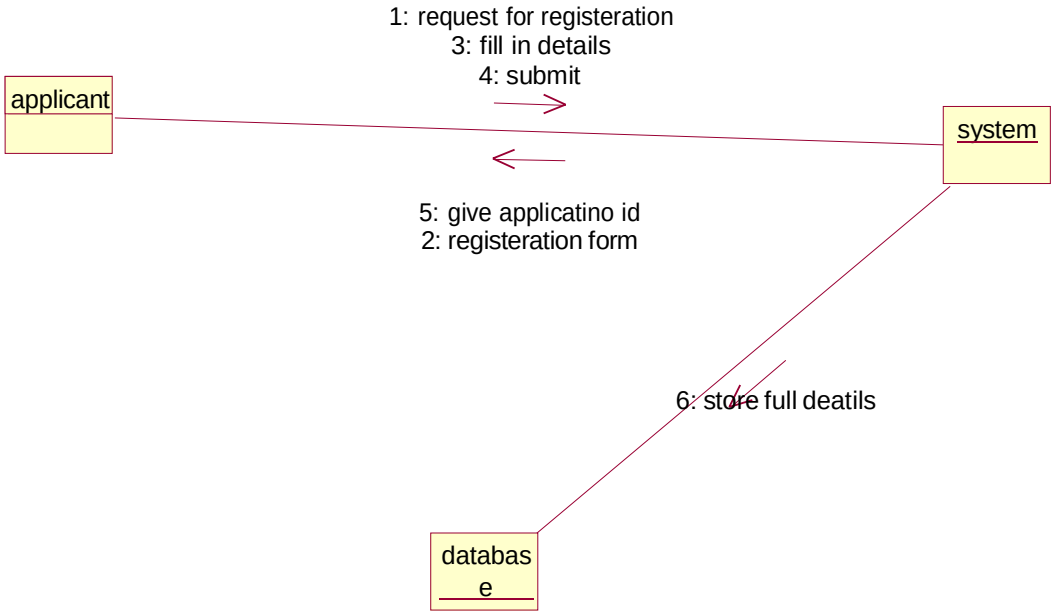
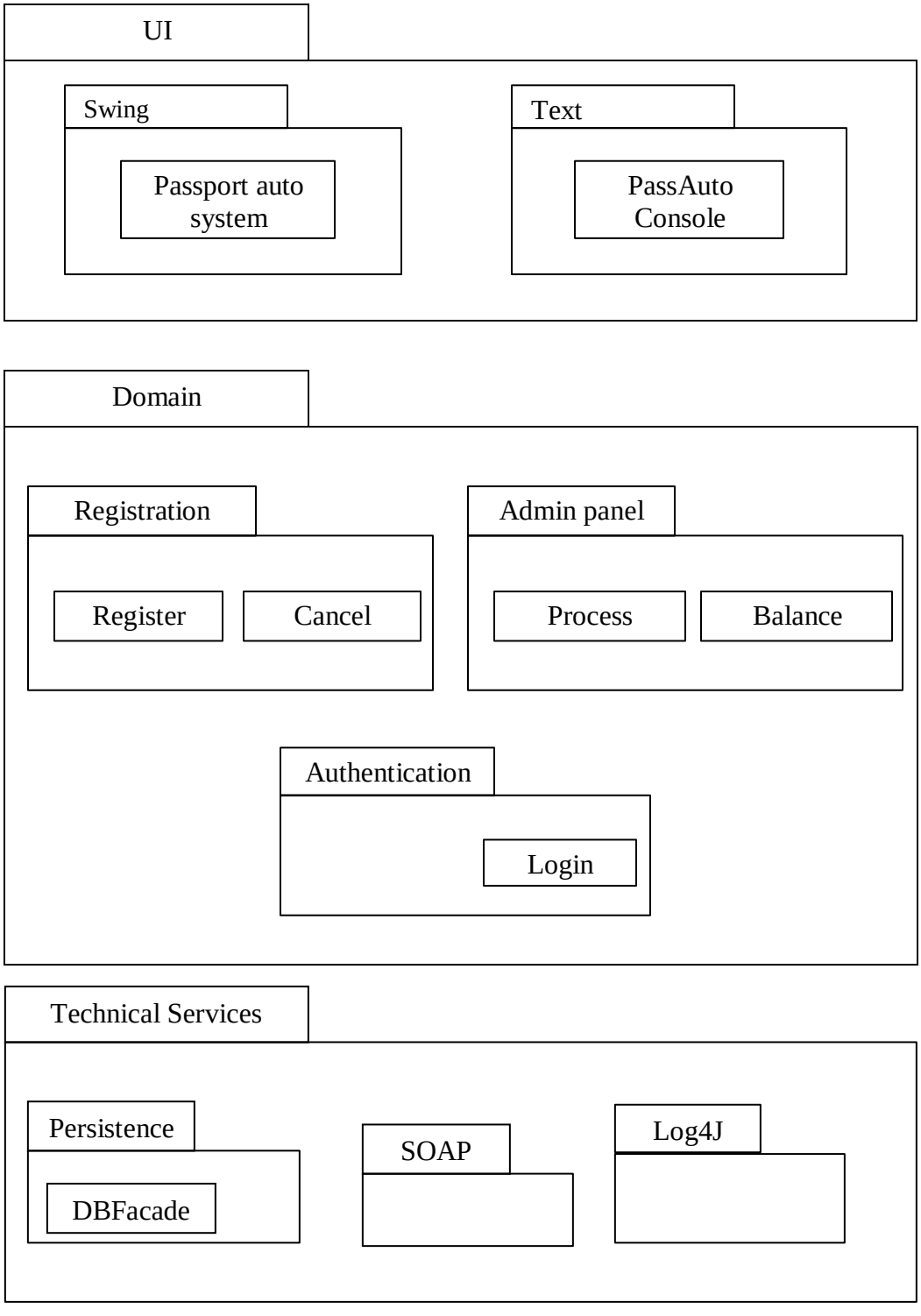


Fig.6.6.COLLABORATION DIAGRAM FOR NEW REGISTRATION

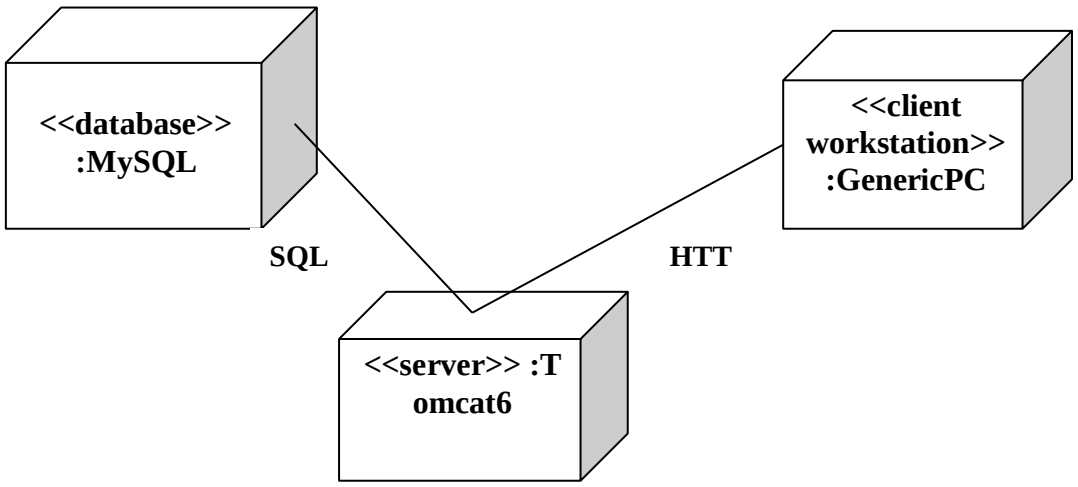
The diagrams show the applicant request the system for registration and the system provide the register form and applicant fill the form and submit and the system give the applicant id. The database stores the full details.

(VII) PARTIAL LAYERD LOGICAL ARCHITECTURE DIAGRAM



(VIII) DEPLOYMENT DIAGRAM AND COMPONENT DIAGRAM

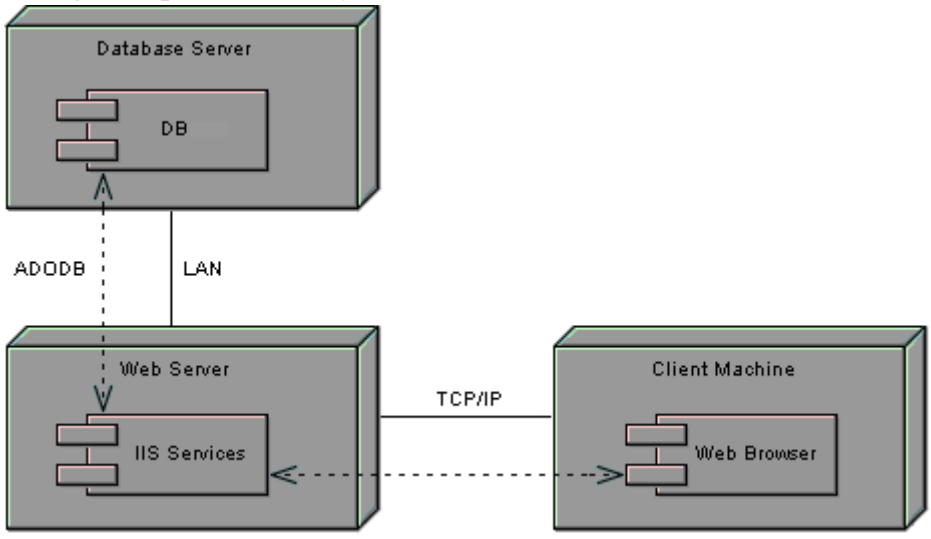
Deployment diagrams are used to visualize the topology of the physical components of a system where the software components are deployed.



DEPLOYMENT DIAGRAM

COMPONENT DIAGRAM

Component diagrams are used to visualize the organization and relationship among components in system



RESULT:

Thus the mini project for passport automation system has been successfully executed and codes are generated.

EXPERIMENT : 13 (b)

BOOK BANK SYSTEM

AIM:

To create a system to perform book bank operation

(I) PROBLEM STATEMENT:

A Book Bank lends books and magazines to member, who is registered in the system. Also it handles the purchase of new titles for the Book Bank. Popular titles are brought into multiple copies. Old books and magazines are removed when they are out or date or poor in condition. A member can reserve a book or magazine that is not currently available in the book bank, so that when it is returned or purchased by the book bank, that person is notified. The book bank can easily create, replace and delete information about the tiles, members, loans and reservations from the system.

(II) SOFTWARE REQUIREMENTS SPECIFICATION:

2.1SOFTWARE INTERFACE

- **Front End Client** - The Student and Librarian online interface is built using JSP and HTML. The Librarians local interface is built using Java.
- **Web Server** - Glassfish application server (Oracle Corporation).
- **Back End** - Oracle database

2.2HARDWARE INTERFACE

The server is directly connected to the client systems. The client systems have access to the database in the server.

(III)USE-CASE DIAGRAM:

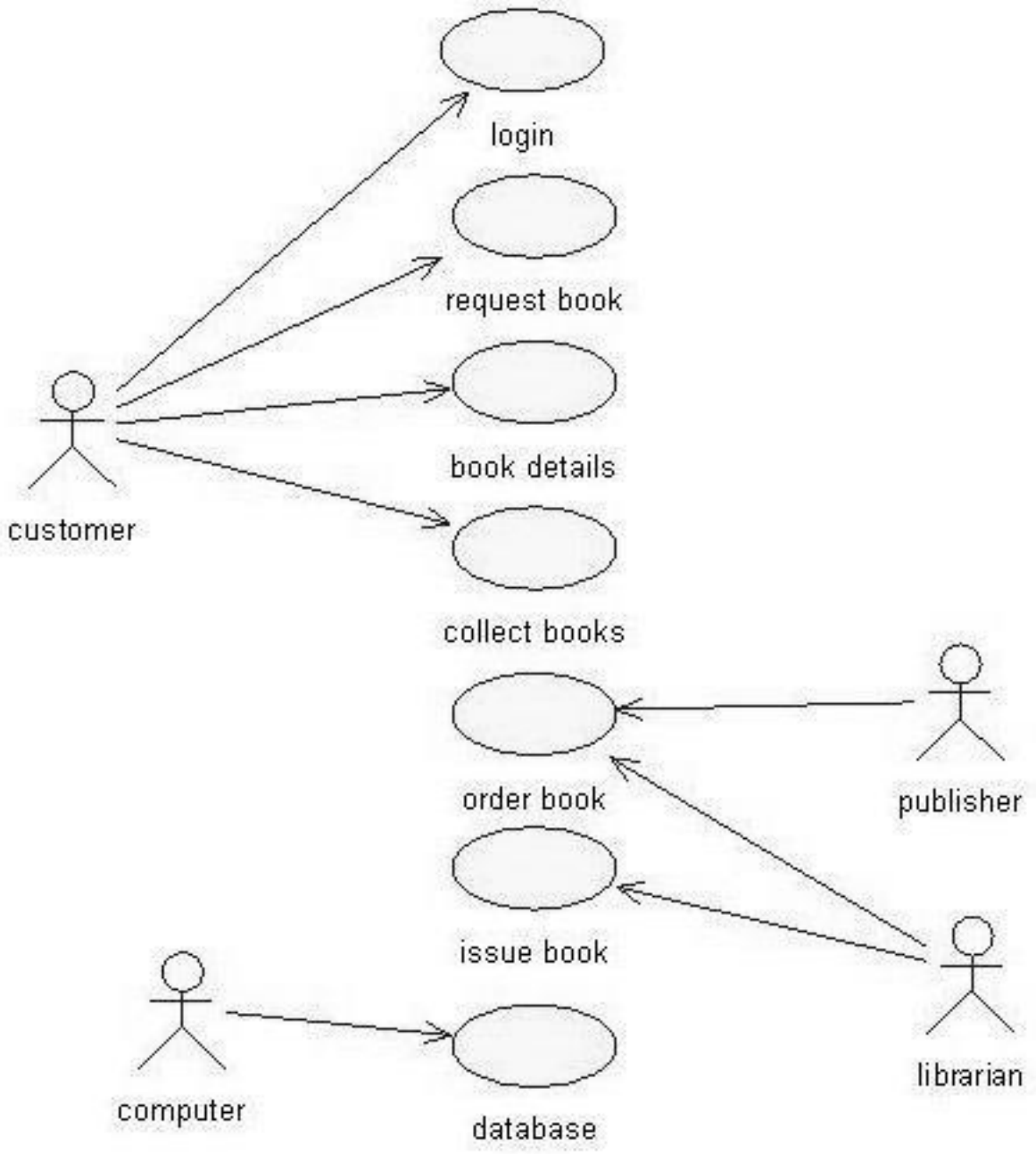


Fig 3. USE-CASE DIAGRAM FOR BOOK BANK SYSTEM

(IV) ACTIVITY DIAGRAM:

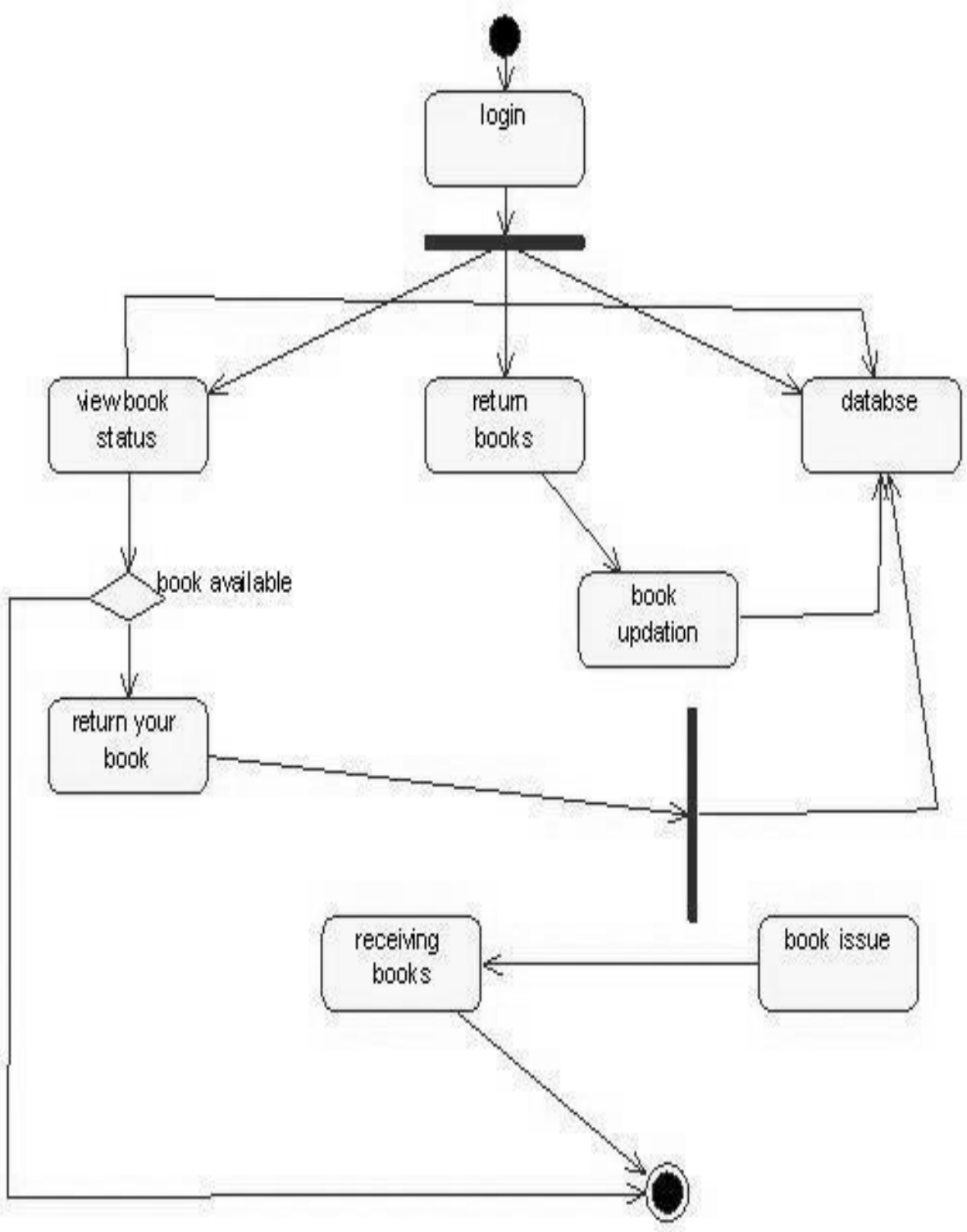


Fig.4. ACTIVITY DIAGRAM

(V) CLASS DIAGRAM:

The class diagram, also referred to as object modeling is the main static analysis diagram. The main task of object modeling is to graphically show what each object will do in the problem domain. The problem domain describes the structure and the relationships among objects.

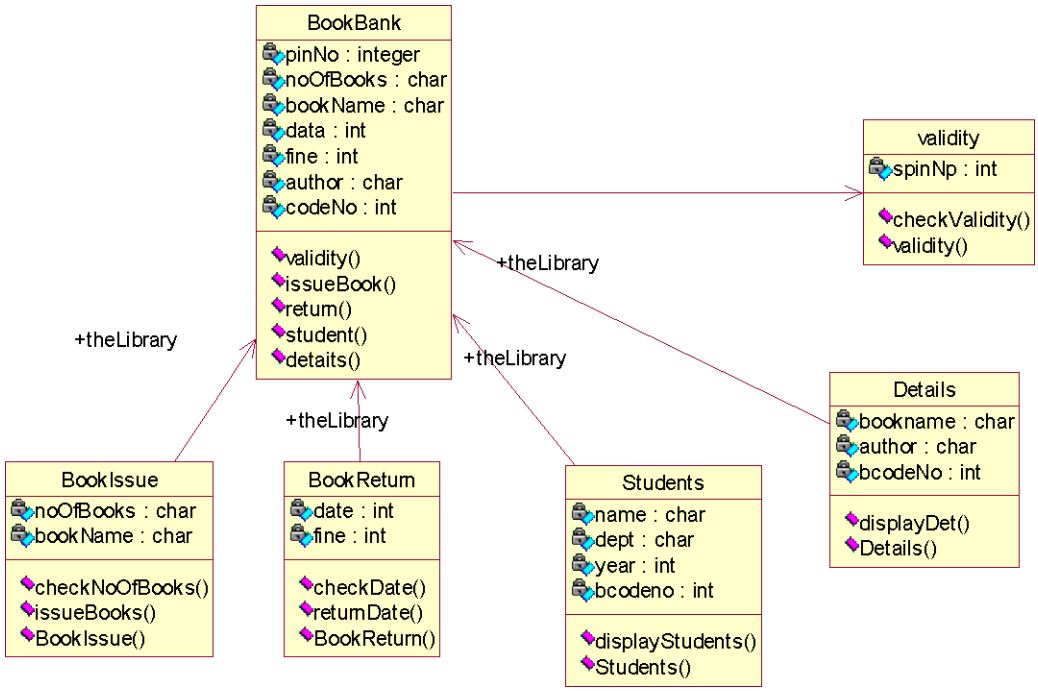


Fig.5. CLASS DIAGRAM FOR BOOK BANK SYSTEM

(VI) 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.

An event also is considered to be any action by an object that sends information. The event line represents a message sent from one object to another, in which the “form” object is requesting an operation be performed

by the “to” object. The “to” object performs the operation using a method that the class contains.

It is also represented by the order in which things occur and how the objects in the system send message to one another.

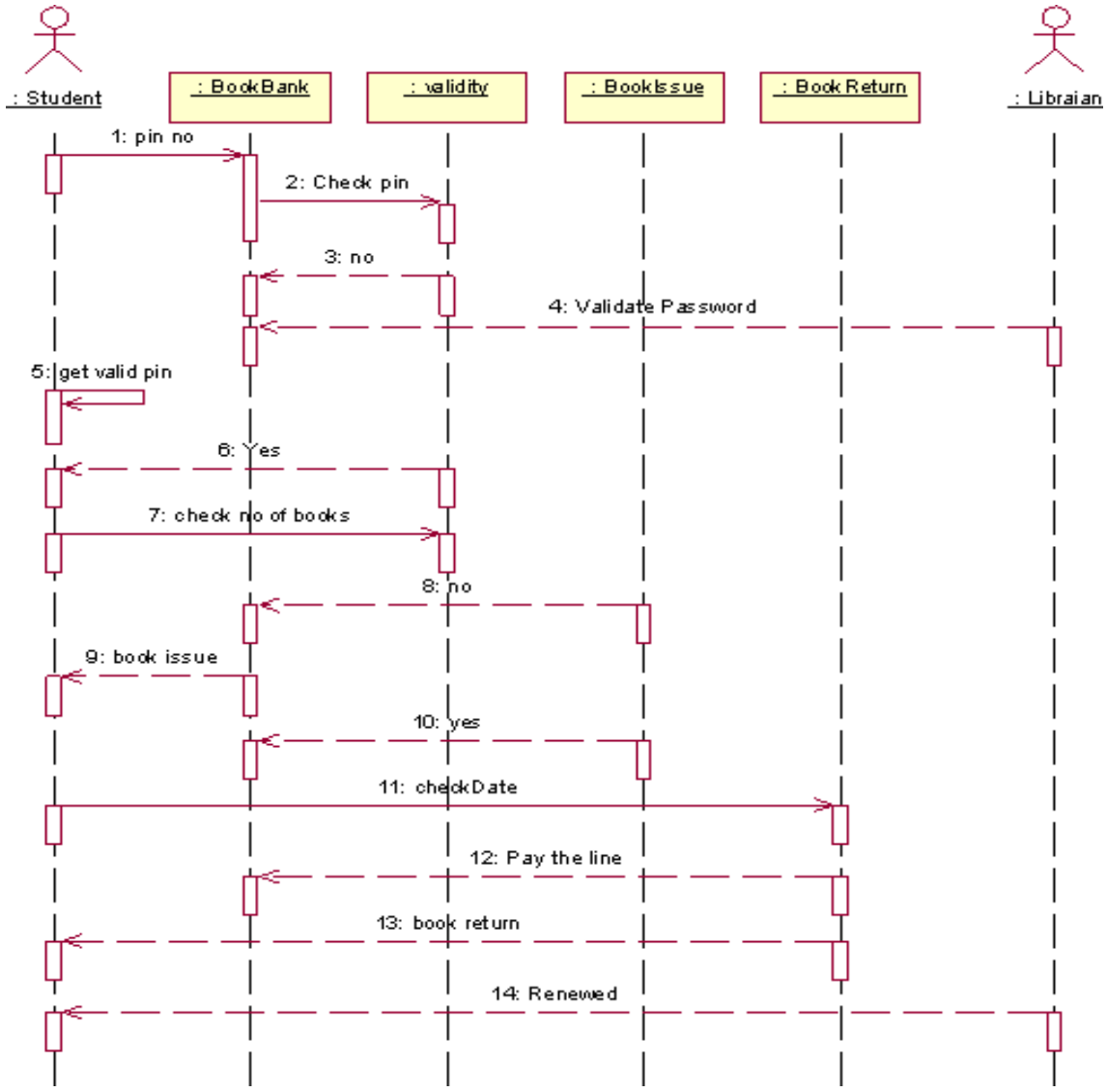


Fig. 6.1. SEQUENCE DIAGRAM FOR DEPOSIT PROCESS

The diagrams show the pin no is entered and check the pin .Get no and validate password check the condition based on condition book issue and return are done. Pay the online and renewed.

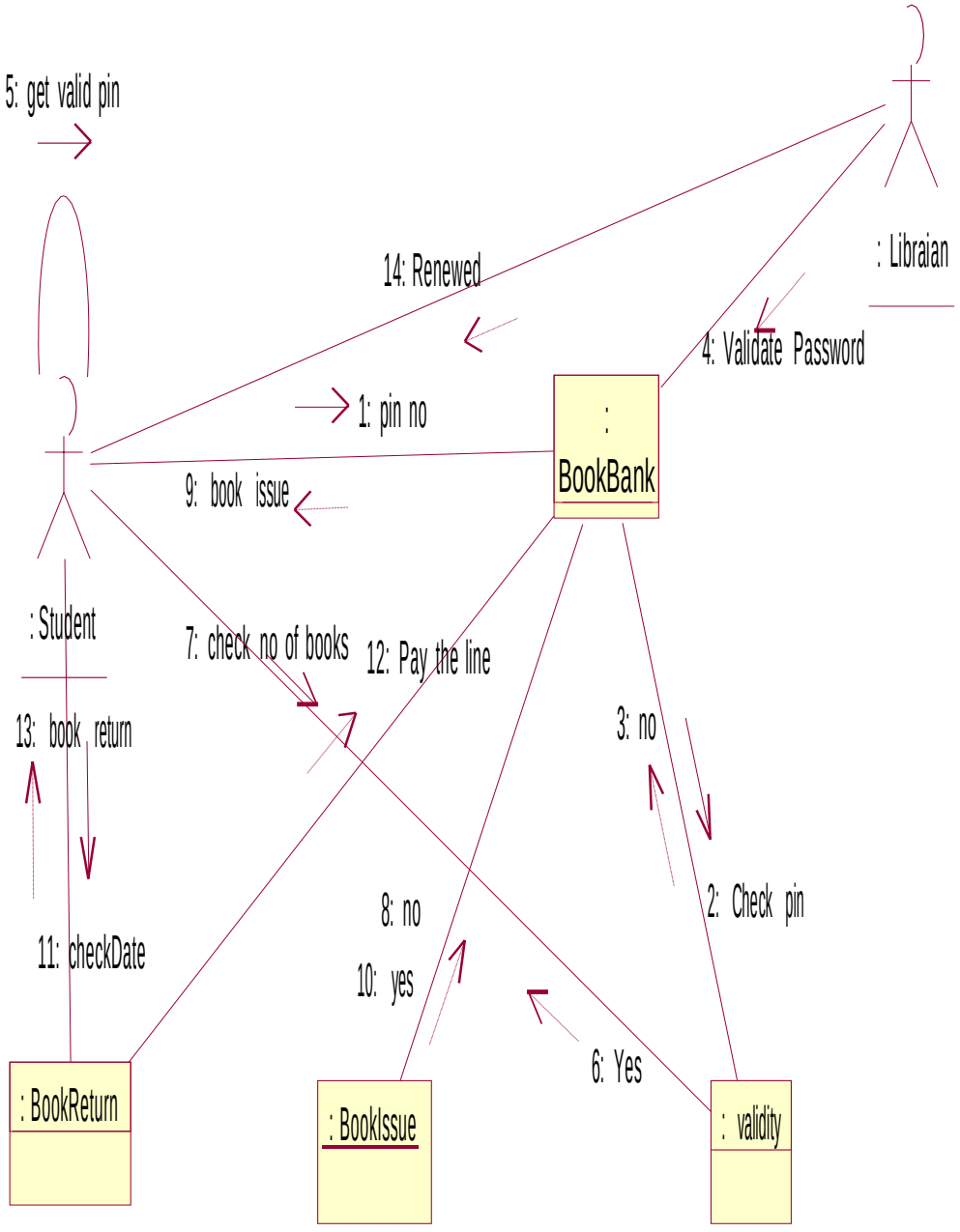
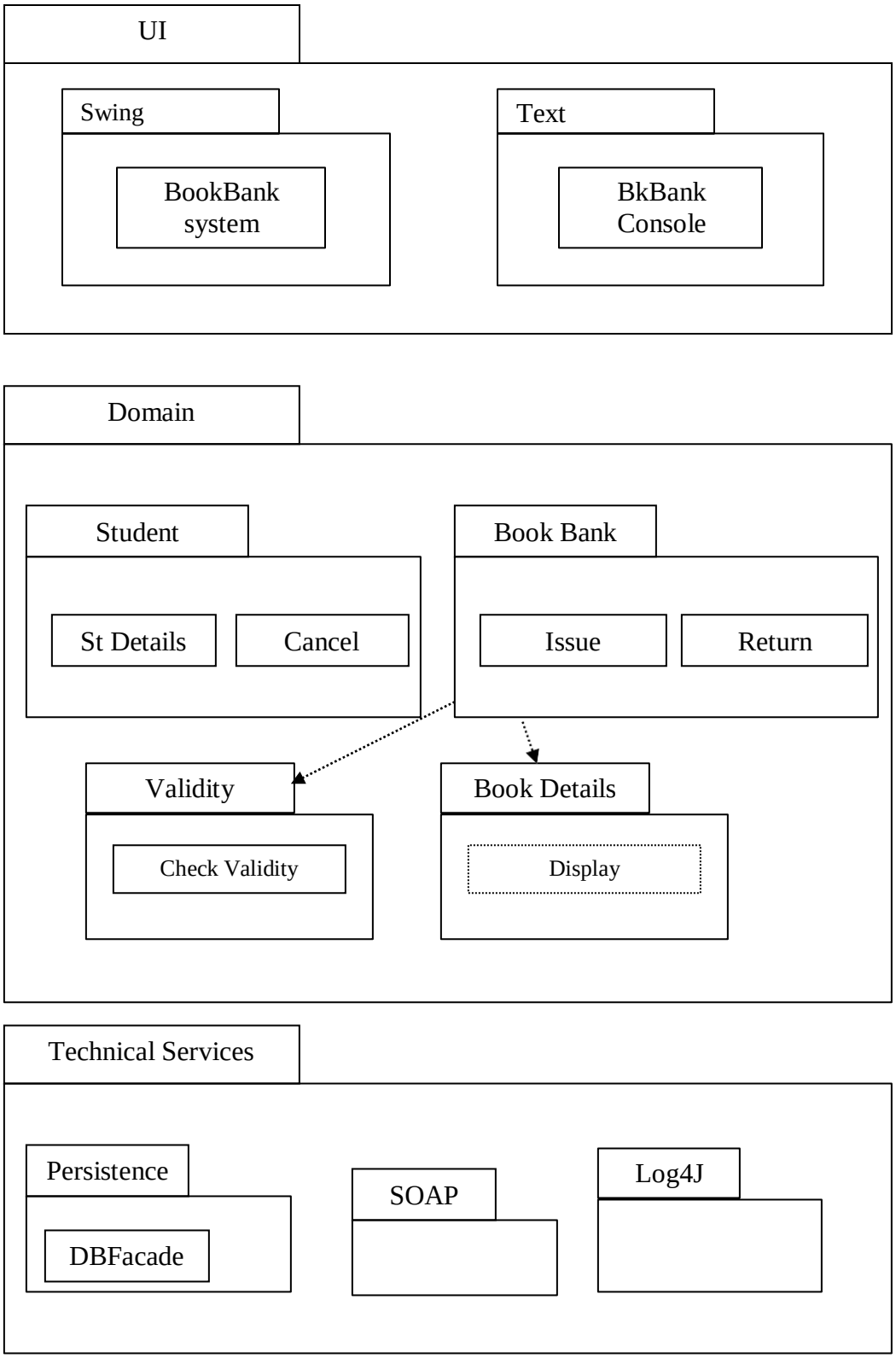


Fig. 6.2. COLLABORATION DIAGRAM FOR DEPOSIT PROCESS

(VII) PARTIAL LAYERD LOGICAL ARCHITECTURE DIAGRAM:



(VIII) DEPLOYMENT DIAGRAM AND COMPONENT DIAGRAM

Deployment diagrams are used to visualize the topology of the physical components of a system where the software components are deployed.

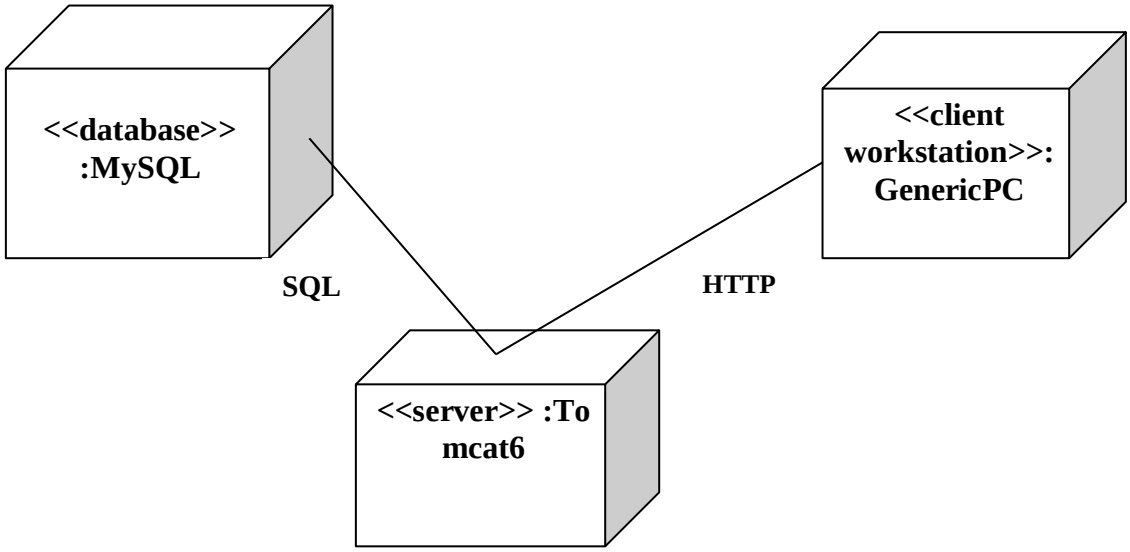


Fig.8.1.DEPLOYMENT DIAGRAM

COMPONENT DIAGRAM

Component diagrams are used to visualize the organization and relationships

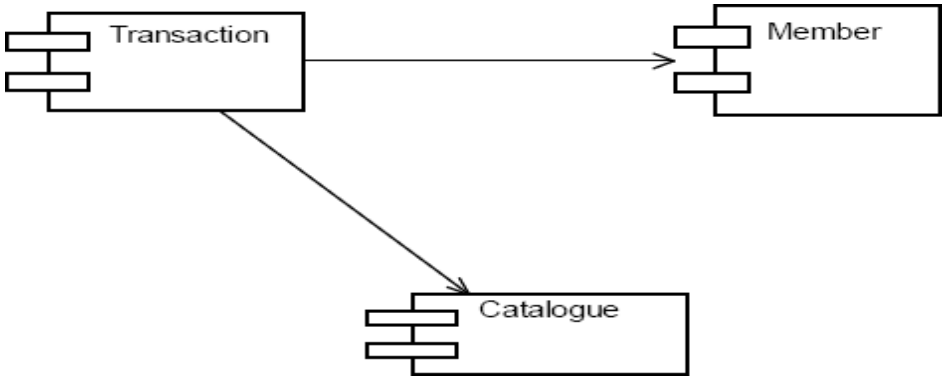


Fig.8.2.COMPONENT DIAGRAM

RESULT:

Thus the mini project for Book Bank System has been successfully executed and codes are generated.

Ex no:4

EXPERIMENT : 13 (c)

Date:

EXAM REGISTRATION SYSTEM

AIM:

To create a system to perform the Exam Registration system

(I) PROBLEM STATEMENT:

Exam Registration system.is used in the effective dispatch of registration form to all of the students. 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, reg.no etc.,) filled by the student whose testament is verified for its genuineness by the Exam Registration System with respect to the already existing information in the database.

(II) SOFTWARE REQUIREMENT SPECIFICATION:

2.1 SOFTWARE INTERFACE

- **Front End Client** - The student and Controller online interface is built using
  JSP and HTML. The Exam Controller's local interface is built using Java.
- **Web Server** - Glassfish application server(SQlCorporation).
- **Back End** - SQL database.

2.2 HARDWARE INTERFACE

The server is directly connected to the client systems. The client systems have
access to the database in the server.

(III) USECASE DIAGRAM:

The Exam Registration use cases in our system are:

1. Login
2. View exam details
3. View fees details

4. Pay fee
5. Display details
6. Logout

USECASE DIAGRAM :

Fig. 3.USECASE DIAGRAM FOR EXAM REGISTRATION SYSTEM

(IV) ACTIVITY DIAGRAM:

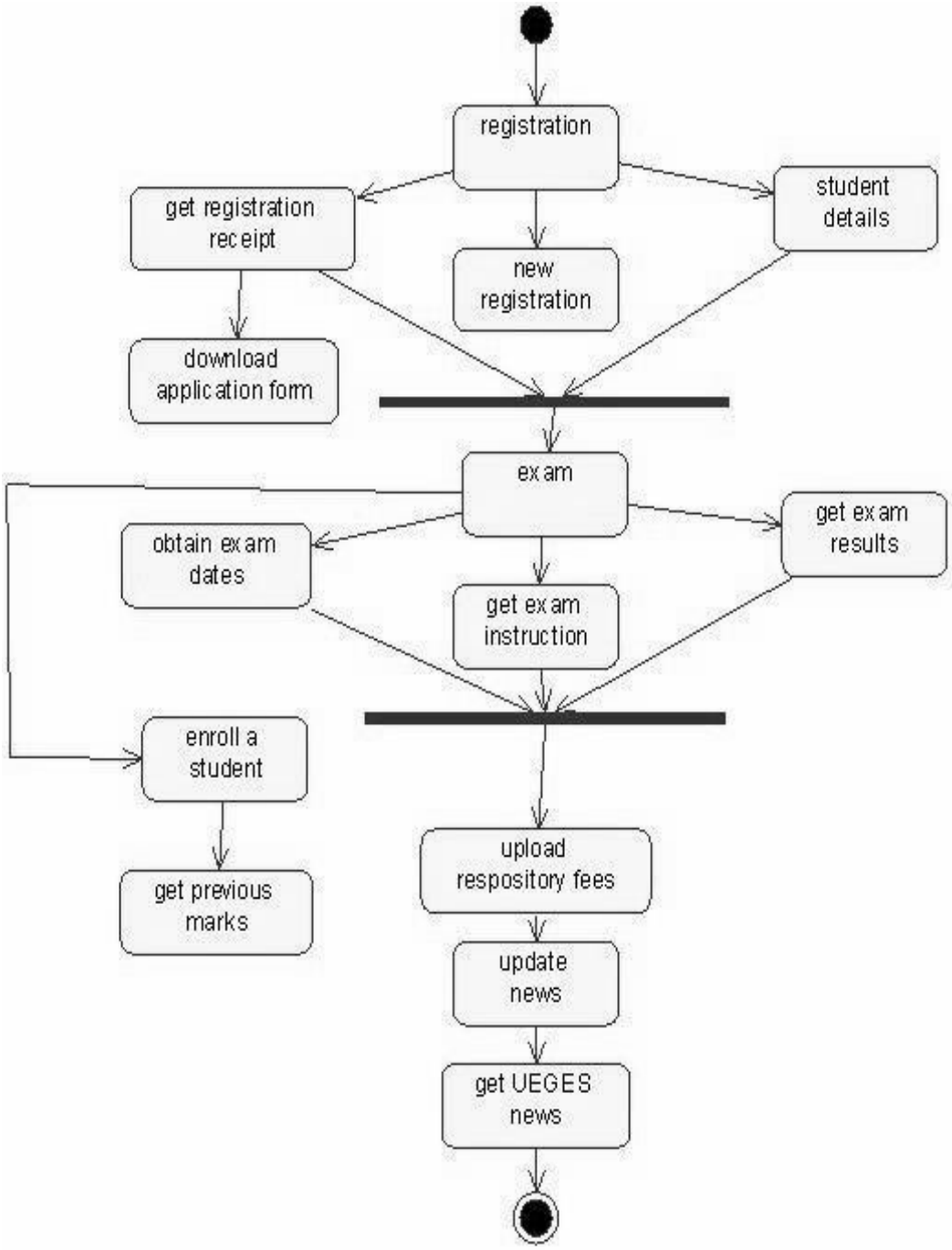


Fig. 4.USECASE DIAGRAM FOR EXAM REGISTRATION SYSTEM

(V)CLASS DIAGRAM:

The class diagram, also referred to as object modeling is the main static analysis diagram. The main task of object modeling is to graphically show what each object will do in the problem domain. The problem domain describes the structure and the relationships among objects.

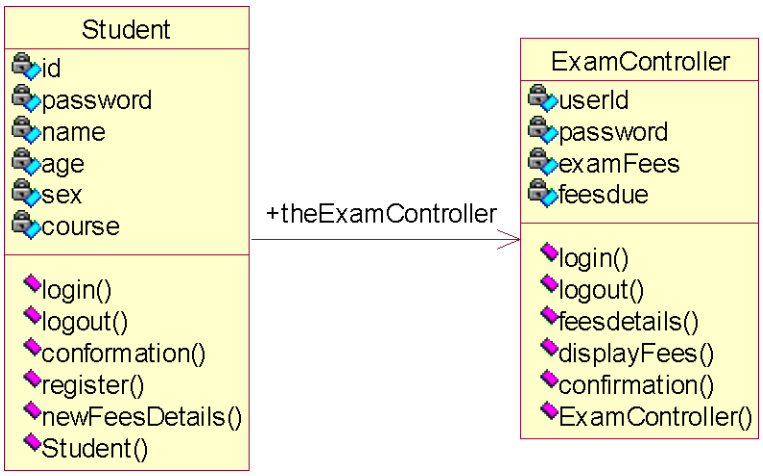


Fig.5. CLASS DIAGRAM FOR EXAM REGISTRATION SYSTEM

(VI)INTERACTION 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.

An event also is considered to be any action by an object that sends information. The event line represents a message sent from one object to another

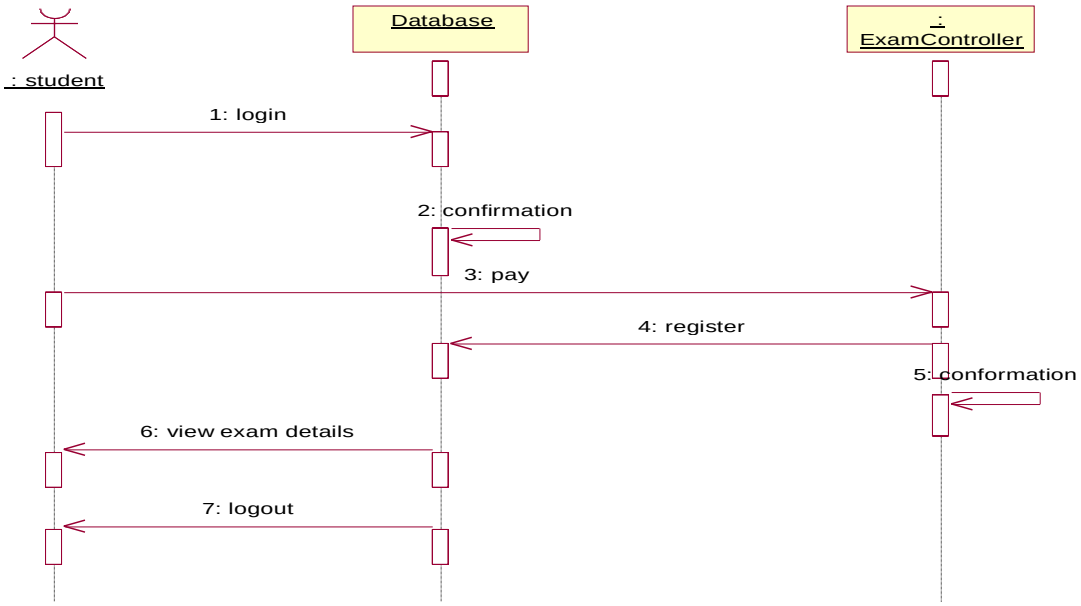


Fig. 6.1. SEQUENCE DIAGRAM FOR REGISTRATION SYSTEM

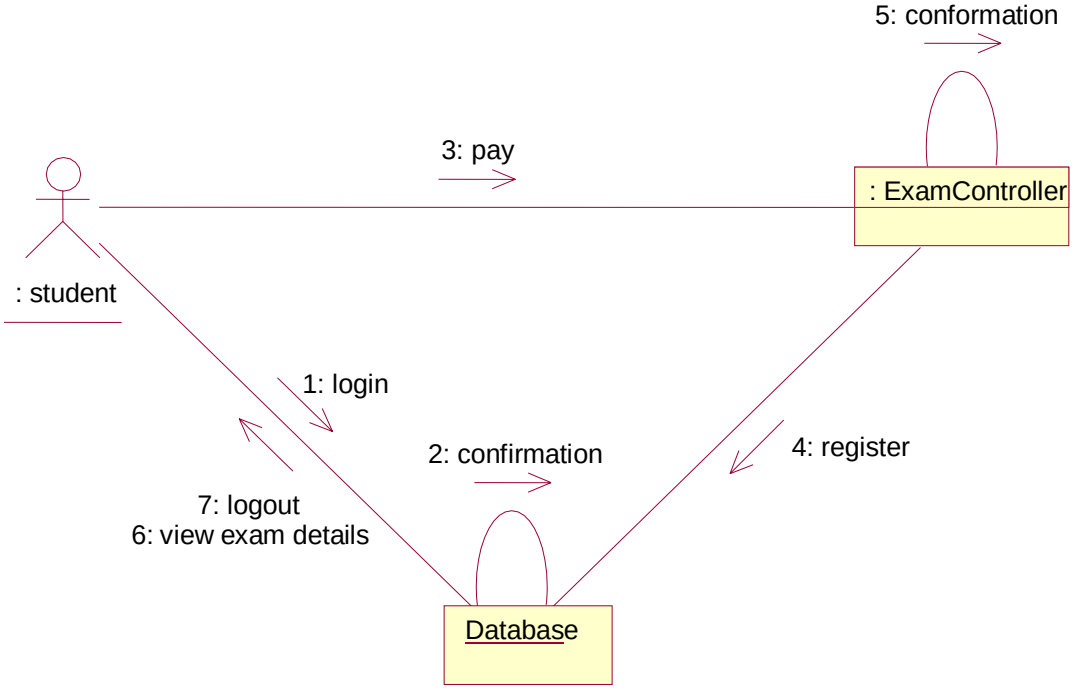
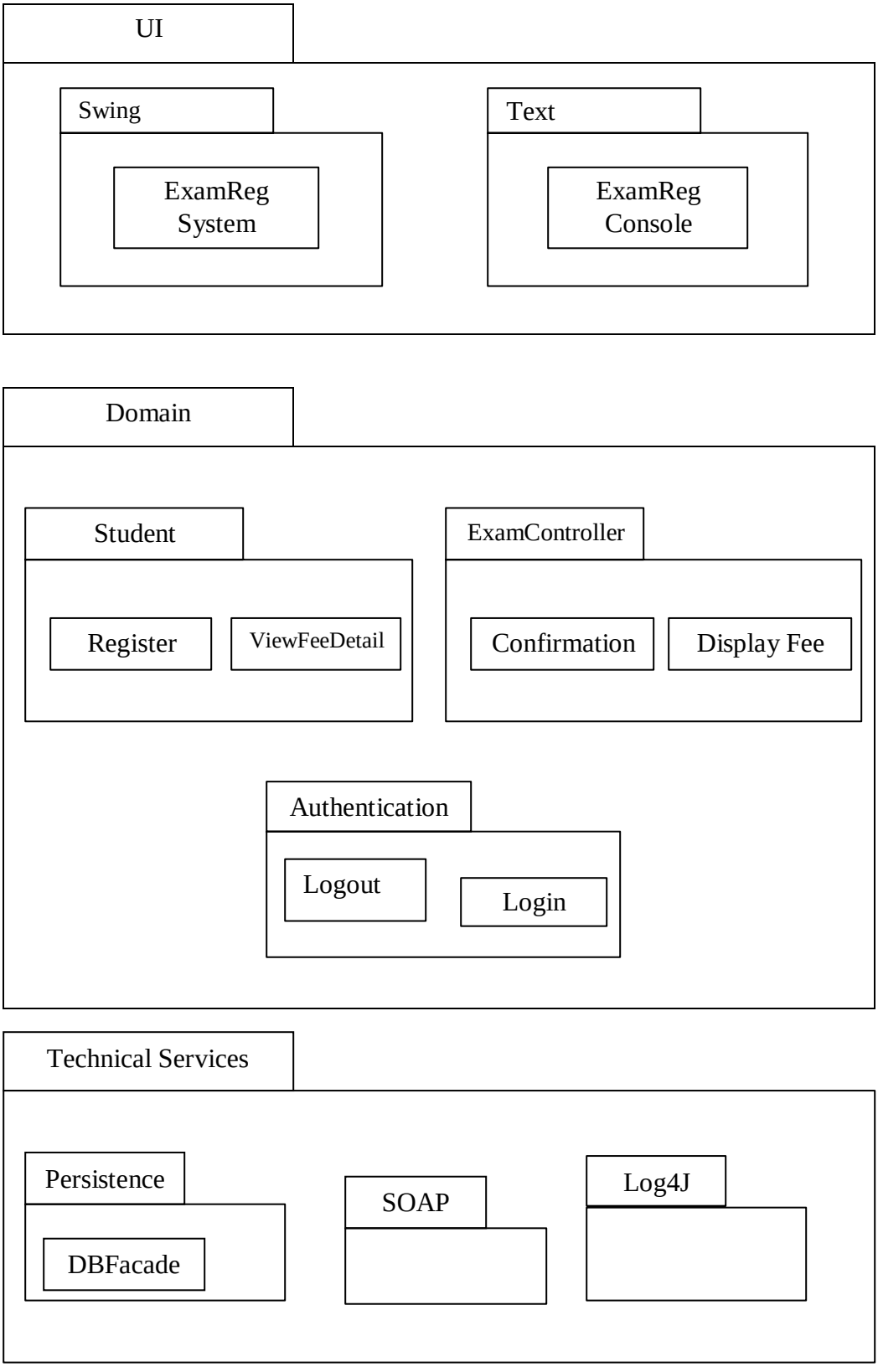


Fig. 6.2. COLLABORATION DIAGRAM FOR REGISTRATION SYSTEM

(VII) PARTIAL LAYERD LOGICAL ARCHITECTURE DIAGRAM:



(VIII) DEPLOYMENT DIAGRAM AND COMPONENT DIAGRAM

Deployment diagrams are used to visualize the topology of the physical components of a system where the software components are deployed.

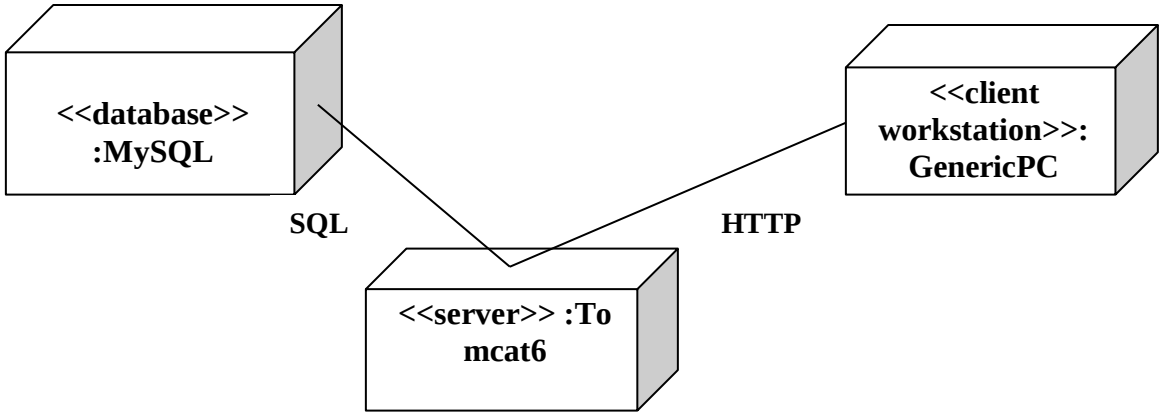


Fig.7.1.DEPLOYMENT DIAGRAM

COMPONENT DIAGRAM

Component diagrams are used to visualize the organization and relationships among components in a system.

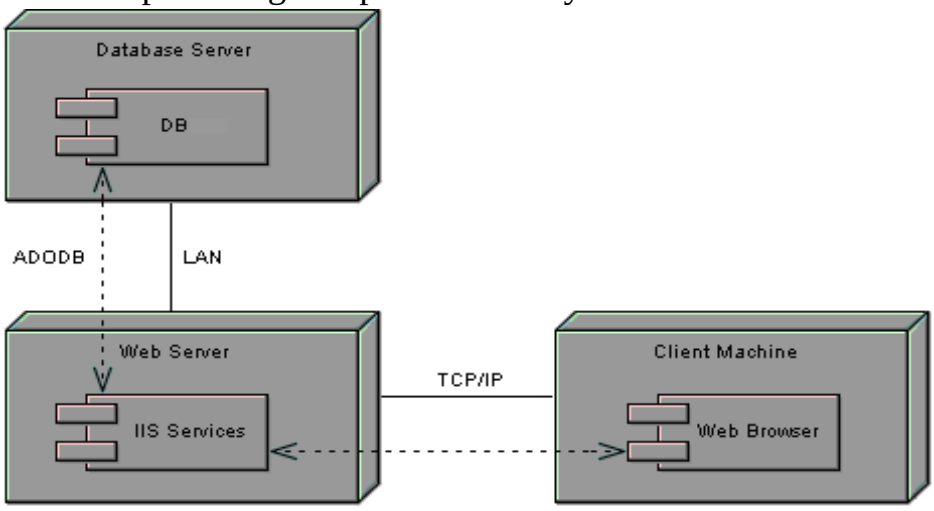


Fig.7.2.COMPONENT DIAGRAM

RESULT:

Thus the mini project for Exam Registration system has been successfully executed and codes are generated.

EXPERIMENT : 13 (d)

STOCK MAINTENANCE

AIM:

To create a system to perform the Stock maintenance

(I) PROBLEM STATEMENT

The stock maintenance system must take care of sales information of the company and must analyze the potential of the trade. It maintains the number of items that are added or removed.The sales person initiates this Use case. The sales person is allowed to update information and view the database.

(II) SOFTWARE REQUIREMENT SPECIFICATION

1.1 PURPOSE

The entire process of Stock maintenanceis done in a manual manner Considering the fact that the number of customers for purchase is increasing every year, a maintenance system is essential to meet the demand. So this system uses several programming and database techniques to elucidate the work involved in this process.

1.2 SCOPE

• The System provides an interface to the customer where they can fill in orders for the item needed.

• The sales person is concerned with the issue of items and can use this system.

• Provide a communication platform between the customer and the sales person.

1.3 TOOLS TO BE USED

• Eclipse IDE (Integrated Development Environment)

• Rational Rose tool (for developing UML Patterns)

(III) USE CASE DIAGRAM

The functionality of a system can be described in a number of different use-cases, each of which represents a specific flow of events in a system. It is a graph of actors, a set of use-cases enclosed in a boundary, communication, associations between the actors and the use-cases, and generalizationamong the use-cases

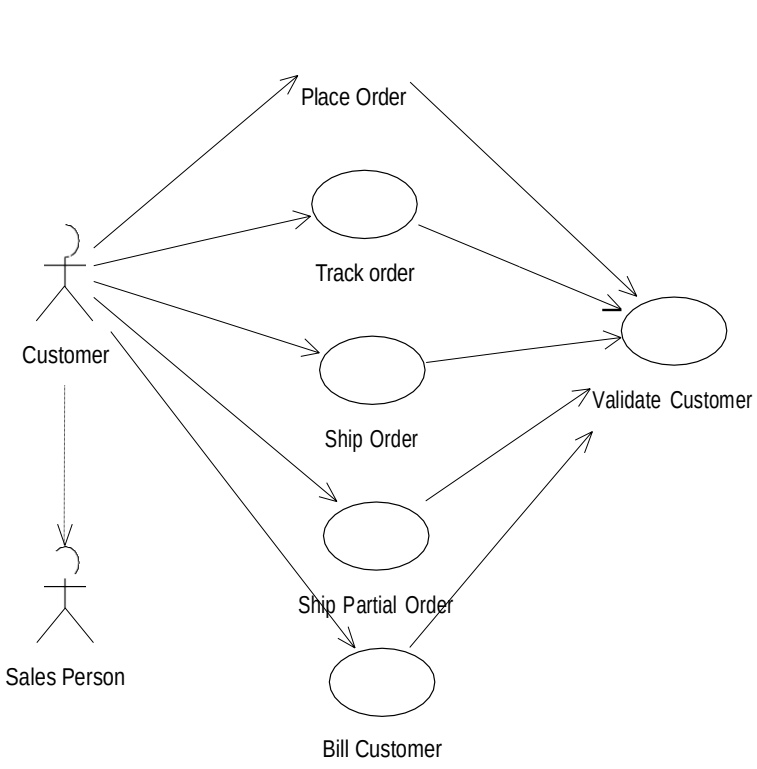


Fig.3. USE CASE DIAGRAM

(IV) ACTIVITY DIAGRAM

It shows organization and their dependence among the set of components. These diagrams are particularly useful in connection with workflow and in describing behavior that has a lot of parallel processing. An activity is a state of doing something: either a real-world process, or the execution of a software routine.

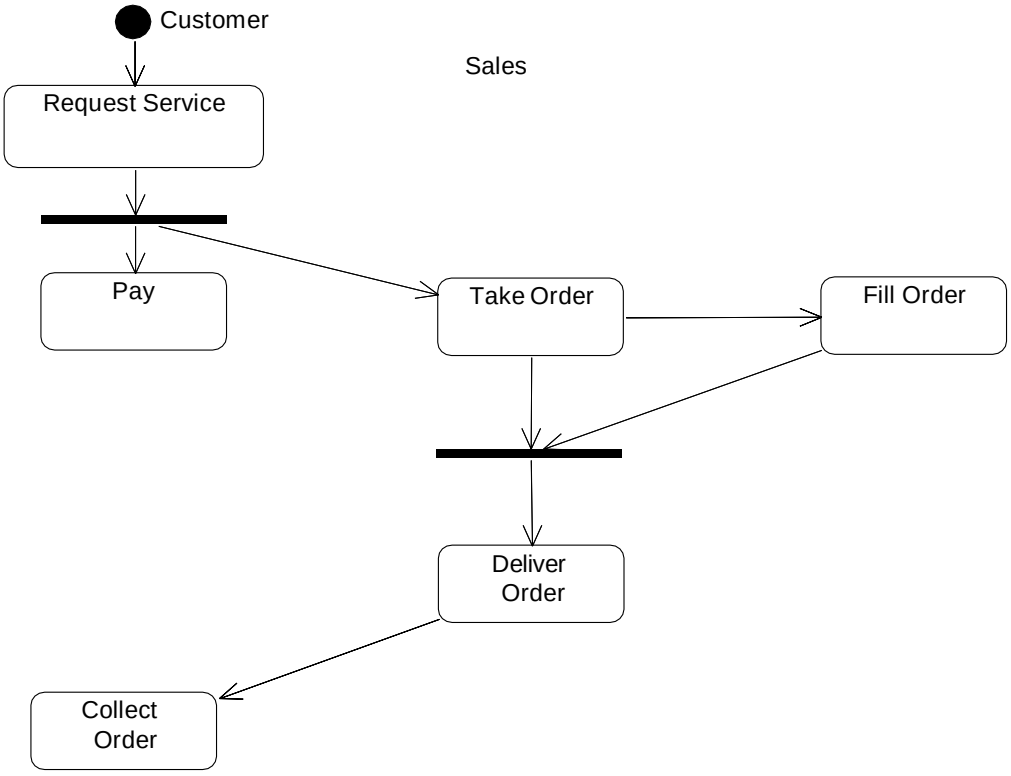


Fig.4. ACTIVITY DIAGRAM

(V) CLASS DIAGRAM

Description:

- A class diagram describes the type of objects in system and various kinds of relationships that exists among them.
- Class diagrams and collaboration diagrams are alternate representations of object models.

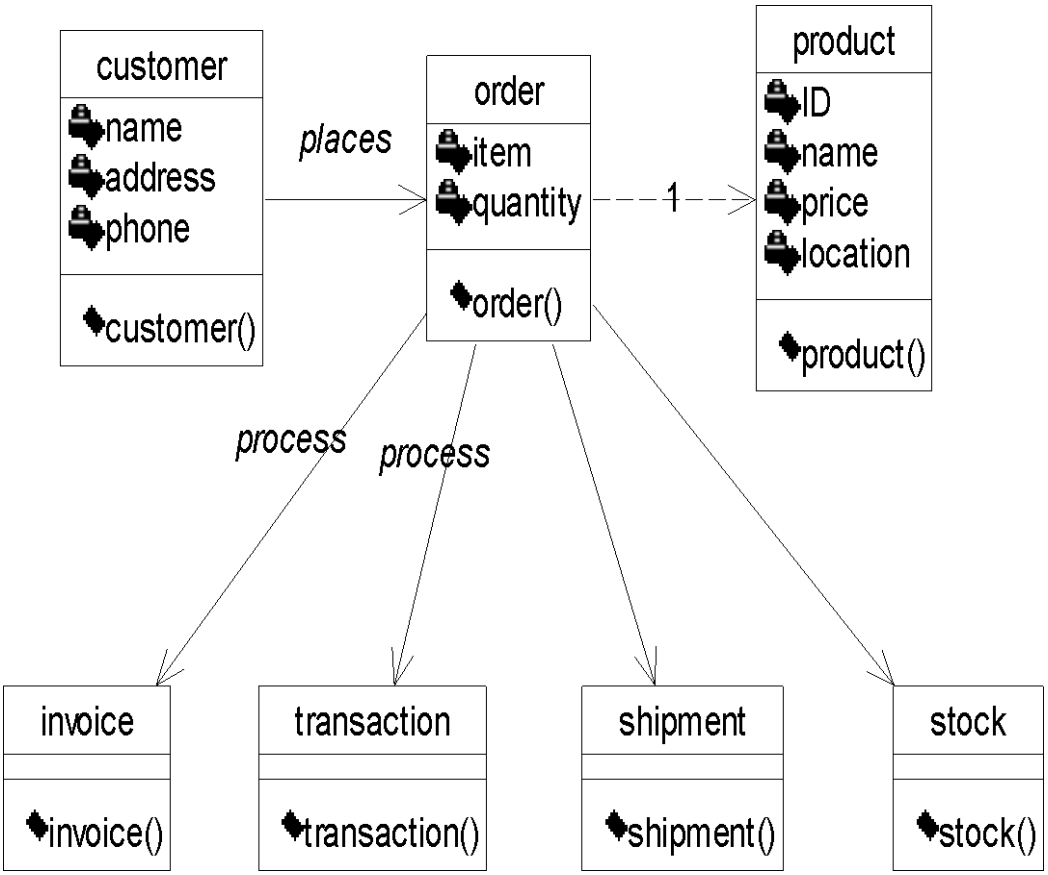


Fig.5. CLASS DIAGRAM

(VI) UML INTERACTION DIAGRAMS

It is the combination of sequence and collaboration diagram. It is used to depict the flow of events in the system over a timeline. The interaction diagram is a dynamic model which shows how the system behaves during dynamic execution.

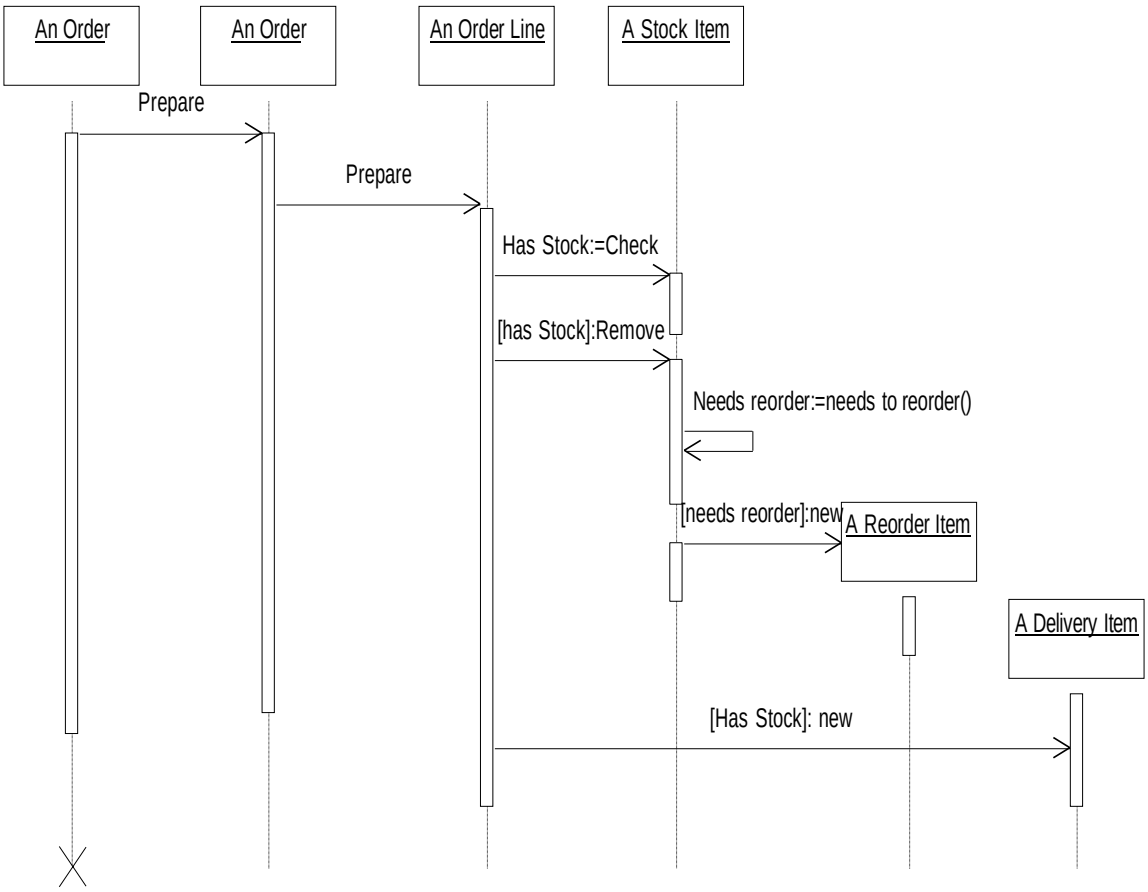


Fig.6.1 SEQUENCE DIAGRAM

COLLABORATION DIAGRAM

Collaboration diagram and sequence diagrams are alternate representations of an interaction. A collaboration diagram is an interaction diagram that shows the order of messages that implement an operation or a transaction.

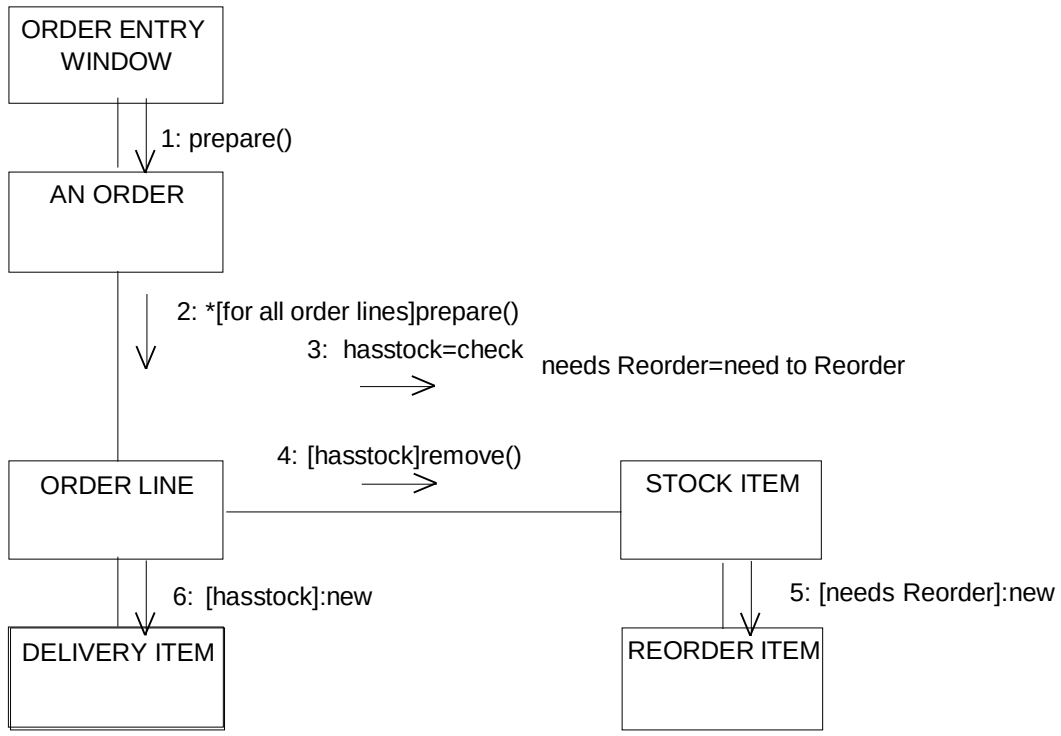
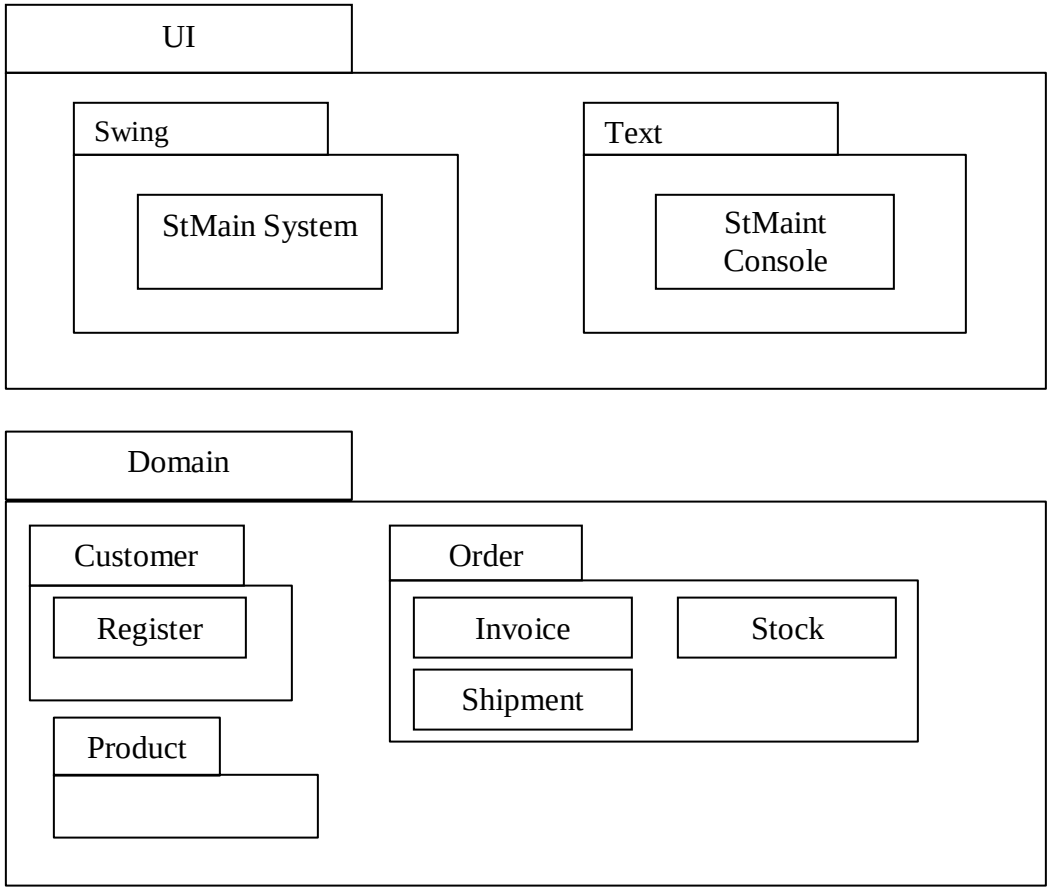


Fig.6.2 COLLABORATION DIAGRAM

(VII) PARTIAL LAYERD LOGICAL ARCHITECTURE DIAGRAM



(VIII) DEPLOYMENT DIAGRAM AND COMPONENT DIAGRAM

Deployment diagrams are used to visualize the topology of the physical components of a system where the software components are deployed.

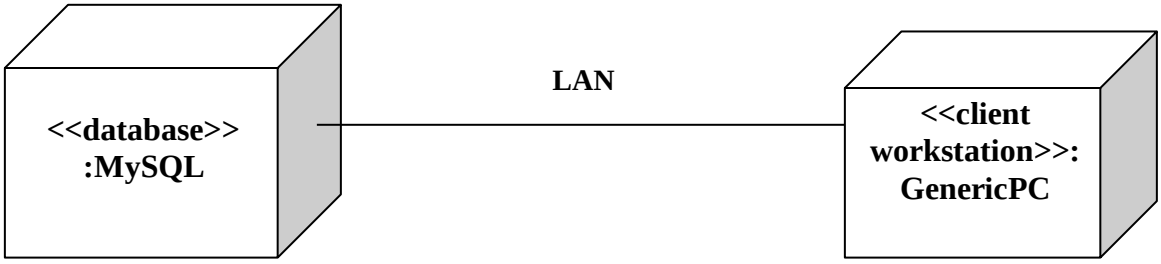


Fig.8.1.DEPLOYMENT DIAGRAM

Component Diagram

Component diagrams are used to visualize the organization and relationships among components in a system.

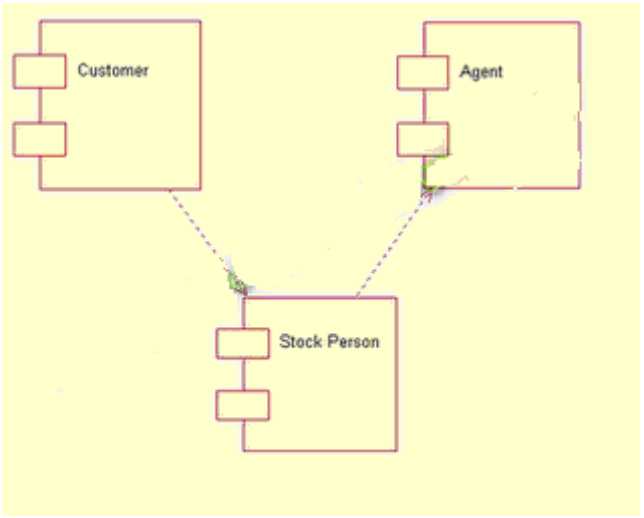


Fig.8.2.COMPONENT DIAGRAM

RESULT:

Thus the mini project for stock maintenance system has been successfully executed and codes are generated.

EXPERIMENT : 13 (e)

ONLINE COURSE RESERVATION SYSTEM

AIM

To design an object oriented model for course reservation system.

(I) PROBLEM STATEMENT

a. Whenever the student comes to join the course he/she should be provided with the list of course available in the college.

b. The system should maintain a list of professor who is teaching the course. At the end of the course the student must be provided with the certificate for the completion of the course.

(II) SYSTEM REQUIEMENT SPECIFICATION

OBJECTIVES

a. The main purpose of creating the document about the software is to know about the list of the requirement in the software project part of the project to be developed.

b. It specifies the requirement to develop a processing software part that completes the set of requirement.

SCOPE

a. In this specification, we define about the system requirements that are about from the functionality of the system.

b. It tells the users about the reliability defined in usecase specification

FUNCTIONALITY

Many members of the process line to check for its occurrences and transaction, we are have to carry over at sometimes

USABILITY

The user interface to make the transaction should be effectively

PERFORMANCE
It is the capability about which it can performed function for many user at sometimes efficiently (ie) without any ever occurrences

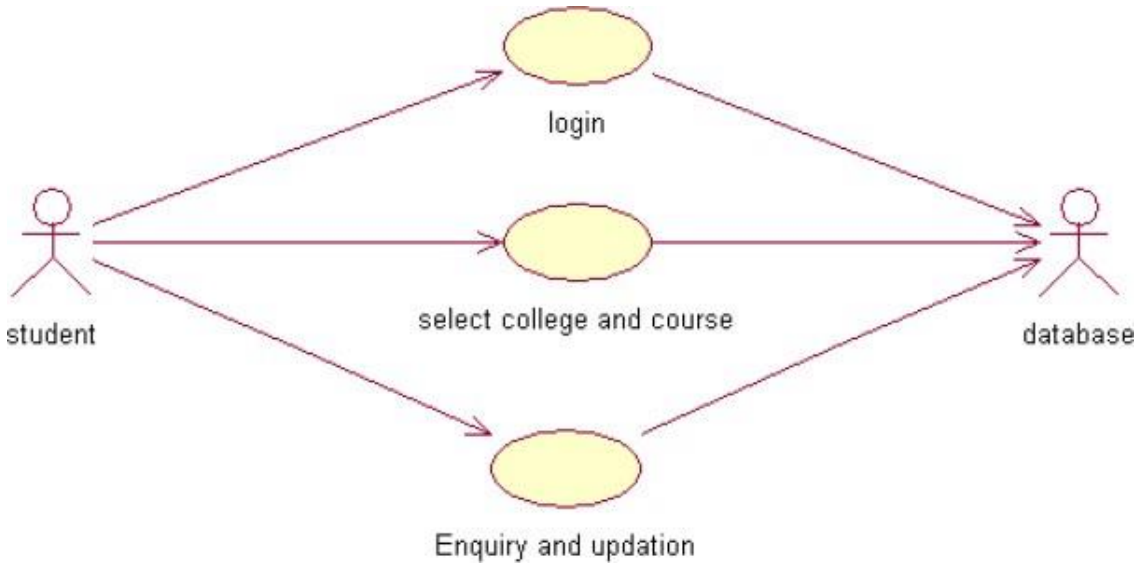
RELIABILITY
The system should be able to the user through the day to day transaction

(III) USERCASE DIAGRAM

a. Use case is a sequence of transaction in a system whose task is to yield result of measurable value to individual author of the system

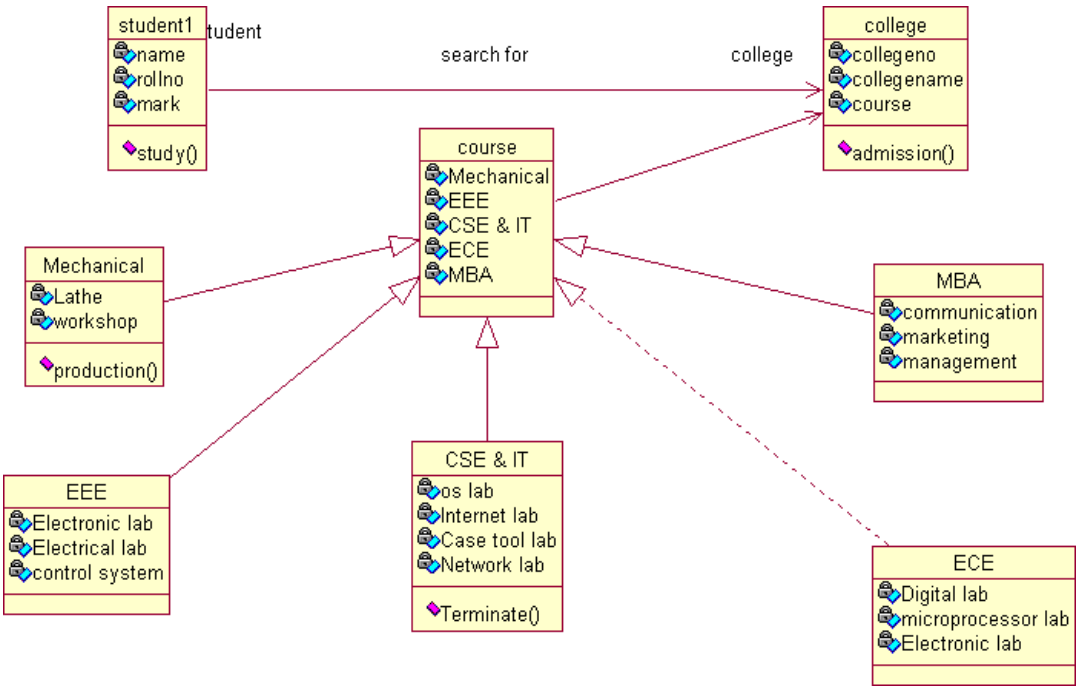
b. Use case is a set of scenarios together by a common user goal

c. A scenario is a sequence of step describing as interaction between a user and a system



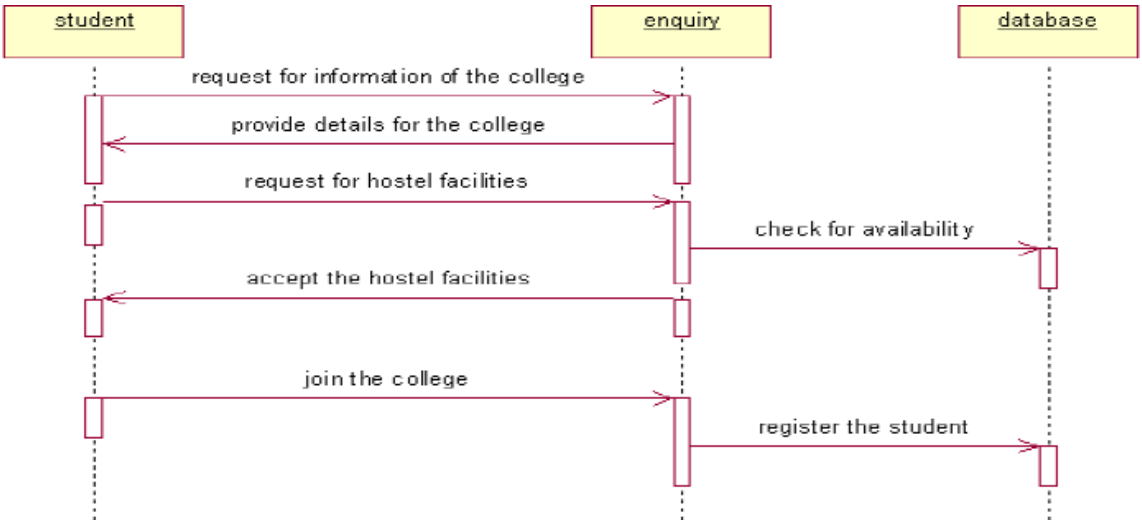
CLASS DIAGRAM:

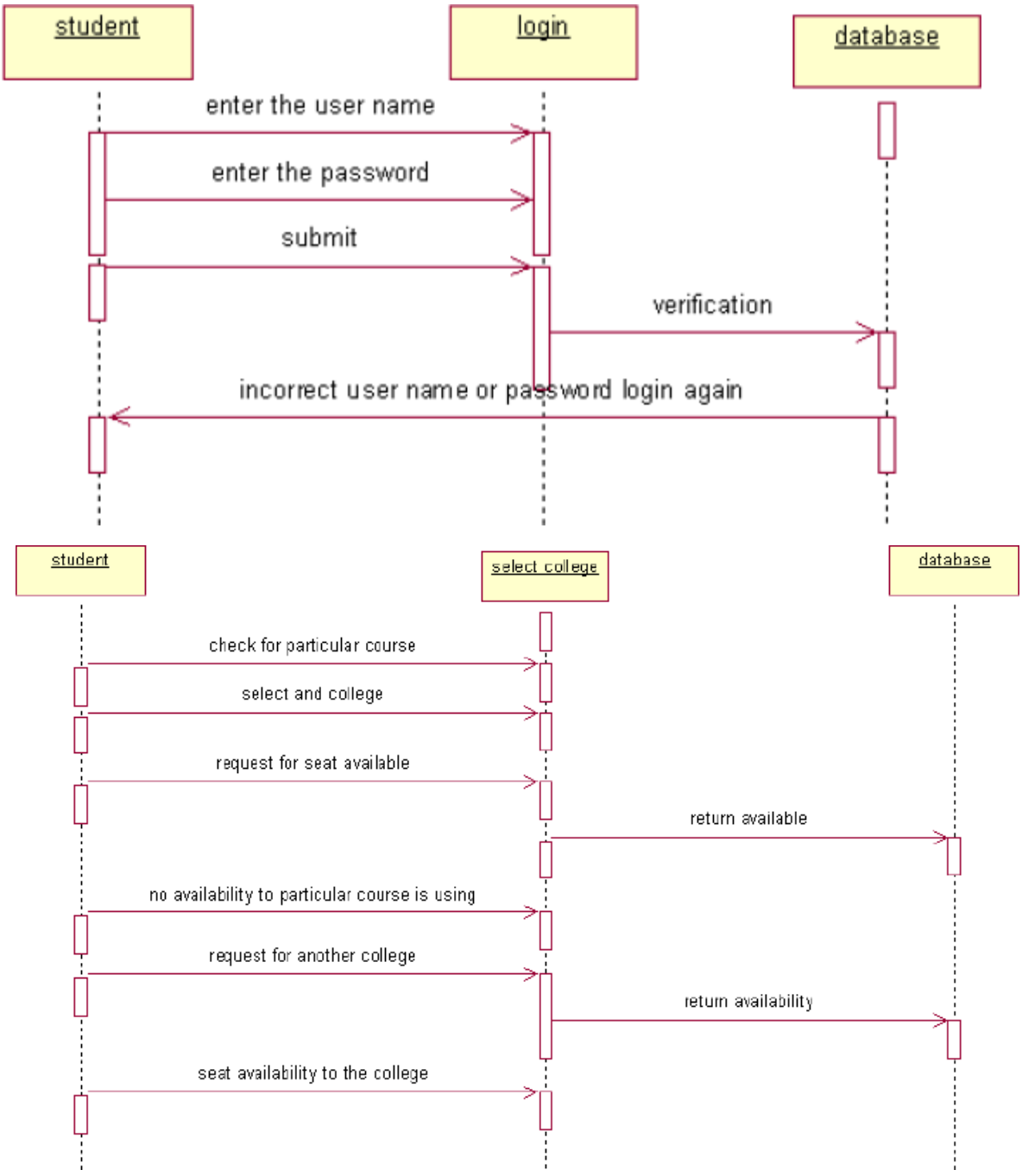
A class diagram describes the type of objectors in the system the various kinds of static relationship that exist among them.



SEQUENCE DIAGRAM

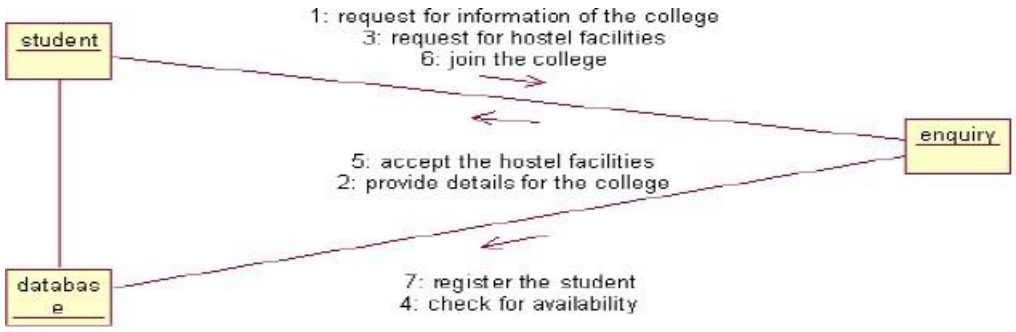
A sequence diagram is one that includes the object of the projects and tells the lifetimes and also various action performed between objects.

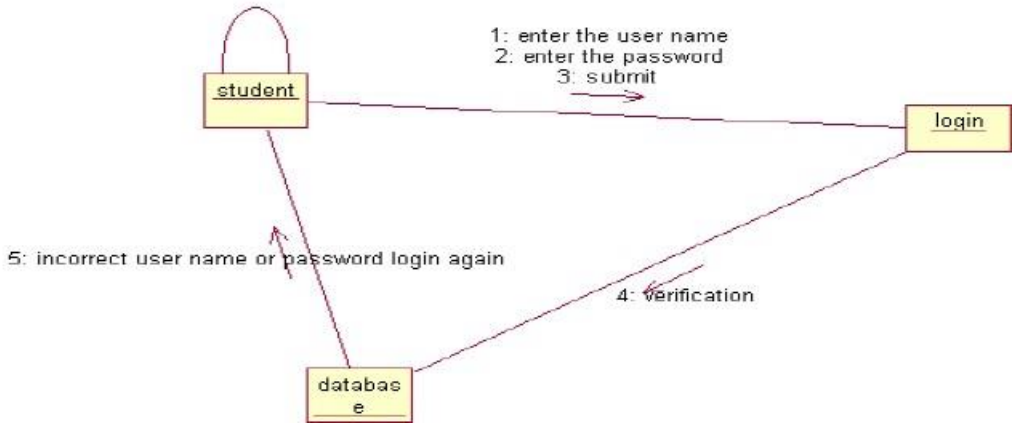


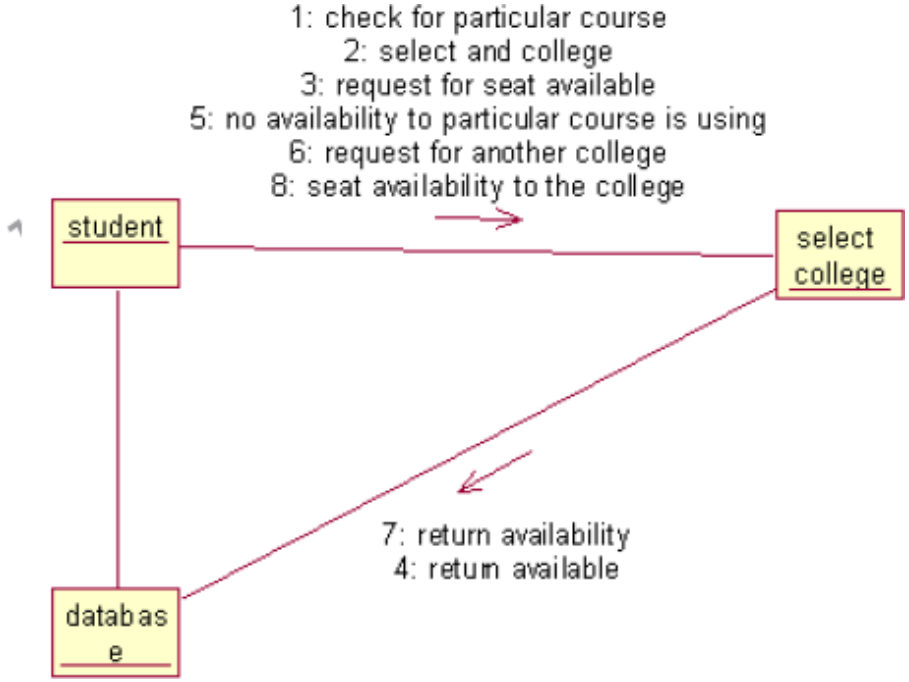


COLLOBORATIION DIAGRAM

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.

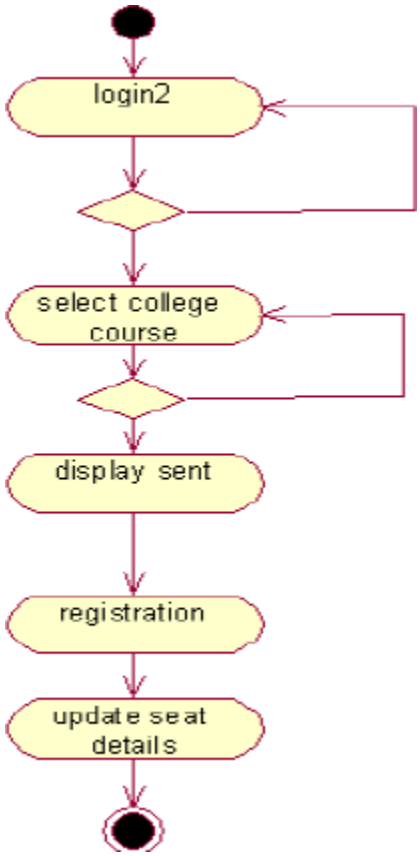

student
1: request for information of the college
3: request for hostel facilities
6: join the college
enquiry
5: accept the hostel facilities
2: provide details for the college
databas
e
7: register the student
4: check for availability


student
1: enter the user name
2: enter the password
3: submit
login
5: incorrect user name or password login again
4: verification
databas
e


1: check for particular course
2: select and college
3: request for seat available
5: no availability to particular course is using
6: request for another college
8: seat availability to the college
student
select
college
7: return availability
4: return available
databas
e

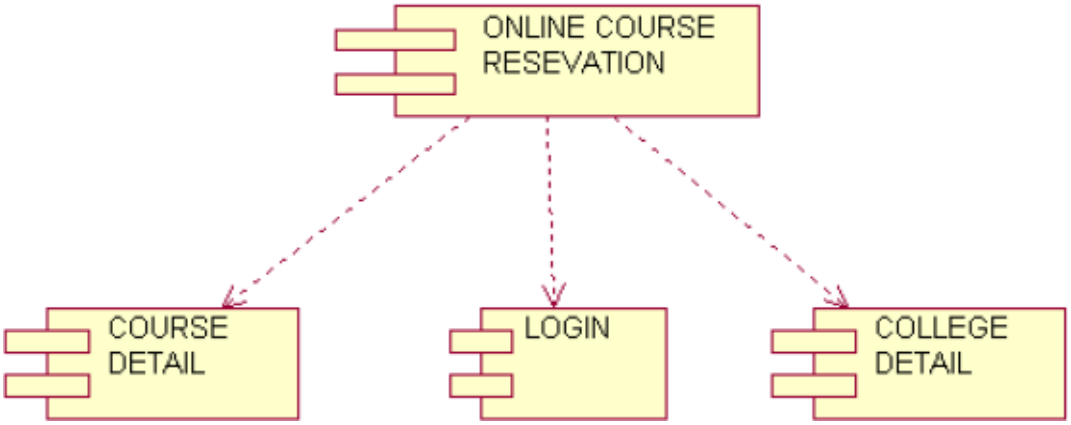
ACTIVIY DIAGRAM

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



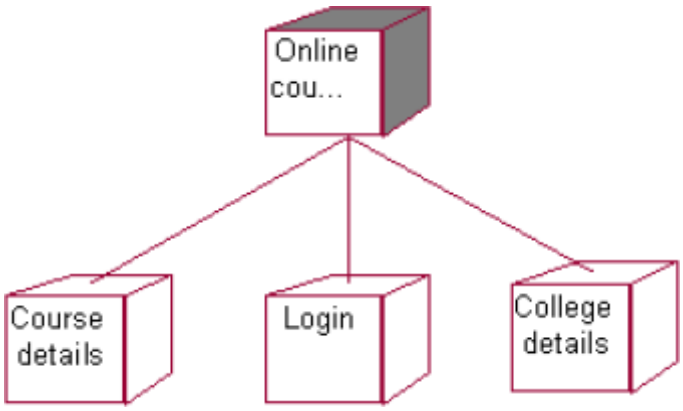
COMPONENT DIAGRAM

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



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-dimentional box. Dependencies are represented by communication association. The basic element of a deployment diagram is a node of two types



PACKAGE DIAGRAM

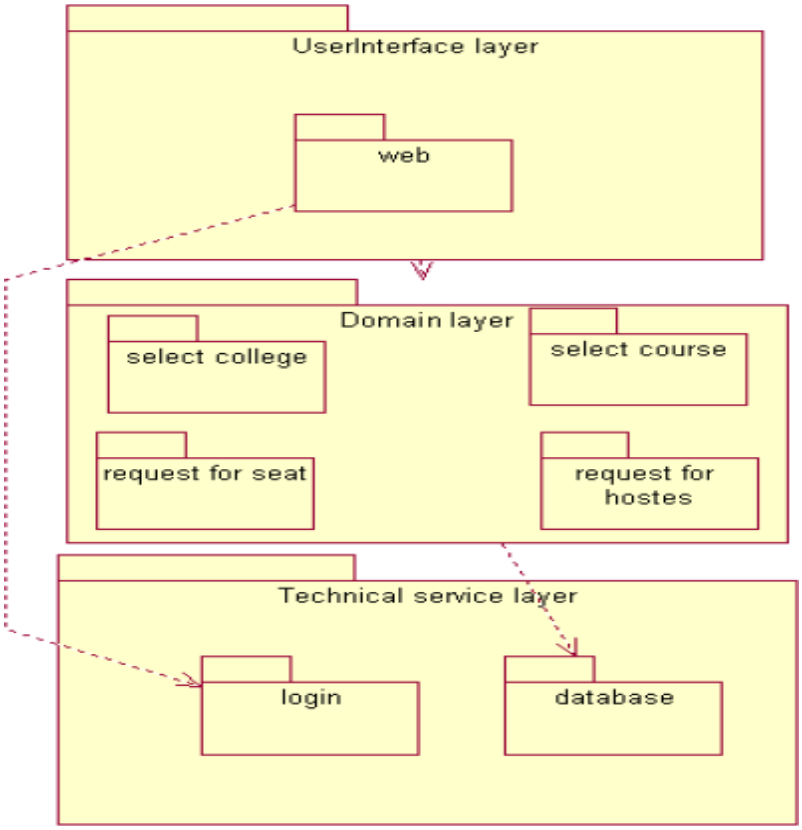
A package diagram is represented as a folder shown as a large rectangle with a top attached to its upper left corner. A package may contain both sub ordinate package and ordinary model elements. All uml models and

diagrams are organized into package. A package diagram in unified modeling language that depicts the dependencies between the packages that make up a 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

a. User interface layer

b. Domain layer

c. Technical services layer



RESULT

Thus the mini project for online course reservation system has been successfully executed and codes are generated.