Experiment 1: Apply the knowledge of SRS and prepare Software Requirement Specification (SRS) document in IEEE format for the project

Learning Objective: Students will be able to List various hardware and software requirements, Distinguish between functional and nonfunctional requirements, indicate the order of priority for various requirements, analyze the requirements for feasibility.

Tools: IEEE template and MS Word

Theory:

The srs should contain the following Table of Contents

Table of Contents

Software Requirements Specification

Revision History

Document Approval

1. Introduction.

1.1 Purpose

1.2 Scope

1.3 Definitions , Acronyms , And Abbreviations

1.4 References

1.5 Overview

2. General Description.

2.1 Product Perspective

2.2 Product Functions

2.3 User Characteristics

2.4 General Constraints

3. Specific Requirements.

3.1 External Interface Requirements

3.1.1 User Interfaces.

3.1.2 Hardware Interfaces.

I. Introduction

An Enterprise resource planning (ERP) is a System that manages the records of students regarding admission, achievement and examination. An Enterprise resource planning (ERP) is designed to help colleges for management of dental students. Extensive information is available at your fingertips through this System. Viewing student data, managing admission and reshuffling, managing seats, quota, board, semester, faculty, category and for examination, block allocation, subject management, scheduling exams, results and related issues are made simple and easy. There are custom search capabilities to aid in finding student information and working on student records. This can make the system easier to navigate and to use maximizing the effectiveness of time and other resources. SMS allows the keeping of personnel data in a form that can be easily accessed and analyzed in a consistent way.

1.1 Purpose

The project is about to handle all the information of the student regarding admission and examination. Also it manages resources which were managed and handled by manpower previously. The main purpose of the project is to integrate distinct sections of the organization into a consistent manner so that complex functions can be handled smoothly by any technical or non-technical persons.

The project aims at the following matters:

- Automation of admission and enrolment as per board, quota, category and available seats.
- Assistance in decision-making.
- To manage information about students, faculty and courses.
- Consistently update information of all the students.

The main purpose of the Admin Module is to introduce new things and configure important aspects. For e.g. only admin is authorized to introduce quota, board, subject, category, etc. and only admin is allowed to configure exams and set fees structure. So the master screens for all These are visible to only admin roles. This is done by the Admin Module. It also can create the users and Physical and Logical Locations. Thus the main purpose of the Admin Module is to managing the dynamic working of the system

1.2 Scope

The scope of the project includes the following:

- Any college can use this system as it is not client centric.
- All admission and examination related work for the student can be done using this system.
- Deliver Electronic Workplace
- Provide Bi-lingual support
- Application Support & Maintenance after deployment to production
- The Admin Module can be reused for projects as well which have many users with different rights. Hence it is reusable.

1.3 Definitions, Acronyms, and Abbreviations

Definitions:

The Enterprise resource planning is an automated version of manual ERP Management System. It can handle all details about a student. The details include college details, subject details, student personnel details, academic details, exam details etc. Our system has two types of accessing modes, administrator and user The management system is managed by an administrator. It is the job of the administrator to insert, update and monitor the whole process. When a user logs in to the system. He would only view details of the student. He can't perform any changes.

Acronyms:

ERP: Enterprise resource planning

LM: Login Module

RUM: Registered Users Module

NUM: Normal Users Module

AM: Administrator Module

SM: Server Module

DB: Database

1.5 Overview

Enterprise resource planning (ERP) is a web-based application that tracks current student's academic information. It maintains academic information for ready access by office staff, students, their faculty advisors, and committee members. Instead of tedious paperwork,students will be able to submit required information electronically, and the departments will be able to evaluate the submissions with a much quicker turnaround. The Student Management System has been modularized into following modules.

A. Login Module

The purpose of this module is to provide entry to the system or website. Based on the type of login, the user is provided with various facilities and functionalities. The main function of this module is to allow the user to use SMS. This module provides two types of login: Admin login and Student login

B. Administrator Module

In this module when the administrator will enter his/her username and password, then he/she will enter into the administrator page and this page consists of two following sub modules.

- ***Student Addition / Updation / Deletion:*** In SMS each Student is added, updated or deleted according to its branch.

- ***Notice/Attendance / Result Generation:*** In SMS information about notice, attendance and Internal result is generated.
- ***Fee Detail and Schedules:*** Fee information detail and schedule detail are managed.

C. Student Module

In this module when a user enters his student id and password, then he can visit all the following pages.

- ***Profile View:*** When the student clicks on this link he/she will get his/her information like student id, student name, password, father name, date of birth, nationality, city, address, country, phone number, mobile number, email. If he/she wants then he/she can change the profile.

- ***Notice View:*** When the student clicks on this link, he can see the latest notices released by the administrator.

- ***Attendance View:*** When the student clicks on this one, the student can get his overall attendance percentage (present and absent).

- ***Internal Results View:*** When the student clicks on this, he/she will get the internals

result in all the subjects. How much grade point he/she secured out of 20 he/she can know.

- ***Fee Detail View:*** When the student clicks this link he/she can get all the fees structure semester wise and annual fee.

- ***The Student Helpdesk:*** This helpdesk is staffed by faculty who are there to help you. You may contact (faculty phone no.).

II. General Description

There are many departments of administration for the maintenance of college information and student databases in any institution. All these departments provide various records regarding students. Most of these track records need to maintain information about the students. This information could be the general details like student name, address, performance, attendance etc. or specific information related to departments like collection of data. All the modules in college administration are interdependent. They are maintained manually. So they need to be automated and centralized as, Information from one module will be needed by other modules. For example, when a student needs his course completion certificate it needs to check many details about the student like his name, reg. number, year of study, exams he attended and many other details. So it needs to contact all the modules that are once, department and examination and the result of students. With that in mind, we overhauled the existing

ERP Database Management System and made necessary improvements to streamline the processes. Administrators using the system will find that the process of recording and retrieving students' information and managing their classes, including marking of attendance, is now a breeze. In general, this project aims to enhance efficiency and at the same time maintain information accurateness. Later in this report, features and improvements that allow achievement to this goal will be demonstrated and highlighted.

2.1 Product Perspective

The various system tools that have been used in developing both the front end, the back end and other tools of the project are being discussed in this section.

- Front End

- JSP, HTML, CSS, JAVA SCRIPTS are utilized to implement the frontend.
- Java Server Page (JSP)
- Different pages in the applications are designed using JSP. A java server page
- component is a type of java server that is designed to fulfill the role of a user interface
- for a java web application. Web development write JSPs as text files that combine
- HTML or XHTML code, XML elements, and embedded JSP actions and commands.

- Using JSP, one can collect input from users through a web page.
- HTML (HyperText Mark-up Language)
- HTML is a syntax used to format a text document on the web.
- CSS (Cascading Style Sheets)
- CSS is a style sheet language used for describing the look and formatting of
- document written in a mark-up language.
- JavaScript is a dynamic computer programming language. It is most commonly used as part of web browsers, whose implementations allow client•side scripts to interact with the user, control the browser, communicate asynchronously, and alter the document content that is displayed

- BACK END

- **MYSQL** is the world's second most widely used open source relational database management system (RDMS). The SQL phrase stands for structured query.

- **PHP** is a server side scripting language designed for web development but also used as a general purpose programming language. PHP code is interpreted by a web server with a PHP processor module, which generates the resulting web page: PHP commands can be embedded directly into an HTML source document rather than calling an external file to process data.

- **SMS GATEWAY is** An SMS gateway that allows a computer to send or receive short message services (SMS) transmissions to or from a telecommunications network. Most messages are eventually routed into the mobile phone networks. Many SMS gateways support media conversion from email and other formats. A direct to a mobile gateway is a device which has built-in wireless. GSM connectivity. It allows SMStext messages to be sent or received by email, from web pages or from other software applications by acquiring a unique identifier from the mobile phone's subscriber identity module, or "SIM card". Direct-to-Mobile gateways are different from SMS aggregators, because they are installed on an organization's own network and connect to a local mobile network. The connection to the mobile network is made by acquiring a SIM card number from the mobile operator and installing it in the gateway.

2.2 Product Functions

The primary function of the Student Management System web server is essentially to save the whole system information sequentially into the database server. The administration department will have access to the whole system environment and that can be modified as per their needs. The architecture of the whole system is made easy so that any person can login to the system and use the functions. The system database is only

accessible to admin and admin can only modify the changes.

2.3 User Characteristics

The user profiles are identified to have interaction with ERP Management System that anyone can register and login into the system and use the required resources. The students can easily fill up the registration form online and submit it. And the admin will check the details that the student is eligible as per the admission criteria. After the student will successfully registered he can use college/school system environments as per their limits decided by admin

2.4 General Constraints

The Student Management System can be accessed successfully by any client location and it's not necessary that every registration is genuine, so there are chances of fake registration that may reflect some errors. So the system is designed in such a way that the database will keep updated by the administrator and there are better security options available on the server that can prevent fake IP addresses to the access system.

III. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The ERP Management System web server must provide a user interface that will be accessible through any internet browser, the major ones being Google Chrome and Internet Explorer 12.

3.1.2 Hardware Interfaces

All components able to be executed on personal computers with Windows OS platforms and other platforms like Linux, Unix.

3.1.3 Software Interfaces

All the interfaces will be ASPX pages running within the internet browser. The SMS must integrate with the DB through SQL Interface. The system will be hosted in a web server running on Windows Server 2005.

3.1.4 Communications Interfaces

Connections to the system will be over TCP/IP connection.

3.2 Functional Requirements

3.2.1 Log in Module (LM)

Users (admin, student and teachers) shall be able to load the Login Module in the internet

browser. The LM shall support the user to log into the system. The login panel shall contain fields to contain a user name and a field for password. The password field shall be masked with symbols when the user types. It shall also contain a button labelled as Login. When the user clicks on the Login button the username and password will be verified by the database administrator and then only the user will be able to use the system functions.

3.2.2 Registered Users Module (RUM)

After successful login, users shall be able to continue navigating through the website and view school/college detailed information. After successful login, user (admin, student and teachers) shall be able to update and maintain their profile, such as changing password and personal details.

3.2.3 Normal Users Module (NUM)

Users who visit SMS but have not registered, are able to navigate through the website. Users shall be able to view currently held events & upcoming institute schedules. Users shall be able to view school/college timings and their faculties information. Users are able to register themselves as registered users, by clicking on the register now button.

3.2.4 Administrator Module (AM)

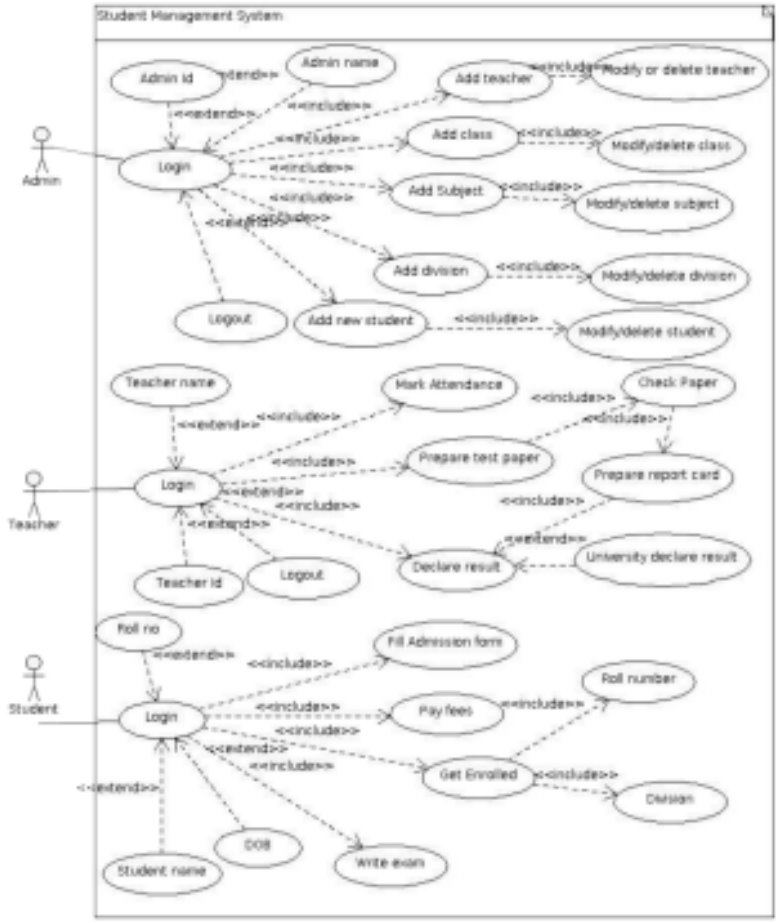
After successful login, the system shall display administrative functions. Administrative functions shown shall be added and updated. When an administrator clicks on the add button, the system shall display a section where the administrator can add new student details, remove unused student details and many more. When an administrator clicks on the update button, the system shall display a section where the administrator can update student details and schedule of lectures which are currently stored in the database. When an administrator adds, updates or deletes an entry, the AM module will send the request to the Server Module which will do the necessary changes to the DB.

3.2.9 Server Module (SM)

SM shall be between the various modules and the DB. SM shall receive all requests and format the pages accordingly to be displayed. SM shall validate and execute all requests from the other modules

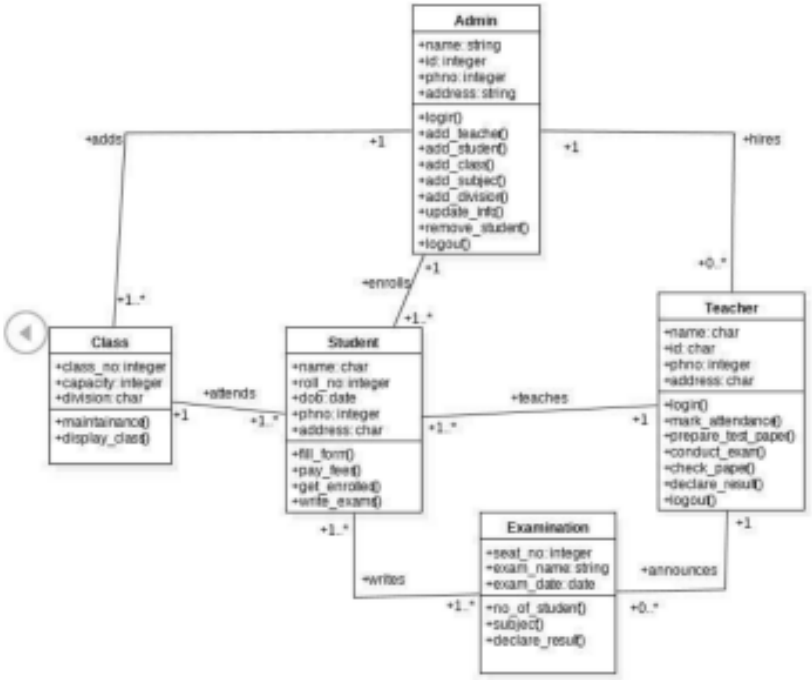
3.3 Use Cases

3.3.1 Use Case #1



3.4 Classes / Objects

3.4.1 Class / Object #1



3.5 Non-functional Requirements

Non-functional requirements may exist for the following attributes. Often these requirements must be achieved at a system wide level rather than at a unit level. State the requirements in the following sections in measurable terms (e.g. 95% of transactions shall be processed in less than a second, system downtime may not exceed 1 minute per day, > 30 day MTBF value, etc.)

3.5.1 Performance

The Student Management System shall be built upon the web development technique and put on the web server online. The system and the server must be capable of handling the real time error functionality that occurs by the defined users. In addition, the system must be safety critical. All failures reported by the server side must be handled instantaneously to allow for user and system safety.

3.5.2 Reliability

The system is safety critical. If it moves out of normal operation mode, the requirement is to drop or down the server and fix it as soon as possible and open it again. This emergency behaviour shall not occur without reason.

3.5.3 Availability

When in normal operating conditions, requests by a user for an online system shall be handled within 1 second. Immediate feedback of the system's activities shall be communicated to the user by clearing the system and giving space and speed to their hospitality.

3.5.4 Security

There shall be a strong security mechanism should be placed in the server side of the system to keep unwanted users from hacking or damaging the system. However, all users of the system give and store the details of privacy related to personal information and many others. However, our system can be accessed online so we need a very secured system as far as security is concerned.

3.5.5 Maintainability

There shall be design documents describing maintenance of the software and database used to save the user details as well as the daily updates and modifications done in the system. There shall be an access on the control system by the admin to maintain it properly at the front end as well as at the back end.

3.5.6 Portability

There is a portability requirement as far as our system is concerned because it is an online as well as offline (local server based) system so we can access it from anywhere through the internet connection. And we have to maintain the copy of stored data into our database.

3.6 Inverse Requirements

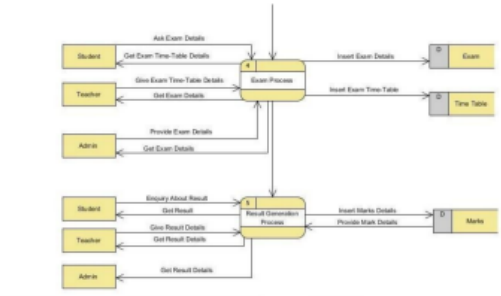
As far as Inverse Requirement is concerned, our system has the best inverse requirement which is most important according to our system view. That is if a student wants to cancel the registration so admin can access the student details for cancellation and will refund the fees that was fetched from student and not complete payment will be back which is the trend and rule by every school/college management.

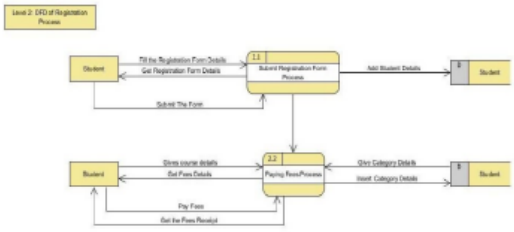
3.7 Logical Database Requirements

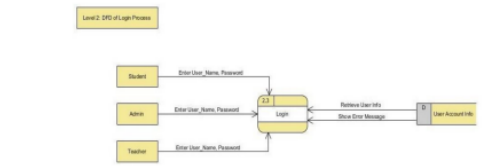
A one to many relational database shall be used in order to validate various student requests and details can be mismatched. Moreover, mismatches are to be logged for reference. The database shall be concurrent with the performance requirements of the Student Management System.

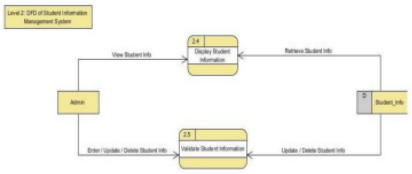
3.8 Other Requirements

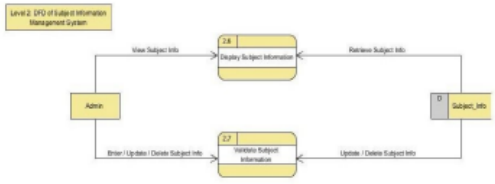
A degraded mode of operation should be possible in which each system can operate independently of central scheduling. The software shall have failure and error recognition codes acting as a safety net, thus keeping the software from performing any major catastrophic functions.

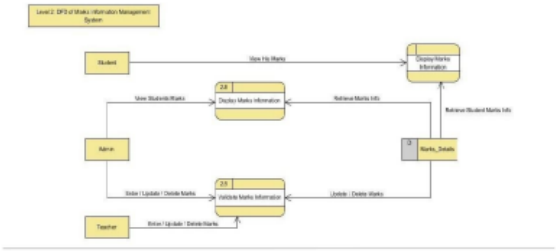












4.2 State-Transition Diagrams (STD)

A. Appendices

A.1 Appendix- 1

Activity diagram manual

UMLActivity Diagram

Overview:

Activity diagram is another important diagram in UML to describe dynamic aspects of the system.

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system.

So the control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent. Activity diagrams deal with all types of flow control by using different elements like fork, join etc.

Purpose:

The basic purposes of activity diagrams are similar to the other four diagrams. It captures the dynamic behavior of the system. Other four diagrams are used to show the message flow from one object to another but the activity diagram is used to show message flow now from one activity to another.

Activity is a particular operation of the system. Activity diagrams are not only used for visualizing the dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques. The only missing thing in the activity diagram is the message part.

It does not show any message flow from one activity to another. Activity diagram is sometimes considered as the flow chart. Although the diagram looks like a flow chart, it is not. It shows different flows like parallel, branched, concurrent and single.

So the purposes can be described as:

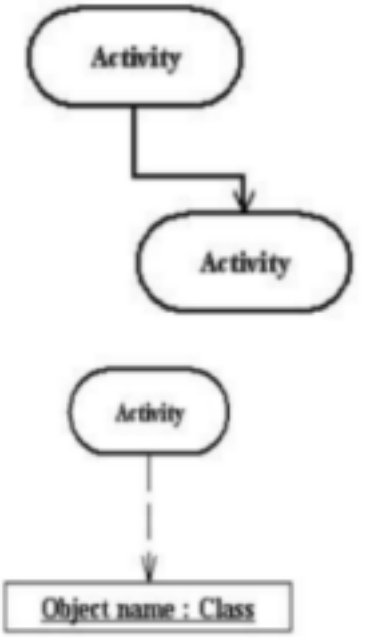
- Draw the activity flow of a system.
- Describe the sequence from one activity to another.
- Describe the parallel, branched and concurrent flow of the system.

Basic Activity Diagram Symbols and Notations

Action states

Action states represent the non interruptible actions of objects. You can draw an action state in SmartDraw using a rectangle with rounded corners.

Action Flow



vs illustrate the relationships among action states.

Object Flow

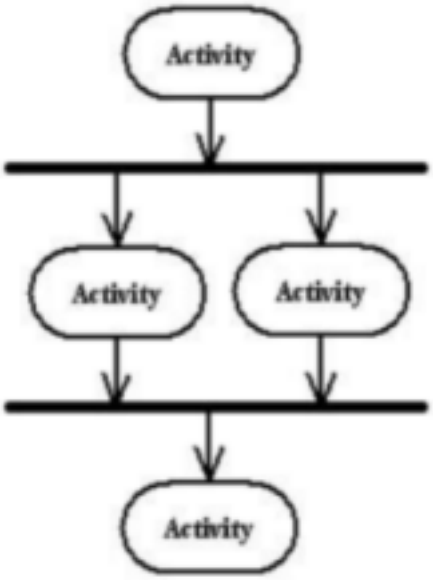
Object flow refers to the creation and modification of objects by activities. An object flow arrow from an action to an object means that the action creates or influences the object. An object flow arrow from an object to an action indicates that the action state uses the object.

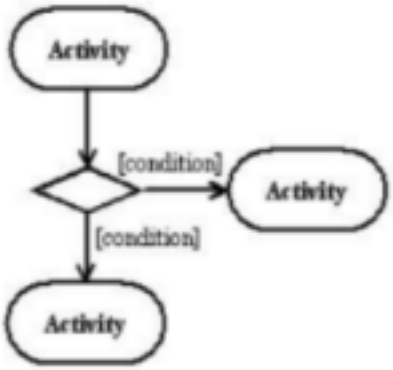
A filled circle followed by an arrow represents the initial action state.

An arrow pointing to a filled circle nested inside another the circle represents the final action state.

Branching

A diamond represents a decision with alternate paths. The outgoing alternates should be labeled with a condition or guard expression. You can also label one of the paths "else."





Synchronization

- A synchronization bar helps illustrate parallel transitions.
- Synchronization is also called forking and joining.

Swimlanes

- Swimlanes group related activities into one column

Where to use Activity Diagrams?

The basic usage of the activity diagram is similar to the other four UML diagrams. The specific usage isto model the control flow from one activity to another. This control flow does not include messages. The activity diagram is suitable for modeling the activity flow of the system. An application can have multiple systems. Activity diagram also captures these systems and describes flow from one system to another. This specific usage is not available in other diagrams. These systems can be database, external queues or any other system. Now we will look into the practical applications of the activity diagram. From the above discussion it is clear that an activity diagram is drawn from a very high level. So it gives a high level view of a system.This high level view is mainly for business users or any other person who is not a technical person. This diagram is used to model the activities which are nothing but business requirements. So the diagram has more impact on business understanding rather than implementation details.

Following are the main usages of activity diagram:

- Modeling workflow by using activities.
- Modeling business requirements.
- High level understanding of the system's functionalities.
- Investigate business requirements at a later stage

A.2 Appendix-2

UML Statechart Diagram

Overview:

The name of the diagram itself clarifies the purpose of the diagram and other details. It describes different states of a component in a system. The states are specific to a component/object of a system. A Statechart diagram describes a state machine. Now to clarify it, a state machine can be defined as a machine which defines different states of an object and these states are controlled by external or internal events. Activity diagram explained in the next chapter, is a special kind of a Statechart diagram. As the Statechart diagram defines states it is used to model the lifetime of an object.

Purpose:

Statechart diagram is one of the five UML diagrams used to model the dynamic nature of a system. They define different states of an object during its lifetime. And these states are changed by events. So Statechart diagrams are useful to model reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

Statechart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. So the most important purpose of a Statechart diagram is to model the lifetime of an object from creation to termination.

Statechart diagrams are also used for forward and reverse engineering of a system. But the main purpose is to model a reactive system.

Following are the main purposes of using Statechart diagrams:

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

Learning Outcomes: Students should have the ability to

LO1: List various hardware and software requirements
LO2: Distinguish between functional and nonfunctional requirements
LO3: Indicate the order of priority for various requirements
LO4: Analyze the requirements for feasibility Course

Course Outcomes: Upon completion of the course students will be able to prepare SRS document

Conclusion: Successfully Sketched a Software Requirement Specification for ERP System

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				

Experiment 2: Use project management tool to prepare schedule for the project

Learning Objective: Students will be able to List the various activities in the project, analyze the various activities for schedule, estimate the time for each activity and develop a Gantt Chart for the activities.

Tools: Gantt Chart using MSExcel

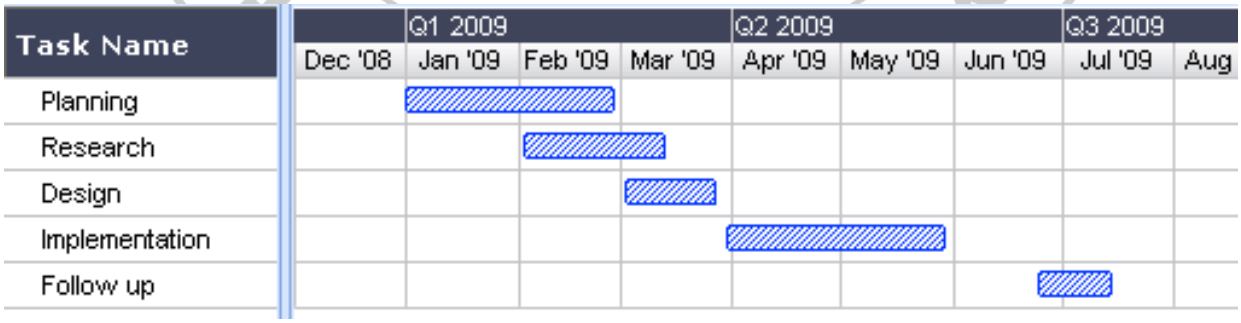
Theory:

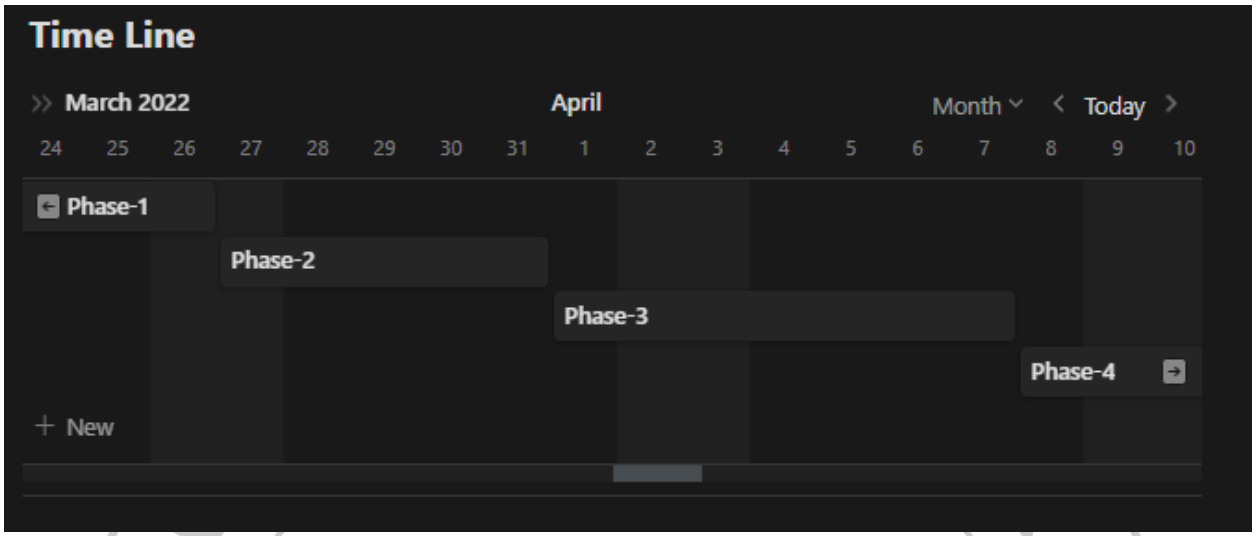
The main aim of PROJECT SCHEDULING AND TRACKING is to get the project completed on time. Program evaluation and review technique (PERT) and Gantt chart are two project scheduling methods that can be applied to software development

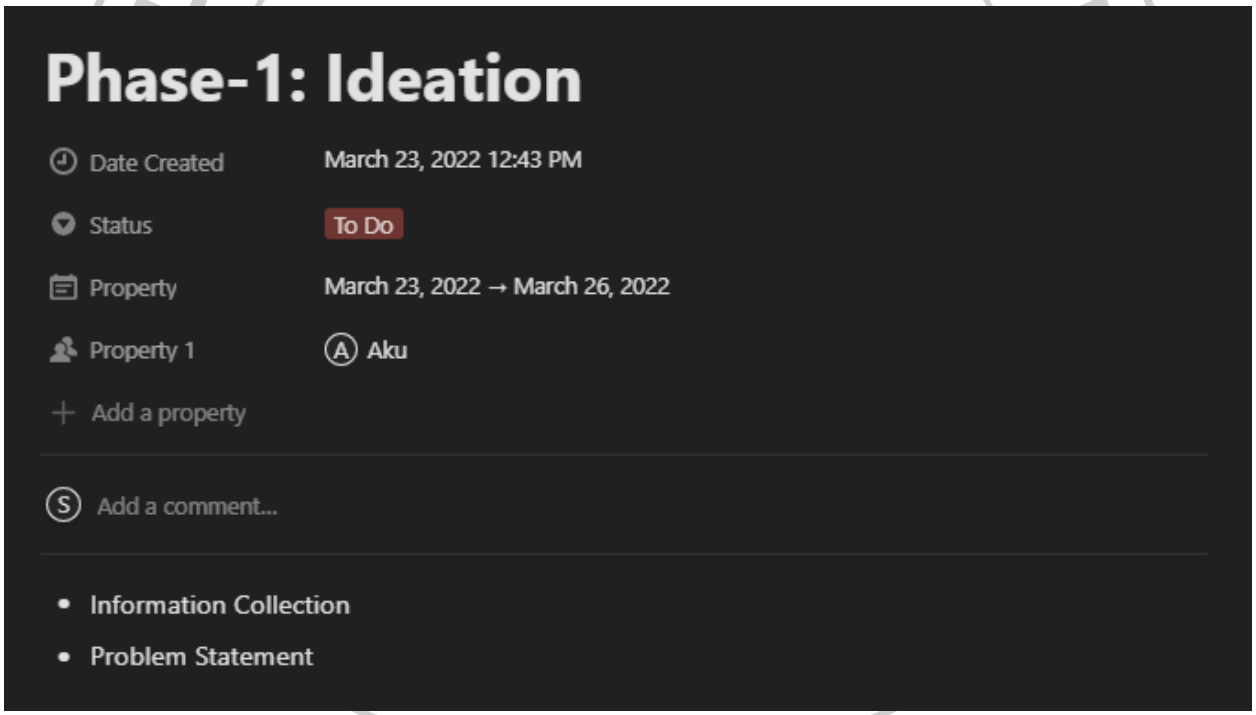
Split the project into tasks and estimate time and resources required to complete each task. Organize tasks concurrently to make optimal use of the workforce. Minimize task dependencies to avoid delays caused by one task waiting for another to complete

Gantt chart: A Gantt chart, commonly used in project management, is one of the most popular and useful ways of showing activities (tasks or events) displayed against time. On the left of the chart is a list of the activities and along the top is a suitable time scale. Each activity is represented by a bar; the position and length of the bar reflects the start date, duration and end date of the activity. This allows you to see at a glance:

- What the various activities are
- When each activity begins and ends
- How long each activity is scheduled to last
- Where activities overlap with other activities, and by how much
- The start and end date of the whole project




Time Line
March 2022
April
Month
Today
24 25 26 27 28 29 30 31 1 2 3 4 5 6 7 8 9 10
Phase-1
Phase-2
Phase-3
Phase-4
New


Phase-1: Ideation
Date Created
March 23, 2022 12:43 PM
Status
To Do
Property
March 23, 2022 → March 26, 2022
Property 1
Aku
Add a property
Add a comment...
Information Collection
Problem Statement

Phase-2: ERP diagrams and Database

Date Created	March 23, 2022 12:48 PM
Status	To Do
Property	Empty
Property 1	Empty

+ Add a property

Add a comment...

- Mind Map and Affinity Diagrams
- Time Line and Dataflow Diagram

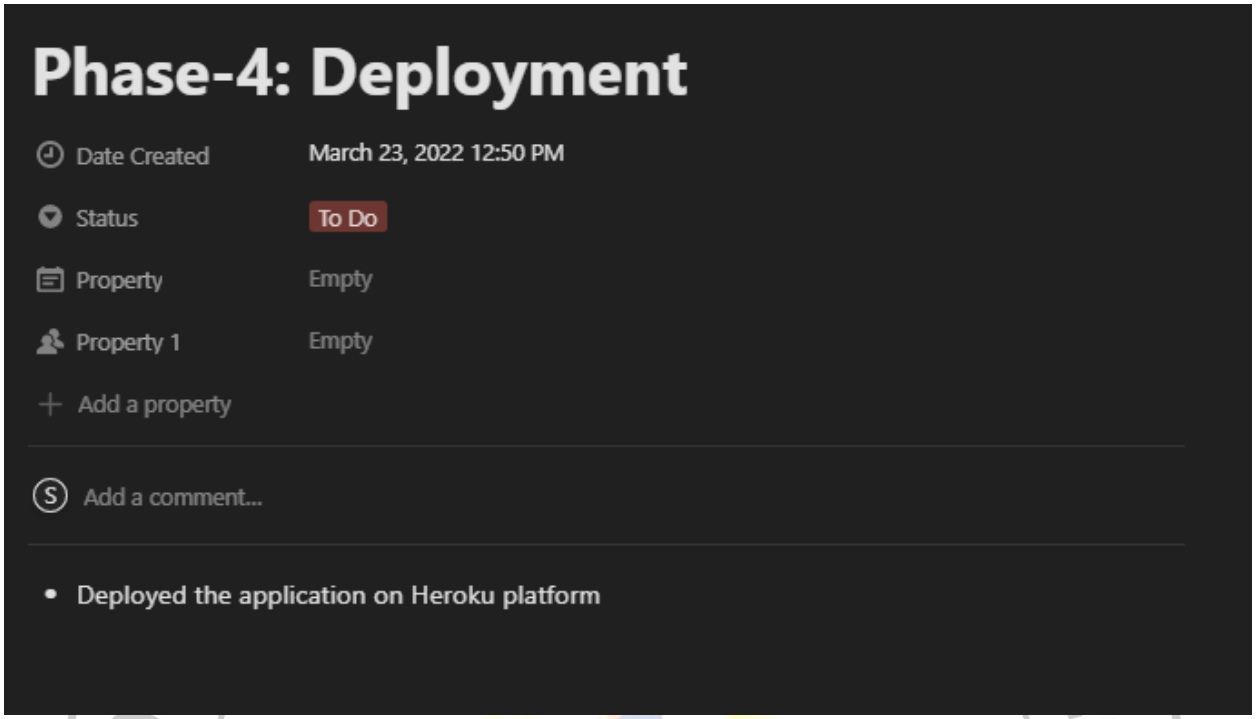
Phase-3: Development

Date Created	March 23, 2022 12:48 PM
Status	To Do
Property	Empty
Property 1	Empty

+ Add a property

Add a comment...

- Prototype Design
- Making separate modules for Admin, Students, Faculty and Examination Cell
- Connecting the UI/UX with the databases (MongoDB) using the Node.js Modules



Learning Outcomes: Students should have the ability to

LO1: List the various activities in the project.
LO2: Analyze the various activities for schedule.
LO3: Estimate the time for each activity.
LO4: Develop a Gantt Chart for the activities.

Outcomes: Upon completion of the course students will be able to use project management tools to prepare schedules for the project.

Conclusion:

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				

Experiment 3: Draw DFD (upto 2 levels) and prepare Data Dictionary for the project

Learning Objective: Students will able to identify the data flows, processes, source and destination for the project, Analyze and design the DFD upto 2 levels, develop a data dictionary for the project

Tools: Dia, StarUML

Theory:

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

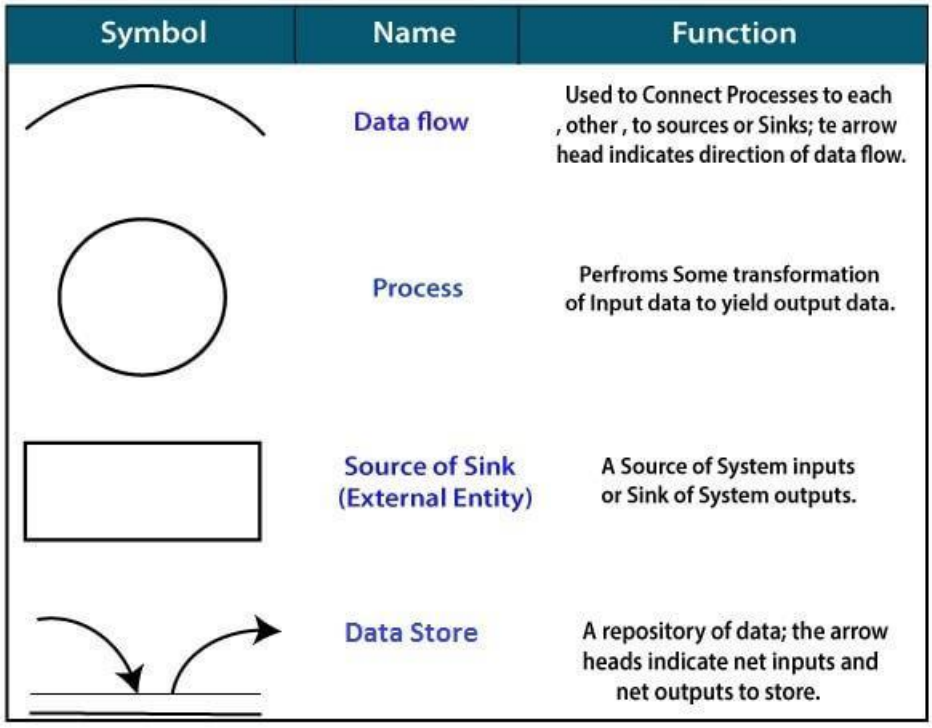
It shows how data enters and leaves the system, what changes the information, and where data is stored.

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

The following observations about DFDs are essential:

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

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



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

Symbols for Data Flow Diagrams

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

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

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

Levels in Data Flow Diagrams (DFD)

The DFD may be used to perform a system or software at any level of abstraction. Infact, DFDs may be partitioned into levels that represent increasing information flow and functional detail. Levels in DFD are numbered 0, 1, 2 or beyond. Here, we will see primarily three levels in the data flow diagram, which are: 0-level DFD, 1-level DFD, and 2-level DFD.

0-level DFD

It is also known as the fundamental system model, or context diagram that represents the entire software requirement as a single bubble with input and output data denoted by incoming and outgoing arrows. Then the system is decomposed and described as a DFD with multiple bubbles. Parts of the system represented by each of these bubbles are then decomposed and documented

as more and more detailed DFDs. This process may be repeated at as many levels as necessary until the program at hand is well understood. It is essential to preserve the number of inputs and outputs between levels, this concept is called leveling by DeMacro. Thus, if bubble "A" has two inputs x_1 and x_2 and one output y, then the expanded DFD, that represents "A" should have exactly two external inputs and one external output as shown in fig:

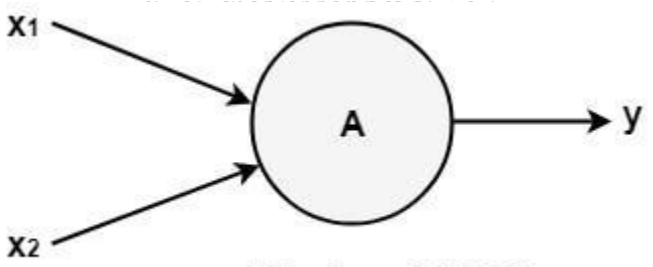


Fig: Level-0 DFD.

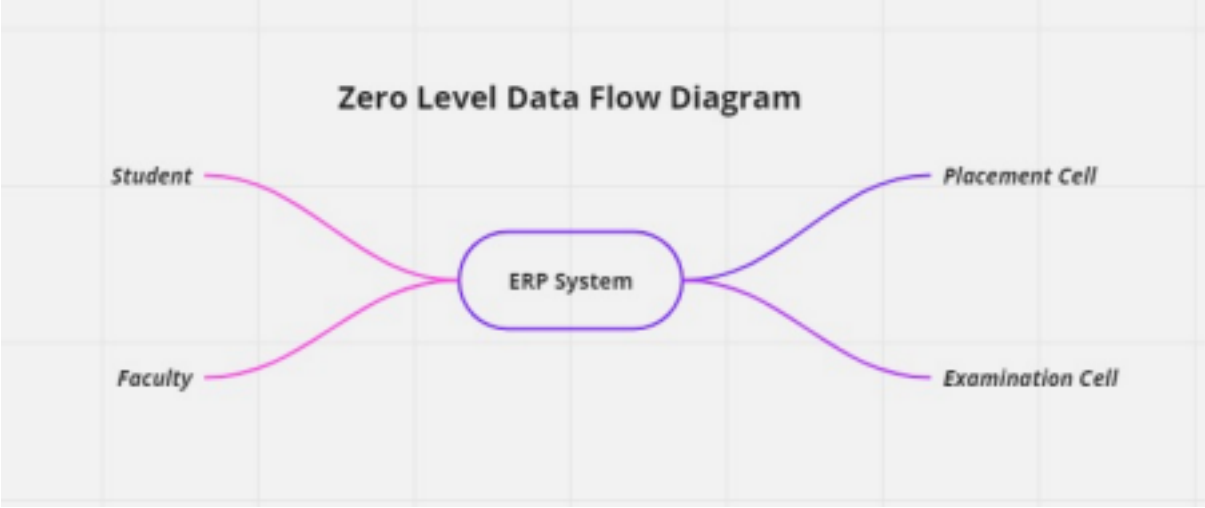
1-level DFD

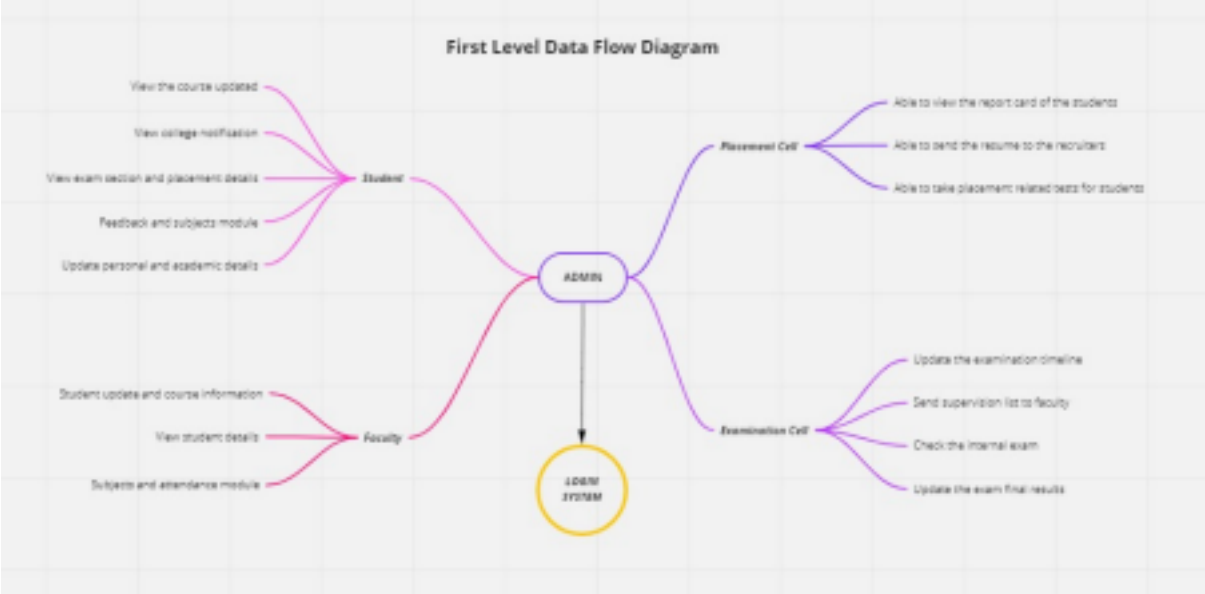
In 1-level DFD, a context diagram is decomposed into multiple bubbles/processes. In this level, we highlight the main objectives of the system and break down the high-level process of 0-level DFD into subprocesses.

2-Level DFD

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

Data Flow Diagram for ERP System (Level-0 and Level-1)





Learning Outcomes: Students should have the ability to

LO1: Identify the dataflows, processes, source and destination for the project.
LO2: Analyze and design the DFD upto 2 levels
LO3: Develop a data dictionary for the project

Outcomes: Upon completion of the course students will be able to prepare Draw DFD (upto 2 levels) and prepare Data Dictionary for the project

Conclusion: Successfully sketched a Data Flow Diagram upto 2 Levels

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				

Experiment 04- Implement UML Use-case diagram

Learning Objective: To implement UML use-case diagrams for the project.

Tools: MS Word, draw.io

Theory:

Use case diagrams

Use case diagrams belong to the category of behavioral diagrams of UML diagrams. Use case diagrams aim to present a graphical overview of the functionality provided by the system. It consists of a set of actions (referred to as use cases) that the concerned system can perform one or more actors, and dependencies among them.

Actor: An actor can be defined as an object or set of objects, external to the system, which interacts with the system to get some meaningful work done. Actors could be human, devices, or even other systems.

For example, consider the case where a customer *withdraws cash* from an ATM. Here, the customer is a human actor.

Actors can be classified as below:

- **Primary actor**: They are principal users of the system, who fulfill their goal by availing some service from the system. For example, a customer uses an ATM to withdraw cash when he needs it. A customer is the primary actor here.
- **Supporting actor**: They render some kind of service to the system. "Bank representatives", who replenishes the stock of cash, is such an example. It may be noted that replenishing stock of cash in an ATM is not the prime functionality of an ATM.

In a use case diagram primary actors are usually drawn on the top left side of the diagram.

Use Case

A use case is simply a functionality provided by a system.

Continuing with the example of the ATM, *withdraw cash* is a functionality that the ATM provides. Therefore, this is a use case. Other possible use cases include, *check balance*, *change PIN*, and so on.

Use cases include both successful and unsuccessful scenarios of user interactions with the system. For example, authentication of a customer by the ATM would fail if he entered the wrong PIN. In such cases, an error message is displayed on the screen of the ATM.

Subject: Subject is simply the system under consideration. Use cases apply to a subject. For example, an ATM is a subject, having multiple use cases, and multiple actors interact with it. However, one should be careful of external systems interacting with the subject as actors.

Graphical Representation: An actor is represented by a stick figure and the name of the actor is written below it. A use case is depicted by an ellipse and the name of the use case is written inside it. The subject is shown by drawing a rectangle. Labels for the system could be put inside it. Use cases are drawn inside the rectangle, and actors are drawn outside the rectangle, as shown in figure - 01.

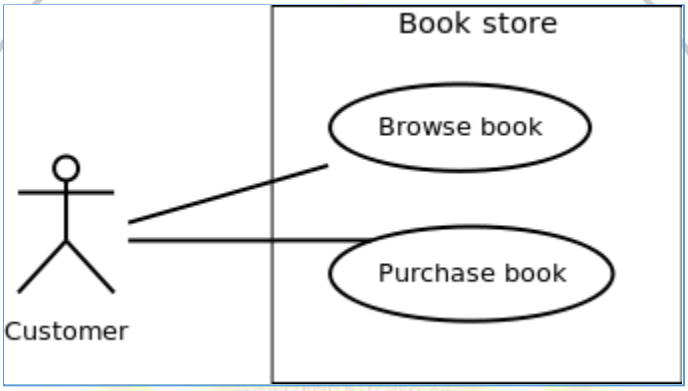


Figure - 01: A use case diagram for a bookstore

Association between Actors and Use Cases

A use case is triggered by an actor. Actors and use cases are connected through binary associations indicating that the two communicate through message passing. An actor must be associated with at least one use case. Similarly, a given use case must be associated with at least one actor. Association among the actors is usually not shown. However, one can depict the class hierarchy among actors.

Use Case Relationships

Three types of relationships exist among use cases:

- Include relationship
- Extend relationship
- Use case generalization

Include Relationship

Include relationships are used to depict common behavior that are shared by multiple use cases. This could be considered analogous to writing functions in a program in order to avoid repetition of writing the same code. Such a function would be called from different points within the program.

Example

For example, consider an email application. A user can send a new mail, reply to an email he has received, or forward an email. However, in each of these three cases, the user must be logged in to perform those actions. Thus, we could have a *login* use case, which is included by *compose mail*, *reply*, and *forward email* use cases. The relationship is shown in figure - 02.

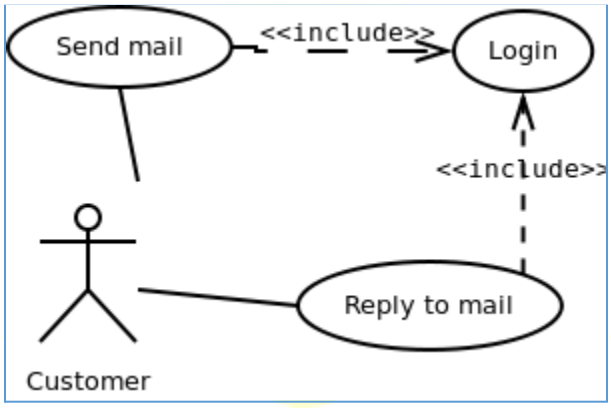


Figure - 02: Include relationship between use cases

Notation

Include relationship is depicted by a dashed arrow with a «include» stereotype from the including use case to the included use case.

Extend Relationship

Use case extensions are used to depict any variation to an existing use case. They are used to specify the changes required when any assumption made by the existing use case becomes false.

Example

Let's consider an online bookstore. The system allows an authenticated user to buy selected book(s). While the order is being placed, the system also allows specifying any special shipping

instructions, for example, calling the customer before delivery. This *Shipping Instructions* step is optional, and not a part of the main *Place Order* use case. Figure - 03 depicts such a relationship.

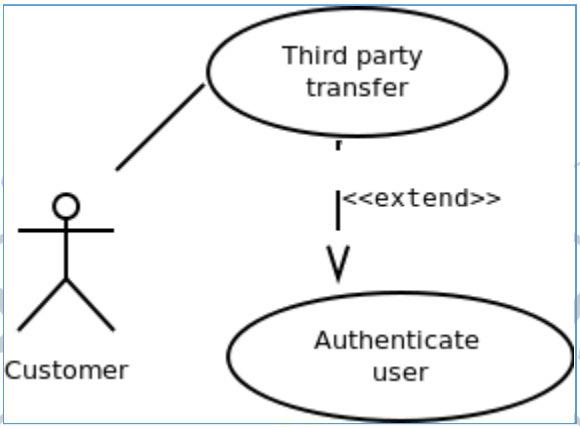


Figure - 03: Extend relationship between use cases

Notation

Extend relationship is depicted by a dashed arrow with a «extend» stereotype from the extending use case to the extended use case.

Generalization Relationship

Generalization relationship is used to represent the inheritance between use cases. A derived use case specializes in some functionality it has already inherited from the base use case.

Example

To illustrate this, consider a graphical application that allows users to draw polygons. We could have a use case *draw polygon*. Now, a rectangle is a particular instance of a polygon having four sides at right angles to each other. So, the use case *draw rectangle* inherits the properties of the use case *draw polygon* and overrides its drawing method. This is an example of a generalization relationship. Similarly, a generalization relationship exists between *draw rectangle* and *draw square* use cases. The relationship has been illustrated in figure - 04.

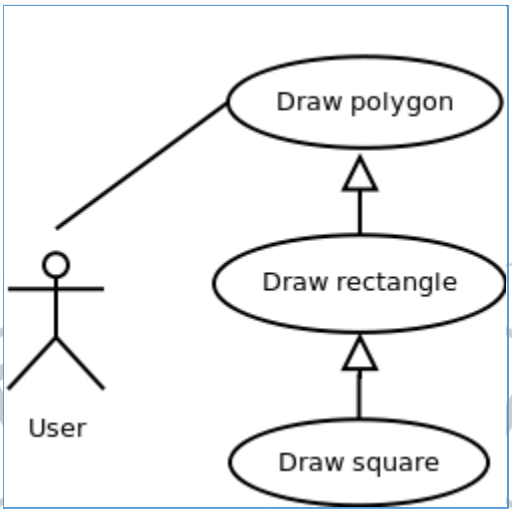


Figure - 04: Generalization relationship among use cases

Notation

Generalization relationship is depicted by a solid arrow from the specialized (derived) use case to the more generalized (base) use case.

Identifying Actors

Given a problem statement, the actors could be identified by asking the following questions :

- Who gets most of the benefits from the system? (The answer would lead to the identification of the primary actor)
- Who keeps the system working? (This will help to identify a list of potential users)
- What other software / hardware does the system interact with?
- Any interface (interaction) between the concerned system and any other system?

Identifying Use cases

Once the primary and secondary actors have been identified, we have to find out their goals i.e. what the functionality they can obtain from the system is. Any use case name should start with a verb like, "Check balance".

Guidelines for drawing Use Case diagrams

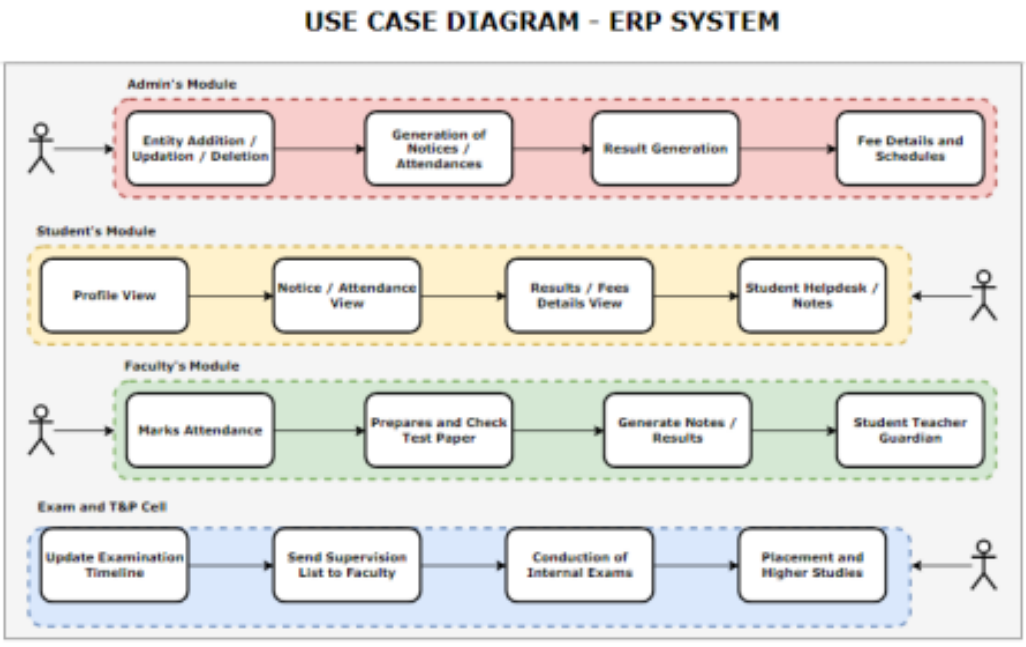
Following general guidelines could be kept in mind while trying to draw a use case diagram :

- Determine the system boundary
- Ensure that individual actors have well-defined purpose
- Use cases identified should let some meaningful work done by the actors
- Associate the actors and use cases -- there shouldn't be any actor or use case floating without any connection
- Use include relationship to encapsulate common behavior among use cases , if any

Procedure:

A Use Case model can be developed by following the steps below.

1. Identify the Actors (role of users) of the system.
2. For each category of users, identify all roles played by the users relevant to the system.
3. Identify what are the users required for the system to be performed to achieve these goals.
4. Create use cases for every goal.
5. Structure the use cases.
6. Prioritize, review, estimate and validate the users.



Result and Discussion:

Q.1) What is a use-case diagram? Draw at least two use-cases for your projects.

Learning Outcomes: The student should have the ability to:

LO 1: Identify the importance of use-case diagrams.

LO 2: Draw use-case diagrams for a given scenario.

Course Outcomes: Upon completion of the course students will be able to understand and demonstrate use-case diagrams.

Conclusion: Thus, students have understood and successfully drawn use-case diagrams.

Viva Questions:

1. What use-case diagrams are used for?
2. Enumerate the type of relationships that exist for use-case diagrams.

For Faculty Use:

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				

Experiment 05- Implement UML Class Diagram

Learning Objective: To implement UML class diagrams for the project.

Tools: MS Word, draw.io

Theory:

Class Diagrams:

Classes are the structural units in an object oriented system design approach, so it is essential to know all the relationships that exist between the classes in a system. All objects in a system are also interacting to each other by means of passing messages from one object to another.

Elements in class diagram

Class diagram contains the system classes with its data members, operations and relationships between classes.

Class

A set of objects containing similar data members and member functions is described by a class. In UML syntax, class is identified by solid outline rectangle with three compartments which contain

- **Class name**

A class is uniquely identified in a system by its name. A textual string is taken as the class name. It lies in the first compartment in the class rectangle.

- **Attributes**

Property shared by all instances of a class. It lies in the second compartment in the class rectangle.

- **Operations**

An execution of an action can be performed for any object of a class. It lies in the last compartment in the class rectangle.

Example

To build a structural model for an Educational Organization, 'Course' can be treated as a class which contains attributes 'courseName' & 'courseID' with the operations 'addCourse()' & 'removeCourse()' allowed to be performed for any object to that class.

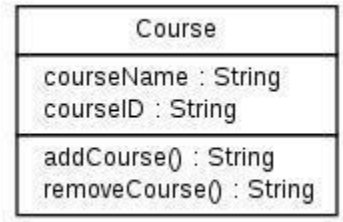


Figure-01

- **Generalization/Specialization**

It describes how one class is derived from another class. Derived class inherits the properties of its parent class.

Example

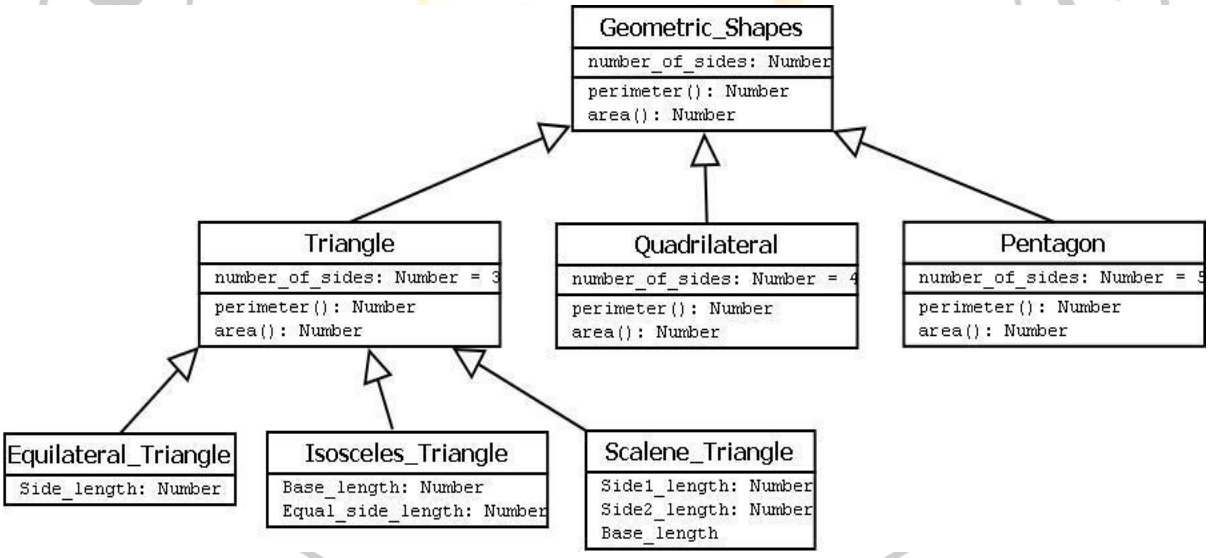


Figure-02

Geometric_Shapes is the class that describes how many sides a particular shape has. Triangle, Quadrilateral and Pentagon are the classes that inherit the property of the Geometric_Shapes class. So the relations among these classes are generalized. Now Equilateral_Triangle, Isosceles_Triangle and Scalene_Triangle, all these three classes inherit the properties of the Triangle class as each one of them has three sides. So, these are specializations of Triangle class.

Relationships

Existing relationships in a system describe legitimate connections between the classes in that system.

- **Association**

It is an instance level relationship that allows exchanging messages among the objects of both ends of association. A simple straight line connecting two class boxes represents an association. We can give a name to association and also at the both end we may indicate role names and multiplicity of the adjacent classes. Association may be uni-directional.

Example

In the structure model for a system of an organization an employee (instance of 'Employee' class) is always assigned to a particular department (instance of 'Department' class) and the association can be shown by a line connecting the respective classes.

Figure-03

- **Aggregation**

It is a special form of association which describes a part-whole relationship between a pair of classes. It means, in a relationship, when a class holds some instances of related classes, then that relationship can be designed as an aggregation.

Example

For a supermarket in a city, each branch runs some of the departments they have. So, the relation among the classes 'Branch' and 'Department' can be designed as aggregation. In UML, it can be shown as in the fig. below.

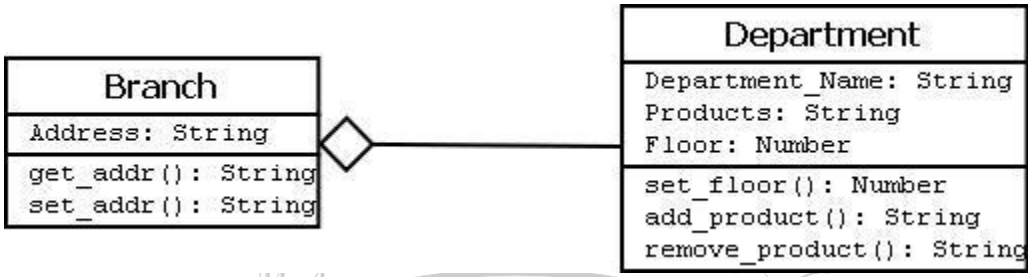


Figure-04

- Composition

It is a strong form of aggregation which describes that whole completely owns its part. Life cycle of the part depends on the whole.

Example

Consider a shopping mall that has several branches in different locations in a city. The existence of branches completely depends on the shopping mall as if it does not exist any branch of it will no longer exist in the city. This relation can be described as composition and can be shown as below

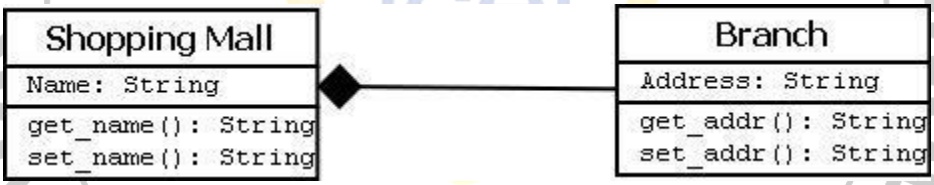


Figure-05

- **Multiplicity**

It describes how the number of instances of one class is related to the number of instances of another class in an association.

Notation for different types of multiplicity:

Single instance	1
Zero or one instance	0..1
Zero or more instance	0..*
One or more instance	1..*
Particular range(two to six)	2..6

Figure-06

Example

One vehicle may have two or more wheels

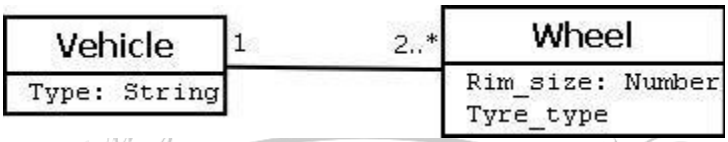


Figure-07:

Procedure:

When required to describe the static view of a system or its functionalities, we would be required to draw a class diagram. Here are the steps you need to follow to create a class diagram.

Step 1: Identify the class names

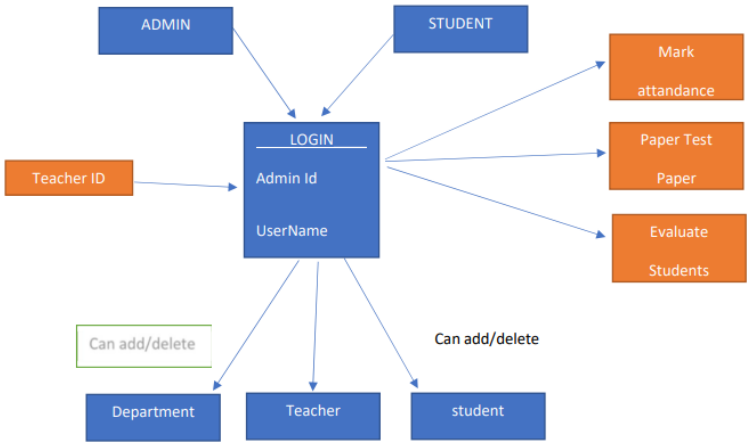
The first step is to identify the primary objects of the system.

Step 2: Distinguish relationships

Next step is to determine how each of the classes or objects are related to one another. Look out for commonalities and abstractions among them; this will help you when grouping them when drawing the class diagram.

Step 3: Create the Structure

First, add the class names and link them with the appropriate connectors. You can add attributes and functions/ methods/ operations later.



Result and Discussion:

Q.1) What is a class diagram? Draw at least two class diagrams for your projects.

Learning Outcomes: The student should have the ability to:

LO 1: Identify the importance of class diagrams.

LO 2: Draw class diagrams for a given scenario.

Course Outcomes: Upon completion of the course students will be able to understand and demonstrate class diagrams.

Conclusion: Thus, students have understood and successfully drawn class diagrams.

Viva Questions:

1. What are class diagrams used for?
2. Enumerate various relationships in a class diagram.

For Faculty Use:

Correction Parameters	**Formative Assessment [40%]**	**Timely completion of Practical [40%]**	**Attendance / Learning Attitude [20%]**	
Marks Obtained				

Experiment 06- Implement State/Activity diagrams

Learning Objective: To implement a dynamic view of a system using Activity/State diagrams.

Tools: MS Word, draw.io

Theory:

Capturing the dynamic view of a system is very important for a developer to develop the logic for a system. State chart diagrams and activity diagrams are two popular UML diagrams to visualize the dynamic behavior of an information system.

In this experiment, we will learn about the different components of activity diagram and state chart diagram and how these can be used to represent the dynamic nature of an information system.

State-chart Diagrams

In the case of Object Oriented Analysis and Design, a system is often abstracted by one or more classes with some well defined behavior and states. A *state chart diagram* is a pictorial representation of such a system, with all its states, and different events that lead transition from one state to another.

To illustrate this, consider a computer. Some possible states that it could have are: running, shutdown, hibernate. A transition from running state to shutdown state occurs when the user presses the "Power off" switch, or clicks on the "Shutdown" button as displayed by the OS. Here, clicking on the shutdown button, or pressing the power off switch act as external events causing the transition.

State-chart diagrams are normally drawn to model the behavior of a complex system. For simple systems this is optional.

Building Blocks of a State-chart Diagram

State

A state is any "distinct" stage that an object (system) passes through in its lifetime. An object remains in a given state for a finite time until "something" happens, which makes it move to another state. All such states can be broadly categorized into following three types:

- Initial: The state in which an object remain when created
- Final: The state from which an object do not move to any other state [optional]
- Intermediate: Any state, which is neither initial, nor final

As shown in figure-01, an initial state is represented by a circle filled with black. An intermediate state is depicted by a rectangle with rounded corners. A final state is represented by an unfilled circle with an inner black-filled circle.

Figure-01: Representation of initial, intermediate, and final states of a state chart diagram

Intermediate states usually have two compartments, separated by a horizontal line, called the name compartment and internal transitions compartment. They are described below:

- Name compartment: Contains the name of the state, which is a short, simple, descriptive string
- Internal transitions compartment: Contains a list of internal activities performed as long as the system is in this state

The internal activities are indicated using the following syntax: action-label / action-expression. Action labels could be any condition indicator. There are, however, four special action labels:

- Entry: Indicates activity performed when the system enter this state
- Exit: Indicates activity performed when the system exits this state
- Do: indicate any activity that is performed while the system remain in this state or until the action expression results in a completed computation
- Include: Indicates invocation of a sub-machine

Any other action label identifies the event (internal transition) as a result of which the corresponding action is triggered. Internal transition is almost similar to self transition, except that the former doesn't result in execution of entry and exit actions. That is, system doesn't exit or re-enter that state. Figure-02 shows the syntax for representing a typical (intermediate) state

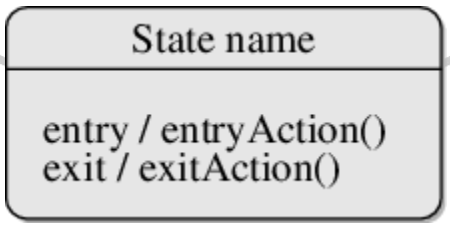


Figure-02: A typical state in a state chart diagram

States could again be either simple or composite. Here, however, we will deal only with simple states.

Transition

Transition is movement from one state to another state in response to an external stimulus (or any internal event). A transition is represented by a solid arrow from the current state to the next state. It is labeled by: event [guard-condition]/[action-expression], where

- Event is the what is causing the concerned transition (mandatory) -- Written in past tense
- Guard-condition is (are) precondition(s), which must be true for the transition to happen
- Action-expression indicate action(s) to be performed as a result of the transition

It may be noted that if a transition is triggered with one or more guard-condition(s), which evaluates to false, the system will continue to stay in the present state. Also, not all transitions do result in a state change. For example, if a queue is full, any further attempt to append will fail until the delete method is invoked at least once. Thus, the state of the queue doesn't change in this duration.

Action

An action represents behavior of the system. While the system is performing any action for the current event, it doesn't accept or process any new event. The order, in which different actions are executed, is given below:

1. Exit actions of the present state
2. Actions specified for the transition
3. Entry actions of the next state

Activity Diagrams

Activity diagrams fall under the category of behavioral diagrams in Unified Modeling Language. It is a high level diagram used to visually represent the flow of control in a system. It has similarities with traditional flow charts. However, it is more powerful than a simple flow chart since it can represent various other concepts like concurrent activities, their joining, and so on.

Activity diagrams, however, cannot depict the message passing among related objects. As such, it can't be directly translated into code. These kinds of diagrams are suitable for confirming the logic to be implemented with the business users. These diagrams are typically used when the business logic is complex. In simple scenarios it can be avoided entirely.

Components of an Activity Diagram

Below we describe the building blocks of an activity diagram.

Activity

An activity denotes a particular action taken in the logical flow of control. This could simply be invocation of a mathematical function, alter an object's properties and so on. An activity is represented with a rounded rectangle, as shown in table-01. A label inside the rectangle identifies the corresponding activity.

There are two special types of activity nodes: initial and final. They are represented with a filled circle, and a filled in circle with a border respectively (table-01). Initial node represents the starting point of a flow in an activity diagram. There could be multiple initial nodes, which means that invoking that particular activity diagram would initiate multiple flows.

A final node represents the end point of all activities. Like an initial node, there could be multiple final nodes. Any transition reaching a final node would stop all activities.

Flow

A flow (also termed as edge or transition) is represented with a directed arrow. This is used to depict transfer of control from one activity to another, or to other types of components, as we will see below. A flow is often accompanied with a label, called the guard condition, indicating the necessary condition for the transition to happen. The syntax to depict it is [guard condition].

Decision

A decision node, represented with a diamond, is a point where a single flow enters and two or more flows leave. The control flow can follow only one of the outgoing paths. The outgoing edges often have guard conditions indicating true-false or if-then-else conditions. However, they can be omitted in obvious cases. The input edge could also have guard conditions. Alternatively, a note can be attached to the decision node indicating the condition to be tested.

Merge

This is represented with a diamond shape, with two or more flows entering, and a single flow leaving out. A merge node represents the point where at least a single control should reach before further processing could continue.

Fork

Fork is a point where parallel activities begin. For example, when a student has been registered with a college, he can in parallel apply for student ID card and library card. A fork is graphically depicted with a black bar, with a single flow entering and multiple flows leaving out.

Join

A join is depicted with a black bar, with multiple input flows, but a single output flow. Physically it represents the synchronization of all concurrent activities. Unlike a merge, in case of a join all of the incoming controls **must be completed** before any further progress could be made. For example, a sales order is closed only when the customer has received the product, **and** the sales company has received its payment.

Note

UML allows attaching a note to different components of a diagram to present some textual information. The information could simply be a comment or may be some constraint. A note can be attached to a decision point, for example, to indicate the branching criteria.

Partition

Different components of an activity diagram can be logically grouped into different areas, called partitions or swim lanes. They often correspond to different units of an organization or different actors. The drawing area can be partitioned into multiple compartments using vertical (or horizontal) parallel lines. Partitions in an activity diagram are not mandatory. The following table shows commonly used components with a typical activity diagram.

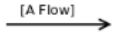
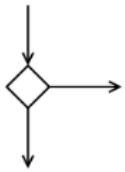
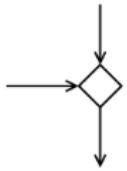
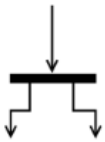
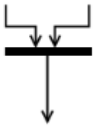
Component	Graphical Notation
Activity	
Flow	
Decision	
Merge	
Fork	
Join	
Note	

Table-01: Typical components used in an activity diagram

A Simple Example

Figure-04 shows a simple activity diagram with two activities. The figure depicts two stages of a form submission. At first a form is filled up with relevant and correct information. Once it is verified that there is no error in the form, it is then submitted. The two other symbols shown in the figure are the initial node (dark filled circle), and final node (outer hollow circle with inner filled circle). It may be noted that there could be zero or more final node(s) in an activity diagram.

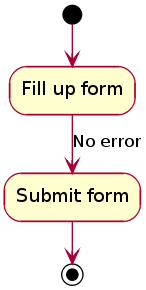


Figure-04: A simple activity diagram

Procedure:

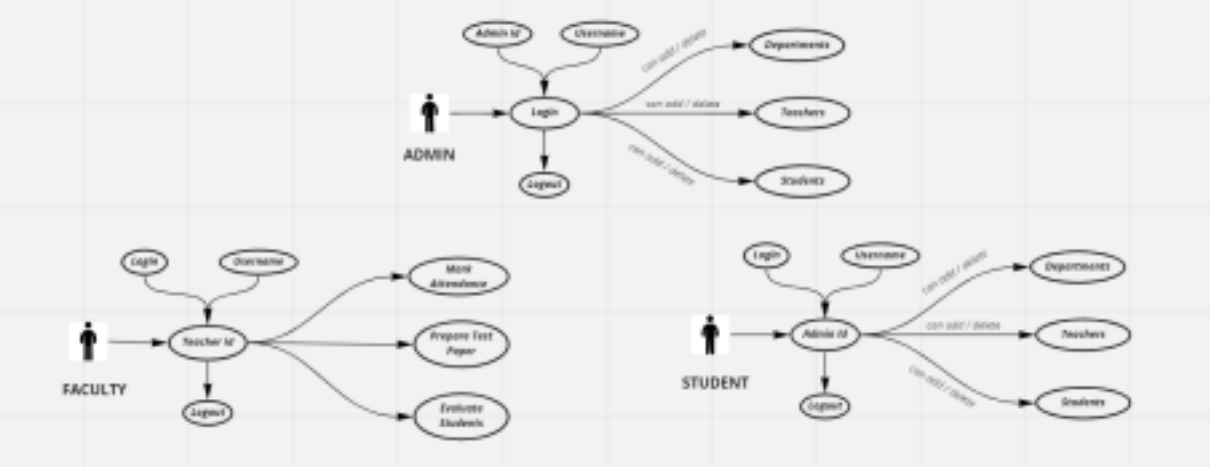
Guidelines for drawing State chart Diagrams

Following steps could be followed, to draw a state chart diagram:

- For the system to developed, identify the distinct states that it passes through
- Identify the events (and any precondition) that cause the state transitions. Often these would be the methods of a class as identified in a class diagram.
- Identify what activities are performed while the system remains in a given state

1. Activity Diagram

2. State Transition Diagram



State transition diagram – Login Activity
(ERP System)

Result and Discussion:

Q.1) What is a dynamic view of a system? Draw at least one state diagram and one activity diagram for your mini project.

Learning Outcomes: The student should have the ability to:

LO 1: Identify the importance of a state diagram.

LO 2: Draw activity diagrams for a given scenario.

Course Outcomes: Upon completion of the course students will be able to understand and demonstrate state and activity diagrams.

Conclusion: Thus, students have understood and successfully drawn state and activity diagrams.

Viva Questions:

1. What is a state diagram used for?
2. Enumerate the steps to draw an activity diagram.

For Faculty Use:

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				

Experiment 7: Sketch Sequence and Collaboration diagram for the project

Learning Objective: Students will able to draw Sequence and Collaboration diagram for the project

Tools: Dia, StarUML

Theory:

A sequence diagram shows, as parallel vertical lines (*lifelines*), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

Sequence Diagram representation

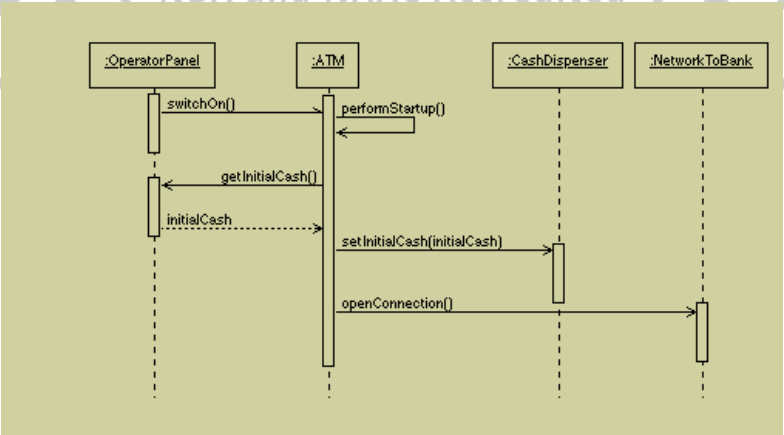
Call Message: A message defines a particular communication between Lifelines of an Interaction.

Destroy Message: Destroy message is a kind of message that represents the request of destroying the lifecycle of the target lifeline.

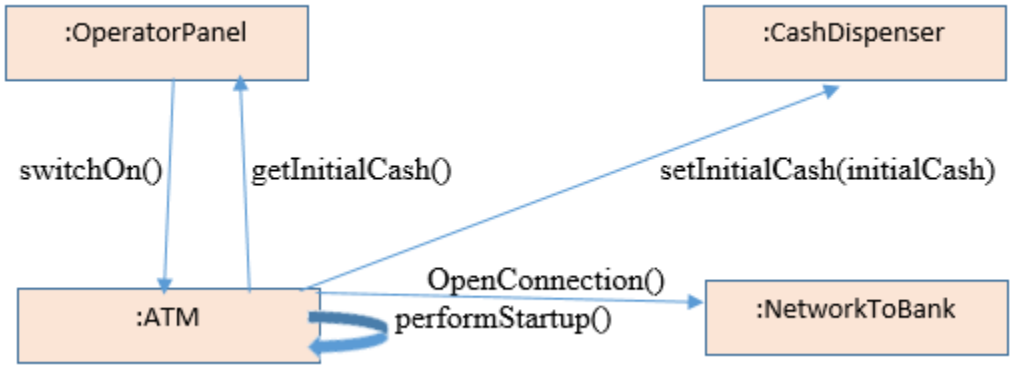
LifeLine: A lifeline represents an individual participant in the Interaction.

Recursive Message: Recursive message is a kind of message that represents the invocation of a message of the same lifeline. Its target points to an activation on top of the activation where the message was invoked from.

Sequence Diagram: Example for ATM System startup



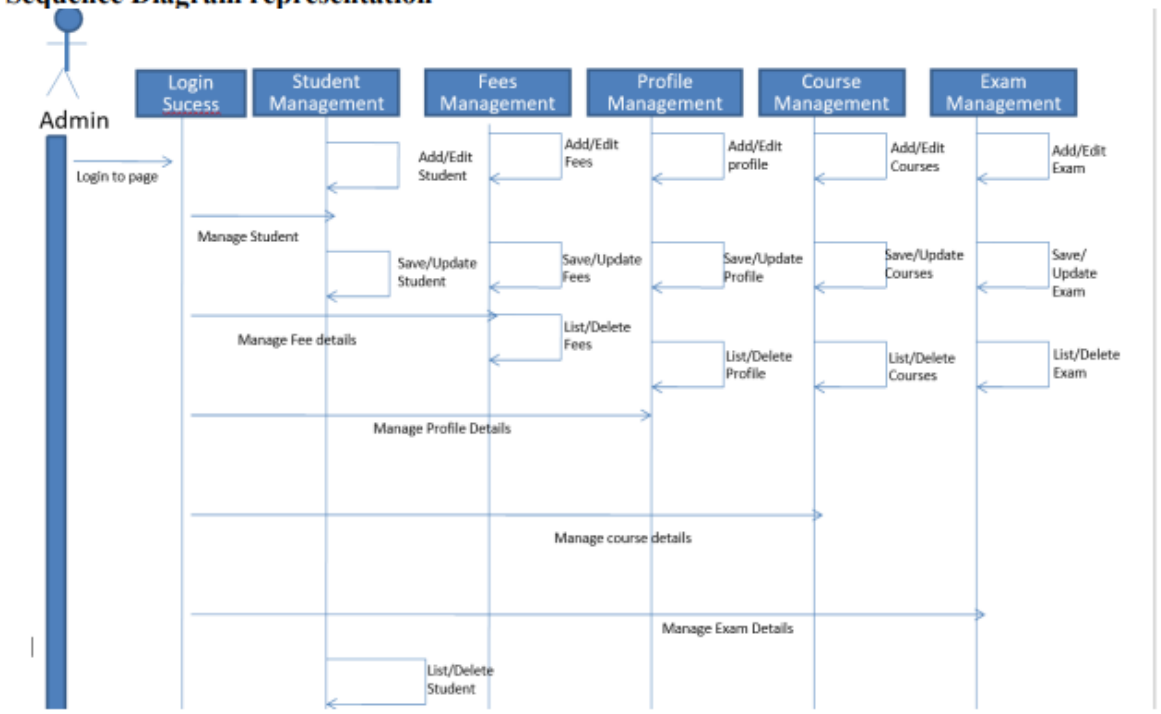
Collaboration diagram for ATM System startup

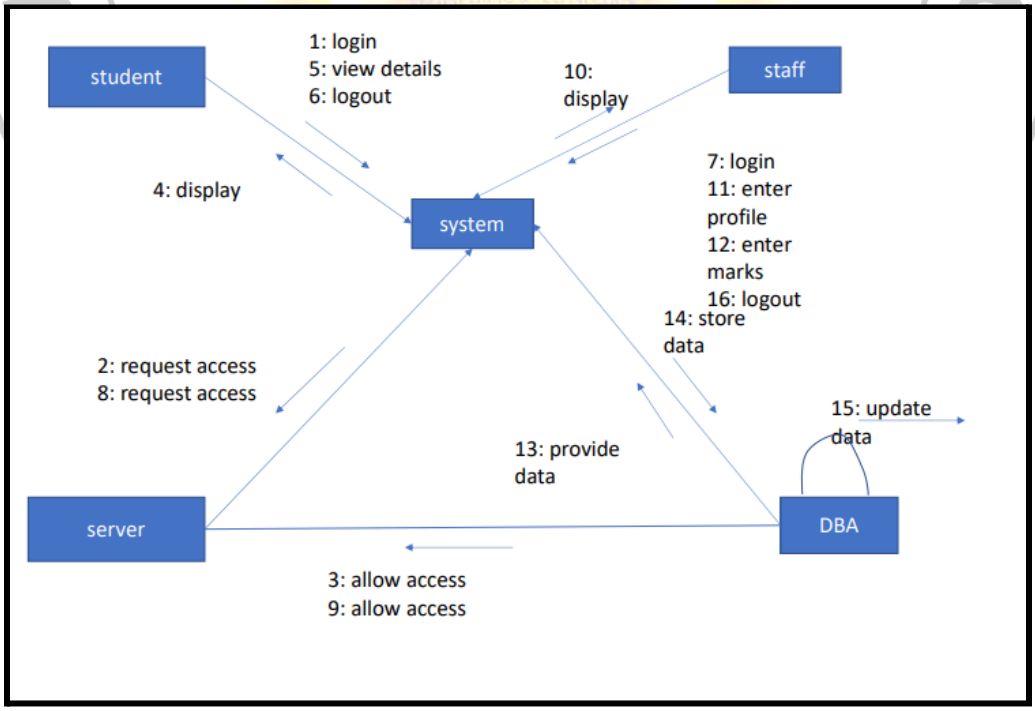


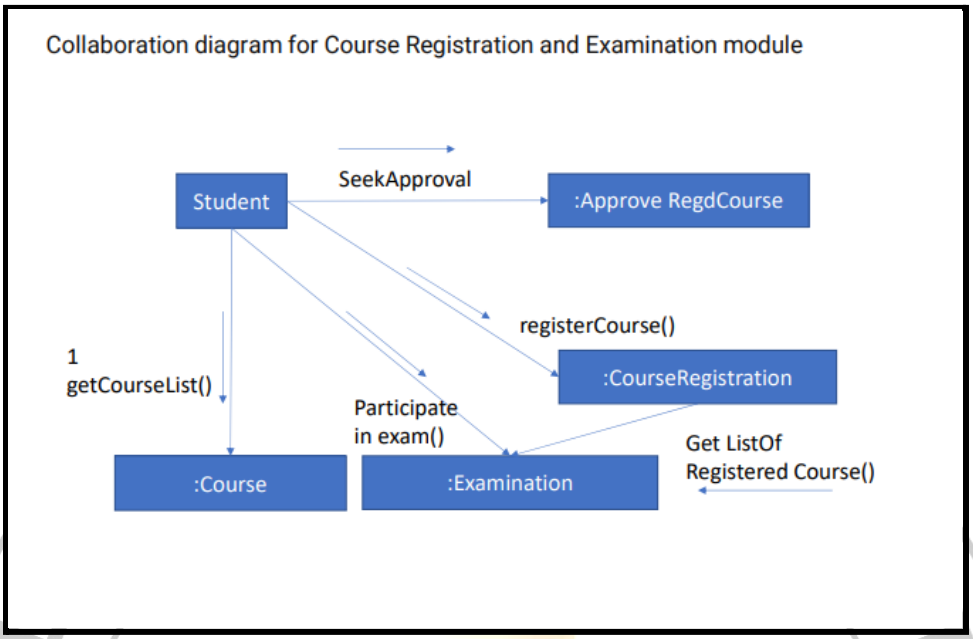
It is clear that sequence charts have a number of very powerful advantages. They clearly depict the sequence of events, show when objects are created and destroyed, are excellent at depicting concurrent operations, and are invaluable for hunting down race conditions. However, with all their advantages, they are not perfect tools. They take up a lot of space, and do not present the interrelationships between the collaborating objects very well.

A collaboration diagram, also known as a communication diagram, depicts the relationships and interactions among software objects in the UML diagrams.
Collaboration diagrams are best suited to the portrayal of simple interactions among relatively small numbers of objects.
Collaboration diagrams are used to visualize the structural organization of objects and their interactions. Sequence diagrams focus on the order of messages that flow between objects.

Sequence Diagram representation







Learning Outcomes: Students should have the ability to

LO1: Identify the classes and objects.
LO2: Identify the interactions between the objects
LO3: Develop a sequence diagram for different scenarios
LO4: generate the collaboration diagram

Outcomes: Upon completion of the course students will be able to draw the sequence and collaboration diagram for the project.

Conclusion: Successfully sketched the Sequence diagram and Collaboration diagram

Viva Questions:

1. What is a sequence diagram
2. Difference between sequence and collaboration diagram?
3. What are entities in a sequence diagram?
4. Explain its relation with the class diagram?

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				

Experiment 8: Change specification and use any SCM Tool to make different versions for the project

Learning Objective: Students will able to create versions using Github tool

Tools: Github

Theory:

Software configuration management: The traditional software configuration management (SCM) process is looked upon by practitioners as the best solution to handling changes in software projects. It identifies the functional and physical attributes of software at various points in time, and performs systematic control of changes to the identified attributes for the purpose of maintaining software integrity and traceability throughout the software development life cycle.

Software configuration management is a part of software engineering, which focuses mainly on maintaining, tracking and controlling the changes done to the software configuration items. Configuration management is present in all phases of software development. The configuration items can be all the objects which come as an output of the development process e.g. coding phase produces source code, exes and obj files. The various configuration items can be:

1. Source code,

2. Documents

3. Data used in the programs

In Configuration management, there can be multiple versions created for any configuration item (Source code/ documents). Each version can be identified by unique configuration or an attribute which is associated with each version. E.g. the version number.

Terminologies used in version control

1. SCI – Software configuration items, i.e. the documents and code which will be having version number and saved.

2. Repository- it is the system where all the SCIs will be stored.

3. Check in- to store the tested and qualified source code.

4. Checkout- to get a copy of the stored SCI from the repository.

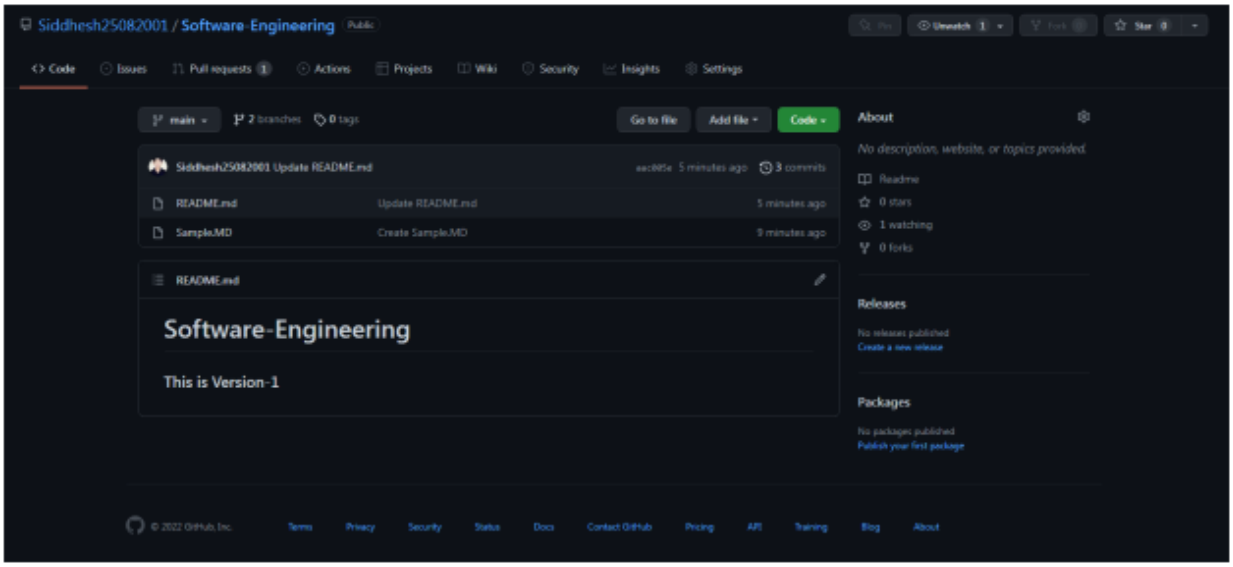
5. Add – Add to the local repo and keep ready for commit

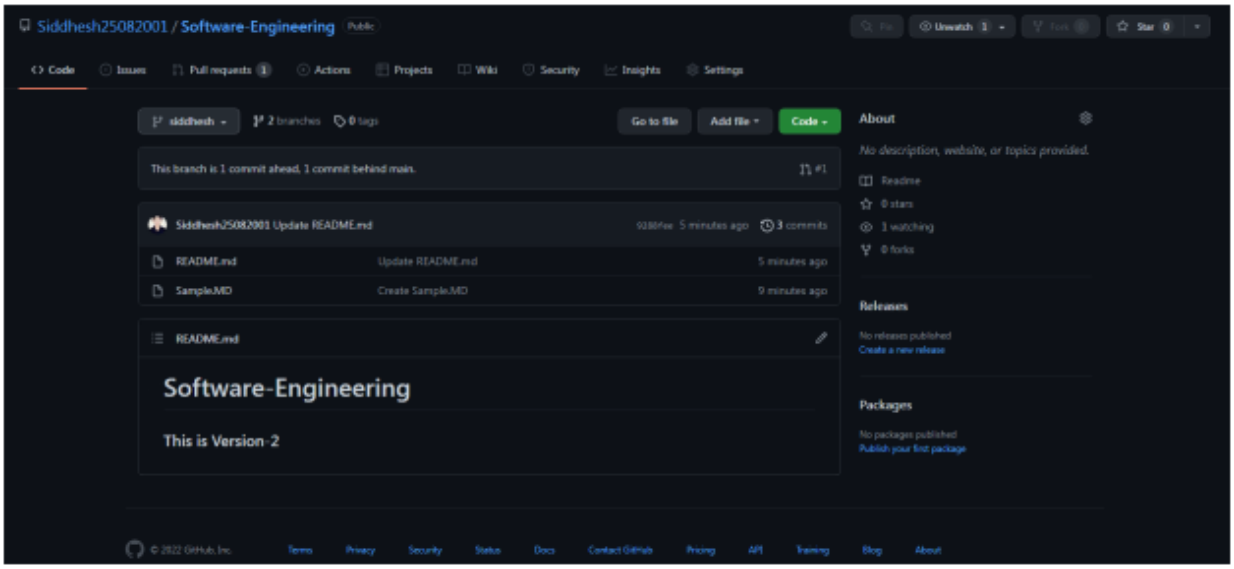
6. Commit- to save the file in repository and create a version

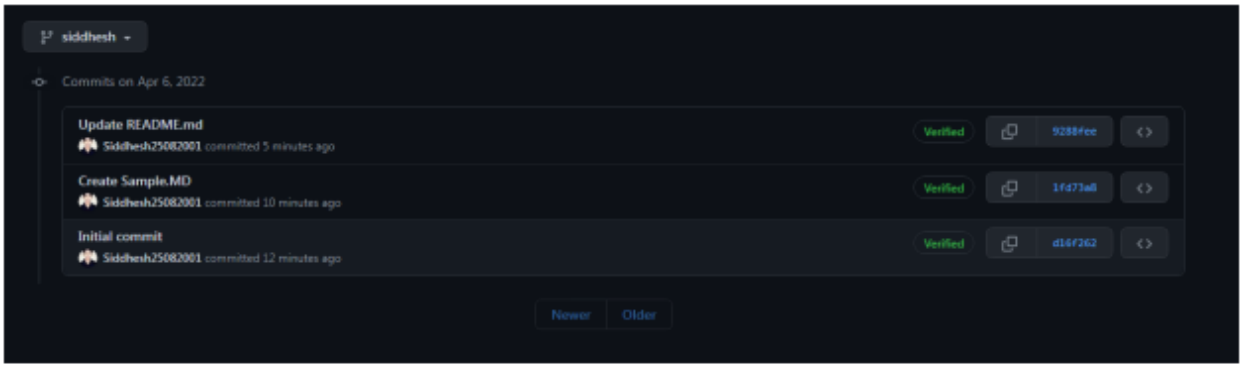
Advantages

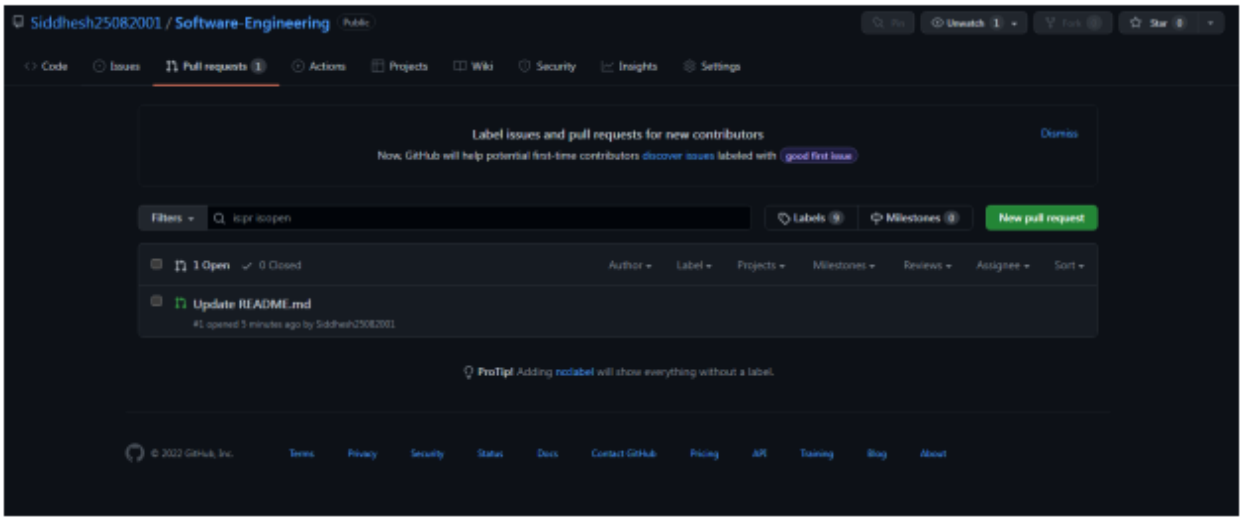
1. The versions are stored in the repository; hence they are available as backups.
2. Multiple people can work simultaneously on same files/source code, without losing the changes made by other developers
3. It is easy to find the files with specifications as a versions are stored with version numbers

GitHub offers all of the distributed revision control and source code management (SCM) functionality of Git as well as adding its own features. Unlike Git, which is strictly a command-line tool, GitHub provides a Web-based graphical interface and desktop as well as mobile integration. It also provides access control and several collaboration features such as bug tracking, feature requests, task management for every project.




Siddhesh25082001 / Software-Engineering
Public
Unwatch 1
Star 0
Code
Issues
Pull requests 1
Actions
Projects
Wiki
Security
Insights
Settings
siddhesh
2 branches
0 tags
Go to file
Add file
Code
This branch is 1 commit ahead, 1 commit behind main.
Siddhesh25082001 Update README.md
9288fee 5 minutes ago
3 commits
README.md
Update README.md
5 minutes ago
SampleMD
Create SampleMD
9 minutes ago
README.md
Software-Engineering
This is Version-2
About
No description, website, or topics provided.
Readme
0 stars
1 watching
0 forks
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
© 2022 GitHub, Inc.
Terms
Privacy
Security
Status
Docs
Contact GitHub
Pricing
API
Training
Blog
About


siddhesh
Commits on Apr 6, 2022
Update README.md
Siddhesh25082001 committed 5 minutes ago
Verified
9288fee
Create SampleMD
Siddhesh25082001 committed 10 minutes ago
Verified
1fd73a8
Initial commit
Siddhesh25082001 committed 12 minutes ago
Verified
d16f262
Newer
Older


Siddhesh25082001 / Software-Engineering
Public
Unwatch 1
Star 0
Code
Issues
Pull requests 1
Actions
Projects
Wiki
Security
Insights
Settings
Label issues and pull requests for new contributors
Now, GitHub will help potential first-time contributors discover issues labeled with good first issue
Dismiss
Filters
is:pr is:open
Labels 9
Milestones 0
New pull request
1 Open
0 Closed
Author
Label
Projects
Milestones
Reviews
Assignee
Sort
Update README.md
#1 opened 5 minutes ago by Siddhesh25082001
ProTip! Adding no:label will show everything without a label.
© 2022 GitHub, Inc.
Terms
Privacy
Security
Status
Docs
Contact GitHub
Pricing
API
Training
Blog
About

Learning Outcomes: Students should have the ability to

LO1: to understand the need of doing configuration management.
LO2: Identify the dissimilarity between version and variant
LO3: provide the knowledge of the benefits of using version control
LO4: To understand the types of version control system

Outcomes: Upon completion of the course students will be able to create versions for the project.

Conclusion: Successfully able to create versions using Github tool

Viva Questions:

1. What is the difference between git and Github?
2. What is version control? Why is it required?
3. What are other tools for version control?
4. What are different types of version control?

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				

Experiment 9: Apply the knowledge of test cases for the project using white box testing.

Learning Objective: Students will able to create unit test cases

Tools: Junit

Theory:

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing also provides an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation.

Unit Testing:

Unit testing focuses on the building blocks of the software system, that is, objects and subsystems. The specific candidates for unit testing are chosen from the object model and the system decomposition. In principle, all the objects developed during the development process should be tested, which is often not feasible because of time and budget constraints. The minimal set of objects to be tested should be the participating objects in the use cases. Subsystems should be tested after each of the objects and classes within that subsystem have been tested individually. Unit testing focuses verification effort on the smallest unit of software design—the software component or module.The unit test is white-box oriented. . In Unit testing the following are tested,

1. The module interface is tested to ensure that information properly flows into and out of the program unit under test.
2. The local data structure is examined to ensure that data stored temporarily maintains its integrity.
3. Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing.
4. All independent paths through the control structure are exercised to ensure that all statements in a module have been executed at least once.
5. And finally, all error handling paths are tested

Write a program to calculate the square of a number in the range 1-100

```
#include <stdio.h>

int main()

{

   int n, res;

   printf("Enter a number: ");

   scanf("%d", &n);

   if (n >= 1 && n <= 100)

   {

         res = n * n;

         printf("\n Square of %d is %d\n", n, res);

   }

   else if (n<= 0 || n > 100)

         printf("Beyond the range");

   return 0;

}
```

Sr no	Input	Output
1	-2	Beyond the range
2	0	Beyond the range
3	1	Square of 1 is 1
4	100	Square of 100 is 10000
5	101	Beyond the range
6	4	Square of 4 is 16
7	62	Square of 62 is 3844

Test Cases

Test case 1 : {I1 ,O1}

Test case 2 : {I2 ,O2}

Test case 3 : {I3, O3}

Test case 4 : {I4, O4}

Test case 5 : {I5, O5}

Test case 6 : {I6, O6}

Test case 7 : {I7, O7}

Learning Outcomes: Students should have the ability to

LO1: Students will be able to understand Software Testing Concepts and the various Software standards.
LO2: to test a software with the help of Junit
LO3: create test cases
LO4: To understand different tools for testing

Outcomes: Upon completion of the course students will be able to write test cases for the project.

Conclusion: Successfully able to create unit test cases

Viva Questions:

5. What is the difference between git and Github?
6. What is version control? Why is it required?
7. What are other tools for version control?
8. What are different types of version control?

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				