

A Laboratory Manual for

SOFTWARE ENGINEERING

(3150711)

B.E. Semester V

(Department of Computer Engineering)



Directorate of Technical Education, Gandhinagar,
Gujarat.

Government Engineering College, Modasa

Certificate

This is to certify that Mr./Ms. _____
_____ Enrollment No. _____ of B.E. Semester _____ Computer
Engineering of this Institute (GTU Code: 016) has satisfactorily completed the
Practical / Tutorial work for the subject Software Engineering (3150711) for the
academic year 2022-23.

Place: _____

Date: _____

Name and Sign of Faculty member

Head of the Department

Preface

The main motto of any laboratory/practical/field work is to enhance required skills and create ability amongst students to solve real-time problems by developing relevant competencies in the psychomotor domain. By keeping this in view, GTU has designed a competency-focused outcome based curriculum for engineering degree programs where sufficient weightage is given to practical work. It shows the importance of enhancement of skills amongst the students, and it pays attention to utilizing every second of time allotted for practicals amongst students, instructors, and faculty members to achieve relevant outcomes by performing the experiments rather than merely study-type experiments. It is a must for the effective implementation of a competency-focused outcome-based curriculum that every practical is keenly designed to serve as a tool to develop and enhance relevant competency required by the various industry among every student. These psychomotor skills are very difficult to develop through traditional chalk-and-board content delivery methods in the classroom. Accordingly, this lab manual is designed to focus on industry-defined relevant outcomes rather than the old practice of conducting practicals to prove concepts and theories.

By using this lab manual, students can go through the relevant theory and procedure in advance before the actual performance, which creates interest, and students can have a basic idea prior to the performance. This, in turn, enhances pre-determined outcomes amongst students. Each experiment in this manual begins with competency, industry-relevant skills, course outcomes as well as practical outcomes (objectives). The students will also achieve safety and necessary precautions to be taken while performing practicals.

This manual also provides guidelines to faculty members to facilitate student-centric lab activities through each experiment by arranging and managing necessary resources in order that the students follow the procedures with required safety and necessary precautions to achieve the outcomes. It also gives an idea of how students will be assessed by providing rubrics.

Software Engineering is an application of a systematic, defined, and measurable approach that begins with requirement specification and progresses with planning, modeling, and testing, and concludes with deployment. It is a layered paradigm that comprises processes, methods, and tools with the bedrock of quality focus. The Software Engineering approach's main purpose is committed to developing the software products within the stipulated time and budget with more quality. Quality product motivates firmness, commodity, and delight.

Utmost care has been taken while preparing this lab manual; however, there is always a chance of improvement. Therefore, we welcome constructive suggestions for improvement and removal of errors, if any.

Practical – Course Outcome matrix

<p>Course Outcomes (COs):</p> <p>CO-1: Prepare SRS (Software Requirement Specification) document and SPMP (Software Project Management Plan) document.</p> <p>CO-2: Apply the concept of Functional Oriented and Object-Oriented Approaches for Software Design.</p> <p>CO-3. Recognize how to ensure the quality of software products, different quality standards, and software review techniques.</p> <p>CO-4. Apply various testing techniques and test plans in.</p> <p>CO-5. Able to understand modern Agile Development</p>						
Sr. No.	Objective(s) of Experiment	CO1	CO2	CO3	CO4	CO5
1.	Define the Problem statement of Software development, and after identifying the requirements based on the requirement engineering tasks, prepare the Software Requirement Specification (SRS) document.	√				√
2.	Prepare the Software Project Management Plan (SPMP), including the following: <ul style="list-style-type: none"> • Estimation of Size, Cost, Duration, Effort • Prepare the Schedule, Milestones using the Gantt chart 	√				√
3.	Prepare the following components of the Data Flow Model: <ul style="list-style-type: none"> • Data Dictionary • Data Flow Diagram Structure Chart 		√			√
4.	Prepare the user's view analysis: Describe different scenarios and Draw Use case diagrams using UML		√			√
5.	Prepare the structural view: Draw the Class diagram and object diagram.		√			√
6.	Prepare the behavioral view: Draw a Sequence diagram and a Collaboration diagram.		√			√
7.	Prepare the behavioral view: Draw a State-chart diagram and Activity diagram.		√			√
8.	Prepare the implementation view: Draw the Component and Deployment diagram.		√			√
9.	Prepare the quality management plan.			√		√
10.	To perform various testing using the testing tool unit testing, integration testing design test cases.				√	√
11	Case Study: Study of Open-source tools in DevOps for Infrastructure Automation, Configuration Management, Deployment Automation, Performance Management, and Log Management. Monitoring					√

Index (Progressive Assessment Sheet)

Sr. No.	Objective(s) of Experiment	Page No.	Date of performance	Date of submission	Assessment Marks	Sign. of Teacher with date	Remarks
1	Define the Problem statement of Software development and after identifying the requirements based on the requirement engineering tasks prepare the Software Requirement Specification (SRS) document.						
2	Prepare the Software Project Management Plan (SPMP) including following: <ul style="list-style-type: none"> • Estimation of Size, Cost, Duration, Effort • Prepare the Schedule, Milestones using Gantt chart 						
3	Prepare the following components of Data Flow Model: <ul style="list-style-type: none"> • Data Dictionary • Data Flow Diagram Structure Chart 						
4	Prepare the user's view analysis: Describe different scenarios and Draw Use case diagrams using UML						
5	Prepare the structural view: Draw Class diagram and object diagram using UML.						
6	Prepare the behavioural view: Draw Sequence diagram and Collaboration diagram using UML.						
7	Prepare the behavioural view: Draw State-chart diagram and Activity diagram using UML.						
8	Prepare the implementation view: Draw Component and Deployment diagram using UML.						
9.	Prepare the quality management plan.						
10.	To perform various testing using the testing tool unit testing, integration testing design test cases.						
11.	Case Study: Study of Open-source tools in DevOps for Infrastructure Automation, Configuration Management, Deployment Automation, Performance Management, Log Management. Monitoring						
Total							

GUJARAT TECHNOLOGICAL UNIVERSITY, AHMEDABAD,
COURSE CURRICULUM

COURSE TITLE: Software Engineering (Code: 3150711)

Degree Program in which this course is offered	Semester in which offered
Computer Engineering	5 th Semester

❖ **RATIONALE**

- To study Software Development Life Cycle, Development models and Agile Software development.
- To study fundamental concepts in software testing, including software testing objectives, process, criteria, strategies, and methods.
- To discuss various software testing issues and solutions in software unit test; integration, regression, and system testing.
- To learn the process of improving the quality of software work products.
- To gain the techniques and skills on how to use modern software testing tools to support software testing projects.
- To expose Software Process Improvement and Reengineering

2. COMPETENCY

- The course content should be taught and analyze with the aim to develop different types of skills so that students are able to acquire following competency:
- Software engineering models to development of the complex engineering software based on the software development life cycle.

3. COURSE OUTCOMES

After learning the course, the students should be able to:

1. Prepare SRS (Software Requirement Specification) document and SPMP (Software Project Management Plan) document.
2. Apply the concept of Functional Oriented and Object-Oriented Approach for Software Design.
3. Recognize how to ensure the quality of software product, different quality standards and software review techniques.
4. Apply various testing techniques and test plan in.
5. Able to understand modern Agile Development.

4. TEACHING AND EXAMINATION SCHEME

Teaching and Examination Scheme:

Teaching Scheme			Credits	Examination Marks				Total Marks
L	T	P	C	Theory Marks		Practical Marks		
				ESE (E)	PA (M)	ESE (V)	PA (I)	
3	0	2	4	70	30	30	20	150

5. SUGGESTED LEARNING RESOURCES

A. LIST OF BOOKS

1. Roger S.Pressman, Software Engineering- A practitioner's Approach, McGraw-Hill International Editions
2. Ian Sommerville, Software engineering, Pearson education Asia
3. Pankaj Jalote, Software Engineering – A Precise Approach Wiley
4. Behhforoz & Frederick Hudson, Software Engineering Fundamentals, OXFORD
5. Rajib Mall, Fundamentals of software Engineering, Prentice Hall of India.
6. Deepak Gaikwad, Viral Thakkar, DevOps Tools from Practitioner's ViewPoint, Wiley
7. Merlin Dorfman (Editor), Richard H. Thayer (Editor), Software Engineering
8. Robert C. "Uncle Bob" Martin, Clean Architecture: A Craftsman's Guide to Software Structure and Design

B. LIST OF SOFTWARE / LEARNING WEBSITES

- www.onesmartclick.com/engsineering/software-engineering.html
- www.sei.cmu.edu
- <https://www.edx.org/school/uc-berkeleyx>
- <https://devops.com/most-popular-open-source-devops-tools/> □ <https://www.guru99.com/devops-tutorial.html>

Industry Relevant Skills

The following industry relevant competency are expected to be developed in the student by undertaking the practical work of this laboratory.

1. Apply knowledge of Machine Learning to solve real world problems
2. Understand and analyze the data analytically to use them wisely to build the ML model

Guidelines for Faculty members

1. Teacher should provide the guideline with demonstration of practical to the students with all features.
2. Teacher shall explain basic concepts/theory related to the experiment to the students before starting of each practical
3. Involve all the students in performance of each experiment.
4. Teacher is expected to share the skills and competencies to be developed in the students and ensure that the respective skills and competencies are developed in the students after the completion of the experimentation.

5. Teachers should give opportunity to students for hands-on experience after the demonstration.
6. Teacher may provide additional knowledge and skills to the students even though not covered in the manual but are expected from the students by concerned industry.
7. Give practical assignment and assess the performance of students based on task assigned to check whether it is as per the instructions or not.
8. Teacher is expected to refer complete curriculum of the course and follow the guidelines for implementation.

Instructions for Students

1. Students are expected to carefully listen to all the theory classes delivered by the faculty members and understand the COs, content of the course, teaching and examination scheme, skill set to be developed etc.
2. Students shall organize the work in the group and make record of all observations.
3. Students shall develop maintenance skill as expected by industries.
4. Student shall attempt to develop related hand-on skills and build confidence.
5. Student shall develop the habits of evolving more ideas, innovations, skills etc. apart from those included in scope of manual.
6. Student shall refer technical magazines and data books.
7. Student should develop a habit of submitting the experimentation work as per the schedule and s/he should be well prepared for the same.

General Guidelines for Software Engineering Laboratory Work

1. Student has to perform all the practical as described in the practical list.
2. For performing the practical list, student can able to work individually or make a team of maximum 3 students' group.
3. For making the group, students must take care that the all the students of group must be from same batch.
4. After establishing the team, every team will have to identify the problem area / definition for performing the laboratory work.
5. Every team has to approve their problem definition to respective faculty member within 15 days of the beginning of the semester.
6. Once the problem definition is approved by the faculty member, every team has to perform all the practical based on their respective problem definition.

Practical – 1

AIM: Define the Problem statement of Software development and after identifying the requirements based on the requirement engineering tasks prepare the Software Requirement Specification (SRS) document.

- **Objectives:** To learn how to define the problem statement for software development and how to identify the requirements while performing requirements engineering tasks and to learn what are the contents of SRS and how to write the contents in SRS.
- **Theory:**
 - The process of collecting the software requirement from the client then understand, evaluate and document it is called as requirement engineering.
 - Requirement engineering constructs a bridge for design and construction.

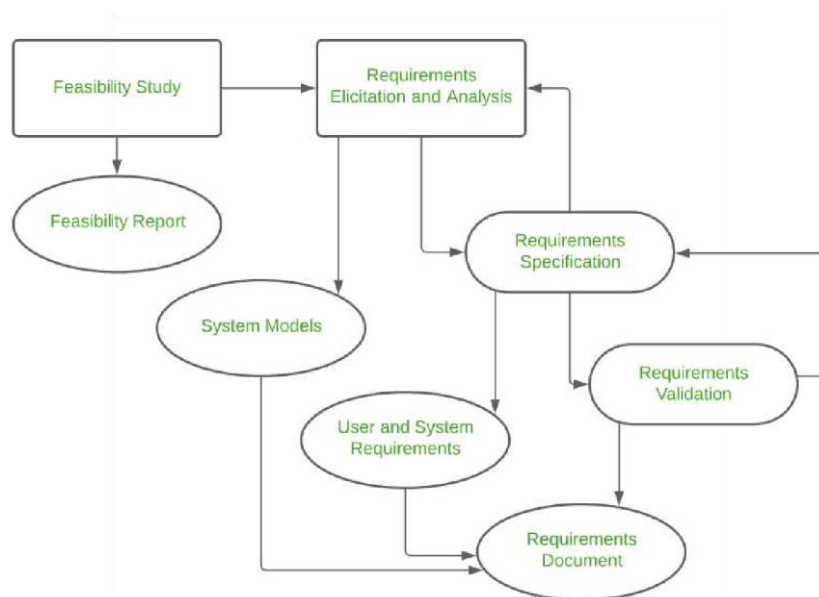


Figure: Requirements Engineering Process

- The production of the requirements stage of the software development process is Software Requirements Specifications (SRS) (also called a requirements document). This report lays a foundation for software engineering activities and is constructed when entire requirements are elicited and analysed. SRS is a formal report, which acts as a representation of software that enables the customers to review whether it (SRS) is according to their requirements. Also, it

comprises user requirements for a system as well as detailed specifications of the system requirements.

- The SRS is a specification for a specific software product, program, or set of applications that perform particular functions in a specific environment. It serves several goals depending on who is writing it. First, the SRS could be written by the client of a system. Second, the SRS could be written by a developer of the system. The two methods create entirely various situations and establish different purposes for the document altogether. The first case, SRS, is used to define the needs and expectation of the users. The second case, SRS, is written for various purposes and serves as a contract document between customer and developer.

- **Characteristics of good SRS**

1. Correctness
2. Completeness
3. Consistency
4. Unambiguousness
5. Ranking for importance and stability
6. Modifiability
7. Verifiability
8. Traceability
9. Design Independence
10. Testability
11. Understandable by the customer
12. The right level of abstraction

The following are the features of a good SRS document:

1. **Correctness:** User review is used to provide the accuracy of requirements stated in the SRS. SRS is said to be perfect if it covers all the needs that are truly expected from the system.
2. **Completeness:** The SRS is complete if, and only if, it includes the following elements:
 - (1). All essential requirements, whether relating to functionality, performance, design, constraints, attributes, or external interfaces.
 - (2). Definition of their responses of the software to all realizable classes of input data in all available categories of situations.
 - (3). Full labels and references to all figures, tables, and diagrams in the SRS and definitions of all terms and units of measure.

- 3. Consistency:** The SRS is consistent if, and only if, no subset of individual requirements described in it conflict. There are three types of possible conflict in the SRS:

- (1). The specified characteristics of real-world objects may conflict. For example, (a) The format of an output report may be described in one requirement as tabular but in another as textual. (b) One condition may state that all lights shall be green while another states that all lights shall be blue.
- (2). There may be a reasonable or temporal conflict between the two specified actions.

For example,

- (a) One requirement may determine that the program will add two inputs, and another determine that the program will multiply them.
 - (b) One condition may state that "A" must always follow "B," while another requires that "A and B" co-occurs.
- (3). Two or more requirements may define the same real-world object but use different terms for that object. For example, a program's request for user input may be called a "prompt" in one requirement's and a "cue" in another. The use of standard terminology and descriptions promotes consistency.

- 4. Unambiguousness:** SRS is unambiguous when every fixed requirement has only one interpretation. This suggests that each element is uniquely interpreted. In case there is a method used with multiple definitions, the requirements report should determine the implications in the SRS so that it is clear and simple to understand.
- 5. Ranking for importance and stability:** The SRS is ranked for importance and stability if each requirement in it has an identifier to indicate either the significance or stability of that particular requirement.

Typically, all requirements are not equally important. Some prerequisites may be essential, especially for life-critical applications, while others may be desirable. Each element should be identified to make these differences clear and explicit. Another way to rank requirements is to distinguish classes of items as essential, conditional, and optional.

- 6. Modifiability:** SRS should be made as modifiable as likely and should be capable of quickly obtaining changes to the system to some extent. Modifications should be perfectly indexed and cross-referenced.
- 7. Verifiability:** SRS is correct when the specified requirements can be verified with a cost effective system to check whether the final software meets those requirements. The requirements are verified with the help of reviews.
- 8. Traceability:** The SRS is traceable if the origin of each of the requirements is clear and if it facilitates the referencing of each condition in future development or enhancement documentation.

There are two types of Traceability:

1. Backward Traceability: This depends upon each requirement explicitly referencing its source in earlier documents.

2. Forward Traceability: This depends upon each element in the SRS having a unique name or reference number.

The forward traceability of the SRS is especially crucial when the software product enters the operation and maintenance phase. As code and design document is modified, it is necessary to be able to ascertain the complete set of requirements that may be concerned by those modifications.

9. **Design Independence:** There should be an option to select from multiple design alternatives for the final system. More specifically, the SRS should not contain any implementation details.
10. **Testability:** An SRS should be written in such a method that it is simple to generate test cases and test plans from the report.
11. **Understandable by the customer:** An end user may be an expert in his/her explicit domain but might not be trained in computer science. Hence, the purpose of formal notations and symbols should be avoided too as much extent as possible. The language should be kept simple and clear.
12. **The right level of abstraction:** If the SRS is written for the requirements stage, the details should be explained explicitly. Whereas, for a feasibility study, fewer analysis can be used. Hence, the level of abstraction modifies according to the objective of the SRS.

Properties of a good SRS document

The essential properties of a good SRS document are the following:

Concise: The SRS report should be concise and at the same time, unambiguous, consistent, and complete. Verbose and irrelevant descriptions decrease readability and also increase error possibilities.

Structured: It should be well-structured. A well-structured document is simple to understand and modify. In practice, the SRS document undergoes several revisions to cope up with the user requirements. Often, user requirements evolve over a period of time. Therefore, to make the modifications to the SRS document easy, it is vital to make the report well-structured.

Black-box view: It should only define what the system should do and refrain from stating how to do these. This means that the SRS document should define the external behaviour of the system and not discuss the implementation issues. The SRS report should view the system to be developed as a black box and should define the

externally visible behaviour of the system. For this reason, the SRS report is also known as the black-box specification of a system.

Conceptual integrity: It should show conceptual integrity so that the reader can merely understand it. **Response to undesired events:** It should characterize acceptable responses to unwanted events. These are called system responses to exceptional conditions.

Verifiable: All requirements of the system, as documented in the SRS document, should be correct. This means that it should be possible to decide whether or not requirements have been met in an implementation.

- **Background / Preparation:**

Requirement engineering consists of seven different tasks as follows:

1. Inception

- Inception is a task where the requirement engineering asks a set of questions to establish a software process.
- In this task, it understands the problem and evaluates with the proper solution.
- It collaborates with the relationship between the customer and the developer.
- The developer and customer decide the overall scope and the nature of the question.

❖ Elicitation

- Elicitation means to find the requirements from anybody.
- The requirements are difficult because the following problems occur in elicitation. **Problem of scope:** The customer give the unnecessary technical detail rather than clarity of the overall system objective.
- **Problem of understanding:** Poor understanding between the customer and the developer regarding various aspect of the project like capability, limitation of the computing environment.
- **Problem of volatility:** In this problem, the requirements change from time to time and it is difficult while developing the project.

❖ Elaboration

- In this task, the information taken from user during inception and elaboration and are expanded and refined in elaboration.
- Its main task is developing pure model of software using functions, feature and constraints of a software.

4. Negotiation

- In negotiation task, a software engineer decides the how will the project be achieved with limited business resources.
- To create rough guesses of development and access the impact of the requirement on the project cost and delivery time.

5. Specification

- In this task, the requirement engineer constructs a final work product.
- The work product is in the form of software requirement specification.
- In this task, formalize the requirement of the proposed software such as informative, functional and behavioral.
- The requirement are formalize in both graphical and textual formats.

6. Validation

- The work product is built as an output of the requirement engineering and that is accessed for the quality through a validation step.
- The formal technical reviews from the software engineer, customer and other stakeholders helps for the primary requirements validation mechanism.

7. Requirement management

- It is a set of activities that help the project team to identify, control and track the requirements and changes can be made to the requirements at any time of the ongoing project.
- These tasks start with the identification and assign a unique identifier to each of the requirement.
- After finalizing the requirement traceability table is developed.
- The examples of traceability table are the features, sources, dependencies, subsystems and interface of the requirement.
- In order to form a [good SRS](#), here you will see some points which can be used and should be considered to form a structure of good SRS. These are as follows :

1. Introduction

- (i) Purpose of this document
- (ii) Scope of this document
- (iii) Overview

2. General description

3. Functional Requirements

4. Interface Requirements

5. Performance Requirements

6. Design Constraints

7. Non-Functional Attributes

8. Preliminary Schedule and Budget
9. Appendices

- **Tools / Material Needed:**

- Hardware:
- Software:

- **Procedure:**

- Every team has to identify the requirements of their projects and arrange as per the requirement engineering document.

- **Steps:**

- Every team has to follow the given format as described in the below example.
- **Example 1:** Withdraw Cash from ATM

R.1: withdraw cash	
Description:	The withdraw cash function first determines the type of account that the user has and the account number from which the user wishes to withdraw cash. It checks the balance to determine whether the requested amount is available in the account. If enough balance is available, it outputs the required cash, otherwise it generates an error message.
R.1.1: Select withdraw amount option	
Input:	“Withdraw amount” option
Output:	user prompted to enter the account type
R.1.2: Select account type	
Input:	user option
Output:	prompt to enter amount
R.1.3: Get required amount	
Input:	amount to be withdrawn in integer values greater than 100 and less than 10,000 in multiples of 100.
Output:	The requested cash and printed transaction statement.
Processing:	The amount is debited from the user’s account if sufficient balance is available, otherwise an error message displayed.

- **Example 2:** Search Book Availability in Library

R.1: Search Book	
Description:	Once the user selects the search option, he would be asked to enter the keywords. The system would search the book in the book list based on the keywords entered. After making the search, the system should output the details of all books whose title or author name match any of the keywords entered. The book details to be displayed include: title, author name, publisher name, year of publication, ISBN number, catalog number, and the location in the library.
R.1.1: Select search option	
Input:	“Search” option
Output:	User prompted to enter the keywords
R.1.2: Search and display	
Input:	Keywords
Output:	Details of all books whose title or author name matches any of the keywords entered by the user. The details displayed would include: title of the book, author name, ISBN number, catalog number, year of publication, number of copies available, and the location in the library.
Processing:	Search the book list based on the keywords
R.2: Renew book	
Description:	When the “renew” option is selected, the user is asked to enter his membership number and password. After password validation, the list of the books borrowed by him are displayed. The user can renew any of his borrowed books by indication them. A requested book cannot be renewed if it is reserved by another user. In this case, an error message would be displayed.
R.2.1: Select renew option	
State:	The user has logged in and the main menu has been displayed
Input:	“Renew” option selection
Output:	Prompt message to the user to enter his membership number and password
R.2.2: Login	
State:	The renew option has been selected
Input:	Membership number and password
Output:	List of the books borrowed by the user is displayed, and user is prompted to select the books to be renewed, if the password is valid. If the password is invalid, the user is asked to reenter the password.
Processing:	Password validation search the books issued to the user form the borrower’s list and display
Next function:	R.2.3 if password is valid and R.2.2 if password is invalid
R.2.3: Renew selected books	
Input:	User choice for books to be renewed out of the books borrowed by him.
Output:	Confirmation of the books successfully renewed and apology message for the books that could not be renewed.

Processing:	Check if anyone has reserved any of the requested books. Renew the books selected by the use in the borrower's list, if no one has reserved those books.
-------------	--

Software Requirement Specification (SRS) Format as name suggests, is complete specification and description of requirements of software that needs to be fulfilled for successful development of software system. These requirements can be functional as well as non-functional depending upon type of requirement. The interaction between different customers and contractor is done because its necessary to fully understand needs of customers.

Document Title
Author(s)
Affiliation
Address
Date
Document Version

Depending upon information gathered after interaction, SRS is developed which describes requirements of software that may include changes and modifications that is needed to be done to increase quality of product and to satisfy customer's demand.

1. **Introduction:**

(i) Purpose of this Document –

- At first, main aim of why this document is necessary and what's purpose of document is explained and described.

(ii) Scope of this document –

- In this, overall working and main objective of document and what value it will provide to customer is described and explained. It also includes a description of development cost and time required.

(iii) Overview –

- In this, description of product is explained. It's simply summary or overall review of product.

2. **General description:**

- In this, general functions of product which includes objective of user, a user characteristic, features, benefits, about why its importance is mentioned. It also describes features of user community.

3. **Functional Requirements:**

- In this, possible outcome of software system which includes effects due to operation of program is fully explained. All functional requirements which may include calculations, data processing, etc. are placed in a ranked order.

4. **Interface Requirements:**

- In this, software interfaces which mean how software program communicates with each other or users either in form of any language, code, or message are fully described and explained. Examples can be shared memory, data streams, etc.

5. **Performance Requirements:**

- In this, how a software system performs desired functions under specific condition is explained. It also explains required time, required memory, maximum error rate, etc.

6. Design Constraints:

- In this, constraints which simply means limitation or restriction are specified and explained for design team. Examples may include use of a particular algorithm, hardware and software limitations, etc.

7. Non-Functional Attributes:

- In this, non-functional attributes are explained that are required by software system for better performance. An example may include Security, Portability, Reliability, Reusability, Application compatibility, Data integrity, Scalability capacity, etc.

8. Preliminary Schedule and Budget:

- In this, initial version and budget of project plan are explained which include overall time duration required and overall cost required for development of project.

9. Appendices:

- In this, additional information like references from where information is gathered, definitions of some specific terms, acronyms, abbreviations, etc. are given and explained.

Software Requirement Specification for Attendance Management system

Authors:

Name: VEKARIYA SANKET KALUBHAI

Name: GOSWAMI VAIBHAVGIRI MANOJGIRI

Institute: Government Engineering College, Modasa

Department: Computer Engineering

Date: 29-Aug-2023

Version: 1.0

1. Introduction:

I. Purpose:

The purpose of a college attendance system in a Software Engineering project is to create a digital solution that automates the process of tracking and managing student attendance. This system can streamline attendance recording, reduce manual effort, improve accuracy, and provide real-time data for teachers, administrators, and students. It could include features such as student checkin through mobile apps, automatic notifications for low attendance, generating attendance reports, and facilitating communication between teachers and students regarding attendance-related matters. The project aims to enhance efficiency, transparency, and accountability within the college attendance process.

II. Scope:

The scope of an Attendance Management System (AMS) software engineering project is multifaceted and involves a comprehensive set of tasks and objectives. At its core, the project aims to develop a robust software solution that automates and streamlines attendance tracking and management processes for organizations, institutions, or events. This encompasses defining the requirements of various stakeholders, such as administrators, teachers, students, and parents, and determining the specific functionalities required, including user authentication, attendance recording methods (e.g., biometrics, mobile apps), real-time tracking, reporting, and notifications. The project scope extends to designing the database schema, selecting an appropriate system architecture and technology stack, developing user-friendly interfaces, and ensuring data security and privacy. Integration with other systems, scalability to accommodate growth, and compliance with relevant regulations are also within the project's scope. Additionally, user training, deployment planning, ongoing maintenance, and performance optimization are critical aspects to consider, making the scope of an AMS project a comprehensive endeavor that seeks to enhance attendance management efficiency and accuracy.

III. Overview:

An Attendance Management System (AMS) is a software solution designed to automate and streamline the process of recording and managing attendance for individuals within an organization or institution. It plays a crucial role in maintaining accurate attendance records, tracking patterns, and facilitating administrative tasks related to attendance. Here's an overview of an Attendance Management System. In summary, an Attendance Management System is a valuable tool for organizations and institutions to automate and improve attendance tracking processes, thereby enhancing operational efficiency and accuracy in managing attendance records. The specific features and architecture of an AMS can vary based on the needs and goals of the organization or institution using it..

2. General description:

2.1 Product Perspective The product Student Management system, is an independent product and does not depend on any other product or system. The product will automate various tasks associated with handling student details and better organizing the stored information and optimum performance, thus helping the Colleges to ensure smooth working of these processes.

2.2 Product Functions Our system has two types of accessing modes,

2.2.1 Administrator

2.2.2 User

2.2.3 Teacher

2.2.4 Student

i) Administrator: SMS is managed by Administrator. Administrator has to update and monitor the registered student details, add a new student, provide register number for all students, assign each student a course etc., Administrator can update his profile, and also can give help to the teachers and students. ii) User: There are two users: a. Student: User can only view their personal details, course assigned, and edit their assigned course and can view their attendance. b. Teacher: User can add them onto the portal and view their schedules, marks attendance of the students, also can view the students details in graphical order, also of a single student and about the views from the students.

2.3 User Characteristics This software gives access to two kinds of users. 1. Administrator: The personnel and College administrator will have administrator access to add, delete and modify information stored in the database. 2. Authorized User: Teaching staff will have access to only view the data stored in the database and can update the student's attendance in the form of formatted reports.

2.4 Assumptions and Dependencies • We assume that the Office personnel do all the data entry based and the correct values obtained from forms and registers. • We assume that the computers that will use the software will be part of the college LAN. • Users with administrator access should be careful in deleting or modifying any information knowingly or unknowingly which will lead to inconsistency of the database. • The end users of this software are assumed to have basic level of computer knowledge i.e. point and click

3. Functional Requirements:

(R.1): User Authentication:	
Description	It serves as the first line of defense against unauthorized access and ensures that only authorized personnel can interact with the system to manage attendance records.
R.1.1: User Authentication	
Input	User credentials (Username and Password)..
Output	Confirmation of successful login or error message

Processing	- Verify the entered credentials against the database and grant access if valid.- Storage of patient information in the database.
(R.2): User Registration	
Description	Student or Teacher can register.
R.2.1: User Registration:	
Input	- User details (Name, Employee ID, Department, etc.)..
Output	Confirmation of successful registration or error message
Processing	- Store employee information in the database.
(R.3): Attendance Recording	
Description	tracking and managing the attendance of employees or students. This process involves the systematic capture, storage, and management of attendance data, ensuring accuracy and reliability.
R.3.1: Attendance Recording:	
Input	- Employee ID or biometric data.
Output.	- Recorded attendance data (date, time, and employee details).
Processing.	- Capture and store the employee's attendance information when they clock in or out..
(R.4): Leave Management	
Description	Leave management is a crucial component of an Attendance Management System, designed to efficiently handle and track employee or student absences.
R.4.1: Leave Management	
Input	- Leave application details (employee ID, leave type, start date, end date, reason).
Output	- Confirmation of leave request submission or error message
Processing	- Validate and store leave requests in the system. Update employee attendance records accordingly
(R.5): Reporting:	
Description	Reporting is a critical functionality within an Attendance Management System (AMS) that enables organizations to analyze attendance data,.
R.5.1: Reporting:	
Input	- Various filters (e.g., date range, department, employee)
Output	- Attendance reports (summary or detailed) in a downloadable format.
Processing	- Generate attendance reports based on the selected filters and present them to the user.
(R.6): Attendance Modification	
Description	Attendance modification is a crucial feature within an Attendance Management System (AMS) that allows authorized personnel to make adjustments to attendance records when errors occur or when exceptions need to be addressed
R.6.1: Attendance Modification.	

Input	- Request to edit attendance records (employee ID, date, time).
Output	- Confirmation of modification or error message.
Processing	- Validate modification requests and update attendance records accordingly, with proper audit trails.
(R.7): Notifications	
Description	This module is sent notification of you're recorded attendance or updates
R.7.1: Notifications	
Input	- Triggers (e.g., late arrival, excessive leaves).
Output	- Email/SMS notifications to employees or supervisors
Processing	- Email/SMS notifications to employees or supervisors

4. Interface Requirements:

- I. User Interface:
 - 1: Attendance Management System
 - 1.1: This software will transmit the attendance of a class to a database on a machine via Internet.
 - 1.2: The user will be allowed to modify attendance records at any time.
 - 1.3: If the user forgets to transmit the information, the system will automatically send it for them at the end of the class.
 - 2: Database The Attendance Management System will communicate with the database to perform the following options.
 - 2.1: To allow a user to enter attendance.
 - 2.2: To allow a user to modify attendance.
 - 2.3: To allow a user to query a system to gain statistics concerning individual and class attendance
- II. Hardware Interface:
 - 1) Device: Mobile, Laptop, PCs
 - 2) Network: Wi-Fi Router This system should need a particular network to use it and also need the particular handy device which support this system.
- III. Software Interface:
 - 1) Operating system: Android, iOS, Windows
 - 2) MySQL servers
 - 3) JDK 1.8

5. Performance Requirements:

Performance requirements for an Attendance Management System (AMS) in a software engineering project are pivotal to ensure efficient and reliable system operation. The system must respond promptly to user interactions, such as logging in, marking attendance, or generating reports, with response times typically measured in milliseconds or seconds. It should handle concurrent users effectively, accommodating peak loads during critical times, such as the start of classes or work shifts. Scalability is essential, enabling the system to expand seamlessly as the user base grows and attendance data accumulates. High throughput is a must, specifying the number of transactions or requests the system can manage per unit of time. Database performance requirements ensure speedy data retrieval, especially for complex reports and analytics. Low network latency is crucial for seamless integration with external systems or remote users. Security measures, while necessary, must not significantly impact performance, and efficient data backup and recovery processes are essential for minimizing downtime. Load testing is critical to identify and address bottlenecks. Continuous performance monitoring with defined thresholds for triggering alerts and the ability to handle mobile app responsiveness, even during peak usage, round out the comprehensive performance requirements for a robust and dependable AMS

6. Design Constraints:

Design constraints for an Attendance Management System (AMS) encompass budgetary and time limitations, hardware and software compatibility requirements, data privacy regulations, and integration complexities with existing systems. These constraints also extend to considerations of network bandwidth, accessibility compliance, scalability expectations, performance benchmarks, and adherence to user experience (UX) and mobile device constraints. Additionally, cultural and organizational policies may influence design choices, as well as the need to accommodate multiple languages and locales. Balancing these constraints is essential to create an effective AMS that aligns with the organization's goals and limitations while ensuring security, usability, and regulatory compliance.

7. Non-functional Attributes:

- Performance: The system should respond quickly to user interactions, such as marking attendance or generating reports, with minimal delay. It should handle concurrent users efficiently, ensuring optimal performance during peak periods.
- Scalability: The AMS should be able to scale seamlessly as the user base and attendance data volume increase. It should accommodate growth without a significant drop in performance.
- Reliability: The system should be highly reliable, minimizing downtime and ensuring data integrity. It should have mechanisms in place for failover and disaster recovery.
- Security: Data security is paramount. The system should employ encryption, access control, and authentication mechanisms to protect sensitive attendance data. It should comply with data privacy regulations and follow best practices for securing user information.
- Availability: The AMS should be available and accessible to users whenever needed, with minimal planned downtime for maintenance. It should have redundancy and backup systems to ensure continuous availability.
- Usability: The user interfaces should be intuitive and user-friendly, promoting ease of use for all types of users. Accessibility features should be in place to cater to users with disabilities.
- Data Backup and Recovery: Regular data backups should be automated, and there should be a reliable plan for data recovery in case of system failures or data loss.
- Performance Monitoring: Implement tools for continuous performance monitoring to track system health and identify bottlenecks or issues proactively

8. Appendices:

In a Software Requirements Specification (SRS) document for an Attendance Management System (AMS), you may include appendices to provide additional information, details, or supporting documentation. Here are some common appendices that can be included in an SRS for an AMS:

- Glossary of Terms: A list of technical terms, acronyms, and abbreviations used throughout the SRS to help readers understand the terminology.
- Use Case Diagrams: Visual representations of various use cases and their relationships in the system, helping to illustrate how different users interact with the system.
- Data Flow Diagrams (DFDs) Diagrams that depict the flow of data within the system, including data sources, processes, and data destinations.
- Entity-Relationship Diagram (ERD) A graphical representation of the database schema, showing how data entities are related in the system.
- System Architecture Diagram An overview of the system's architecture, illustrating the high-level components, their interactions, and the flow of data and control.
- User Interface Mockups Visual representations of the user interfaces, including screen layouts, forms, and reports, to give stakeholders a sense of the system's look and feel.

- Sample Reports Examples of the types of reports that the system can generate, including attendance summaries, detailed logs, and analytics.
- Performance Test Results Data and results from performance testing, including response times, scalability tests, and stress tests..

Tools/Materials Needed:

- Hardware Requirement:
 - 1) Device: Mobile, Laptop, PCs
 - 2) Memory: Min 64GB
 - 3) RAM: Min 4GB
- Software Requirement:
 - 1) OS: Android, iOS, Windows, iMac
 - 2) MySQL servers
 - 3) Minimum android 7 for mobiles and android supported devices Minimum windows XP or upper for windows operating system.

Quiz:

1. Which are properties of good SRS?
2. What is functional and non-functional requirement?
3. Which are different requirement engineering tasks? Suggested Reference:
 1. Lecture on "System Analysis and Design", NPTEL
 2. When Telepathy Won't Do: Requirements Engineering Key Practices
 3. Requirements Analysis: Process of requirements gathering and requirement definition
 4. IEEE Recommended Practice for Software Requirements Specifications 5. Requirements Trace-ability and Use Cases

Rubric wise marks obtained:

Rubrics	1	2	3	4	5	Total
Marks	Complete implementation as asked	Complete implementation as asked Problem analysis	Complete implementation as asked Problem analysis Development of the Solution	Complete implementation as asked Problem analysis Development of the Solution Concept Clarity & understanding	Complete implementation as asked Problem analysis Development of the Solution Concept Clarity & understanding Correct answer to all questions	

Signature of Faculty:

Practical – 2

AIM: Prepare the Software Project Management Plan (SPMP) including following: 1) Estimation of Size, Cost, Duration, Effort 2) Prepare the Schedule, Milestones using Gantt chart

- **Objectives:**
 1. To perform Estimation of Size, Cost, Duration, Effort
 2. To Prepare the Schedule, Milestones
- **Theory:**

Once a project is found to be feasible, software project managers undertake project planning. Project planning is undertaken and completed even before any development activity starts. Project planning consists of the following essential activities:

Estimating the following attributes of the project:

Project size: What will be problem complexity in terms of the effort and time required to develop the product?

Cost: How much is it going to cost to develop the project?

Duration: How long is it going to take to complete development?

Effort: How much effort would be required?

The effectiveness of the subsequent planning activities is based on the accuracy of these estimations.

- Scheduling manpower and other resources.
- Staff organization and staffing plans.
- Risk identification, analysis, and abatement planning
- Miscellaneous plans such as quality assurance plan, configuration management plan, etc.

Organization of SPMP Document

Introduction (Objectives, Major Functions, Performance Issues, Management and Technical Constraints)

Project Estimates (Historical Data, Estimation Techniques, Effort, Cost, and Project Duration Estimates)

Project Resources Plan (People, Hardware and Software, Special Resources)

Schedules (Work Breakdown Structure, Task Network, Gantt Chart Representation, PERT Chart Representation)

Risk Management Plan (Risk Analysis, Risk Identification, Risk Estimation, Abatement Procedures)

Project Tracking and Control Plan

Miscellaneous Plans (Process Tailoring, Quality Assurance)

Function Points

STEP 1: measure size in terms of the amount of functionality in a system. Function points are computed by first calculating an unadjusted function point count (UFC).

Counts are made for the following categories

External inputs—those items provided by the user that describe distinct application-oriented data (such as file names and menu selections)

External outputs—those items provided to the user that generate distinct application-oriented data (such as reports and messages, rather than the individual components of these)

External inquiries—interactive inputs requiring a response

External files—machine-readable interfaces to other systems

Internal files—logical master files in the system

STEP 2: Multiply each number by a weight factor, according to complexity (simple, average or complex) of the parameter, associated with that number. The value is given by a table:

Parameter	simple	average	complex
users inputs	3	4	6
users outputs	4	5	7
users requests	3	4	6
files	7	10	15
external interfaces	5	7	10

STEP 3: Calculate the total UFP(Unadjusted Function Points)

STEP 4: Calculate the total TCF(Technical Complexity Factor) by giving a value between 0 and 5 according to the importance of the following points(next slide): Technical Complexity Factors:

1.Data Communication 2.Distributed Data Processing 3.Performance Criteria 4.Heavily Utilized Hardware 5.High Transaction Rates 6.Online Data Entry 7.Online Updating 8.End-user Efficiency

9.Complex Computations 10.Reusability 11.Ease of Installation 12.Ease of Operation 13.Portability 14.Maintainability

STEP 5: Sum the resulting numbers to obtain DI(degree of influence)

STEP 6: TCF(Technical Complexity Factor) by given by the formula

– $TCF = 0.65 + 0.01 * DI$

STEP 6: Function Points are by given by the formula

– $FP = UFP * TCF$

The Constructive Cost Model (COCOMO) is the most widely used software estimation model.

The COCOMO model predicts the effort and duration of a project based on inputs relating to the size of the resulting systems and a number of "cost drives" that affect productivity.

The most important factors contributing to a project's duration and cost is the Development Mode Organic Mode: The project is developed in a familiar, stable environment, and the

product is similar to previously developed products. The product is relatively small, and requires little innovation.

Semidetached Mode: The project's characteristics are intermediate between Organic and Embedded.

Embedded Mode: The project is characterized by tight, inflexible constraints and interface requirements. An embedded mode project will require a great deal of innovation.

Mode	Effort Formula
Organic	$E = 2.4 * (S^{1.05})$
Semidetached	$E = 3.0 * (S^{1.12})$
Embedded	$E = 3.6 * (S^{1.20})$

Example:

```
Size = 200 KLOC
Effort = a * Sizeb
Organic — E = 2.4 * (2001.05) = 626 staff-months
Semidetached — E = 3.0 * (2001.12) = 1133 staff-months
Embedded — E = 3.6 * (2001.20) = 2077 staff-months
```

Project-task scheduling is an important project planning activity. It involves deciding which tasks would be taken up when. In order to schedule the project activities, a software project manager needs to do the following:

1. Identify all the tasks needed to complete the project.
2. Break down large tasks into small activities.
3. Determine the dependency among different activities.
4. Establish the most likely estimates for the time durations necessary to complete the activities.
5. Allocate resources to activities.
6. Plan the starting and ending dates for various activities.
7. Determine the critical path.

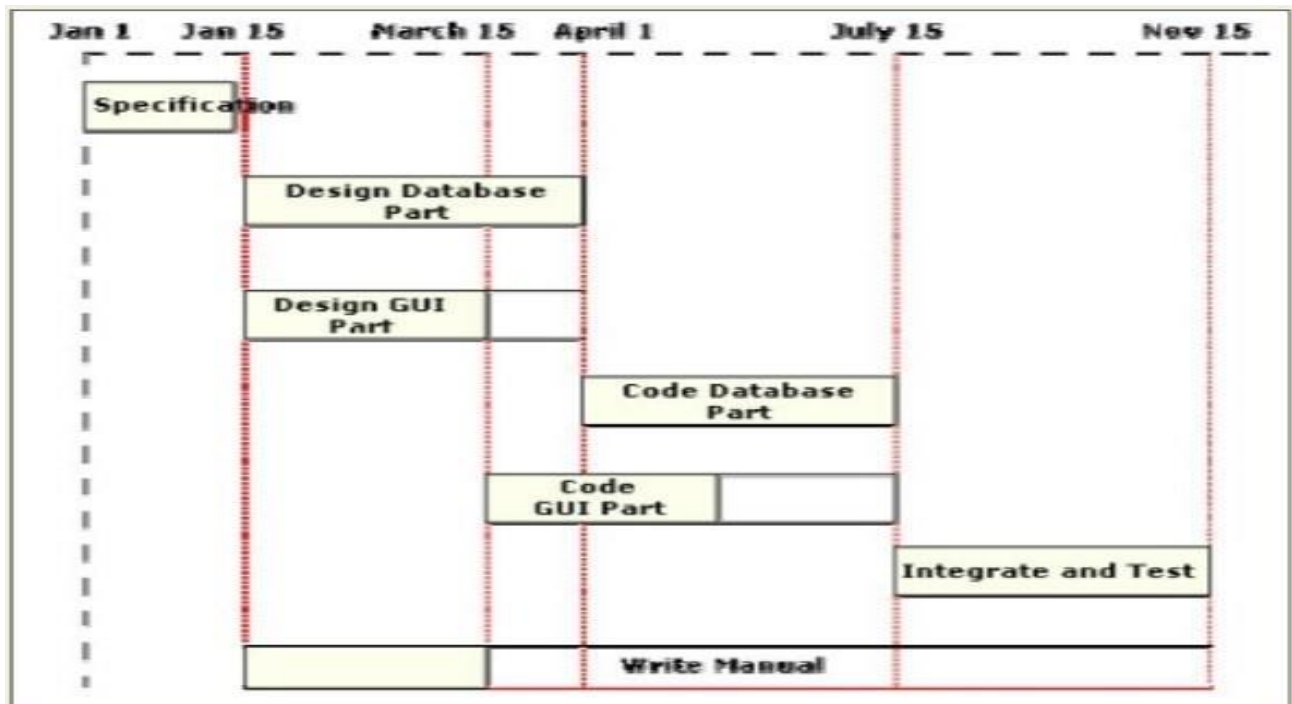
A critical path is the chain of activities that determines the duration of the project. The first step in scheduling a software project involves identifying all the tasks necessary to complete the project. A good knowledge of the intricacies of the project and the development process helps the managers to effectively identify the important tasks of the project. Next, the large tasks are broken down into a logical set of small activities which would be assigned to different engineers. The work breakdown structure formalism helps the manager to breakdown the tasks systematically after the project manager has broken down the tasks and created the work breakdown structure, he has to find the dependency among the activities.

Dependency among the different activities determines the order in which the different activities would be carried out. If an activity A requires the results of another activity B, then

activity A must be scheduled after activity B. In general, the task dependencies define a partial ordering among tasks, i.e. each tasks may precede a subset of other tasks, but some tasks might not have any precedence ordering defined between them (called concurrent task). The dependency among the activities is represented in the form of an activity network. Once the activity network representation has been worked out, resources are allocated to each activity.

Resource allocation is typically done using a Gantt chart. After resource allocation is done, a PERT chart representation is developed. The PERT chart representation is suitable for program monitoring and control. For task scheduling, the project manager needs to decompose the project tasks into a set of activities. The time frame when each activity is to be performed is to be determined. The end of each activity is called milestone. The project manager tracks the progress of a project by monitoring the timely completion of the milestones. If he observes that the milestones start getting delayed, then he has to carefully control the activities, so that the overall deadline can still be met.

Gantt charts are mainly used to allocate resources to activities. The resources allocated to activities include staff, hardware, and software. Gantt charts (named after its developer Henry Gantt) are useful for resource planning. A Gantt chart is a special type of bar chart where each bar represents an activity. The bars are drawn along a time line. The length of each bar is proportional to the duration of time planned for the corresponding activity. Gantt charts are used in software project management are actually an enhanced version of the standard Gantt charts. In the Gantt charts used for software project management, each bar consists of a white part and a shaded part. The shaded part of the bar shows the length of time each task is estimated to take. The white part shows the slack time, that is, the latest time by which a task must be finished.



Organization of SPMP Document

Introduction:

Objectives: This document gives the detailed functional and non-functional requirements for project of Health Management System. The purpose of this document is that the requirement mentioned in this document should be fulfil to implement the system.

Project Estimation Techniques:

- Function Points Analysis (FPA)
- COCOMO (Constructive Cost Model)

Function Points:

FPA is an estimation technique that assesses the functionality provided by a software application from the user's perspective. It quantifies the functionality in terms of function points, which are categorized into different types, including external inputs, external outputs, external inquiries, internal logical files, and external interface files. FPA helps in estimating the size and complexity of a software project based on the identified functions and their complexity.

- **Step 1: count of all information domain:**

External inputs:	10
External outputs:	10
External inquiries:	10
External files:	6
Internal files:	9

- **Step 2: Calculate the total UFP (Unadjusted Function Points)**

Measurement Parameter	Count		Weighting Factor Average	
External inputs	10	*	4	40
External outputs	10	*	5	50
External inquiries	10	*	4	40

External files	6	*	7	42
Internal files	9	*	10	90
			Total Count:	262

- **Step 3: Calculate the total VAF (Value Adjustment Factor) by giving a value between 0 and 5 according to the importance of the following:**

1. Data Communication	5
2. Distributed Data Processing	5
3. Performance Criteria	4
4. Heavily Utilized Hardware	2
5. High Transaction Rates	1
6. Online Data Entry	2
7. Online Updating	2
8. End user Efficiency	3
9. Complex Computations	4
10. Reusability	5
11. Ease of Installation	4
12. Ease of Operation	4
13. Portability	4
14. Maintainability	4

$$\begin{aligned}
 \text{Total Degree of influence (TDI):} & \quad 49 \\
 \text{Value Adjustment Factor (VAF)} & \quad = 0.65 + 0.01 * \text{DI} \\
 & \quad = 0.65 + 0.01 * 49 \\
 \text{VAF} & \quad = 1.14
 \end{aligned}$$

- **Step 4: calculate FP using Below Formula:**

$$\begin{aligned}
 \text{FP} &= \text{UFP} * \text{VAF} \\
 &= 262 * [0.65 + [0.01 * 52]] \\
 &= 262 * [0.65 + 0.52] \\
 &= 262 * 1.17 \\
 &= 306.54 \\
 \text{Function point is } &306.54
 \end{aligned}$$

COCOMO MODE

COCOMO is a family of estimation models, including Basic COCOMO and Intermediate COCOMO, which help in estimating the effort, cost, and duration of a software project. COCOMO models consider various factors, such as project size, complexity, development environment, and team experience.

While Function Points Analysis estimates the size of the software, COCOMO estimates effort and cost based on a combination of size and other project-specific factors

Codes of line (COL): 350 KLOC

Organic	$E = 2.04 * (COL)^{1.05}$
Semi-detached	$E = 3.00 * (COL)^{1.12}$
embedded	$E = 3.60 * (COL)^{1.20}$

- Effort (Semi-detached):

$$E = 3.00 * (COL)^{1.12}$$

$$E = 3.00 * (270)^{1.12}$$

$$E = 3.00 * (1.12)$$

$$E = 49.295 \text{ person month}$$

- Duration (Semi-detached):

$$D = 2.5 * (E)^{0.35}$$

$$D = 2.5 * (49.295)^{0.35}$$

$$D = 9.78 \text{ Months}$$

- Average staff size (Semi-detached):

$$S = E/D$$

$$S = 49.295/9.78$$

$$S = 5.040 \text{ person}$$

- Productivity:

$$P = \text{Codes of line (COL)} / \text{Average staff size}$$

$$P = 270 / 5.040$$

$$P = 53.57$$

- Calculate Cost:

$$\text{Cost per Person-Month} = 15000$$

$$\text{Cost (in currency)} = \text{Effort} * \text{Cost per Person-Month}$$

$$\text{Cost} \approx 49.295 * 15000$$

$$\text{Cost} \approx 739425$$

Schedules:

1. Identify all the tasks needed to complete the project.

- Project Initiation
- Requirements Gathering and Analysis
- Planning
- Design
- Development
- Testing
- Training and Documentation
- User Acceptance Testing (UAT)
- Maintenance and Support
- Evaluation and Improvement
- Data Management and Privacy
- Marketing and Adoption
- Reporting and Analytics
- Project Closure

2. Break down large tasks into small activities.

Attendance Management System

Account	Gathering and Analysis	Design	Development
Registration	Stakeholder Interviews	System Architecture	Frontend user interface
Verification	Functional Requirement	User Interface (UI)	Authentication and access control
Login	Non functional Requirement	Data flow Diagrams	Data storage and Retrieval
Log out	Use Cases	Security measures and Protocols	Third-party APIs and Services
Update	Data storage and retrieval	Plan for data privacy	Reporting and Analytics features

1. Establish the most likely estimates for the time durations necessary to complete the activities.

Activities	Duration (in months)
Account	4
Gathering and Analysis	5
Design	6
Development	7

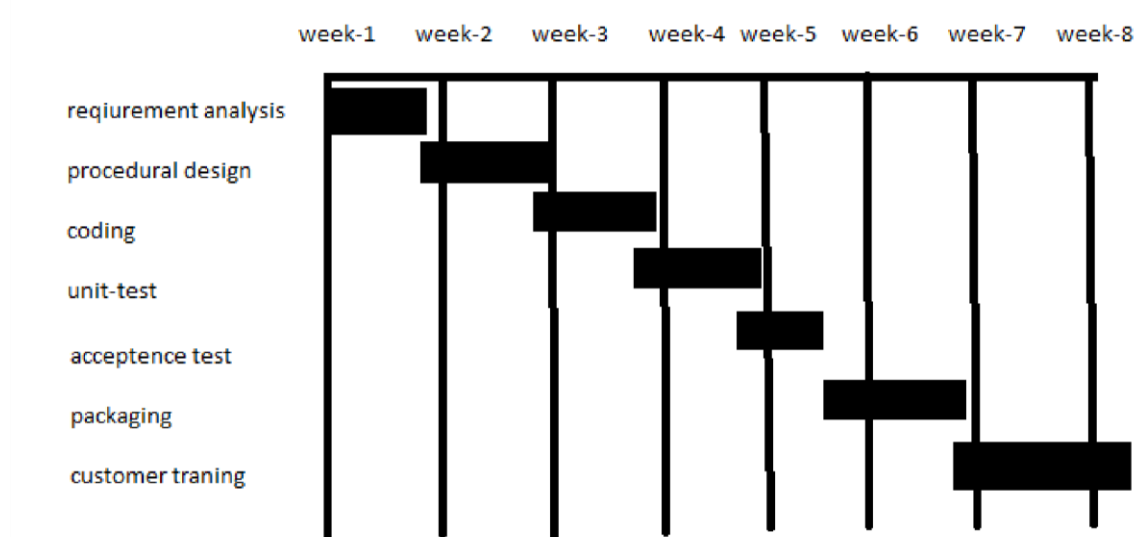
2. Allocate resources to activities.

Attendance Management System:

1. Requirements Gathering:
2. System Design:
3. Development:

4. Testing:
5. Deployment
6. Ongoing Maintenance:
7. Hardware and Software Resources:
8. Facilities:
9. Budgeting:
10. Documentation and Reporting:
11. Training:
12. Risk Management:
13. Communication:
14. Monitoring and Evaluation:
15. Feedback Gathering:
16. Documentation

Plan the starting and ending dates for various activities.



Quiz:

- 1) Explain project scheduling process. Explain Gantt Chart in detail.
- 2) A project size of 300 KLOC is to be developed. Software development team has beginner experience on similar type of projects. The project schedule is not very tight. Calculate the Effort, development time, average staff size, and productivity of the project. 3) Explain Software metrics used for software cost estimation

Suggested Reference:

- I. COCOMO
- II. Halstead complexity measures
- III. COCOMO (Constructive Cost Model), Seminar on Software Cost Estimation WS 2002 / 2003, presented by Nancy Merlo – Schett
- IV. The Halstead metrics
- V. Software Engineering, National Program on Technology Enhanced Learning VI. Halstead Metrics, Verifysoft Technology

Rubric wise marks obtained:

Rubrics	1	2	3	4	5	Total
Marks	Complete implementation as asked	Complete implementation as asked Problem analysis	Complete implementation as asked Problem analysis Development of the Solution	Complete implementation as asked Problem analysis Development of the Solution Concept Clarity & understanding	Complete implementation as asked Problem analysis Development of the Solution Concept Clarity & understanding Correct answer to all questions	

Signature of Faculty:

Practical – 3

AIM: Prepare the following components of Data Flow Model: Data Dictionary, Data Flow

Diagram and Structure Chart Objectives:

1. To learn how to prepare data dictionary.
2. To learn how to draw data flow diagrams.
3. To convert a DFD into structure chart.

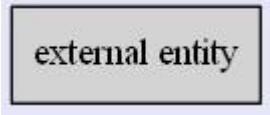


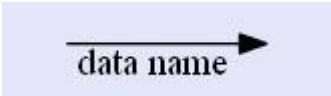
Theory:

- o 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.
- o It shows how data enters and leaves the system, what changes the information, and where data is stored.
- o The objective of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communication tool between a system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called as a data flow graph or bubble chart.
- o The following observations about DFDs are essential:
 - ☐ All names should be unique. This makes it easier to refer to elements in the DFD.
 - ☐ Remember that DFD is not a flow chart. Arrows in a flow chart that represents the order of events; arrows in DFD represents flowing data. A DFD does not involve any order of events.
 - ☐ Suppress logical decisions. If we ever have the urge to draw a diamond-shaped box in a DFD, suppress that urge! A diamond-shaped box is used in flow charts

to represents decision points with multiple exists paths of which the only one is taken. This implies an ordering of events, which makes no sense in a DFD.

- Do not become bogged down with details. Defer error conditions and error handling until the end of the analysis.

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

Term	Notation	Remarks
External entity		Name of the external entity is written inside the rectangle
Process		Name of the process is written inside the circle
Data store		A left-right open rectangle is denoted as data store; name of the data store is written inside the shape
Data flow		Data flow is represented by a directed arc with its data name

Explanation of Symbols used in DFD

- Process: Processes are represented by circle. The name of the process is written into the circle. The name of the process is usually given in such a way that represents the functionality of the process. More detailed functionalities can be shown in the next Level if it is required. Usually it is better to keep the number of processes less than 7. If we see that the number of processes becomes more than 7 then we should combine some the processes to a single one to reduce the number of processes and further decompose it to the next level .

- External entity: External entities are only appear in context diagram. External entities are represented by a rectangle and the name of the external entity is written into the shape. These send data to be processed and again receive the processed data.
- Data store: Data stores are represented by a left-right open rectangle. Name of the data store is written in between two horizontal lines of the open rectangle. Data stores are used as repositories from which data can be flown in or flown out to or from a process.
- Data flow: Data flows are shown as a directed edge between two components of a Data Flow Diagram. Data can flow from external entity to process, data store to process, in between two processes and vice-versa.

• Background / Preparation:

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 DFDM

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

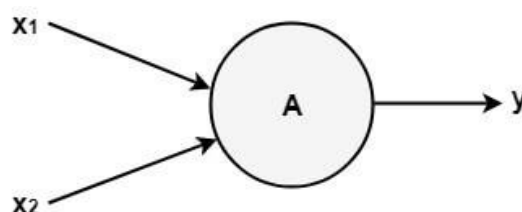


Fig: Level-0 DFD.

The Level-0 DFD, also called context diagram of the result management system is shown in fig. As the bubbles are decomposed into less and less abstract bubbles, the corresponding data flow may also be needed to be decomposed.

1-level DFD

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

2-Level DFD

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

- Tools / Material Needed:
 - o Hardware:
 - o Software:
- Procedure / Steps:
 - o Example: ATM System

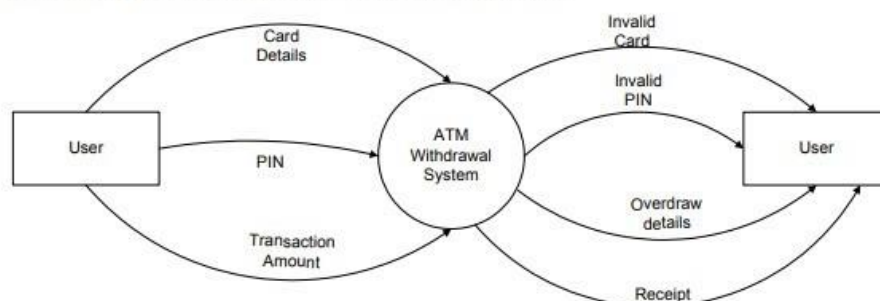
The data flow diagram is a hierarchy of diagram consist of:

- Context Diagram (conceptually level zero)
- The Level-1 DFD
- And possible Level-2 DFD and further levels of functional decomposition depending on the complexity of your system.

Context DFD

The figure below shows a context Data Flow Diagram that is drawn for an Android supermarket app. It contains a process (shape) that represents the system to model, in this case, the "ATM System". It also shows the participants who will interact with the system, called the external entities. In this example, there is only one external entity, which is the User. In between the process and the external entity, there is a uni-directional connector,

Level 0 DFD (Context Diagram) – ATM Cash Withdrawal

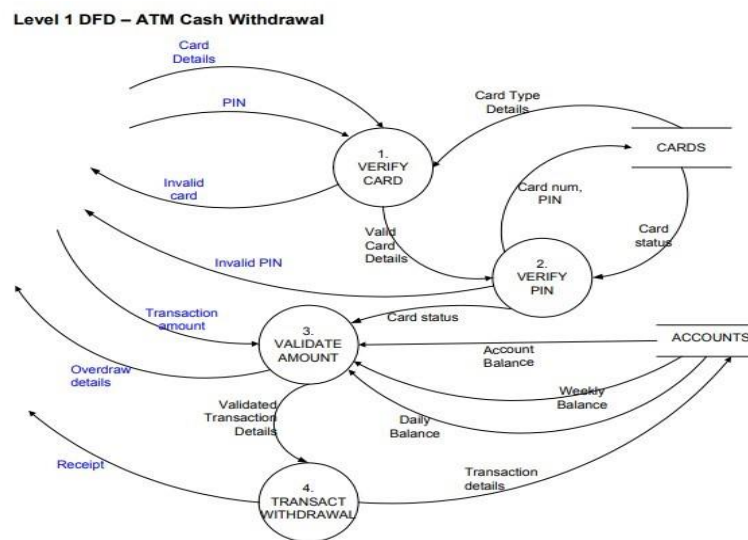


which indicates the existence of information exchange between user and the ATM System, and the information flow is bidirectional.

Context DFD is the entrance of a data flow model. It contains one and only one process and does not show any data store, which makes the diagram simple.

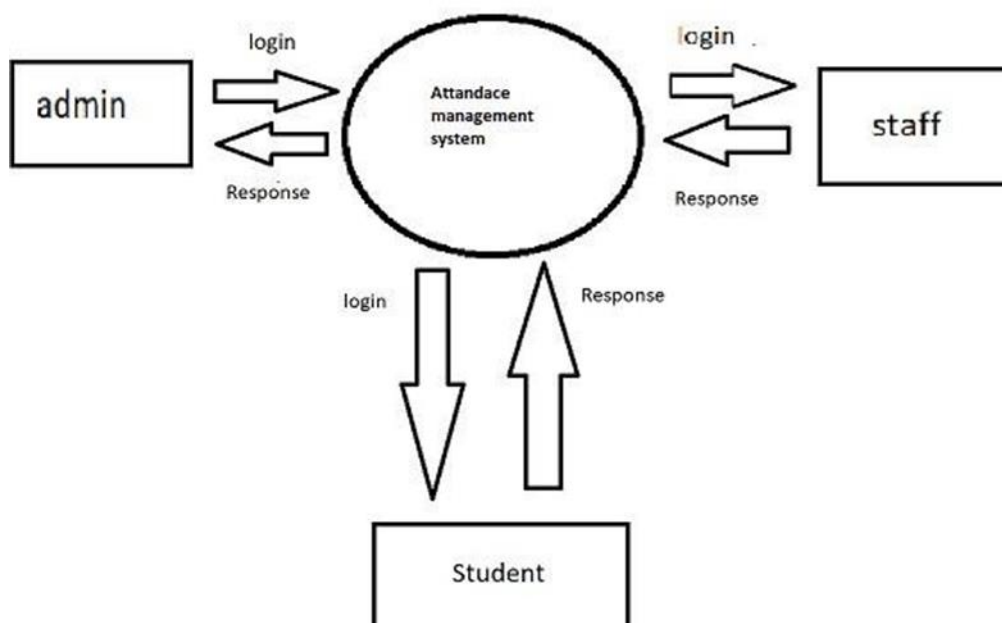
Level 1 DFD

The figure below shows the level 1 DFD, which is the decomposition (i.e. break down) of the ATM System process that is shown in the context DFD. Read through the diagram and then we will introduce some of the key concepts based on this diagram.



Procedure / Steps:

(i) DFD Level - 0 :



(ii)DFD Level - 1:



Quiz:

- 1) Draw the Data Flow Diagram for Hotel Management System.
- 2) What is DFD? Give advantage and disadvantages of DFD.
- 3) Types and Components of Data Flow Diagram (DFD)

Suggested Reference:

- 1) NPTEL course material - System Analysis and Design
- 2) Booch, G. et al. The Unified Modeling Language User Guide. Chapters 15, 18, 27. AddisonWesley.
- 3) Jacobson, I. et al. Object-Oriented Software Engineering: A Use-Case Driven Approach. Addison-Wesley.

- 4) Fowler, M. UML Distilled: A Brief Guide to the Standard Object Modelling Language.
Chapter
5. Addison Wesley.

Rubric wise marks obtained:

Rubrics	1	2	3	4	5	Total
Marks	Complete implementation as asked	Complete implementation as asked Problem analysis	Complete implementation as asked Problem analysis Development of the Solution	Complete implementation as asked Problem analysis Development of the Solution Concept Clarity & understanding	Complete implementation as asked Problem analysis Development of the Solution Concept Clarity & understanding Correct answer to all questions	

Signature of Faculty:

Practical – 4

AIM: Prepare the user's view analysis : Describe different scenarios and Draw Use case diagrams using UML

- Objectives:
 1. To write different scenarios of the system's execution.
 2. To explore various UML use case diagram components to draw USECASE diagram.
 - Theory:
 - o A use case diagram is used to represent the dynamic behavior of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and also tells how the user handles a system.
 - o Purpose of Use Case Diagrams
 - The main purpose of a use case diagram is to portray the dynamic aspect of a system. It accumulates the system's requirement, which includes both internal as well as external influences. It invokes persons, use cases, and several things that invoke the actors and elements accountable for the implementation of use case diagrams. It represents how an entity from the external environment can interact with a part of the system.
- Following are the purposes of a use case diagram given below:
1. It gathers the system's needs.
 2. It depicts the external view of the system.
 3. It recognizes the internal as well as external factors that influence the system.
 4. It represents the interaction between the actors.
- o In a use-case diagram, an actor is a user of the system (i.e. Something external to the system; can be human or non-human) acting in a particular role. o A use-case is a task which the actor needs to perform with the help of the system, e.g., find details of a book or print a copy of a receipt in a bookshop.
 - o We can draw a box (with a label) around a set of use cases to denote the system boundary, as on the previous slide ("library system").

Inheritance can be used between actors to show that all use cases of one actor are available to the other:

If several use cases include, as part of their functionality, another use case, we have a special way to show this in a use-case diagram with an <<include>> relation.

If a use-case has two or more significantly different outcomes, we can show this by extending the use case to a main use case and one or more subsidiary cases.

- **Background / Preparation:**

How to draw a Use Case diagram?

It is essential to analyze the whole system before starting with drawing a use case diagram, and then the system's functionalities are found. And once every single functionality is identified, they are then transformed into the use cases to be used in the use case diagram.

After that, we will enlist the actors that will interact with the system. The actors are the person or a thing that invokes the functionality of a system. It may be a system or a private entity, such that it requires an entity to be pertinent to the functionalities of the system to which it is going to interact.

Once both the actors and use cases are enlisted, the relation between the actor and use case/system is inspected. It identifies the no of times an actor communicates with the system. Basically, an actor can interact multiple times with a use case or system at a particular instance of time.

Following are some rules that must be followed while drawing a use case diagram:

1. A pertinent and meaningful name should be assigned to the actor or a use case of a system.
2. The communication of an actor with a use case must be defined in an understandable way.
3. Specified notations to be used as and when required.
4. The most significant interactions should be represented among the multiple no of interactions between the use case and actors. The purposes of use case diagrams can be as follows:

- Used to gather requirements of a system.
- Used to get an outside view of a system.
- Identify external and internal factors influencing the system.
- Show the interacting among the requirements are actors.

Scenarios

- Scenarios are real-life examples of how a system can be used.
- They should include
 - A description of the starting situation;
 - A description of the normal flow of events;
 - A description of what can go wrong;
 - Information about other concurrent activities;

A description of the state when the scenario finishes.

- Tools / Material Needed:
 - o Hardware:
 - o Software:
- Procedure / Steps:
 - o Developing Use Cases:
 - o Step One – Define the set of actors that will be involved in the story
 - ☐ Actors are people, devices, or other systems that use the system or product within the context of the function and behavior that is to be described
 - ☐ Actors are anything that communicate with the system or product and that are external to the system itself
 - o Step Two – Develop use cases, where each one answers a set of questions
 - o Example: ATM System

1. Identification of use cases.

2. Identification of actors.

3. Sample input:

1. Actor: User

2. Use case: Login

- Withdrawal Use Case :

A withdrawal transaction asks the customer to choose a type of account to withdraw from (e.g. checking) from a menu of possible accounts, and to choose a dollar amount from a menu of possible amounts. The system verifies that it has sufficient money on hand to satisfy the request before sending the transaction to the bank. (If not, the customer is informed and asked to enter a different amount.) If the transaction is approved by the bank, the appropriate amount of cash is dispensed by the machine before it issues a receipt. A withdrawal transaction can be cancelled by the customer pressing the Cancel key any time prior to choosing the dollar amount.

- Deposit Use Case :

A deposit transaction asks the customer to choose a type of account to deposit to (e.g. checking) from a menu of possible accounts, and to type in a dollar amount on the keyboard. The transaction is initially sent to the bank to verify that the ATM can accept a deposit from this customer to this account. If the transaction is approved, the machine accepts an envelope

from the customer containing cash and/or checks before it issues a receipt. Once the envelope has been received, a second message is sent to the bank, to confirm that the bank can credit the customer's account – contingent on manual verification of the deposit envelope contents by an operator later.

A deposit transaction can be cancelled by the customer pressing the Cancel key any time prior to inserting the envelope containing the deposit. The transaction is automatically cancelled if the customer fails to insert the envelope containing the deposit within a reasonable period of time after being asked to do so.

- Transfer UseCase:

A transfer transaction asks the customer to choose a type of account to transfer from (e.g. checking) from a menu of possible accounts, to choose a different account to transfer to, and to type in a dollar amount on the keyboard. No further action is required once the transaction is approved by the bank before printing the receipt.

A transfer transaction can be cancelled by the customer pressing the Cancel key any time prior to entering a dollar amount.

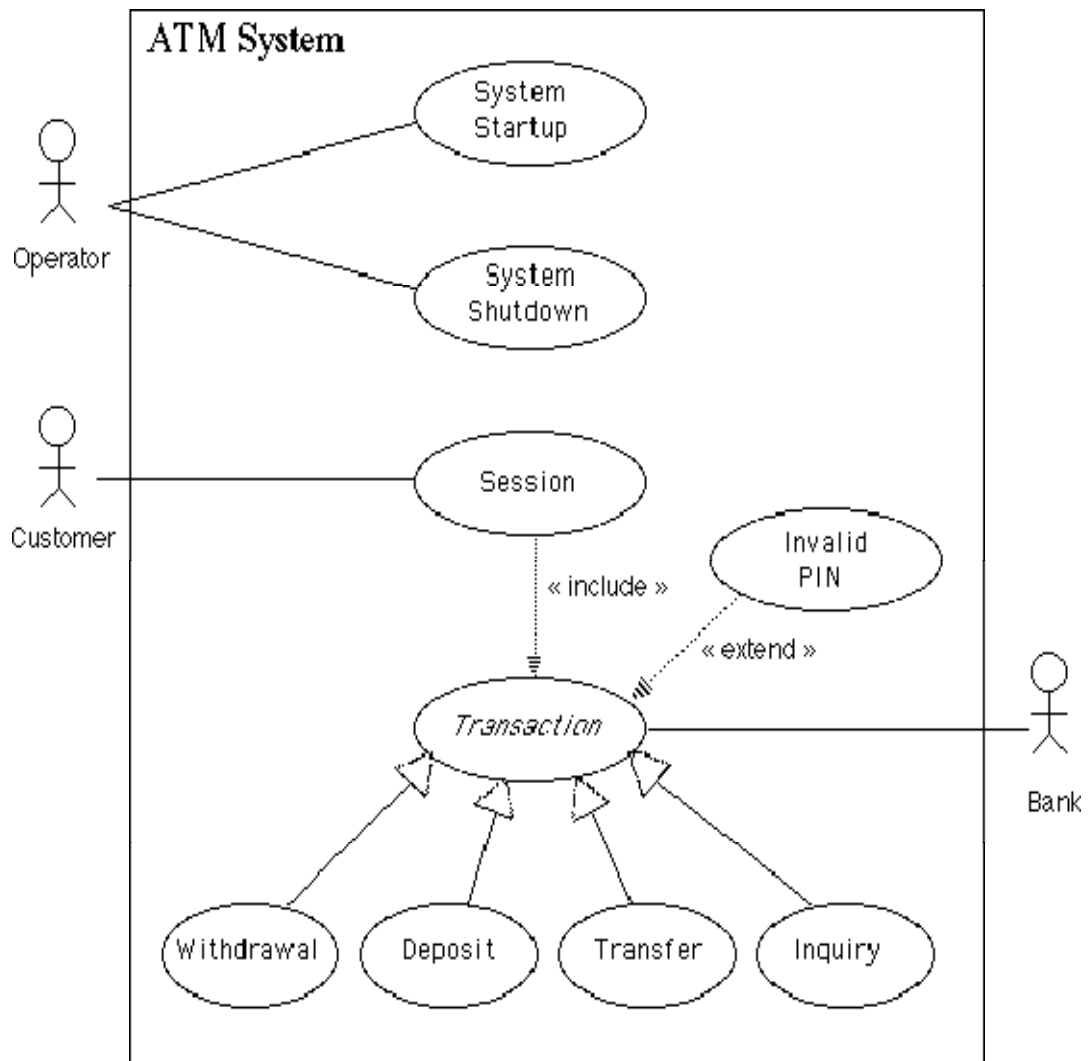
- Inquiry Use Case :

An inquiry transaction asks the customer to choose a type of account to inquire about from a menu of possible accounts. No further action is required once the transaction is approved by the bank before printing the receipt. An inquiry transaction can be cancelled by the customer pressing the Cancel key any time prior to choosing the account to inquire about.

- Validate User usecase:

This usecase is for validate the user i.e check the pin number, when the bank reports that the customer's transaction is disapproved due to an invalid PIN. The customer is required to re-enter the PIN and the original request is sent to the bank again. If the bank now approves the transaction, or disapproves it for some other reason, the original use case is continued; otherwise the process of re-entering the PIN is repeated. Once the PIN is successfully re-entered.

If the customer fails three times to enter the correct PIN, the card is permanently retained, a screen is displayed informing the customer of this and suggesting he/she contact the bank, and the entire customer session is aborted.



Actors:

Admin: A user with administrative privileges who manages the system.

Instructor: A user responsible for marking attendance.

Student: A user whose attendance is recorded.

Use Cases:

Login/Logout:

Actors: Admin, Instructor, Student

Description: Users can log in and out of the system.

Manage Users:

Actors: Admin

Description: Admin can add, edit, or delete user accounts (Instructors and Students).

Manage Courses:

Actors: Admin

Description: Admin can add, edit, or delete courses.

Manage Classes:

Actors: Admin

Description: Admin can create, schedule, or cancel classes.

Mark Attendance:

Actors: Instructor

Description: Instructors can mark attendance for students in their classes.

View Attendance:

Actors: Instructor, Student

Description: Instructors and students can view their own attendance records.

Generate Reports:

Actors: Admin

Description: Admin can generate attendance reports for courses, classes, or individual students.

Notify Absence:

Actors: Student

Description: Students can request excused absences or provide a reason for their absence.

Review Absence Requests:

Actors: Instructor, Admin

Description: Instructors and admin can review and approve/deny absence requests submitted by students.

Associations:

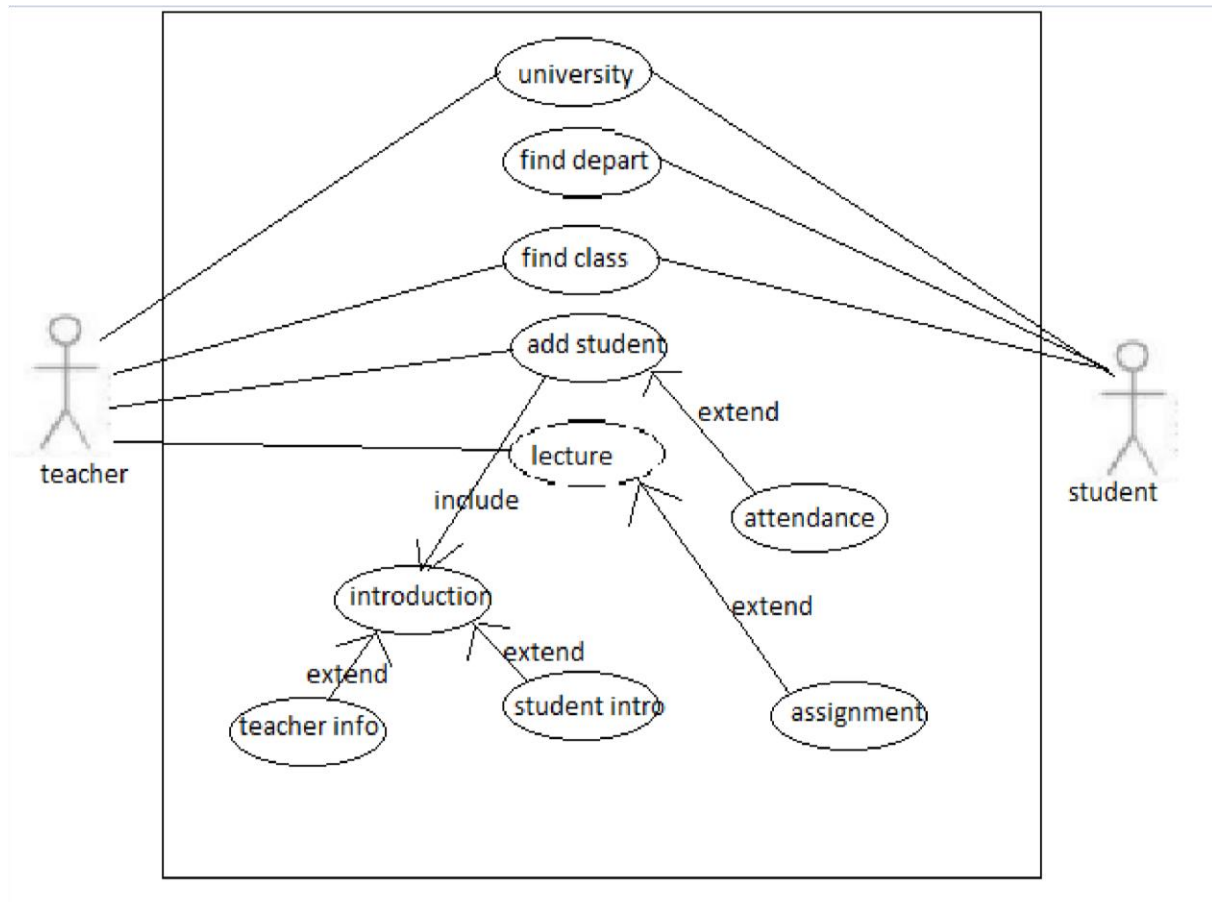
Admin is associated with all use cases, as they have the highest level of access and can perform all functions.

Instructors are associated with "Mark Attendance" and "View Attendance" use cases.

Students are associated with "View Attendance" and "Notify Absence" use cases.

System Boundary:

The system boundary encapsulates all the use cases and actors, representing the Attendance Management System as a whole.



Quiz:

1. What is use case diagram? Draw use case for library management system.
2. List relationship used in use case.
3. What Tests Can Help Find Useful Use Cases?

Suggested Reference:

- 1) Use Case Diagrams.
- 2) Unified Modeling Language, Superstructure, V2.1.2.
- 3) "Functional Requirements and Use Cases", Ruth Malan and Dana Bredemeyer, Bredemeyer Consulting.
- 4) "A Use Case Template: draft for discussion", Derek Coleman, Hewlett-Packard Software Initiative X. J. Zheng, X. Liu, & S. Liu. (2010). Use Case and Non-functional Scenario Template-Based Approach to Identify Aspects. Computer Engineering and Applications ICCEA 2010 Second International Conference on (Vol. 2, pp. 89-93).

Rubric wise marks obtained:

Rubrics	1	2	3	4	5	Total
Marks	Complete implementation as asked	Complete implementation as asked Problem analysis	Complete implementation as asked Problem analysis Development of the Solution	Complete implementation as asked Problem analysis Development of the Solution Concept Clarity & understanding	Complete implementation as asked Problem analysis Development of the Solution Concept Clarity & understanding Correct answer to all questions	

Signature

Practical – 5

AIM: Prepare the structural view : Draw Class diagram and object diagram.

- Objectives: to learn about UML class diagram and object diagram components.
- Theory:
 - o Class diagram:
 - The class diagram depicts a static view of an application. It represents the types of objects residing in the system and the relationships between them. A class consists of its objects, and also it may inherit from other classes. A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code.
 - It shows the attributes, classes, functions, and relationships to give an overview of the software system. It constitutes class names, attributes, and functions in a separate compartment that helps in software development. Since it is a collection of classes, interfaces, associations, collaborations, and constraints, it is termed as a structural diagram.

o Purpose of Class Diagrams

The main purpose of class diagrams is to build a static view of an application. It is the only diagram that is widely used for construction, and it can be mapped with objectoriented languages. It is one of the most popular UML diagrams. Following are the purpose of class diagrams given below:

1. It analyses and designs a static view of an application.
2. It describes the major responsibilities of a system.
3. It is a base for component and deployment diagrams.
4. It incorporates forward and reverse engineering.

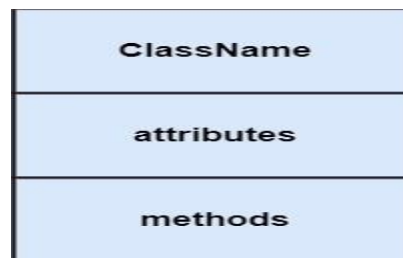
o Benefits of Class Diagrams

1. It can represent the object model for complex systems.
2. It reduces the maintenance time by providing an overview of how an application is structured before coding.
3. It provides a general schematic of an application for better understanding.
4. It represents a detailed chart by highlighting the desired code, which is to be programmed.
5. It is helpful for the stakeholders and the developers.

Vital components of a Class Diagram

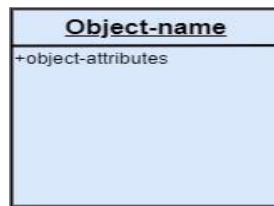
The class diagram is made up of three sections:

2. Upper Section: The upper section encompasses the name of the class. A class is a representation of similar objects that shares the same relationships, attributes, operations, and semantics. Some of the following rules that should be taken into account while representing a class are given below:
 1. Capitalize the initial letter of the class name.
 2. Place the class name in the center of the upper section.
 3. A class name must be written in bold format.
 4. The name of the abstract class should be written in italics format.
3. Middle Section: The middle section constitutes the attributes, which describe the quality of the class. The attributes have the following characteristics:
 1. The attributes are written along with its visibility factors, which are public (+), private (-), protected (#), and package (~).
 2. The accessibility of an attribute class is illustrated by the visibility factors.
 3. A meaningful name should be assigned to the attribute, which will explain its usage inside the class.
4. Lower Section: The lower section contains methods or operations. The methods are represented in the form of a list, where each method is written in a single line. It demonstrates how a class interacts with data.



- o Object diagram:
- o Object diagrams are dependent on the class diagram as they are derived from the class diagram. It represents an instance of a class diagram. The objects help in portraying a static view of an object-oriented system at a specific instant.
- o Both the object and class diagram are similar to some extent; the only difference is that the class diagram provides an abstract view of a system. It helps in visualizing a particular functionality of a system.

Notation of an Object Diagram



o Purpose of Object Diagram

The object diagram holds the same purpose as that of a class diagram. The class diagram provides an abstract view which comprises of classes and their relationships, whereas the object diagram represents an instance at a particular point of time.

The object diagram is actually similar to the concrete (actual) system behavior. The main purpose is to depict a static view of a system.

□ Background / Preparation:

How to draw a Class Diagram?

The class diagram is used most widely to construct software applications. It not only represents a static view of the system but also all the major aspects of an application. A collection of class diagrams as a whole represents a system.

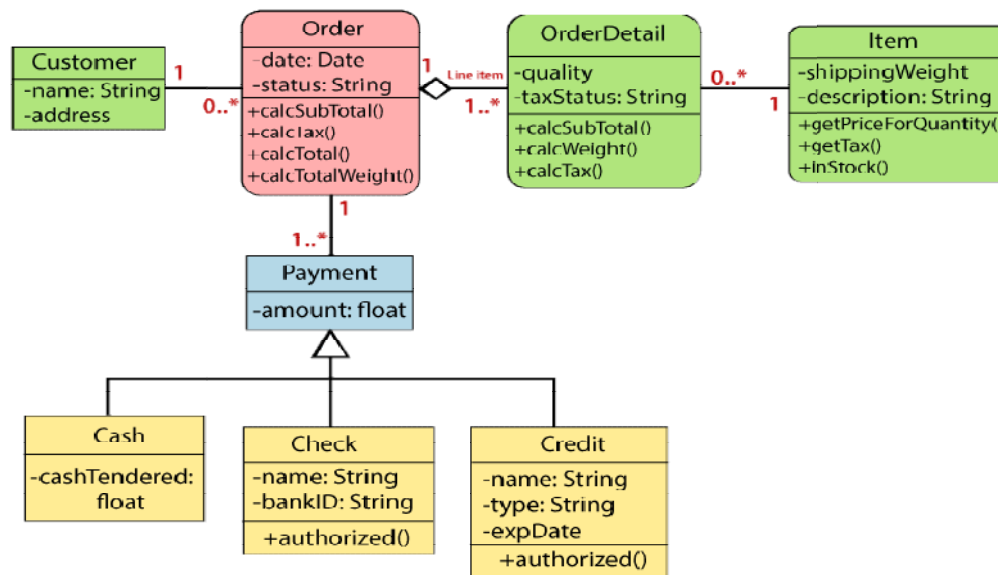
Some key points that are needed to keep in mind while drawing a class diagram are given below:

1. To describe a complete aspect of the system, it is suggested to give a meaningful name to the class diagram.
2. The objects and their relationships should be acknowledged in advance.
3. The attributes and methods (responsibilities) of each class must be known.
4. A minimum number of desired properties should be specified as more number of the unwanted property will lead to a complex diagram.
5. Notes can be used as and when required by the developer to describe the aspects of a diagram.
6. The diagrams should be redrawn and reworked as many times to make it correct before producing its final version.

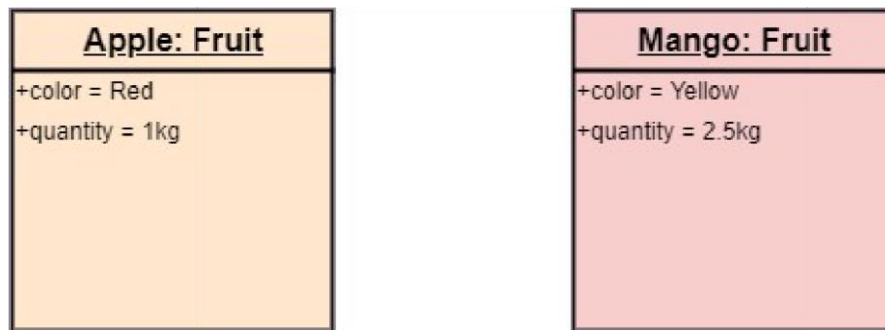
o How to draw an Object Diagram?

1. All the objects present in the system should be examined before start drawing the object diagram.
2. Before creating the object diagram, the relation between the objects must be acknowledged.
3. The association relationship among the entities must be cleared already.

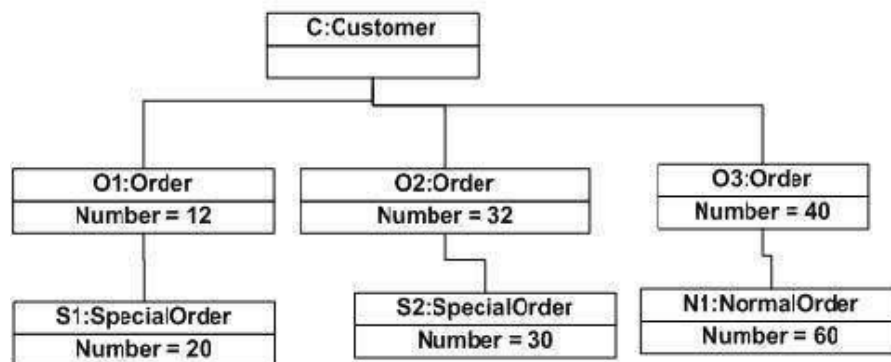
4. To represent the functionality of an object, a proper meaningful name should be assigned.
 5. The objects are to be examined to understand its functionality.
- o How to draw an Object Diagram?
 1. All the objects present in the system should be examined before start drawing the object diagram.
 2. Before creating the object diagram, the relation between the objects must be acknowledged.
 3. The association relationship among the entities must be cleared already.
 4. To represent the functionality of an object, a proper meaningful name should be assigned.
 5. The objects are to be examined to understand its functionality.
 - Tools / Material Needed:
 - o Hardware: o Software:
 - Procedure / Steps:
 - o Example of sales order system



- o Example of Object Diagram



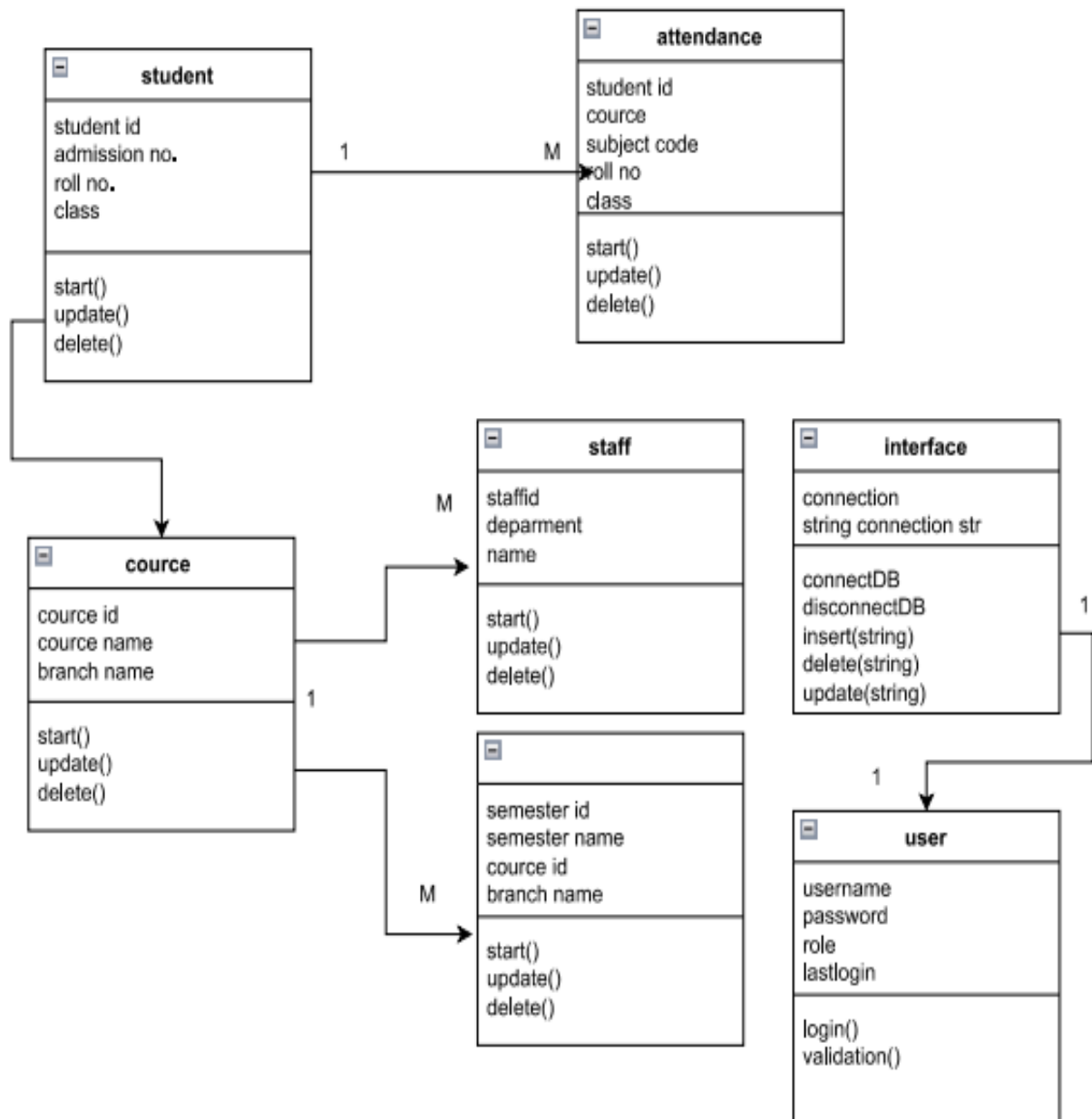
Object diagram of an order management system



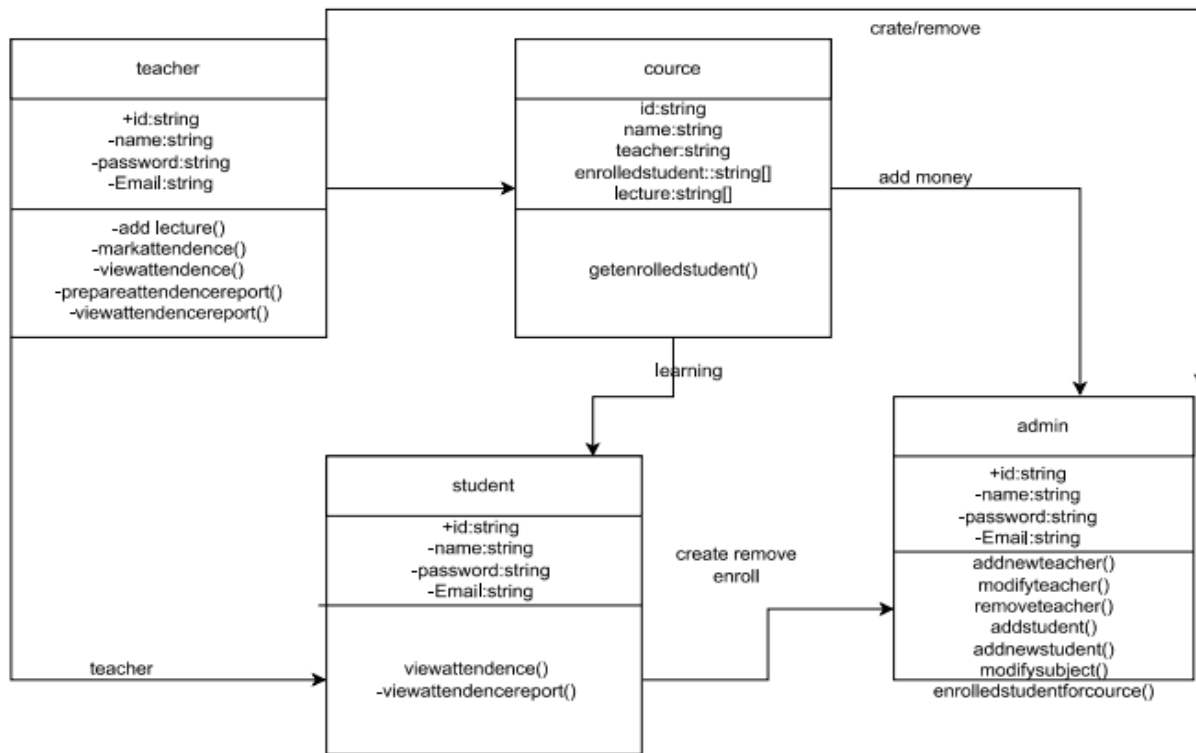
Suggested Reference:

- 1) Domain Analysis.
- 2) Domain Analysis Using Textual Analysis Approach
- 3) Domain model
- 4) Business Modeling – The Domain Model
- 5) I. Y. Song, K. Yano, J. Trujillo, and S. Luján-Mora. "A Taxonomic Class Modeling Methodology for Object-Oriented Analysis", In Information Modeling Methods and Methodologies, Advanced Topics in Databases Series, Ed. (J Krostige, T. Halpin, K. Siau), Idea Group Publishing, 2004, pp. 216-240.

Class diagram :



Object diagram:



Rubric wise marks obtained:

Rubrics	1	2	3	4	5	Total
Marks	Complete implementation as asked	Complete implementation as asked Problem analysis	Complete implementation as asked Problem analysis Development of the Solution	Complete implementation as asked Problem analysis Development of the Solution Concept Clarity & understanding	Complete implementation as asked Problem analysis Development of the Solution Concept Clarity & understanding Correct answer to all questions	

Signature of Faculty:

Practical – 6

AIM: Prepare the behavioral view: Draw Sequence diagram and Collaboration diagram.

- **Objectives:** To explore and use various UML components of sequence diagram and collaboration diagram
- **Theory:**
 - o **Sequence diagram:**
 - The sequence diagram represents the flow of messages in the system and is also termed as an event diagram. It helps in envisioning several dynamic scenarios. It portrays the communication between any two lifelines as a time-ordered sequence of events, such that these lifelines took part at the run time. In UML, the lifeline is represented by a vertical bar, whereas the message flow is represented by a vertical dotted line that extends across the bottom of the page. It incorporates the iterations as well as branching.

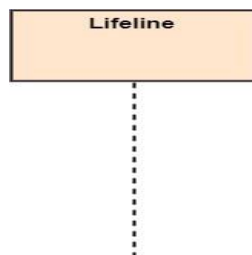
o Purpose of a Sequence Diagram

1. To model high-level interaction among active objects within a system.
2. To model interaction among objects inside a collaboration realizing a use case.
3. It either models generic interactions or some certain instances of interaction.

o Notations of a Sequence Diagram

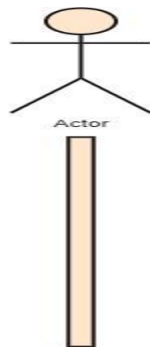
Lifeline

An individual participant in the sequence diagram is represented by a lifeline. It is positioned at the top of the diagram.



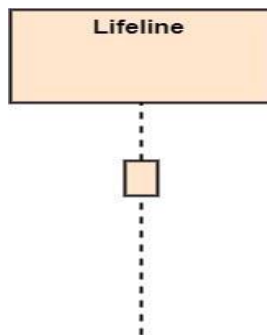
Actor

A role played by an entity that interacts with the subject is called as an actor. It is out of the scope of the system. It represents the role, which involves human users and external hardware or subjects. An actor may or may not represent a physical entity, but it purely depicts the role of an entity. Several distinct roles can be played by an actor or vice versa.



Activation

It is represented by a thin rectangle on the lifeline. It describes that time period in which an operation is performed by an element, such that the top and the bottom of the rectangle is associated with the initiation and the completion time, each respectively.

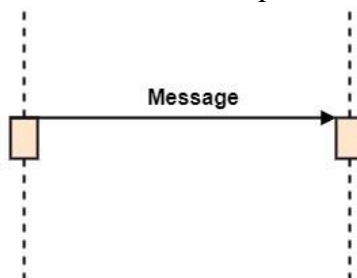


Messages

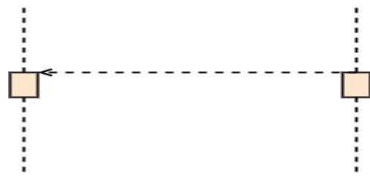
The messages depict the interaction between the objects and are represented by arrows. They are in the sequential order on the lifeline. The core of the sequence diagram is formed by messages and lifelines.

Following are types of messages enlisted below:

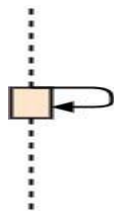
- Call Message: It defines a particular communication between the lifelines of an interaction, which represents that the target lifeline has invoked an operation.



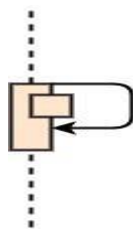
Return Message: It defines a particular communication between the lifelines of interaction that represent the flow of information from the receiver of the corresponding caller message.



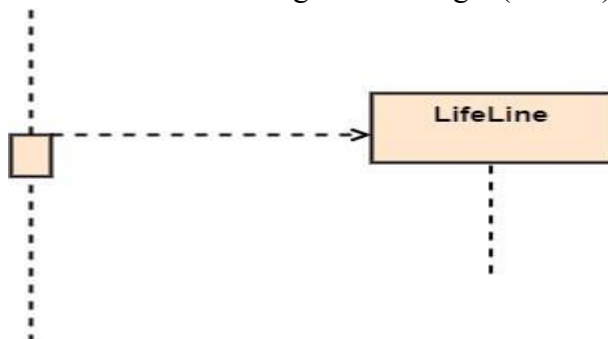
Self Message: It describes a communication, particularly between the lifelines of an interaction that represents a message of the same lifeline, has been invoked.



Recursive Message: A self message sent for recursive purpose is called a recursive message. In other words, it can be said that the recursive message is a special case of the self message as it represents the recursive calls.



Create Message: It describes a communication, particularly between the lifelines of an interaction describing that the target (lifeline) has been instantiated.



Destroy Message: It describes a communication, particularly between the lifelines of an interaction that depicts a request to destroy the lifecycle of the target.



Duration Message: It describes a communication particularly between the lifelines of an interaction, which portrays the time passage of the message while modeling a system.

o Collaboration diagram:

- The collaboration diagram is used to show the relationship between the objects in a system. Both the sequence and the collaboration diagrams represent the same information but differently. Instead of showing the flow of messages, it depicts the architecture of the object residing in the system as it is based on object-oriented programming. An object consists of several features. Multiple objects present in the system are connected to each other. The collaboration diagram, which is also known as a communication diagram, is used to portray the object's architecture in the system.

Notations of a Collaboration Diagram

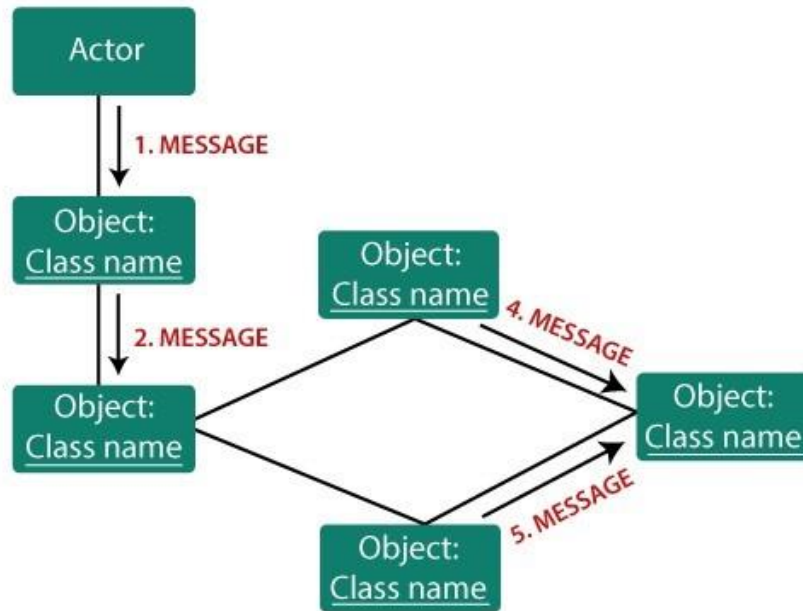
Following are the components of a component diagram that are enlisted below:

1. **Objects:** The representation of an object is done by an object symbol with its name and class underlined, separated by a colon.
In the collaboration diagram, objects are utilized in the following ways:
 - o The object is represented by specifying their name and class. o It is not mandatory for every class to appear. o A class may constitute more than one object.
 - o In the collaboration diagram, firstly, the object is created, and then its class is specified.
 - o To differentiate one object from another object, it is necessary to name them.
2. **Actors:** In the collaboration diagram, the actor plays the main role as it invokes the interaction. Each actor has its respective role and name. In this, one actor initiates the use case.
3. **Links:** The link is an instance of association, which associates the objects and actors. It portrays a relationship between the objects through which the messages are sent. It is represented by a solid line. The link helps an object to

connect with or navigate to another object, such that the message flows are attached to links.

4. Messages: It is a communication between objects which carries information and includes a sequence number, so that the activity may take place. It is represented by a labeled arrow, which is placed near a link. The messages are sent from the sender to the receiver, and the direction must be navigable in that particular direction. The receiver must understand the message.

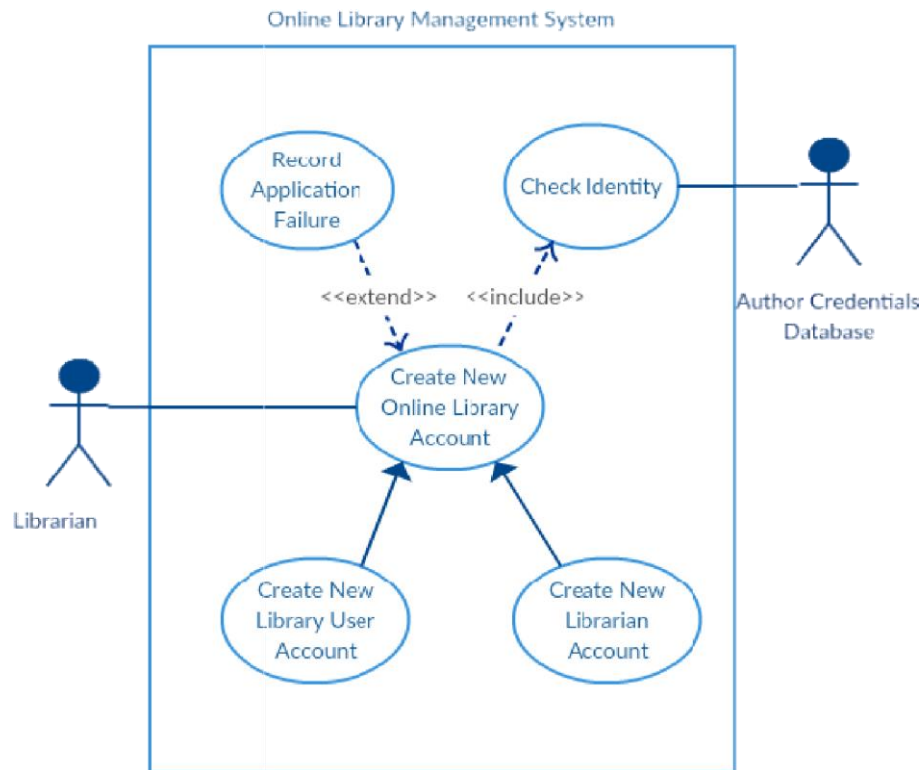
Components of a collaboration diagram



- Background / Preparation:

o How to Draw a Sequence Diagram

- ☐ A sequence diagram represents the scenario or flow of events in one single use case. The message flow of the sequence diagram is based on the narrative of the particular use case.
- ☐ Then, before you start drawing the sequence diagram or decide what interactions should be included in it, you need to draw the use case diagram and ready a comprehensive description of what the particular use case does.



From the above use case diagram example of ‘Create New Online Library Account’, we will focus on the use case named ‘Create New User Account’ to draw our sequence diagram example.

Before drawing the sequence diagram, it’s necessary to identify the objects or actors that would be involved in creating a new user account. These would be;

- Librarian
- Online Library Management system
- User credentials database
- Email system

Once you identify the objects, it is then important to write a detailed description on what the use case does. From this description, you can easily figure out the interactions (that should go in the sequence diagram) that would occur between the objects above, once the use case is executed.

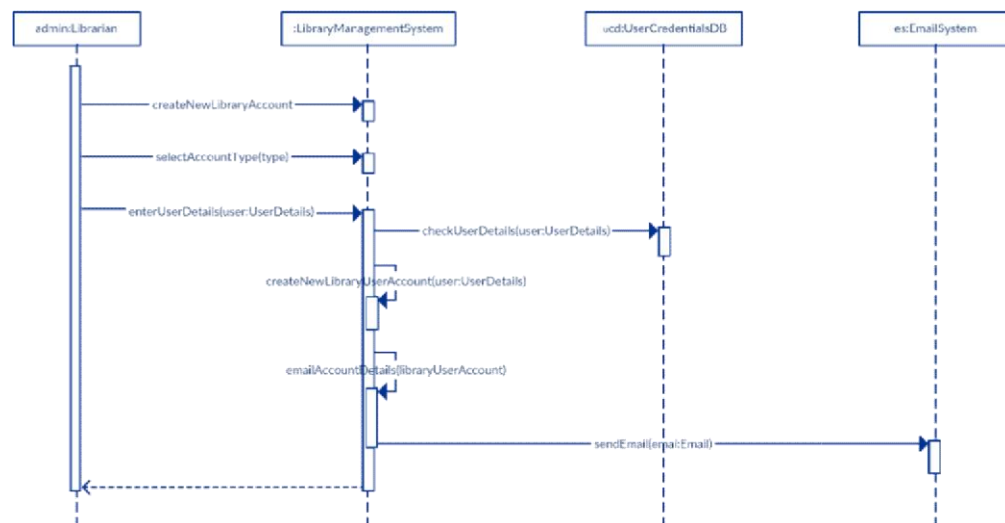
Here are the steps that occur in the use case named ‘Create New Library User Account’.

- The librarian request the system to create a new online library account

- The librarian then selects the library user account type
- The librarian enters the user's details
- The user's details are checked using the user Credentials Database
- The new library user account is created
- A summary of the of the new account's details are then emailed to the user

From each of these steps, you can easily specify what messages should be exchanged between the objects in the sequence diagram. Once it's clear, you can go ahead and start drawing the sequence diagram.

The sequence diagram below shows how the objects in the online library management system interact with each other to perform the function 'Create New Library User Account'.



- Tools / Material Needed:

- Hardware
- Software

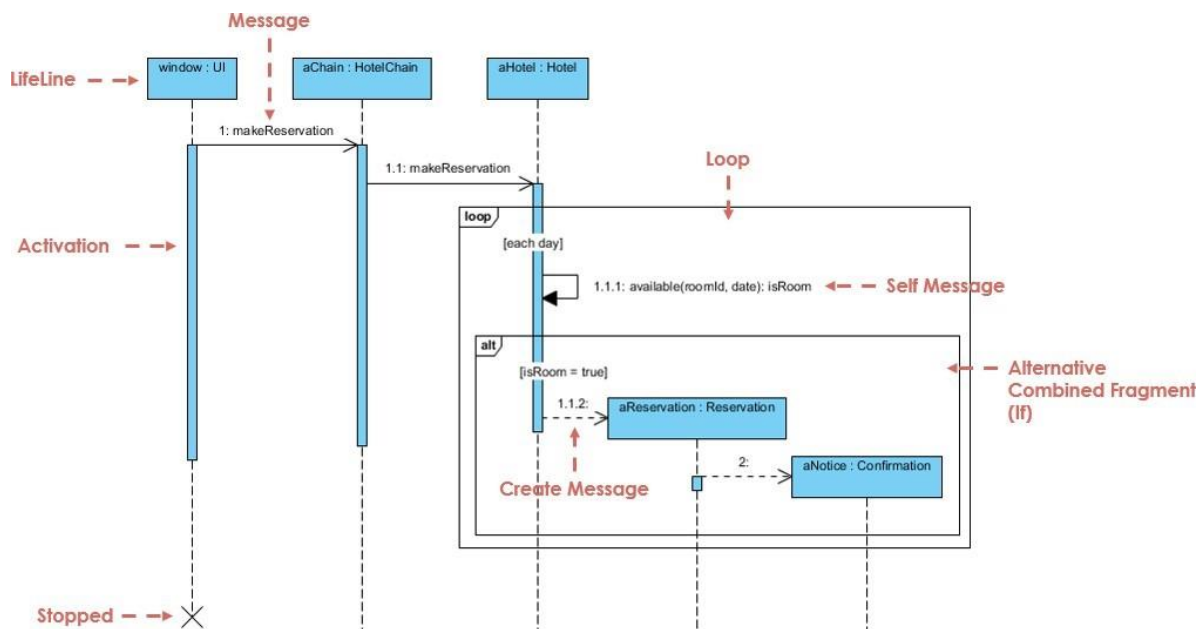
- Procedure / Steps:

o Sequence Diagram Example: Hotel System

Sequence Diagram is an interaction diagram that details how operations are carried out - what messages are sent and when. Sequence diagrams are organized according to time.

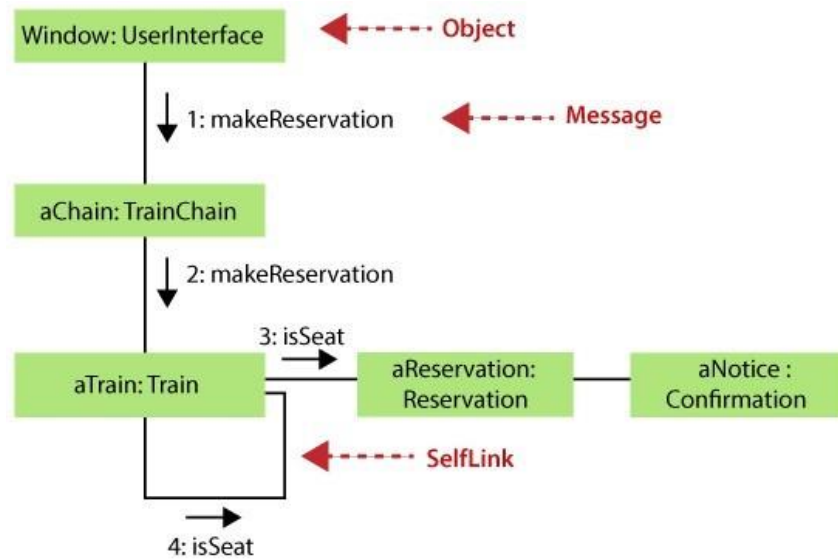
The time progresses as you go down the page. The objects involved in the operation are listed from left to right according to when they take part in the message sequence.

Below is a sequence diagram for making a hotel reservation. The object initiating the sequence of messages is a Reservation window.



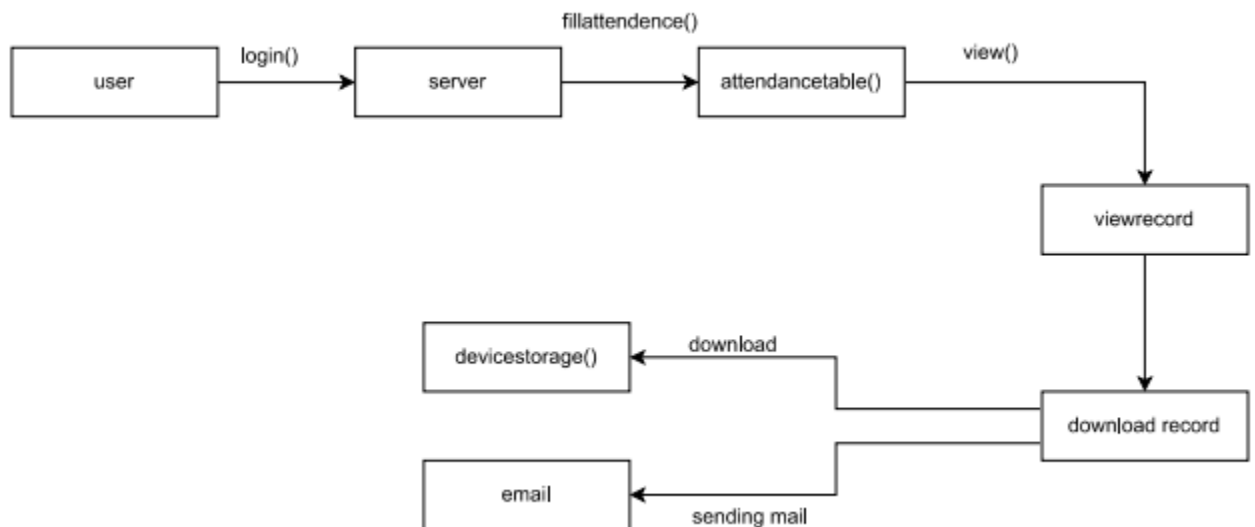
Note That: Class and object diagrams are static model views. Interaction diagrams are dynamic. They describe how objects collaborate.

- Steps for creating a Collaboration Diagram
- Determine the behavior for which the realization and implementation are specified.
- Discover the structural elements that are class roles, objects, and subsystems for performing the functionality of collaboration.
- Choose the context of an interaction: system, subsystem, use case, and operation.
- Think through alternative situations that may be involved.
- Implementation of a collaboration diagram at an instance level, if needed.
- A specification level diagram may be made in the instance level sequence diagram for summarizing alternative situations.
- Example of a Collaboration Diagram



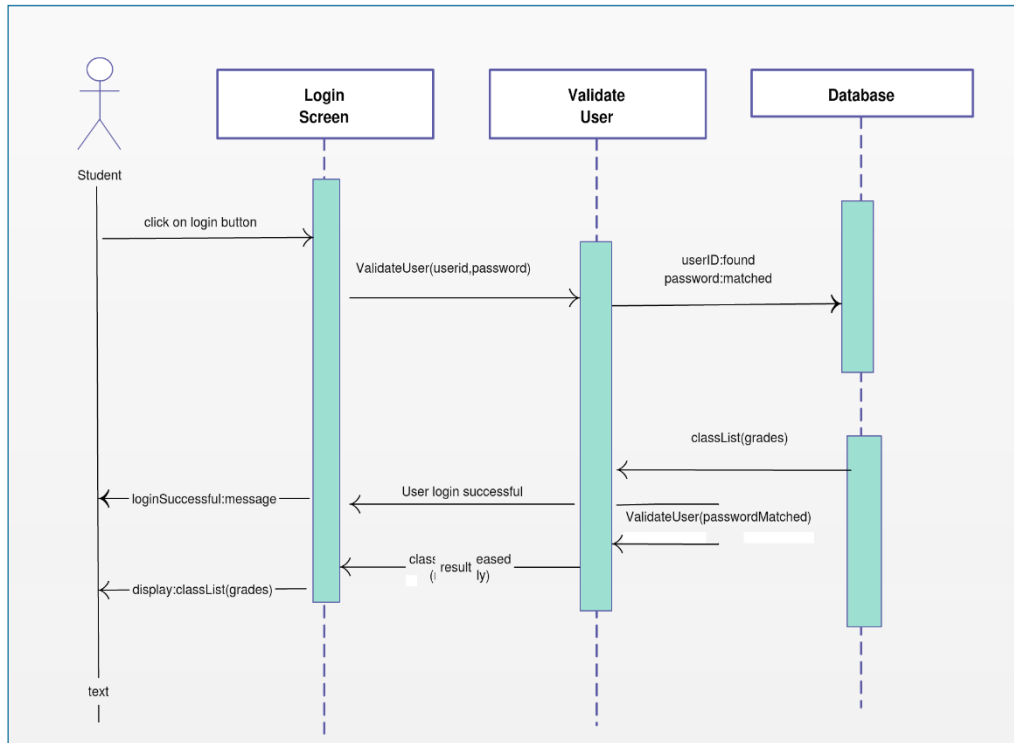
Suggested Reference:

- 1) Booch, G. et al. The Unified Modeling Language User Guide. Chapters 15, 18, 27. AddisonWesley.
- 2) Jacobson, I. et al. Object-Oriented Software Engineering: A Use-Case Driven Approach. Addison-Wesley.
- 3) Fowler, M. UML Distilled: A Brief Guide to the Standard Object Modelling Language. Chapter 5. Addison Wesley.



(Fig. Collaboration diagram)

Sequence-diagram:



Rubric wise marks obtained:

Rubrics	1	2	3	4	5	Total
Marks	Complete implementation as asked	Complete implementation as asked Problem analysis	Complete implementation as asked Problem analysis Development of the Solution	Complete implementation as asked Problem analysis Development of the Solution Concept Clarity & understanding	Complete implementation as asked Problem analysis Development of the Solution Concept Clarity & understanding Correct answer to all questions	

Signature of Faculty:

Practical – 7

AIM: Prepare the behavioral view: Draw State-chart diagram and Activity diagram.

- Objectives:
 - ☐ To explore various components of UML state-chart diagram and use them.
 - ☐ To explore various elements of activity diagram and use them.
- Theory:
 - o State Machine Diagram:
 - ☐ The state machine diagram is also called the Statechart or State Transition diagram, which shows the order of states underwent by an object within the system. It captures the software system's behavior. It models the behavior of a class, a subsystem, a package, and a complete system.
 - ☐ It tends out to be an efficient way of modeling the interactions and collaborations in the external entities and the system. It models event-based systems to handle the state of an object. It also defines several distinct states of a component within the system. Each object/component has a specific state.

Following are the types of a state machine diagram that are given below:

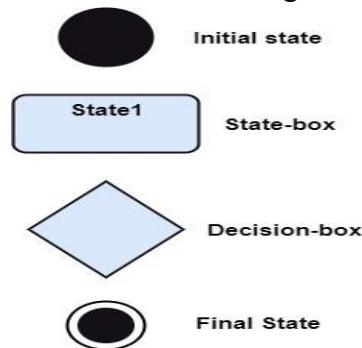
- Behavioral state machine
 - ☐ The behavioral state machine diagram records the behavior of an object within the system. It depicts an implementation of a particular entity. It models the behavior of the system.
- o State Machine Diagram:
 - ☐ The state machine diagram is also called the Statechart or State Transition diagram, which shows the order of states underwent by an object within the system. It captures the software system's behavior. It models the behavior of a class, a subsystem, a package, and a complete system.
 - ☐ It tends out to be an efficient way of modeling the interactions and collaborations in the external entities and the system. It models event-based systems to handle the state of an object. It also defines several distinct states of a component within the system. Each object/component has a specific state.

Following are the types of a state machine diagram that are given below:

- Protocol state machine
 - It captures the behavior of the protocol. The protocol state machine depicts the change in the state of the protocol and parallel changes within the system. But it does not portray the implementation of a particular component.

o Notation of a State Machine Diagram

Following are the notations of a state machine diagram enlisted below:



1. Initial state: It defines the initial state (beginning) of a system, and it is represented by a black filled circle.
2. Final state: It represents the final state (end) of a system. It is denoted by a filled circle present within a circle.
3. Decision box: It is of diamond shape that represents the decisions to be made on the basis of an evaluated guard.
4. Transition: A change of control from one state to another due to the occurrence of some event is termed as a transition. It is represented by an arrow labeled with an event due to which the change has ensued.
5. State box: It depicts the conditions or circumstances of a particular object of a class at a specific point of time. A rectangle with round corners is used to represent the state box.

• Background / Preparation:

o Types of State

The UML consist of three states:

1. Simple state: It does not constitute any substructure.
2. Composite state: It consists of nested states (substates), such that it does not contain more than one initial state and one final state. It can be nested to any level.

3. Submachine state: The submachine state is semantically identical to the composite state, but it can be reused.

When to use an Activity Diagram?

An activity diagram can be used to portray business processes and workflows. Also, it is used for modeling business as well as the software. An activity diagram is utilized for the followings:

1. To graphically model the workflow in an easier and understandable way.
2. To model the execution flow among several activities.
3. To model comprehensive information of a function or an algorithm employed within the system.
4. To model the business process and its workflow.
5. To envision the dynamic aspect of a system.
6. To generate the top-level flowcharts for representing the workflow of an application.
7. To represent a high-level view of a distributed or an object-oriented system.

- Tools / Material Needed:

- Hardware: ◦ Software:

- Procedure / Steps:

- How to Draw a State Machine Diagram?

The state machine diagram is used to portray various states undergone by an object. The change in one state to another is due to the occurrence of some event. All of the possible states of a particular component must be identified before drawing a state machine diagram.

The primary focus of the state machine diagram is to depict the states of a system. These states are essential while drawing a state transition diagram. The objects, states, and events due to which the state transition occurs must be acknowledged before the implementation of a state machine diagram.

Following are the steps that are to be incorporated while drawing a state machine diagram:

1. A unique and understandable name should be assigned to the state transition that describes the behavior of the system.
2. Out of multiple objects, only the essential objects are implemented.

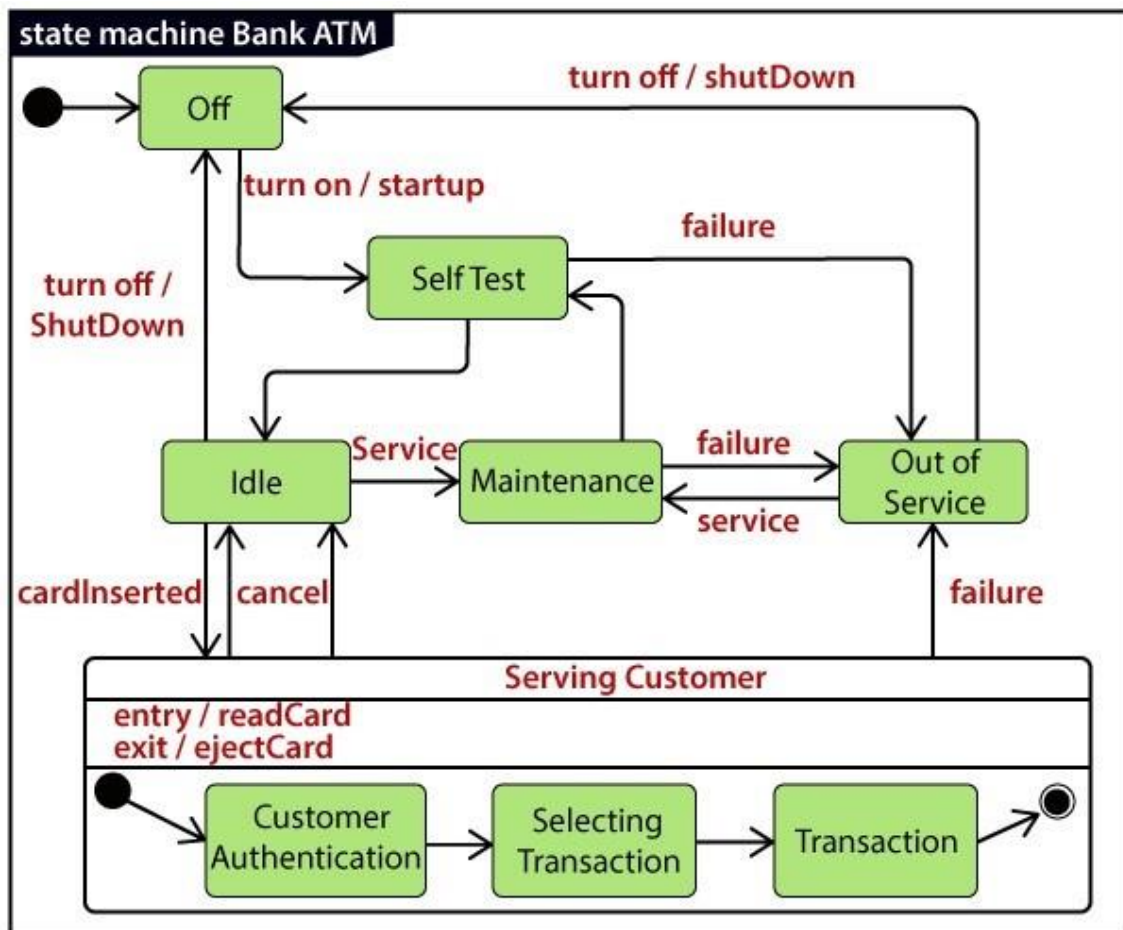
3. A proper name should be given to the events and the transitions.

o Example of a State Machine Diagram

An example of a top-level state machine diagram showing Bank Automated Teller Machine (ATM) is given below.

Initially, the ATM is turned off. After the power supply is turned on, the ATM starts performing the startup action and enters into the Self Test state. If the test fails, the ATM will enter into the Out Of Service state, or it will undergo a triggerless transition to the Idle state. This is the state where the customer waits for the interaction.

Whenever the customer inserts the bank or credit card in the ATM's card reader, the ATM state changes from Idle to Serving Customer, the entry action readCard is performed after entering into Serving Customer state. Since the customer can cancel the transaction at any instant, so the transition from Serving Customer state back to the Idle state could be triggered by cancel event.



Here the Serving Customer is a composite state with sequential substates that are Customer Authentication, Selecting Transaction, and Transaction.

Customer Authentication and Transaction are the composite states itself is displayed by a hidden decomposition indication icon. After the transaction is finished, the Serving Customer encompasses a triggerless transition back to the Idle state. On leaving the state, it undergoes the exit action ejectCard that discharges the customer card.

- o **Activity Diagram:**

- o In UML, the activity diagram is used to demonstrate the flow of control within the system rather than the implementation. It models the concurrent and sequential activities.
- o The activity diagram helps in envisioning the workflow from one activity to another. It put emphasis on the condition of flow and the order in which it occurs. The flow can be sequential, branched, or concurrent, and to deal with such kinds of flows, the activity diagram has come up with a fork, join, etc.
- o It is also termed as an object-oriented flowchart. It encompasses activities composed of a set of actions or operations that are applied to model the behavioral diagram.

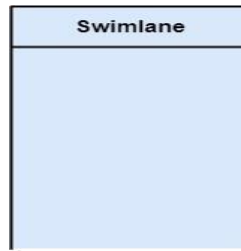
- o **Components of an Activity Diagram**

Following are the component of an activity diagram:

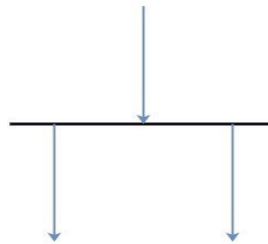
- o **Activities**
- o The categorization of behavior into one or more actions is termed as an activity. In other words, it can be said that an activity is a network of nodes that are connected by edges. The edges depict the flow of execution. It may contain action nodes, control nodes, or object nodes.
- o The control flow of activity is represented by control nodes and object nodes that illustrates the objects used within an activity. The activities are initiated at the initial node and are terminated at the final node.



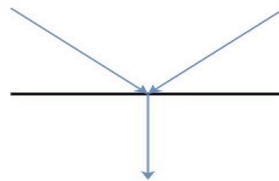
- o **Activity partition /swimlane** o The swimlane is used to cluster all the related activities in one column or one row. It can be either vertical or horizontal. It used to add modularity to the activity diagram. It is not necessary to incorporate swimlane in the activity diagram. But it is used to add more transparency to the activity diagram.



- o Forks
- o Forks and join nodes generate the concurrent flow inside the activity. A fork node consists of one inward edge and several outward edges. It is the same as that of various decision parameters. Whenever a data is received at an inward edge, it gets copied and split crossways various outward edges. It split a single inward flow into multiple parallel flows.



- o Join Nodes
- o Join nodes are the opposite of fork nodes. A Logical AND operation is performed on all of the inward edges as it synchronizes the flow of input across one single output (outward) edge.



- o Pins
 - o It is a small rectangle, which is attached to the action rectangle. It clears out all the messy and complicated thing to manage the execution flow of activities. It is an object node that precisely represents one input to or output from the action.
- o Notation of an Activity diagram

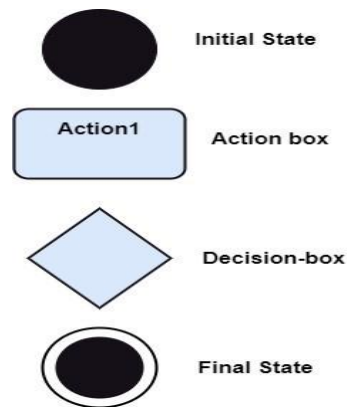
Activity diagram constitutes following notations:

Initial State: It depicts the initial stage or beginning of the set of actions.

Final State: It is the stage where all the control flows and object flows end.

Decision Box: It makes sure that the control flow or object flow will follow only one path.

Action Box: It represents the set of actions that are to be performed.



o How to draw an Activity Diagram?

An activity diagram is a flowchart of activities, as it represents the workflow among various activities. They are identical to the flowcharts, but they themselves are not exactly the flowchart. In other words, it can be said that an activity diagram is an enhancement of the flowchart, which encompasses several unique skills.

Since it incorporates swimlanes, branching, parallel flows, join nodes, control nodes, and forks, it supports exception handling. A system must be explored as a whole before drawing an activity diagram to provide a clearer view of the user. All of the activities are explored after they are properly analyzed for finding out the constraints applied to the activities. Each and every activity, condition, and association must be recognized.

After gathering all the essential information, an abstract or a prototype is built, which is then transformed into the actual diagram.

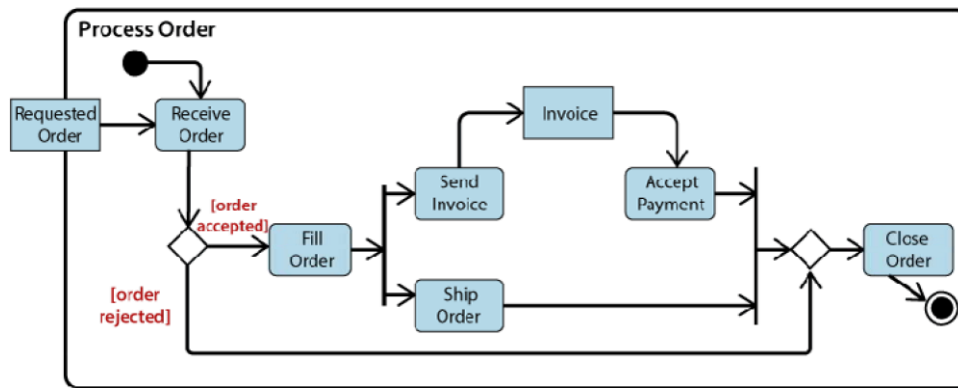
Following are the rules that are to be followed for drawing an activity diagram:

1. A meaningful name should be given to each and every activity.
2. Identify all of the constraints.
3. Acknowledge the activity associations.

o Example of an Activity Diagram

An example of an activity diagram showing the business flow activity of order processing is given below.

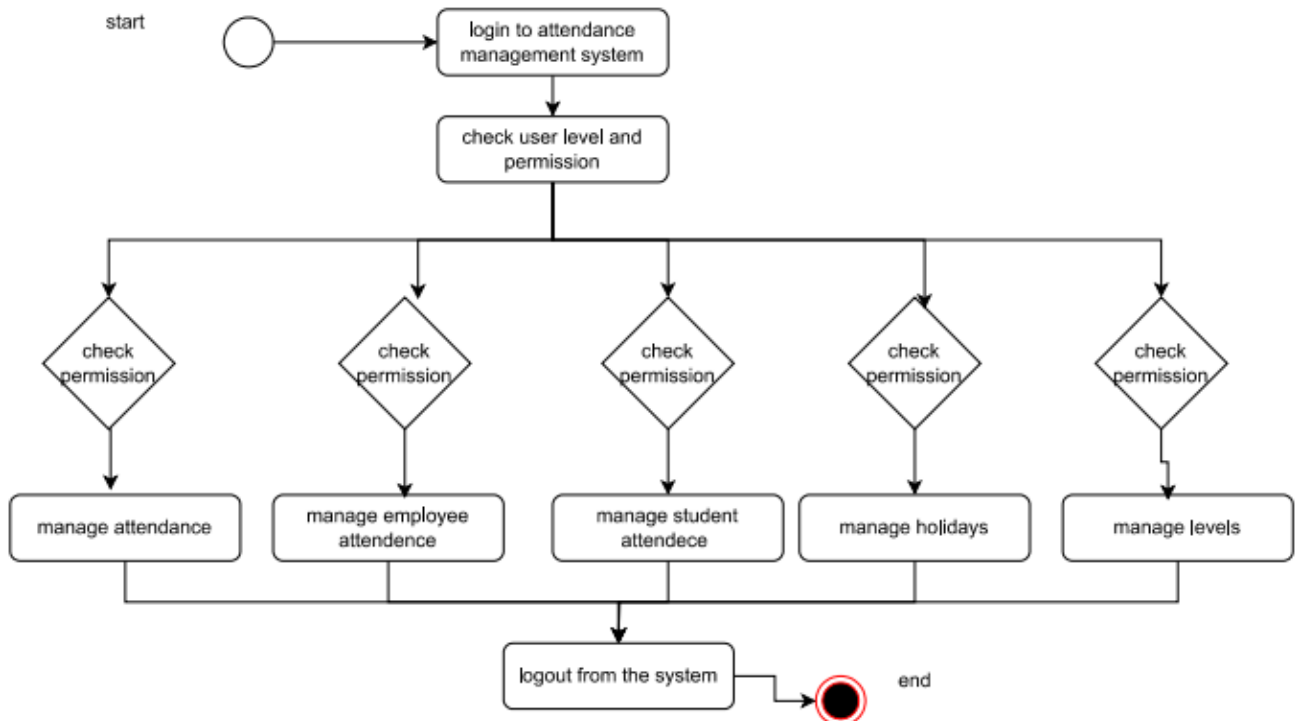
Here the input parameter is the Requested order, and once the order is accepted, all of the required information is then filled, payment is also accepted, and then the order is shipped. It permits order shipment before an invoice is sent or payment is completed.



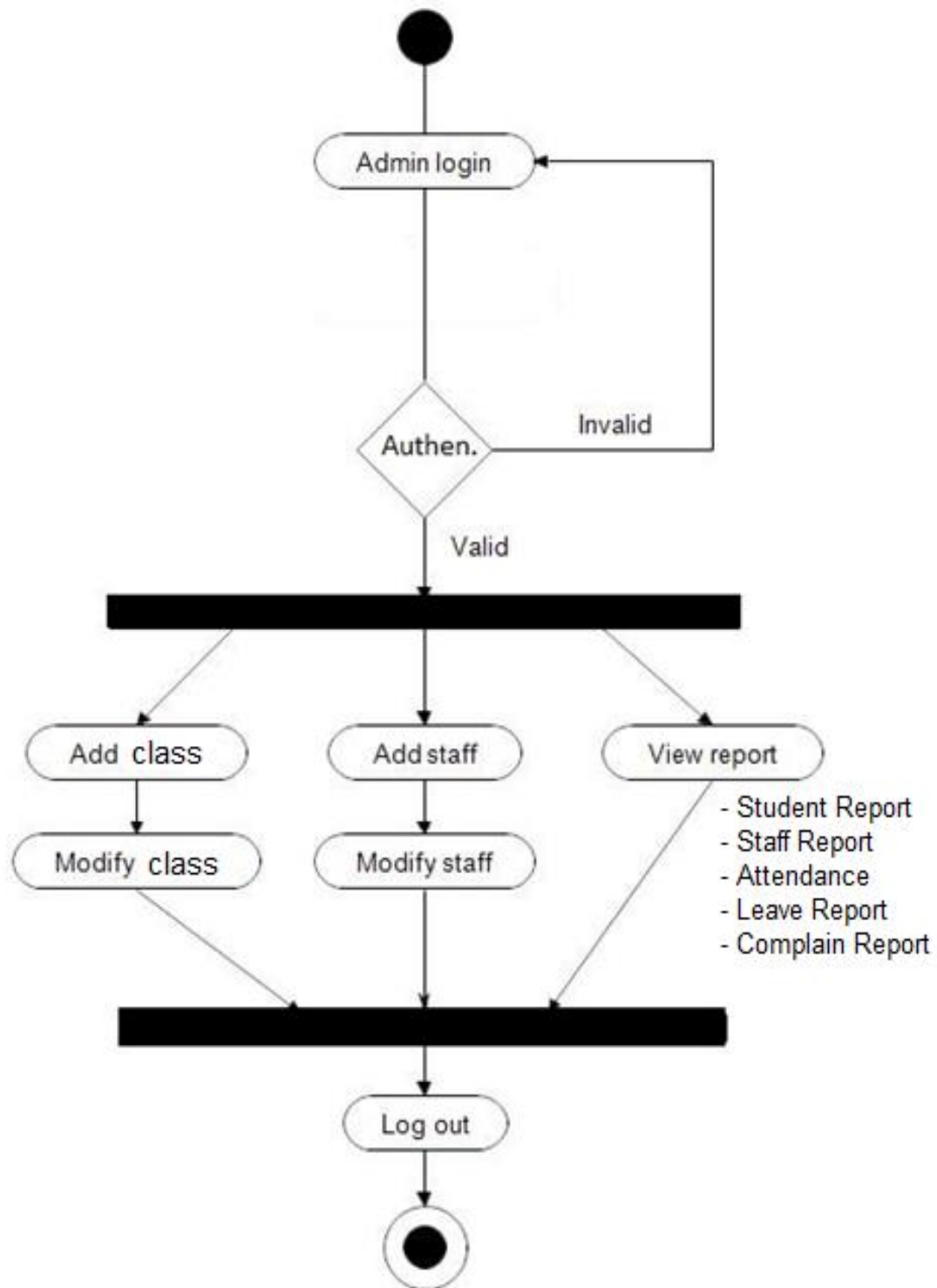
Suggested Reference:

- 1) UML 2 State Machine Diagrams.
- 2) Modeling Behavior with UML Interactions and State charts
- 3) UML 2 State Machine Diagramming Guidelines
- 4) State chart Diagram
- 5) PlantUML (Open-Source tool in Java to draw UML Diagram)

State chart diagram:



Activity diagram:



Rubric wise marks obtained:

Rubrics	1	2	3	4	5	Total
Marks	Complete implementation as asked	Complete implementation as asked Problem analysis	Complete implementation as asked Problem analysis Development of the Solution	Complete implementation as asked Problem analysis Development of the Solution Concept Clarity & understanding	Complete implementation as asked Problem analysis Development of the Solution Concept Clarity & understanding Correct answer to all questions	

Signature of Faculty:

Practical – 8

AIM: Prepare the implementation view: Draw Component diagram and Deployment diagram.

- Objectives: to explore and use various elements UML component diagram and deployment diagram.
- Theory:
 - o A component diagram is used to break down a large object-oriented system into the smaller components, so as to make them more manageable. It models the physical view of a system such as executables, files, libraries, etc. that resides within the node.
 - o It visualizes the relationships as well as the organization between the components present in the system. It helps in forming an executable system. A component is a single unit of the system, which is replaceable and executable. The implementation details of a component are hidden, and it necessitates an interface to execute a function. It is like a black box whose behavior is explained by the provided and required interfaces.

- o Purpose of a Component Diagram

Since it is a special kind of a UML diagram, it holds distinct purposes. It describes all the individual components that are used to make the functionalities, but not the functionalities of the system. It visualizes the physical components inside the system. The components can be a library, packages, files, etc.

The component diagram also describes the static view of a system, which includes the organization of components at a particular instant. The collection of component diagrams represents a whole system.

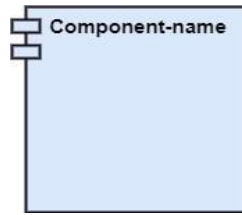
The main purpose of the component diagram are enlisted below:

1. It envisions each component of a system.
2. It constructs the executable by incorporating forward and reverse engineering.
3. It depicts the relationships and organization of components.

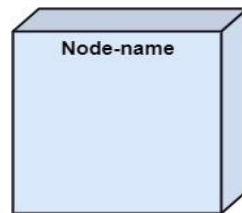
- Background / Preparation:

- o Notation of a Component Diagram

- a) A component



b) A node



o When to use a Component Diagram?

It represents various physical components of a system at runtime. It is helpful in visualizing the structure and the organization of a system. It describes how individual components can together form a single system. Following are some reasons, which tells when to use component diagram:

1. To divide a single system into multiple components according to the functionality.
2. To represent the component organization of the system.

- Tools / Material Needed:

- o Hardware:
 - o Software:

- Procedure / Steps:

- o How to Draw a Component Diagram?

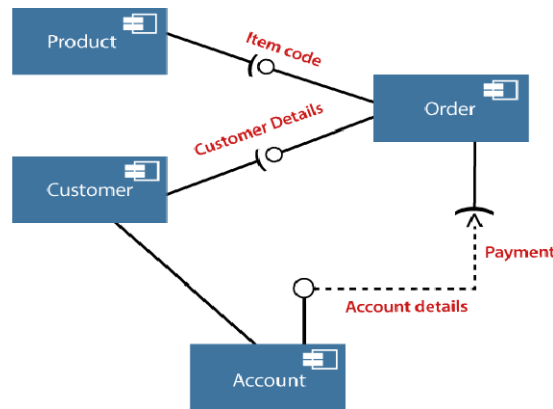
- ☐ The component diagram is helpful in representing the physical aspects of a system, which are files, executables, libraries, etc. The main purpose of a component diagram is different from that of other diagrams. It is utilized in the implementation phase of any application.
 - ☐ Once the system is designed employing different UML diagrams, and the artifacts are prepared, the component diagram is used to get an idea of implementation. It plays an essential role in implementing applications efficiently.
 - ☐ Following are some artifacts that are needed to be identified before drawing a component diagram:

1. What files are used inside the system?
2. What is the application of relevant libraries and artifacts?
3. What is the relationship between the artifacts?

□ Following are some points that are needed to be kept in mind after the artifacts are identified:

1. Using a meaningful name to ascertain the component for which the diagram is about to be drawn.
2. Before producing the required tools, a mental layout is to be made.
3. To clarify the important points, notes can be incorporated.

o **Example of a Component Diagram** o A component diagram for an online shopping system is given below:

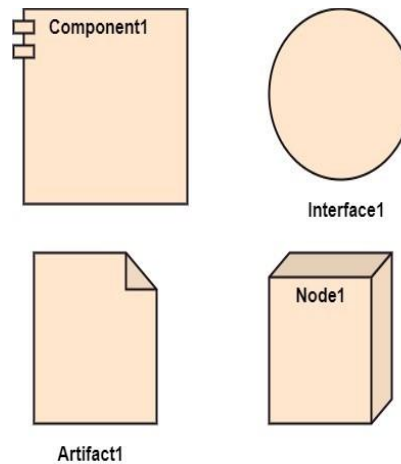


- o The deployment diagram visualizes the physical hardware on which the software will be deployed. It portrays the static deployment view of a system. It involves the nodes and their relationships.
- o It ascertains how software is deployed on the hardware. It maps the software architecture created in design to the physical system architecture, where the software will be executed as a node. Since it involves many nodes, the relationship is shown by utilizing communication paths.
- o **Purpose of Deployment Diagram**

- The main purpose of the deployment diagram is to represent how software is installed on the hardware component. It depicts in what manner a software interacts with hardware to perform its execution.
- Both the deployment diagram and the component diagram are closely interrelated to each other as they focus on software and hardware components. The component diagram represents the components of a system, whereas the deployment diagram describes how they are actually deployed on the hardware.

- o The deployment diagram does not focus on the logical components of the system, but it put its attention on the hardware topology.
- o Following are the purposes of deployment diagram enlisted below:
 - To envision the hardware topology of the system.
 - To represent the hardware components on which the software components are installed.
 - To describe the processing of nodes at the runtime.
- o **Symbol and notation of Deployment diagram** o The deployment diagram consist of the following notations:

1. A component
2. An artifact
3. An interface
4. A node



- o **How to draw a Deployment Diagram?**
 - The deployment diagram portrays the deployment view of the system. It helps in visualizing the topological view of a system. It incorporates nodes, which are physical hardware. The nodes are used to execute the artifacts. The instances of artifacts can be deployed on the instances of nodes.
 - Since it plays a critical role during the administrative process, it involves the following parameters:
 1. High performance
 2. Scalability
 3. Maintainability
 4. Portability
 5. Easily understandable

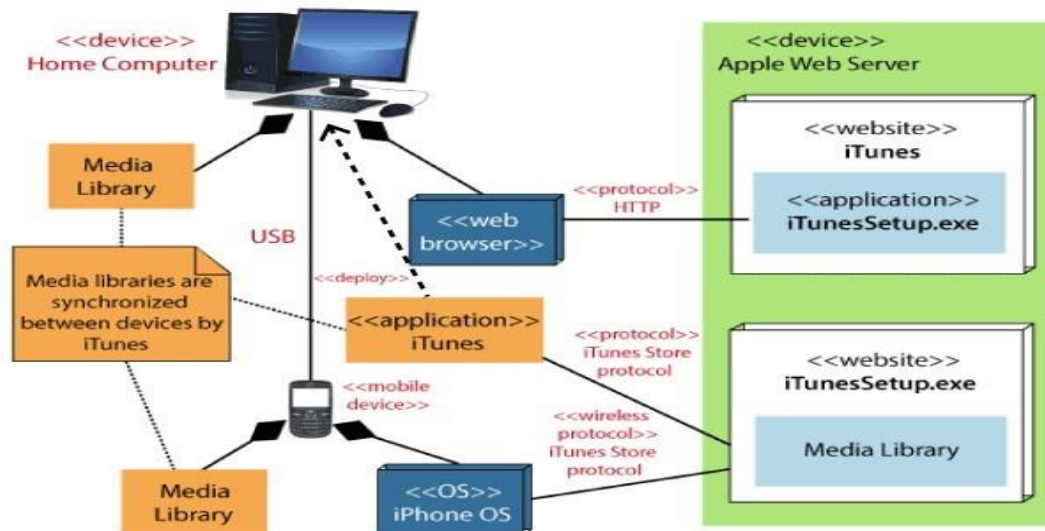
- One of the essential elements of the deployment diagram is the nodes and artifacts. So it is necessary to identify all of the nodes and the relationship between them. It becomes easier to develop a deployment diagram if all of the nodes, artifacts, and their relationship is already known.

- o Example of a Deployment diagram

A deployment diagram for the Apple iTunes application is given below.

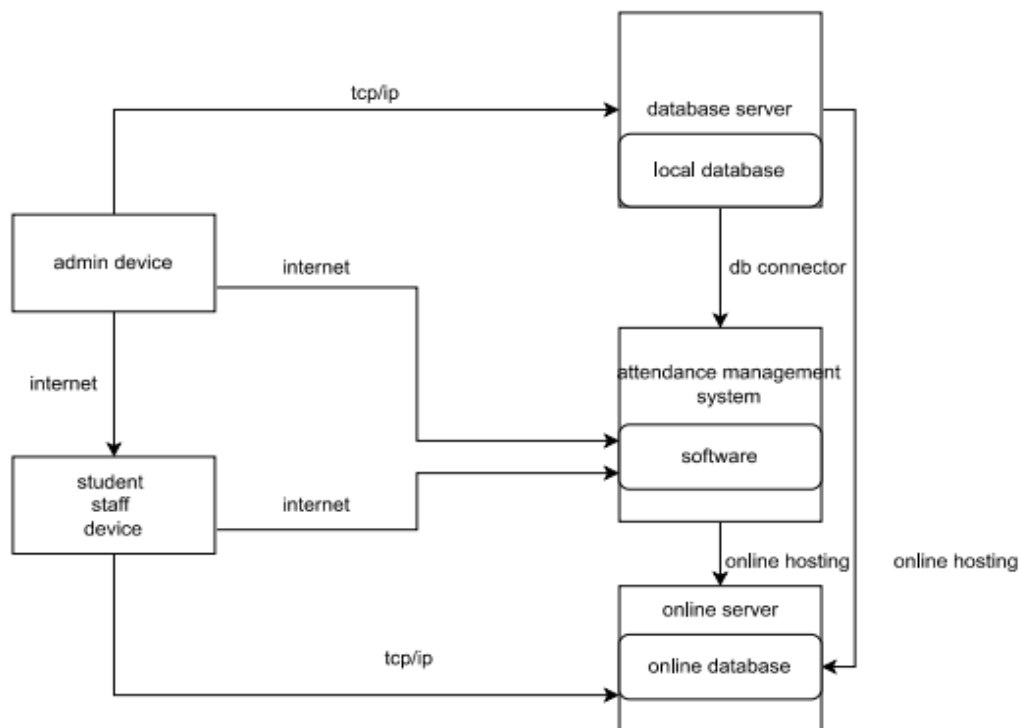
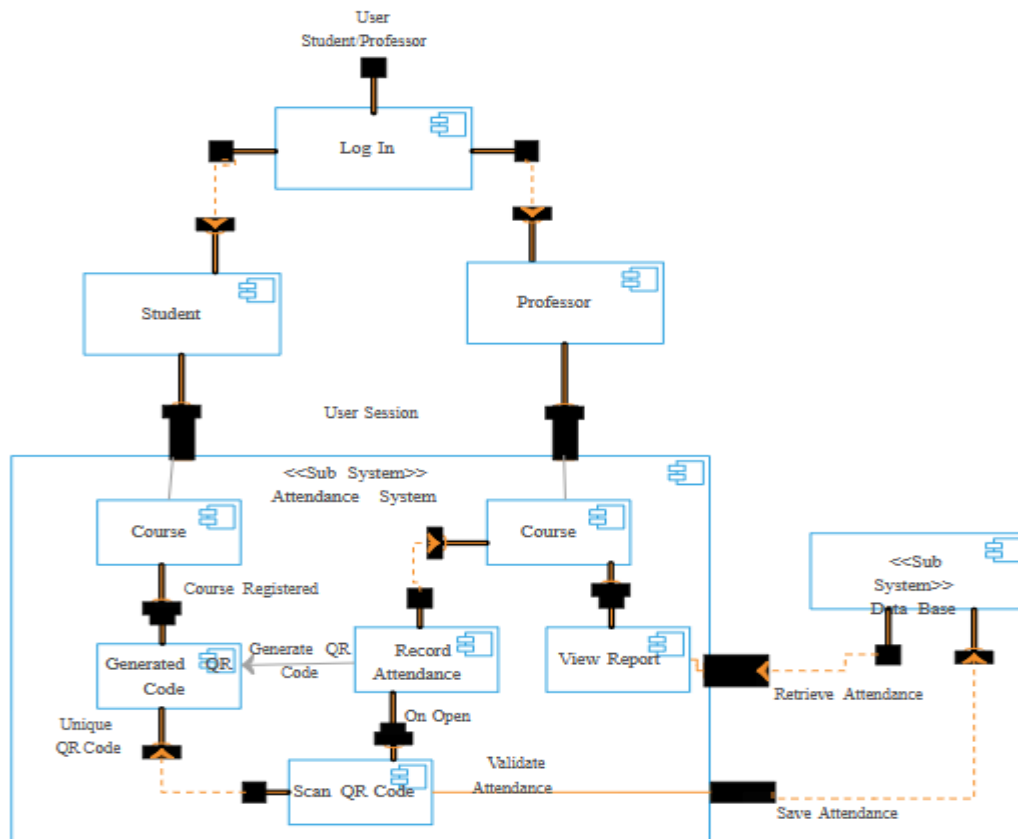
The iTunes setup can be downloaded from the iTunes website, and also it can be installed on the home computer. Once the installation and the registration are done, iTunes application can easily interconnect with the Apple iTunes store. Users can purchase and download music, video, TV serials, etc. and cache it in the media library.

Devices like Apple iPod Touch and Apple iPhone can update its own media library from the computer with iTunes with the help of USB or simply by downloading media directly from the Apple iTunes store using wireless protocols, for example; Wi-Fi, 3G, or EDGE.



Deployment diagram

Component diagram



Suggested Reference:

1. Introduction to the Diagrams of UML 2.X

Rubric wise marks obtained:

Rubrics	1	2	3	4	5	Total
Marks	Complete implementation as asked	Complete implementation as asked Problem analysis	Complete implementation as asked Problem analysis Development of the Solution	Complete implementation as asked Problem analysis Development of the Solution Concept Clarity & understanding	Complete implementation as asked Problem analysis Development of the Solution Concept Clarity & understanding Correct answer to all questions	

Signature of Faculty:

Practical – 9

AIM: Prepare the quality management plan.

- **Objectives:** To explore the contents which is required to create quality management plan
- **Theory:**

Quality Management Plan Content

- Overview of Quality Management Plan
- Purpose
- Overview of Plan, Do, Check, Act
- Customer Quality Objectives
- Identify Customer Quality Objectives
- Identify Quality Threshold for each Quality Objective
- Quality Control Plans
- Address each major deliverable
- Identify Independent Technical Review Team(s)
- Quality Assurance
- Organizational Quality System Requirements (Organizational Quality Management Plan)
- Project-specific requirements
- Other Project Specific Information as required

Sample Quality Management Plan

Quality Management Relationships

	Quality Planning	Quality Control (QC)	Quality Assurance (QA)	Quality Improvement
	Plan	Do	Check	Act
What Is Done	Determine what will be quality on the project and how quality will be measured	Monitor specific project products to determine if they meet performance measurement thresholds defined in the quality management plan	Determine if measurement of quality is appropriate by evaluating overall performance on a regular basis to insure the project will satisfy customer quality expectations	Increase the effectiveness and efficiency of the project when corrective actions such as Change Requests are identified. Changes to the Quality Management Plan and the PMP may be required.
When It Is Done	Project Planning Phase Processes <ul style="list-style-type: none"> • PMP Development • Project Scope & Customer Requirements Definition • Team Establishment • Activity/Schedule Development • Resource Estimate Development • Project Delivery Acquisition Strategy 	Project Execution, & Control Phase Processes <ul style="list-style-type: none"> • Project Execution & Control • Lessons Learned 	Project Execution, & Control Phase Processes <ul style="list-style-type: none"> • Project Execution & Control 	Project Execution, & Control Phase and Project Planning Phase Processes <ul style="list-style-type: none"> • Change Management • PMP Development

Plan

- Identify the customers Quality Objectives. Help customers express quality expectations in objective, quantitative terms.
- Identify professional standards including legal, environmental, economic, code, life safety and health.
- Balance needs and expectations of customers and stakeholders with cost, schedule, and professional standards. Evaluate the costs and benefits of selected quality objectives and the processes to be used to achieve objectives.

- Develop an effective plan and processes, including quality assurance and quality control procedures, to achieve objectives. Consider risk/hazard factors and complexity of the project and adapt processes to provide the requisite level of quality. Document in the risk management plan any project variations from the local QMP requirements.
- Develop performance measure thresholds to ensure agreement on the definition of success relative to Quality Objectives.
- Ensure customer endorsement of all quality objectives included in the Quality Management Plan.

Do

- Do the work according to the approved PMP and standard operating procedures.
- Project execution is a dynamic process. The PDT must communicate, meet on a regular basis, and adapt to changing conditions. The Quality Management Plan and PMP may require modification to ensure that project objectives are met.

Check

- Perform independent technical review, management oversight, and verification to ensure that quality objectives are met consistent with District Quality Management Plans.
- Check performance against the PMP and Customer Quality Objectives performance measures thresholds to verify that performance will accomplish Quality Objectives and to verify sufficiency of the plan. Share findings with all project stakeholders to facilitate continuous improvement.

Act

- If performance measures thresholds are exceeded, take specific corrective actions to fix the systemic cause of any non-conformance, deficiency, or other unwanted effect.
- Document quality improvements that could include appropriate revisions to the quality management plan, alteration of quality assurance and control procedures, and adjustments to resource allocations.

Suggested Reference:

- 1) Rajib Mall, Fundamentals of software Engineering, Prentice Hall of India.
- 2) Pankaj Jalote, Software Engineering – A Precise Approach Wiley

Overview of Quality Management Plan: -

Purpose: A Quality Management Plan (QMP) for an Learn Easy is a crucial document that outlines the strategies, processes, and procedures to ensure that the platform meets or exceeds the desired quality standards.

Overview: The QMP will adhere to the PDCA cycle, which include planning, executing, monitoring and controlling, and continuous improvement.

Customer Quality Objectives:

1. Accuracy of Attendance Records:

Objective: Ensure that the attendance records are accurate and error-free.

Measure: Less than 1% discrepancy between manually recorded attendance and the system's records.

2. Reliability and Uptime:

Objective: Ensure the system is available and reliable at all times.

Measure: 99.9% system uptime, with minimal downtime for maintenance.

3. Usability and User-Friendliness:

Objective: Ensure the system is easy to use and intuitive for all types of users.

Measure: 90% user satisfaction rating based on user feedback surveys.

4. Data Security and Privacy:

Objective: Protect student and employee data from unauthorized access.

Measure: Zero data breaches and compliance with data protection regulations (e.g., GDPR).

5. Scalability:

Objective: Ensure the system can scale to accommodate a growing number of users and records.

Quality Threshold for each Quality Objective:

1. Accuracy of Attendance Records:

Threshold: Less than 1% discrepancy between manually recorded attendance and system records.

2. Reliability and Uptime:

Threshold: 99.9% system uptime, with less than 0.1% downtime for maintenance.

3. Usability and User-Friendliness:

Threshold: Maintain a user satisfaction rating of at least 90% based on user feedback surveys.

4. Data Security and Privacy

Threshold: Zero data breaches; full compliance with data protection regulations.

5. Scalability

Threshold: System performance remains consistent as the number of users and records increases.

6. Response Time:

Threshold: Respond to at least 95% of user interactions within 2 seconds.

7. Error Handling

Threshold: Less than 1% of user interactions result in system errors.

Quality Control Plans:

1. Content Review and Approval Process:

Plan: Define a standardized content creation process, including authoring guidelines and review protocols. Establish a content review team with subject matter experts (SMEs) and instructional designers.

Do: Authors create content according to established guidelines. Content review team evaluates content for accuracy, relevance, and alignment with learning objectives.

Check: Conduct regular content audits to ensure compliance with content creation standards. Review team provides feedback to authors and ensures necessary revisions are made.

Act: Update content creation guidelines based on lessons learned from reviews. Provide additional training or resources to authors for improved content creation.

2. User Experience Testing:

Plan: Define a testing protocol that covers various aspects of user experience (UI/UX). Specify testing environments and user demographics for testing.

Do: Conduct usability testing with representative users to evaluate platform navigation, accessibility, and overall experience.

Check: Analyze usability test results to identify areas for improvement in the user interface and overall experience. Document usability issues and prioritize them based on severity.

Act: Implement UI/UX improvements based on usability test findings. Conduct follow-up usability testing to validate the effectiveness of the implemented changes.

3. Accessibility and Inclusivity Testing:

Plan: Establish testing criteria based on accessibility standards (e.g., WCAG). Define a protocol for testing with assistive technologies.

Do: Conduct accessibility testing using various assistive technologies (e.g., screen readers, voice recognition software). Verify compliance with WCAG guidelines.

Check: Evaluate test results to identify areas of non-compliance or accessibility issues. Document findings and prioritize necessary changes.

Act: Implement accessibility enhancements and conduct retesting to confirm compliance. Provide training to content creators and developers on accessibility best practices.

4. Compatibility Testing:

Plan: Specify the list of target devices, browsers, and operating systems for compatibility testing. Define testing scenarios to cover a range of use cases.

Do: Conduct compatibility testing on identified devices, browsers, and operating systems. Verify that the platform functions consistently across all specified environments.

Check: Review compatibility test results to identify any issues or inconsistencies in platform performance.

Act: Address identified compatibility issues through development or configuration adjustments. Update the list of supported devices, browsers, and operating systems as needed.

Independent Technical Review Team(s):

An Independent Technical Review Team Consisting of Experienced Software engineers and quality Assurance experts will be established. This team will conduct periodic reviews of project deliverables , code , and processes to identify issues and suggest improvements.

Quality Assurance:

The Project will adhere to the organization's overarching quality management plan, which includes documented quality processes, procedures, and guidelines. This includes the organization's commitment to continuous improvement and adherence to industry best practices.

Project-specific requirements:

Project-specific quality requirements will be identified, documented, and communicated to the project team.

User Authentication and Authorization: Implement secure user authentication and authorization mechanisms to ensure that only authorized users have access to the platform.

User Profiles and Personalization: Allow users to create and manage their profiles, including personal information, preferences, and progress tracking.

Course Creation and Management: Enable instructors or content creators to easily create, organize, and manage courses, including uploading various types of content (text, video, audio, quizzes, etc.).

Content Delivery and Accessibility: Ensure that content is accessible on different devices (desktop, laptop, tablet, smartphone) and complies with accessibility standards (e.g., WCAG) to accommodate users with disabilities.

Interactive Learning Features: Include interactive elements such as quizzes, assignments, discussion forums, polls, and collaborative projects to engage learners.

Progress Tracking and Reporting: Provide tools for learners and administrators to track progress, view completion certificates, and generate reports on user performance.

Assessment and Evaluation Tools: Offer various types of assessments (e.g., multiple-choice, essays, peer reviews) and tools for instructors to evaluate learner performance.

Content Search and Filtering: Implement a robust search and filtering system to help users easily find courses, modules, or specific pieces of content.

Notification and Communication System: Enable notifications for important updates, assignments, discussions, and facilitate communication between learners and instructors.

Feedback and Rating System: Allow learners to provide feedback on courses and modules, and enable rating and review features to help others make informed choices.

Other Project Specific Information are required:

Target Audience Demographics: Define the age group, educational background, professions, and any other relevant demographic information about the intended users.

Subject Matter or Industry Focus: Specify the field or industry that the e-learning platform is catering to (e.g., healthcare, technology, language learning, professional development).

Licensing and Copyright Considerations: Determine if there are any specific licensing or copyright requirements for the content being used or generated on the platform.

Timeline and Milestones: Set specific milestones and deadlines for different phases of the project, including development, testing, content creation, and platform launch.

Feedback and Iteration Process: Specify how user feedback will be collected, analyzed, and used to inform ongoing improvements and updates.

Rubric wise marks obtained:

Rubrics	1	2	3	4	5	Total
Marks	Complete implementation as asked	Complete implementation as asked Problem analysis	Complete implementation as asked Problem analysis Development of the Solution	Complete implementation as asked Problem analysis Development of the Solution Concept Clarity & understanding	Complete implementation as asked Problem analysis Development of the Solution Concept Clarity & understanding Correct answer to all questions	

Signature of Faculty:

Practical – 10

AIM: To perform various testing using the testing tool unit testing, integration testing design test cases..

- Objectives: to explore and learn about different testing techniques and use them.
- Theory:
 - o Software Testing is evaluation of the software against requirements gathered from users and system specifications. Testing is conducted at the phase level in software development life cycle or at module level in program code. Software testing comprises of Validation and Verification.
 - o **Software Validation**
 - ☐ Validation is process of examining whether or not the software satisfies the user requirements. It is carried out at the end of the SDLC. If the software matches requirements for which it was made, it is validated.
 - o Validation ensures the product under development is as per the user requirements.
 - o Validation answers the question – "Are we developing the product which attempts all that user needs from this software ?".
 - o Validation emphasizes on user requirements.
 - o **Software Verification**
 - ☐ Verification is the process of confirming if the software is meeting the business requirements, and is developed adhering to the proper specifications and methodologies.
 - ☐ Verification ensures the product being developed is according to design specifications.
 - ☐ Verification answers the question– "Are we developing this product by firmly following all design specifications ?"
 - ☐ Verifications concentrates on the design and system specifications.
 - o Target of the test are -
- Errors - These are actual coding mistakes made by developers. In addition, there is a difference in output of software and desired output, is considered as an error.

- Fault - When error exists fault occurs. A fault, also known as a bug, is a result of an error which can cause system to fail.
- Failure - failure is said to be the inability of the system to perform the desired task. Failure occurs when fault exists in the system.

o Testing Levels

- Testing itself may be defined at various levels of SDLC. The testing process runs parallel to software development. Before jumping on the next stage, a stage is tested, validated and verified.
- Testing separately is done just to make sure that there are no hidden bugs or issues left in the software. Software is tested on various levels - o Unit Testing

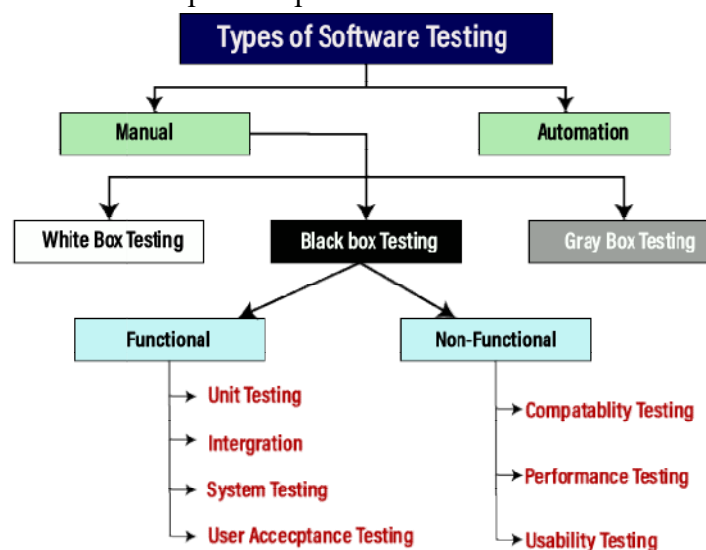
While coding, the programmer performs some tests on that unit of program to know if it is error free. Testing is performed under white-box testing approach. Unit testing helps developers decide that individual units of the program are working as per requirement and are error free.

o Integration Testing

Even if the units of software are working fine individually, there is a need to find out if the units if integrated together would also work without errors. For example, argument passing and data updation etc.

o System Testing

The software is compiled as product and then it is tested as a whole.



- Background / Preparation:

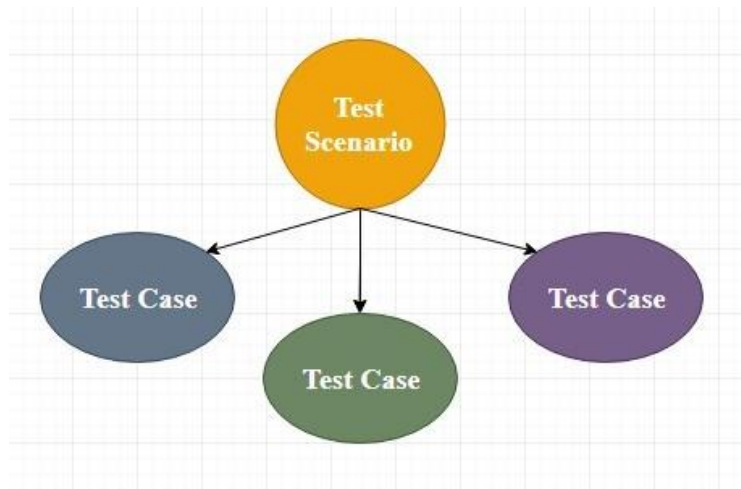
- o **Test management tool** o Test management tools are used to keep track of all the testing activity, fast data analysis, manage manual and automation test cases, various environments, and plan and maintain manual testing as well.
- o Test management tools are used to keep track of all the testing activity, fast data analysis, manage manual and automation test cases, various environments, and plan and maintain manual testing as well.
- o The test management tool is connected with the automation software. These types of tools had various strategies for testing and multiple sets of features. Some of the test management tools had capabilities to design the test case with the help of requirements.
- o It is best for test managing, scheduling, defect logging, tracking, and analysis.
- o Some of the most commonly used test management tools are as follows:
- o Quality center o RTH
- o Testpad o Test Monitor o PractiTest

- Tools / Material Needed:

- o Hardware:
- o Software:

- Procedure / Steps:

- o RTH is another web-based open-source tool, and it stands for the Requirement and testing hub. It is used to manage the requirements, test results, and also have bug tracking facilities. This tool follows a structured approach to extend the visibility of the testing process with the help of a common repository for test cases, test result, requirements, and test plans.
- o Test Cases:
 - The test case is defined as a group of conditions under which a tester determines whether a software application is working as per the customer's requirements or not. Test case designing includes preconditions, case name, input conditions, and expected result. A test case is a first level action and derived from test scenarios.



- o **Test case template** o The primary purpose of writing a test case is to achieve the efficiency of the application.

Header						
Test Case Name/ID :- Release - Version - Application Name - Module						
Test Case Type:- E.T.C I.T.C S.T.C						
Requirement Number:-						
Module:-						
Serverity:- Critical/Major/Minor						
Status:-						
Release:-						
Version:-						
Pre-condition:-						
Test Data:-						
Summary:-						
Body						
Step No.	Descri- pation	Inputs	Expected Result	Actual Reasult	Status	Comments
...
...
Footer						
Author:-			Reviewd By:-			
Date:-			Approved By:-			

As we know, the actual result is written after the test case execution, and most of the time, it would be same as the expected result. But if the test step will fail, it will be different. So, the actual result field can be skipped, and in the Comments section, we can write about the bugs.

And also, the Input field can be removed, and this information can be added to the Description field.

The above template we discuss above is not the standard one because it can be different for each company and also with each application, which is based on the test engineer and the test lead. But, for testing one application, all the test engineers should follow a usual template, which is formulated.

The test case should be written in simple language so that a new test engineer can also understand and execute the same.

In the above sample template, the header contains the following:

Step number

It is also essential because if step number 20 is failing, we can document the bug report and hence prioritize working and also decide if it's a critical bug.

Test case type

It can be functional, integration or system test cases or positive or negative or positive and negative test cases.

Release

One release can contain many versions of the release.

Pre-condition

These are the necessary conditions that need to be satisfied by every test engineer before starting the test execution process. Or it is the data configuration or the data setup that needs to be created for the testing.

For example: In an application, we are writing test cases to add users, edit users, and delete users. The per-condition will be seen if user A is added before editing it and removing it.

Test data

These are the values or the input we need to create as per the per-condition.

For example, Username, Password, and account number of the users.

The test lead may be given the test data like username or password to test the application, or the test engineer may themselves generate the username and password.

- o **Test Cases – Login Page** o Following is the possible list of functional and non-functional test cases for a login page:
- o **Functional Test Cases:**

Sr. No.	Functional Test Cases	Type- Negative/ Positive Test Case
1	Verify if a user will be able to login with a valid username and valid password.	Positive
Sr. No.	Functional Test Cases	Type- Negative/ Positive Test Case
2	Verify if a user cannot login with a valid username and an invalid password.	Negative
3	Verify the login page for both, when the field is blank and Submit button is clicked.	Negative
4	Verify the 'Forgot Password' functionality.	Positive
5	Verify the messages for invalid login.	Positive
6	Verify the 'Remember Me' functionality.	Positive
7	Verify if the data in password field is either visible as asterisk or bullet signs.	Positive
8	Verify if a user is able to login with a new password only after he/she has changed the password.	Positive
9	Verify if the login page allows to log in simultaneously with different credentials in a different browser.	Positive
10	Verify if the 'Enter' key of the keyboard is working correctly on the login page.	Positive
Other Test Cases		
11	Verify the time taken to log in with a valid username and password.	Performance & Positive Testing
12	Verify if the font, text color, and color coding of the Login page is as per the standard.	UI Testing & Positive Testing
13	Verify if there is a 'Cancel' button available to erase the entered text.	Usability Testing

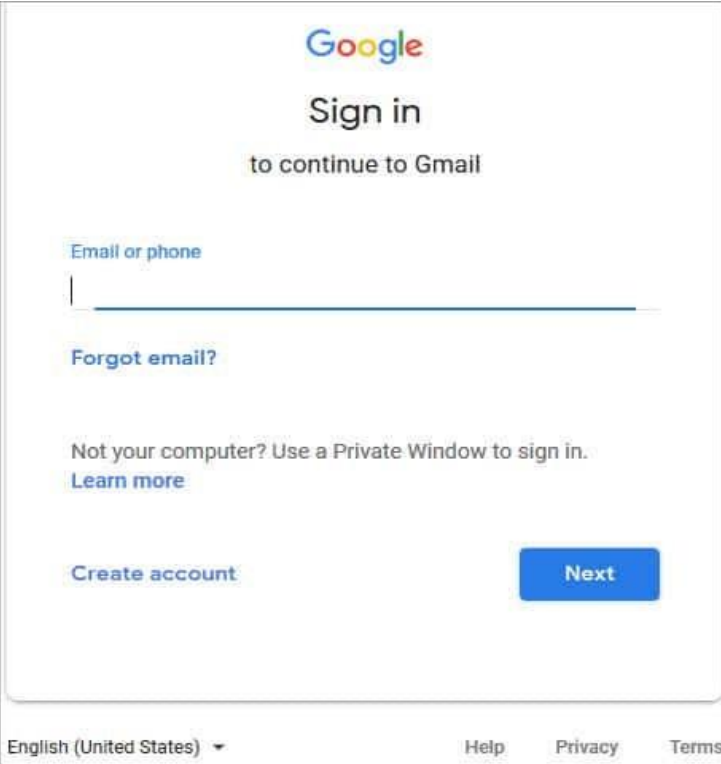
14	Verify the login page and all its controls in different browsers	Browser Compatibility & Positive Testing.
----	--	---

o Non-functional Security Test Cases:

Sr. No.	Security test cases	Type- Negative/ Positive Test Case
1	Verify if a user cannot enter the characters more than the specified range in each field (Username and Password).	Negative
2	Verify if a user cannot enter the characters more than the specified range in each field (Username and Password).	Positive
3	Verify the login page by pressing 'Back button' of the browser. It should not allow you to enter into the system once you log out.	Negative
4	Verify the timeout functionality of the login session.	Positive
Sr. No.	Security test cases	Type- Negative/ Positive Test Case
5	Verify if a user should not be allowed to log in with different credentials from the same browser at the same time.	Negative
6	Verify if a user should be able to login with the same credentials in different browsers at the same time.	Positive
7	Verify the Login page against SQL injection attack.	Negative
8	Verify the implementation of SSL certificate.	Positive

Showing 1 to 8 of 8 entries

We can take an Example of Gmail Login page. Here is the image of it.



Google

Sign in

to continue to Gmail

Email or phone

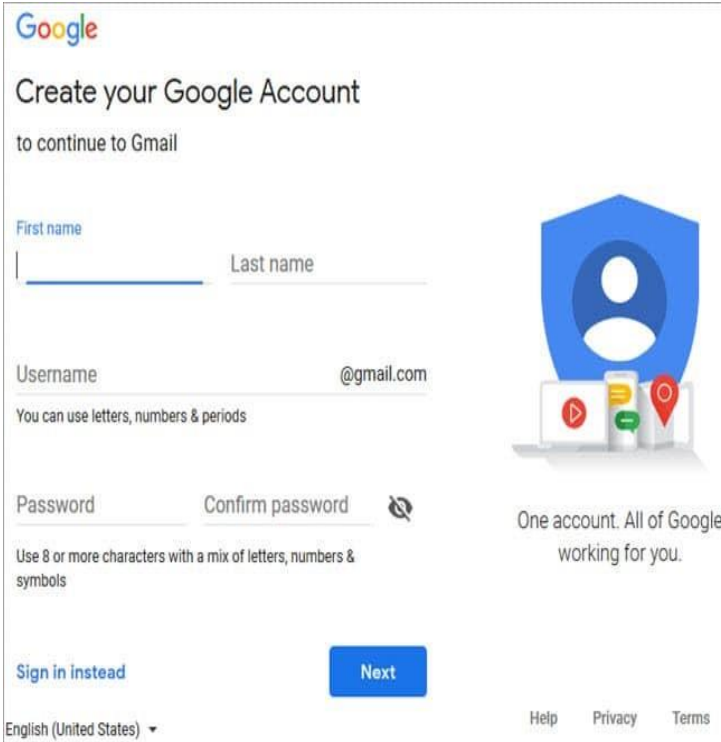
[Forgot email?](#)

Not your computer? Use a Private Window to sign in.
[Learn more](#)

[Create account](#) [Next](#)

English (United States) ▾ [Help](#) [Privacy](#) [Terms](#)

Test Cases for Gmail Login page



Google


Create your Google Account

to continue to Gmail

First name Last name

Username @gmail.com

You can use letters, numbers & periods

Password Confirm password 

Use 8 or more characters with a mix of letters, numbers & symbols

[Sign in instead](#) [Next](#)

English (United States) ▾ [Help](#) [Privacy](#) [Terms](#)

One account. All of Google working for you.

Sr. No.	Test Scenarios
------------	----------------

1	Enter the valid email address & click next. Verify if the user gets an option to enter the password.
2	Don't enter an email address or phone number & just click the Next button. Verify if the user will get the correct message or if the blank field will get highlighted.
3	Enter the invalid email address & click the Next button. Verify if the user will get the correct message.
4	Enter an invalid phone number & click the Next button. Verify if the user will get the correct message.
5	Verify if a user can log in with a valid email address and password.
6	Verify if a user can log in with a valid phone number and password.
7	Verify if a user cannot log in with a valid phone number and an invalid password.
8	Verify if a user cannot log in with a valid email address and a wrong password.
9	Verify the 'Forgot email' functionality.
Sr. No.	Test Scenarios
10	Verify the 'Forgot password' functionality.

Test Scenarios for the Sign-up page

- 1) Verify the messages for each mandatory field.
- 2) Verify if the user cannot proceed without filling all the mandatory fields.
- 3) Verify the age of the user when the DOB is selected.
- 4) Verify if the numbers and special characters are not allowed in the First and Last name.
- 5) Verify if a user can sign-up successfully with all the mandatory details.
- 6) Verify if a user can log in with the valid details.
- 7) Verify if the Password and Confirm Password fields are accepting similar strings only.
- 8) Verify if the Password field will prompt you for the weak passwords.
- 9) Verify if duplicate email address will not get assigned.
- 10) Verify that hints are provided for each field on the form, for the ease of use.

Suggested Reference:

1 Software Testing: A Craftsman's Approach, by Paul C. Jorgensen, Third Edition

2 Software Engineering by Rajib Mall, PHI 2014

Test Case: - Registration

Sr. No.	Functional Test Cases	Type- Negative/ Positive Test Case
1	Verify that the registration page loads.	Positive
2	Verify that all requirements are present.	Positive
3	Verify that user can enter valid email.	Positive
4	Verify that user can't register with an invalid email address.	Negative
5	Verify that user can enter valid password.	Positive
6	Verify that user can not register with an weak password.	Positive
7	Verify that the password confirmation.	Positive
8	Verify that user can click the "Register" button to create an account.	Positive
9	Verify that the user is assigned unique identifier upon successful registration.	Positive
10	Verify that user is informed of any errors or invalid inputs during registration	Positive

Test Case: - Login Page

Sr. No.	Functional Test Cases	Type- Negative/ Positive Test Case
1	Verify if a user will be able to login with a valid username and valid password.	Positive
2	Verify if a user cannot login with a valid username and an invalid password.	Negative
3	Verify the login page for both, when the field is blank and Submit button is clicked.	Negative
4	Verify the 'Forgot Password' functionality.	Positive

5	Verify the messages for invalid login.	Positive
6	Verify the 'Remember Me' functionality.	Positive
7	Verify if the data in password field is either visible as asterisk or bullet signs.	Positive
8	Verify if a user is able to login with a new password only after he/she has changed the password.	Positive
9	Verify if the login page allows to log in simultaneously with different credentials in a different browser.	Positive
10	Verify if the 'Enter' key of the keyboard is working correctly on the login	Positive

Test Case: - Attendance record

Sr. No.	Functional Test Cases	Type- Negative/ Positive Test Case
1	Verify Login as a teacher	Positive
2	Verify admin Select a class and date for attendance..	Positive
3	Verify Mark attendance for each student , Save the attendance and Verify the attendance	Positive
4	report Attempt to log in with invalid credentials	Negative

Test Case :- Generate attendance report

Sr. No.	Functional Test Cases	Type- Negative/ Positive Test Case
1	Login as a teacher/admin.	Positive
2	Go to the attendance management section	Positive
3	Select a class and date range.	Positive
4	Generate the attendance report.	Positive
5.	Mark attendance for a student twice	Negative

Test Case :Leave attendance :-

Sr. No.	Functional Test Cases	Type- Negative/ Positive Test Case
1	Navigate to the "Leave" section.	Positive
2	Fill in invalid leave request details (past dates).	Negative
3	Access the "Leave Requests" section by any user.	Positive
4	Attempt to approve leave request for another manager	Negative

Test-case :Notifications and download records

Sr. No.	Functional Test Cases	Type- Negative/ Positive Test Case
1	User dashboard is displayed.	Positive
2	User receives a notification of attendance being updated.	Positive
3	Teacher/admin dashboard is displayed.	Positive
4	Verify the absence of notification.	Positive
5.	Attendance records are accessible for download..	Positive
6.	Attendance records for the selected period are downloaded.	Positive
7.	System should not allow the download of invalid records.	Negative
8.	Enter an invalid date range and attempt to download records	Negative

Rubric wise marks obtained:

Rubrics	1	2	3	4	5	Total
Marks	Complete implementation as asked	Complete implementation as asked Problem analysis	Complete implementation as asked Problem analysis Development of the Solution	Complete implementation as asked Problem analysis Development of the Solution Concept Clarity & understanding	Complete implementation as asked Problem analysis Development of the Solution Concept Clarity & understanding Correct answer to all questions	

Signature of Faculty:

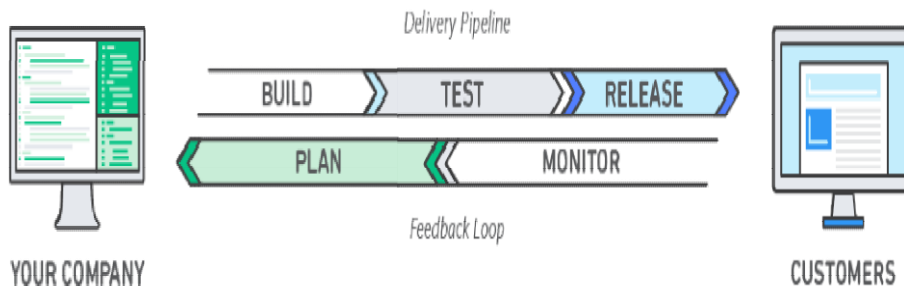
Practical – 11

AIM: Case Study: Study of Open-source tools in DevOps for Infrastructure Automation, Configuration

Management, Deployment Automation, Performance Management, Log Management. Monitoring

- **Objectives:** to explore various case studies where DevOps is applied.
- **Theory:**

DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market.



How Devops works?

Under a DevOps model, development and operations teams are no longer “siloeed.” Sometimes, these two teams are merged into a single team where the engineers work across the entire application lifecycle, from development and test to deployment to operations, and develop a range of skills not limited to a single function.

In some DevOps models, quality assurance and security teams may also become more tightly integrated with development and operations and throughout the application lifecycle. When security is the focus of everyone on a DevOps team, this is sometimes referred to as DevSecOps.

These teams use practices to automate processes that historically have been manual and slow. They use a technology stack and tooling which help them operate and evolve applications quickly and reliably. These tools also help engineers independently accomplish tasks (for example, deploying code or provisioning infrastructure) that normally would have required help from other teams, and this further increases a team's velocity.

Why DevOps matters?

Software and the Internet have transformed the world and its industries, from shopping to entertainment to banking. Software no longer merely supports a business; rather it becomes an integral component of every part of a business. Companies interact with their customers through software delivered as online services or applications and on all sorts of devices. They also use software to increase operational efficiencies by transforming every part of the value chain, such as logistics, communications, and operations. In a similar way that physical goods companies transformed how they design, build, and deliver products using industrial automation throughout the 20th century, companies in today's world must transform how they build and deliver software.

Practise DevOps

Continuous Integration

Continuous integration is a software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.

Continuous Delivery

Continuous delivery is a software development practice where code changes are automatically built, tested, and prepared for a release to production. It expands upon continuous integration by deploying all code changes to a testing environment and/or a production environment after the build stage. When continuous delivery is implemented properly, developers will always have a deployment-ready build artifact that has passed through a standardized test process.

Microservices

The microservices architecture is a design approach to build a single application as a set of small services. Each service runs in its own process and communicates with other services through a welldefined interface using a lightweight mechanism, typically an HTTP-based application programming interface (API). Microservices are built around business capabilities; each service is scoped to a single purpose. You can use different frameworks or programming languages to write microservices and deploy them independently, as a single service, or as a group of services.

Infrastructure as Code

Infrastructure as code is a practice in which infrastructure is provisioned and managed using code and software development techniques, such as version control and continuous integration. The cloud's API-driven model enables developers and system administrators to interact with infrastructure programmatically, and at scale, instead of needing to manually set up and configure resources. Thus, engineers can interface with infrastructure using code-based tools and treat infrastructure in a manner similar to how they treat application code. Because they are defined by code, infrastructure and

servers can quickly be deployed using standardized patterns, updated with the latest patches and versions, or duplicated in repeatable ways.

Configuration Management

Developers and system administrators use code to automate operating system and host configuration, operational tasks, and more. The use of code makes configuration changes repeatable and standardized. It frees developers and systems administrators from manually configuring operating systems, system applications, or server software.

Policy as Code

With infrastructure and its configuration codified with the cloud, organizations can monitor and enforce compliance dynamically and at scale. Infrastructure that is described by code can thus be tracked, validated, and reconfigured in an automated way. This makes it easier for organizations to govern changes over resources and ensure that security measures are properly enforced in a distributed manner (e.g. information security or compliance with PCI-DSS or HIPAA). This allows teams within an organization to move at higher velocity since non-compliant resources can be automatically flagged for further investigation or even automatically brought back into compliance.

Monitoring and Logging

Organizations monitor metrics and logs to see how application and infrastructure performance impacts the experience of their product's end user. By capturing, categorizing, and then analyzing data and logs generated by applications and infrastructure, organizations understand how changes or updates impact users, shedding insights into the root causes of problems or unexpected changes. Active monitoring becomes increasingly important as services must be available 24/7 and as application and infrastructure update frequency increases. Creating alerts or performing real-time analysis of this data also helps organizations more proactively monitor their services.
















Communication and Collaboration

Increased communication and collaboration in an organization is one of the key cultural aspects of DevOps. The use of DevOps tooling and automation of the software delivery process establishes collaboration by physically bringing together the workflows and responsibilities of development and operations. Building on top of that, these teams set strong cultural norms around information sharing and facilitating communication through the use of chat applications, issue or project tracking systems, and wikis. This helps speed up communication across developers, operations, and even other teams like marketing or sales, allowing all parts of the organization to align more closely on goals and projects.

DevOps Tool

The DevOps model relies on effective tooling to help teams rapidly and reliably deploy and innovate for their customers. These tools automate manual tasks, help teams manage complex environments at scale, and keep engineers in control of the high velocity that is enabled by DevOps. AWS provides services that are designed for DevOps and that are built first for use with the AWS cloud. These services help you use the DevOps practices described above.

❖ Open-source tools in DevOps:

Configuration Management	Continuous integration	Microservices	Collaboration	Monitoring	Development
 CHEF	 Jenkins	 docker	 JIRA Software	 MONIT	 Visual Studio
 SALTSTACK	 TeamCity	 MESOS	 slack	 Ganglia	 APACHE ACTIVEMQ
 puppet	 CodeShip	 TRITON Compute	 HipChat	 SNORT	 Vagrant
 ANSIBLE	 circleci	 ElasticBox	 Trello Organize anything, together.	 cacti	 Microsoft Azure

• **Open-source tools in DevOps for Infrastructure Automation:**

Infrastructure automation is a critical component of modern DevOps practices. Open-source tools play a significant role in this space, offering flexibility and cost-effectiveness. Here are some open-source tools specifically designed for infrastructure automation:

1. **Terraform:**

Description: Terraform is an Infrastructure as Code (IaC) tool developed by HashiCorp. It allows you to define, provision, and manage infrastructure using a declarative configuration language.



Key Features:

- ❖ Multi-cloud support (AWS, Azure, GCP, etc.).
- ❖ State management for tracking infrastructure changes.
- ❖ Modular and reusable code with Terraform modules.
- ❖ Ecosystem of providers for various services and resources.

2. Ansible:

Description: Ansible is a popular configuration management and automation tool that uses YAML to define infrastructure as code. It can be used for provisioning, configuration management, and application deployment.



Key Features:

- ❖ Agentless, making it easy to set up and use.
- ❖ Supports a wide range of modules for various tasks.
- ❖ Playbooks for defining automation tasks.
- ❖ Strong community support and a vast collection of roles.

3. Packer:

Description: Packer is a tool for creating machine images from a single source configuration. It allows you to build machine images for different platforms in an automated and repeatable manner.



Key Features:

- ❖ Supports multiple builders (AWS, VirtualBox, Docker, etc.).

- ❖ Infrastructure as code for creating custom machine images.
- ❖ Integration with other provisioning tools like Ansible and Chef.

4. **Vagrant:**

Description: Vagrant is a tool for creating and managing virtualized development environments. It helps developers set up and share consistent development environments easily.



Key Features:

- ❖ Uses a simple configuration file (Vagrantfile).
- ❖ Supports various providers (VirtualBox, VMware, etc.).
- ❖ Enables team collaboration with portable development environments.

5. **SaltStack:**

Description: SaltStack is a configuration management and automation platform designed for speed and scalability. It can be used for remote execution, configuration management, and orchestration.



Key Features:

- ❖ Highly scalable and fast.
- ❖ Supports event-driven automation with the Reactor.
- ❖ Powerful remote execution capabilities.

6. **CloudInit:**

Description: CloudInit is a standard for customizing cloud instances during their initial boot process. While not a standalone tool, it's a critical component for automating cloud infrastructure.



Key Features:

- ❖ Configures instance settings during the first boot.
- ❖ Supported by major cloud providers, including AWS, Azure, and Google Cloud.

7. Rundeck:

Description: Rundeck is an open-source automation platform that provides job scheduling and runbook automation. It's useful for automating operational tasks and workflows



Key Features:

- ❖ Web-based interface for creating and managing automation workflows.
- ❖ Access control and auditing features.
- ❖ Integration with various tools and scripts.

8. OpenStack:

Description: OpenStack is an open-source cloud computing platform that can be used to build private and public clouds. It provides infrastructure automation capabilities, including compute, storage, and networking services.



Key Features:

- ❖ Modular architecture with services like Nova (compute) and Neutron (networking).
- ❖ Scalable and customizable cloud infrastructure.
- ❖ These open-source tools can be used individually or in combination to automate the provisioning, configuration, and management of infrastructure in a flexible and cost-effective manner. Depending on your specific needs and infrastructure, you can choose the tool or combination of tools that best suits your requirements.

❖ Open-source tools in DevOps for Configuration Management:

Configuration management is a critical aspect of DevOps, ensuring that infrastructure and application configurations remain consistent, are version-controlled, and can be easily replicated across different environments. Here are some open-source tools for configuration management:

1. Ansible:

Description: Ansible is an open-source configuration management and automation tool that uses simple, human-readable YAML scripts (playbooks) to define infrastructure and application configurations. It is agentless, which means it doesn't require agents to be installed on target systems.



Key Features:

- Declarative syntax for defining configurations.
- Supports various modules for tasks like package management, file manipulation, and more.
- Ansible Galaxy provides a library of pre-built roles.

- Strong community support and integrations with cloud platforms.

2. Chef:

Description: Chef is a powerful open-source configuration management tool that uses Ruby-based scripts (cookbooks) to automate infrastructure and application provisioning. It follows an "infrastructure as code" approach.



Key Features:

- Infrastructure automation with code-based cookbooks.
- Supports test-driven development (TDD) for infrastructure.
- Chef Supermarket offers a repository of pre-built cookbooks.
- Integrates with cloud providers and platforms.

3. Puppet:

Description: Puppet is an open-source configuration management tool that uses its own declarative language to define configurations. It is designed for managing and automating infrastructure at scale.



Key Features:

- Puppet's DSL for configuration definitions.
- Puppet Forge provides a repository of pre-built modules.
- Supports role-based management of configurations.
- Integrates with cloud services and container platforms.

4. SaltStack:

Description: SaltStack, or Salt, is an open-source, event-driven automation and configuration management tool. It is known for its speed and scalability, making it suitable for managing large infrastructures.



Key Features:

- Highly scalable and fast remote execution.
- Event-driven automation with the Salt Reactor.
- Supports infrastructure automation through states and pillars.
- Strong security features, including ZeroMQ encryption.

5. Terraform:

Description: Terraform is an open-source Infrastructure as Code (IaC) tool developed by HashiCorp. While it's primarily used for provisioning infrastructure, it also includes configuration management capabilities for defining infrastructure resources.



Key Features:

- Multi-cloud support (AWS, Azure, GCP, etc.).
- Infrastructure provisioning and management through Terraform code.
- Modular and reusable code with Terraform modules.

- Integration with configuration management tools like Ansible and Puppet.

6. Rundeck:

Description: Rundeck is an open-source automation platform with a focus on job scheduling and runbook automation. While not a traditional configuration management tool, it helps manage operational tasks and automate workflows.



Key Features:

- Web-based interface for creating and managing automation workflows.
- Access control and auditing features.
- Integration with various tools and scripts.
- Supports scheduling and orchestrating complex tasks.

❖ Open-source tools in DevOps for Deployment Automation:

1. Jenkins for Continuous Integration and Continuous Deployment (CI/CD):

Description: Jenkins is an open-source automation server that can be used for various tasks in the CI/CD pipeline. It is highly extensible and widely used in the DevOps community.



Key Features:

- Automation: Jenkins enables automation of the entire software development lifecycle, from code integration and testing to deployment and monitoring.

Pipeline as Code: It allows you to define CI/CD pipelines as code, which can be versioned, shared, and easily replicated.

- Plugins: Jenkins has a vast ecosystem of plugins, allowing you to integrate it with various tools and services.
- Scalability: Jenkins can be scaled horizontally to handle increasing workloads.

2. Docker for Containerization:

Description: Docker is an open-source platform that automates the deployment of applications inside lightweight, portable containers. Containers package applications and their dependencies in a consistent, isolated environment.



Key Features:

- Isolation: Containers ensure that applications run consistently regardless of the environment.
- Portability: Containers can be moved between environments with minimal changes.
- Version Control: Container images can be versioned and stored in container registries.
- Efficiency: Containers are lightweight and start quickly.

3. Kubernetes for Container Orchestration:

Description: Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications.



kubernetes

Key Features:

- Orchestration: Kubernetes schedules and manages containerized applications across a cluster of machines.
- Scalability: It can scale applications up or down automatically based on demand.

- Self-Healing: Kubernetes monitors the health of applications and restarts containers if needed.
- **Load Balancing: It provides built-in load balancing for containerized services.**

4. Helm for Kubernetes Package Management:

Description: Helm is an open-source package manager for Kubernetes. It simplifies defining, installing, and upgrading even complex Kubernetes applications through Helm charts.



Key Features:

- Charts: Helm charts are packages of pre-configured Kubernetes resources.
- Versioning: Helm charts can be versioned and shared via chart repositories.
- Templating: Helm allows parameterized templates for different environments or configurations.

5. Prometheus and Grafana for Monitoring:

Description: Prometheus is an open-source monitoring and alerting toolkit. Grafana is an open-source platform for observability and visualization.



Key Features (Prometheus):

- Data Collection: Prometheus scrapes metrics from various targets.
- Flexible Query Language: It provides PromQL for querying and analyzing metrics.
- Alerting: Prometheus can trigger alerts based on defined rules.

Key Features (Grafana):

Visualization: Grafana offers flexible and interactive data visualization.

- Dashboards: It supports the creation of custom dashboards for monitoring.
- Alerting: Grafana can integrate with Prometheus for alerting.

❖ Open-source tools in DevOps for Performance Management:

1. JMeter for Load Testing:

Description: Apache JMeter is an open-source tool for load testing, performance testing, and stress testing of web applications. It simulates a heavy load on a server, network, or object to test its strength or analyze overall performance under different load types.



Key Features:

- HTTP/HTTPS Support: JMeter can simulate various web protocols, including HTTP and HTTPS.
- Scripting: It provides a graphical interface for building test plans and scripting tests.
- Scalability Testing: JMeter can simulate a large number of concurrent users, helping identify performance bottlenecks.

2. Prometheus for Monitoring:

Description: Prometheus is an open-source monitoring and alerting toolkit that is designed for reliability and scalability. It collects and stores metrics from various sources, allowing real-time monitoring and alerting.



Key Features:

- Data Collection: Prometheus scrapes metrics from targets such as applications, servers, and services.
- Flexible Query Language: PromQL enables querying and analyzing metrics.
- Alerting: Prometheus can trigger alerts based on defined rules.

3. Grafana for Visualization:

Description: Grafana is an open-source platform for observability and data visualization. It can be used to create dashboards and graphs for monitoring and analyzing data.



Key Features:

- Visualization: Grafana offers flexible and interactive data visualization.
- Dashboards: It supports the creation of custom dashboards for monitoring and analysis.
- Alerting: Grafana can integrate with various monitoring systems, including Prometheus.

4. Apache JMeter for Real User Monitoring:

Description: In addition to load testing, Apache JMeter can be used for real user monitoring (RUM) to capture performance data from actual users and browsers.



Key Features:

- Browser Simulation: JMeter can simulate browser requests and interactions to collect real user data.
- Script Recording: It allows recording and playback of user interactions for performance testing and monitoring.

❖ Open-source tools in DevOps for log management:

1. Elasticsearch for Log Storage:

Description: Elasticsearch is an open-source, distributed search and analytics engine designed for handling large volumes of data. It's often used for log storage and indexing.



Key Features:

- **Scalability:** Elasticsearch can handle large volumes of data and scale horizontally.
- **Full-Text Search:** It offers powerful full-text search capabilities for log data.
- **Real-Time Data:** Elasticsearch provides real-time indexing and search for log data.

2. Logstash for Log Ingestion:

Description: Logstash is an open-source server-side data processing pipeline that ingests, transforms, and enriches log data before sending it to a data store.



Key Features:

- **Data Transformation:** Logstash allows data transformation using various filters and plugins.
- **Data Enrichment:** You can enrich log data with additional information using lookup tables and external sources.

Support for Various Inputs and Outputs: Logstash supports a wide range of input and output sources, making it versatile for log ingestion.

3. Kibana for Log Visualization:

Description: Kibana is an open-source data visualization and exploration tool that is often used with Elasticsearch for log data analysis and visualization.



Key Features:

- Data Visualization: Kibana offers interactive and customizable data visualization.
- Dashboards: You can create custom dashboards to display key log data and metrics.
- Elasticsearch Integration: Kibana seamlessly integrates with Elasticsearch for log data retrieval.

4. Filebeat for Log Shipper:

Description: Filebeat is an open-source lightweight log shipper that collects and forwards log data to a centralized location, often to Elasticsearch or Logstash.



Key Features:

- Lightweight: Filebeat has a small footprint and is resource-efficient.
- Log Data Collection: It can collect log data from various sources, including log files, system logs, and Docker containers.

- Real-Time Data Shipper: Filebeat ships log data in real-time to the central log management system.

❖ Open-source tools in DevOps for Monitoring:

1. Prometheus for Metrics Collection:

Description: Prometheus is an open-source monitoring and alerting toolkit. It is designed for reliability and scalability, collecting metrics from different sources and providing real-time monitoring and alerting.



Key Features:

- Data Collection: Prometheus scrapes metrics from various targets, including applications, servers, and services.
- Flexible Query Language: PromQL allows for querying and analyzing metrics.
- Alerting: Prometheus can trigger alerts based on predefined rules.

2. Grafana for Data Visualization:

Description: Grafana is an open-source platform for observability and data visualization. It is commonly used to create dashboards and visualizations for monitoring and data analysis.



Key Features:

- Visualization: Grafana provides flexible and interactive data visualization.
- Dashboards: It supports the creation of custom dashboards for monitoring and analysis.

- Alerting: Grafana can integrate with various monitoring systems, including Prometheus.

3. ELK Stack for Log Management:

Description: The ELK Stack consists of three open-source tools: Elasticsearch, Logstash, and Kibana. It is used for log management, log analysis, and log visualization.



Key Features:

- Elasticsearch: Indexes and stores log data for efficient searching and analysis.
- Logstash: Collects, processes, and enriches log data before sending it to Elasticsearch.
- Kibana: Provides visualization and dashboards for log analysis.

4. Zabbix for Infrastructure Monitoring:

Description: Zabbix is an open-source monitoring solution that can be used for network monitoring, server monitoring, and application monitoring.



Key Features:

- Auto Discovery: Zabbix can automatically discover and monitor devices and services.
- Custom Monitoring Items: It allows you to create custom monitoring items for specific metrics.
- Alerting: Zabbix supports alerting and notification when predefined thresholds are breached.

Suggested Reference:

- 1 Deepak Gaikwad, Viral Thakkar, DevOps Tools from Practitioner's ViewPoint, Wiley.
- 2 The DevOps Handbook - Gene Kim et. al.

Rubric wise marks obtained:

Rubrics	1	2	3	4	5	Total
Marks	Complete implementation as asked	Complete implementation as asked Problem analysis	Complete implementation as asked Problem analysis Development of the Solution	Complete implementation as asked Problem analysis Development of the Solution Concept Clarity & understanding	Complete implementation as asked Problem analysis Development of the Solution Concept Clarity & understanding Correct answer to all questions	

Signature of Faculty: