Department of Information Technology
BABU BANARASI DAS
Institute of Technology & Management, Lucknow
Recognized by AICTE, Govt.of India & Affiliated to Dr. A.P.J Abdul Kalam Technical University,Lucknow,AKTU College Code=054

LAB MANUAL

OF

SOFTWARE ENGINEERING LAB

[BCS-651]

B.TECH, 3rd Year, Semester VI

Dr. A.P.J. Abdul Kalam Tech. University
Uttar Pradesh

2025-26

Department of Information Technology

MANUAL CONTENTS

This manual is intended for the 3rd year students of Information Technology in the subject of Software Engineering Lab. This manual typically contains practical/lab sessions related Software Engineering covering various aspects related to the subject of enhanced understanding.

Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.

Good Luck! for your enjoyable laboratory sessions.

PREFACE

This practical manual will be helpful for students of Information Technology for understanding the course from the point of view of applied aspects. Though all the efforts have been made to make this manual error free, yet some errors might have crept in inadvertently. Suggestions from the readers for the improvement of the manual are most welcomed.

INDEX

S.No	Content	Page No.
1.	Vision & Mission of Institute	
2.	Vision & Mission of Department	
3.	Program Educational Objectives	
4.	Program Outcomes	
5.	Program Specific Objectives	
6.	Course Evaluation Scheme	
7.	List of Programs	
8.	Course Outcome of Practical	
9.	CO-PO-PSO Mapping	
10.	Experiment Details	
11.	Internal Marks Assessment Rubrics	

VISION OF THE INSTITUTE

"To become a leading institute of providing professionally competent and socially responsive technocrats with high moral values."

MISSION OF THE INSTITUTE

- To create an eco-system for the dissemination of technical knowledge, to achieve academic excellence.
- To develop technocrats with creative skills and leadership qualities, to solve local and global challenges.
- To impart human values and ethics in students, to make them socially and Eco-friendly responsible.

VISION OF THE DEPARTMENT

"To transform the students to be skilled IT professionals, Innovative leaders and environmentally receptive citizens"

MISSION OF THE DEPARTMENT

M1: To develop and deliver quality academic programs in emerging & innovative fields of engineering to empower the students to meet industry standards.

M2: To build students community with ethical standards to undertake R&D in thrust areas of national & international needs.

M3: To inculcate professional behaviour, positive attitude and communication skills**.**

.

Program Educational Objectives (PEOs) of Department

PEO1: To develop the ability among students to understand the concept of core Information Technology subjects that will facilitate understanding of any new technology.

PEO2: To embed a strong foundation in the engineering fundamentals to solve analyze and design real time engineering products.

PEO3: To give exposures to emerging edge technologies adequate training and opportunities to work as a team on multi-disciplinary projects with Effective communication skills and leadership qualities.

Program Outcomes: (PO)

PO 1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering Fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO 2.**Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO 3.**Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO 4.Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO 5.Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

PO 6.The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

PO 7.Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO 8.Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO 9.Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO 10.Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear

PO11.Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as

a member and leader in a team, to manage projects and in multidisciplinary environments.

PO 12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs) of the Department

PSO1: Use and apply current technical concepts and practices in the core Information Technologies of Human Machine Learning, Information Management, Programming, Cloud Computing.

PSO2: Design, develop and test software systems for worldwide network of computers to provide solutions to real world problems.

Course Evaluation Scheme

S. No.	Subject Code	Name of the subject	Periods			Evaluation Scheme				Subject Total	Credit
						Sessional Assessment			ESE		
			L	T	P	CT	TA	Total			
1	BCS-651	SE Lab	0	0	2		50	50	50	100	1

List of Programs

S.No	Name of the Program	COs
1.	Understanding an SRS. Prepare a SRS document in line with IEEE recommended standards.	CO1
2.	Draw the use case diagram and specify the role of each of the actors using Star UML. Also state the precondition, post condition and function of each use case.	CO1
3.	To draw a sample Data Flow Diagram.	CO2
4.	To draw a sample Entity Relationship Diagram.	CO2
5.	Prepare State Chart Diagram.	CO3
6.	To draw a sample activity diagram.	CO3
7.	To draw a sample Collaboration diagram.	CO4
8.	To draw a sample Sequence diagram.	CO4
9.	To draw a sample Class diagram.	CO5
10.	Estimation of project matrices.	CO5

Course Outcomes

CO's	Course Outcome	PO	Bloom
CO1	Identify ambiguities, inconsistencies and incompleteness from a requirements specification and state functional and non-functional requirement.	PO1, PO2, PO3, PO5, PSO1, PSO2	L2
CO2	Identify different actors and use cases from a given problem statement and draw use case diagram to associate use cases with different types of relationship.	PO1, PO2, PO3, PO5, PSO1, PSO2	L3
CO3	Draw a class diagram after identifying classes and association among them.	PO1, PO2, PO3, PO5, PSO1, PSO2	L4
CO4	Graphically represent various UML diagrams, and associations among them and identify the logical sequence of activities undergoing in a system, and represent them pictorially.	PO1, PO2, PO3, PO5, PSO1, PSO2	L5
CO5	Able to use modern engineering tools for specification, design, implementation and testing.	PO1, PO2, PO3, PO5, PSO1, PSO2	L4

CO-PO-PSO Mapping

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
CO1	2	3	-	-	-	-	-	-	-	-	-	-	-	1
CO2	-	-	-	-	3	-	-	-	-	-	3	-	3	1
CO3	-	-	2	-	2	-	-	-	-	-	-	-	2	-
CO4	-	3	-	-	-	-	-	-	-	-	-	-	-	-
CO5	1	2	2	-	-	-	-	-	-	-	-	-	-	-

Experiment-1

AIM: Understanding an SRS. Prepare a SRS document in line with IEEE recommended standards.

REQUIREMENTS:

Hardware Requirements:

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

Software Requirements: Star-UML, Windows

THEORY:

An SRS is basically an organization's understanding (in writing) of a customer or potential client's system requirements and dependencies at a particular point in time (usually) prior to any actual design or development work. It's a two-way

insurance policy that assures that both the client and the organization understand the other's requirements from that perspective at a given point in time.

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

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

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

It provides feedback to the customer. An SRS is the customer's assurance that the development organization understands the issues or problems to be solved and the software behavior necessary to address those problems. Therefore, the SRS should be written in natural language (versus a formal language, explained later in this article), in an unambiguous manner that may also include charts, tables, data flow diagrams, decision tables, and so on.

It decomposes the problem into component parts. The simple act of writing down software requirements in a well-designed format organizes information, places borders around the problem, solidifies ideas, and helps break down the problem into its component parts in an orderly fashion.

It serves as an input to the design specification. As mentioned previously,

the SRS serves as the parent document to subsequent documents, such as the software design specification and statement of work. Therefore, the SRS must contain sufficient detail in the functional system requirements so that a design solution can be devised.

It serves as a product validation check. The SRS also serves as the parent document for testing and validation strategies that will be applied to the requirements for verification.

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

SRS should address the following

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

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

Characteristics of a good SRS

An SRS should be:

a) Correct

b) Unambiguous

c) Complete

d) Consistent

e) Ranked for importance and/or stability

f) Verifiable

g) Modifiable

h) Traceable

A sample of basic SRS Outline:

1. Introduction

1.1 Purpose

1.2 Document conventions

1.3 Intended audience

1.4 Additional information

1.5 Contact information/SRS team members

1.6 References

2. Overall Description

2.1 Product perspective

2.2 Product functions

2.3 User classes and characteristics

2.4 Operating environment

2.5 User environment

2.6 Design/implementation constraints

2.7 Assumptions and dependencies

3. External Interface Requirements

3.1 User interfaces

3.2 Hardware interfaces

3.3 Software interfaces

3.4 Communication protocols and interfaces

4. System Features

4.1 System feature A

4.2 Description and priority

4.3 Action/result

4.4 Functional requirements

4.5 System feature B

5. Other Non-functional Requirements

5.1 Performance requirements

5.2 Safety requirements

5.3 Security requirements

5.4 Software quality attributes

5.5 Project documentation

5.6 User documentation

6. Other Requirements

Appendix A:

Terminology/Glossary/Definitions

list Appendix B: To be determined

Experiment-2

AIM: Draw the use case diagram and specify the role of each of the actors using Star UML. Also state the precondition, post condition and function of each use case.

REQUIREMENTS:

Hardware Interfaces

- Pentium(R)4 CPU 2.26GHz, 128 MBRAM
- Screenresolutionofatleast800x600requiredforproper and complete viewing of screens. Higher resolution would not be a problem.
- CD ROM Driver

Software Interfaces

- Any window-based operating system(Windows 95/98/2000/XP/NT)
- WordPad or Microsoft Word
- Star UML

THEORY:

According to the UML specification a use case diagram is —a diagram that shows the relationships among actors and use cases within a system. Use case diagrams are often used to: Provide an overview of all or part of the usage requirements for a system or organization in the form of an essential model or a business model Communicate the scope of a development project Model your analysis of your usage requirements in the form of a system use case model.

Use case models should be developed from the point of view of your projects take holders and not from the (often technical) point of view of developers. There are guidelines for:

Use Cases

Actor

Relationships

System Boundary Boxes

1. Use Cases

A use case describes a sequence of actions that provide a measurable value to an actor. A use case is drawn as a horizontal ellipse on a UML use case diagram.

- Use Case Names Begin with a Strong Verb.
- Name Use Cases Using Domain Terminology.
- Place Your Primary Use Cases in the Top-Left Corner of the Diagram
- Imply Timing Considerations by Stacking Use Cases.

2. Actors

An actor is a person, organization, or external system that plays a role in one or more interactions with your system (actors are typically drawn as stick figures on UML Use Case diagrams).

- Place Your Primary Actor(S) In The Top-Left Corner Of The Diagram
- Draw Actors To The Outside Of A Use Case Diagram
- Name Actors With Singular, Business-Relevant Nouns
- Associate Each Actor With One Or More Use Cases
- Actors Model Roles, Not Positions
- Use<<system>>to Indicate System Actors
- Actors Don‘t Interact With One Another
- Introduce an Actor Called –Time to Initiate Scheduled Events

3. Relationships

There are several types of relationships that may appear on a use case diagram:

- An association between an actor and a use case
- An association between two use cases
- A generalization between two actors
- A generalization between two use cases

Associations are depicted as lines connecting two modeling elements with an optional open-headed arrow head on one end of the line indicating the direction of the initial invocation of the relationship. Generalizations are depicted as a close-headed arrow with the arrow pointing towards the more general modeling element.

- Indicate An Association Between An Actor And A Use Case If The Actor Appears Within The Use Case Logic
- Avoid Arrow heads On Actor-Use Case Relationships
- Apply<<include>>When You Know Exactly When To Invoke The Use Case
- Apply<<extend>>When A Use Case May Be Invoked Across Several Use Case Steps
- Introduce<<extend>>associations sparingly
- GeneralizeUseCasesWhenaSingleConditionResultsInSignificantlyNewBusinessLogic
- Do Not Apply<<uses>>,<<includes>>, or <<extends>>
- Avoid More Than Two Levels Of Use Case Associations
- Place An Included Use Case To The Right Of The Invoking Use Case
- Place An Extending Use Case Below The Parent Use Case
- Apply the–Is Like‖ Rule to Use Case Generalization
- Place an Inheriting Use Case Below The Base Use Case
- Apply the–Is Like‖ Rule to Actor Inheritance
- Place an Inheriting Actor Below the Parent Actor

4. *System Boundary Boxes*

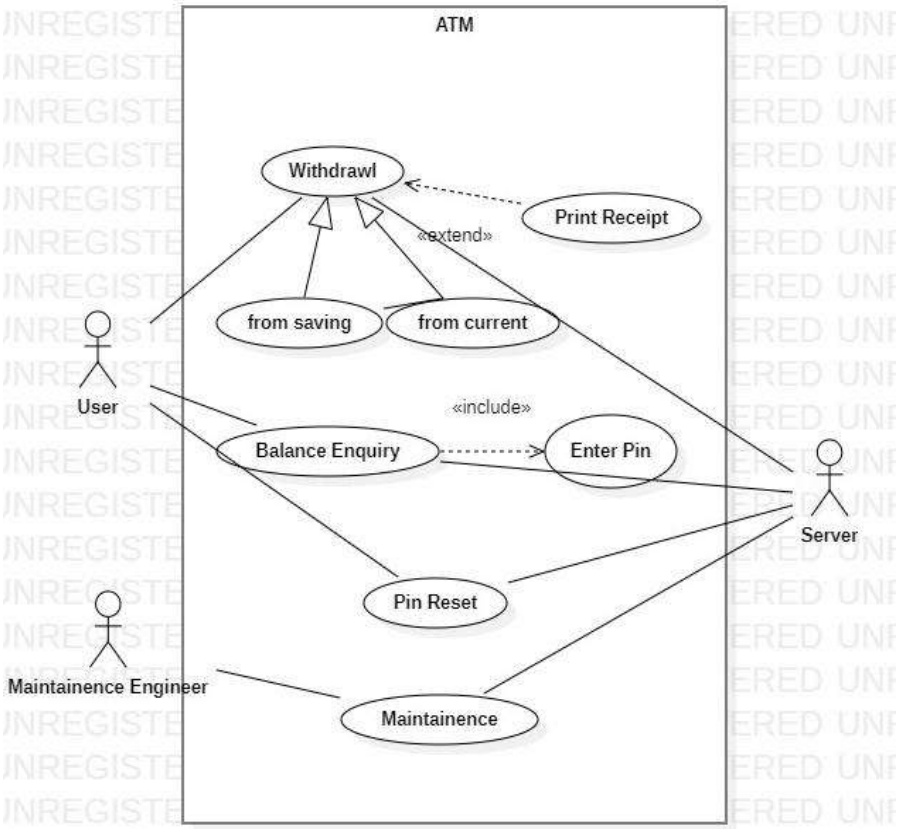
The rectangle around the use cases is called the system boundary box and as the name suggests it indicates the scope of your system – the use cases inside the rectangle represent the functionality that you intend to implement.

- Indicate Release Scope with a System Boundary Box.
- Avoid Meaning less system Boundary Boxes.

Creating Use Case Diagrams

We start by identifying as many actors as possible. You should ask how the actors interact with the system to identify an initial set of use cases. Then, on the diagram, you connect the actors with the use cases with which they are involved. If actor supplies information, initiates the use case, or receives any information as a result of the use case, then there should be an association between them.

Conclusion: The use case diagram was made successfully by following the steps Described above



Some Sample Use Case Diagrams are given below for illustration purpose:

Experiment-3

AIM: Draw Data Flow Diagram using Star UML.

REQUIREMENTS:

Hardware Interfaces

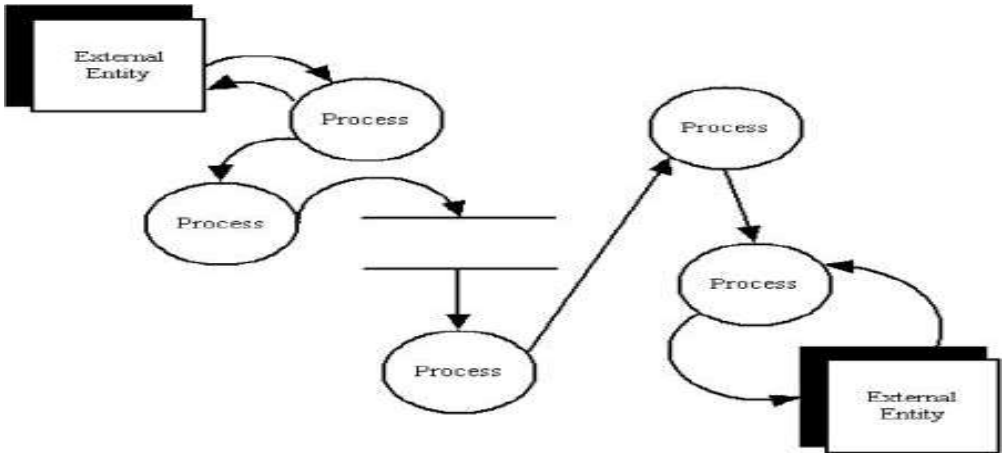
- Pentium(R)4 CPU 2.26GHz, 128 MBRAM
- Screenresolutionofatleast800x600requiredforproper and complete viewing of screens. Higher resolution would not be a problem.
- CD ROM Driver

Software Interfaces

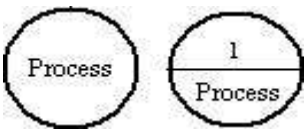
- Any window-based operating system(Windows 95/98/2000/XP/NT)
- WordPad or Microsoft Word
- Star UML

THEORY

Dataflow diagrams illustrate how data is processed by a system in terms of inputs and outputs



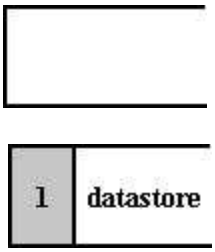
You can use two different types of notations on your data flow diagrams: Yourdon & Coad or Gane & Sarson.



Process Notations

Datastore Notations

Yourdon and Coad
Datastore Notations



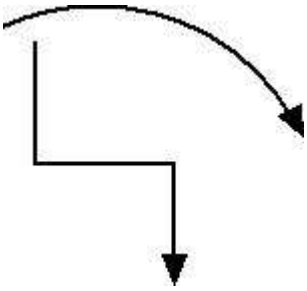
Gane and Sarson

Datastore Notations

Data Store

Data stores are repositories of data in the system. They are sometimes also referred to as files.

Dataflow Notations



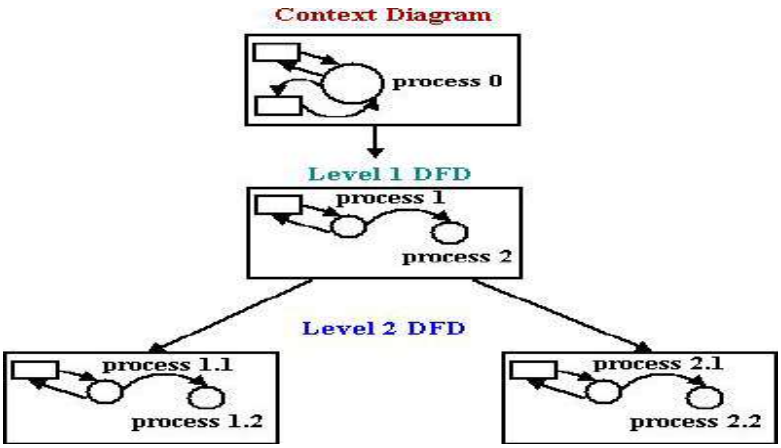
Dataflow

Dataflows are pipelines through which packets of information flow. Label the arrows with the name of the data that moves through it.

HOW TO DRAW DATA FLOW DIAGRAMS (cont'd)

Data Flow Diagram Layers

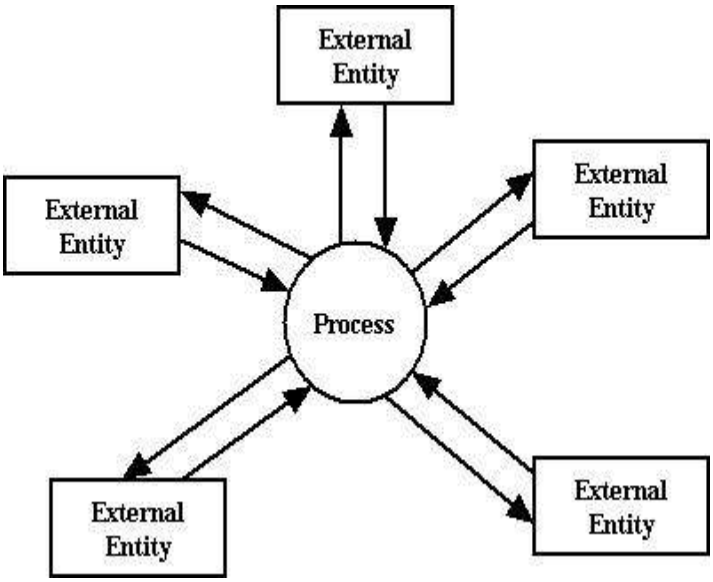
Draw data flow diagrams in several nested layers. A single process node on a high level diagram can be expanded to show a more detailed data flow diagram. Draw the context diagram first, followed by various layers of data flow diagrams.



The nesting of data flow layers

Context Diagrams

A context diagram is a top level (also known as Level0) data flow diagram. It only contains one process node (process 0) that generalizes the function of the entire system in relationship to external entities.



Experiment-4

AIM:- To draw a sample Entity Relationship Diagram.

REQUIREMENTS:

Hardware Interfaces

- Pentium(R)4 CPU 2.26 GHz, 128 MB RAM

- Screen resolution of at least 800x600 required for proper and complete viewing of screens. Higher resolution would not be a problem.
- CD ROM Driver

Software Interfaces

- Any window-based operating system(Windows 95/98/2000/XP/NT)
- WordPad or Microsoft Word
- Star UML

THEORY

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

Entity

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

Relationship

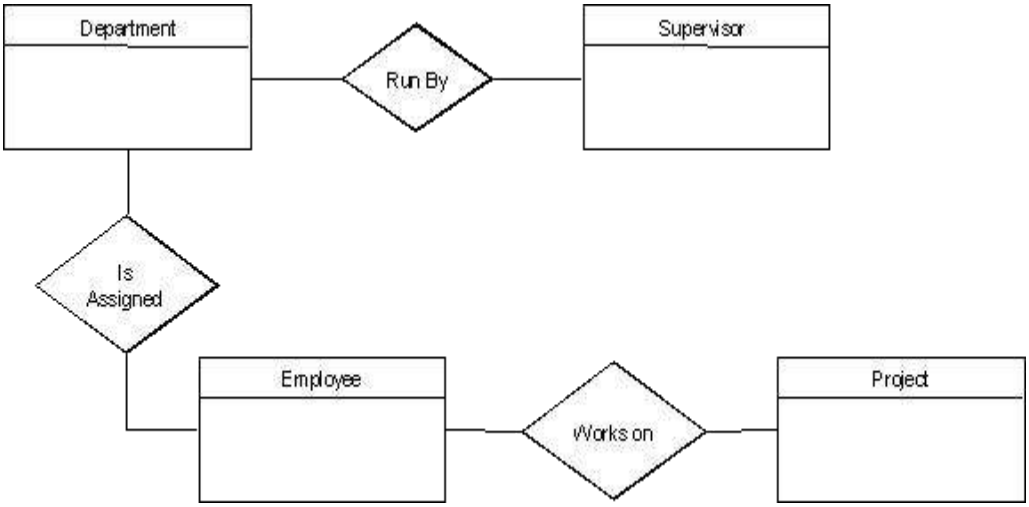
A data relationship is a natural association that exists between one or more entities. E.g. Employees process payments. **Cardinality** defines the number of occurrences of one entity for a single occurrence of the related entity. E.g. an employee may process many payments but might not process any payments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

Attribute

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

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

1.IdentifyEntities	Identify the roles, events, locations, tangible things or concepts about Which the end-users want to store data.
2.FindRelationships	Find the natural associations between pairs of entities using a relationship matrix.
3.DrawRoughERD	Put entities in rectangles and relationships online segments connecting the entities.
4. Fill in Cardinality	Determine the number of occurrences of one entity for a single occurrence of the related entity.
5.DefinePrimaryKeys	Identify the data attribute(s) that uniquely identify one and only one occurrence of each entity.
6.DrawKey-BasedERD	Eliminate Many-to-Many relationships and include primary and foreign keys in each entity.
7.IdentifyAttributes	Name the information details (fields) which are essential to the system under development.
8.MapAttributes	For each attribute, match it with exactly one entity that it describes.
9.Draw fully attributed ERD	Adjust the ERD from step 6 to account for entities or relationships discovered in step 8.
10.CheckResults	Does the final Entity Relationship Diagram accurately depict the system data?



Experiment-5

AIM: To prepare State Chart Diagram.

REQUIREMENTS:

Hardware Interfaces

- Pentium(R)4 CPU 2.26GHz, 128 MBRAM
- Screenresolutionofatleast800x600requiredforproper and complete viewing of screens. Higher resolution would not be a problem.
- CD ROM Driver

Software Interfaces

- Any window-based operating system(Windows 95/98/2000/XP/NT)
- WordPad or Microsoft Word
- Star UML

THEORY:

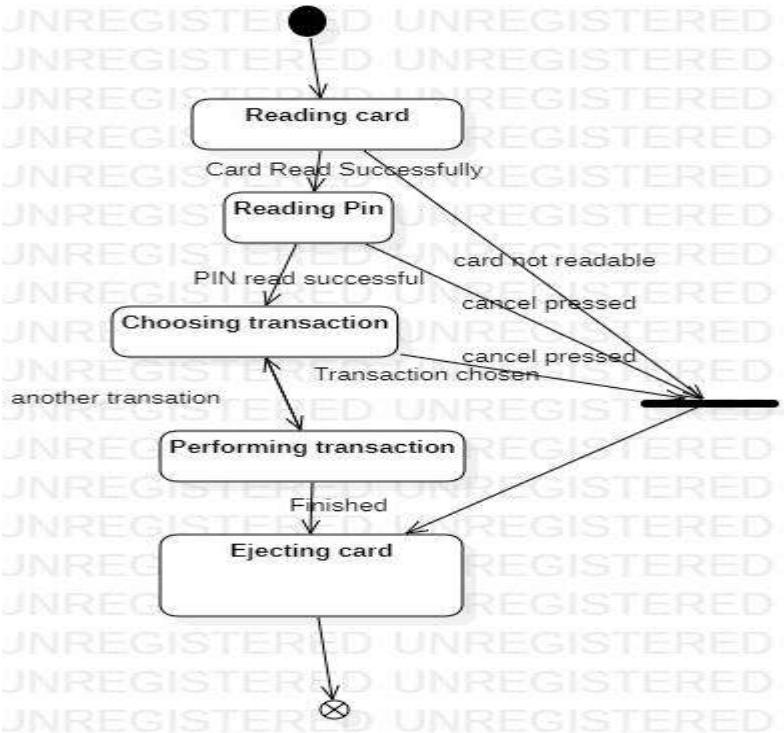
- ✓ State Chart Diagrams provide away to model the various states in which an object can exist.
- ✓ There are two special states: the start state and the stop state. The **Start state** is represented by a **block dot**. The **Stop state** is represented by a **bull's eye**.
- ✓ A condition enclosed in square brackets is called a **guard condition,** and controls when a transition can or cannot occur.
- ✓ **Process** that occur while an object is in certain state are called **actions.**

STEPSTODRAWSTATECHARTDIAGRAMINSTAR UML SOFTWARE

To insert new state diagram secondary click on Logical View.

The state chart diagram was made successfully by following the described steps above.

Conclusion:



Example of State chart diagram for ATM machine

Experiment-6

AIM: To draw a sample activity diagram.

REQUIREMENTS:

Hardware Interfaces

- Pentium(R)4 CPU 2.26GHz, 128 MBRAM
- Screenresolutionofatleast800x600requiredforproper and complete viewing of screens. Higher resolution would not be a problem.
- CD ROM Driver

Software Interfaces

- Any window-based operating system(Windows 95/98/2000/XP/NT)
- WordPad or Microsoft Word
- Star UML

THEORY

UML 2 activity diagrams are typically used for business process modeling, for modeling the logic captured by a single use case or usage scenario, or for modeling the detailed logic of a business rule. Although UML activity diagrams could potentially model the internal logic of a complex operation it would be far better to simply rewrite the operation so that it is simple enough that you don‘t require an activity diagram. In many ways UML activity diagrams are the object-oriented equivalent of flow charts and data flow diagrams (DFDs) from structured development.

Let‘s start by describing the basic notation:

Initial node. The filled in circle is the starting point of the diagram. An initial node isn‘t required although it does make it significantly easier to read the diagram.

Activity final node. The filled circle with a border is the ending point. An activity diagram can have zero or more activity final nodes.

Activity. The rounded rectangles represent activities that occur. An activity may be physical, such as Inspect Forms, or electronic, such as Display Create Student Screen.

Flow/edge. The arrows on the diagram. Although there is a subtle difference between flows and edges, never a practical purpose for the difference although.

Fork. A black bar with one flow going into it and several leaving it. This denotes the beginning of parallel activity.

Join. A black bar with several flows entering it and one leaving it. All flows going into the join must reach it before processing may continue. This denotes the end of parallel processing.

Condition. Text such as [Incorrect Form] on a flow, defining a guard which must evaluate to true in order to traverse the node.

Decision. A diamond with one flow entering and several leaving. The flows leaving include conditions although some modelers will not indicate the conditions if it is obvious.

Merge. A diamond with several flows entering and one leaving. The implication is that one or more incoming flows must reach this point until processing continues, based on any guards on the outgoing flow.

Partition. If figure is organized into three partitions, it is also called swim lanes, indicating who/what is performing the activities (either the Applicant, Registrar, or System).

Sub-activity indicator. The rake in the bottom corner of an activity, such as in the Apply to University activity, indicates that the activity is described by a more finely detailed activity diagram.

Flow final. The circle with the X through it. This indicates that the process stops at this point.

GUIDELINES ASSOCIATED FOR DRAWING AN ACTIVITY DIAGRAM

1. General Guidelines
2. Activities
3. Decision Points
4. Guards
5. Parallel Activities
6. Swim lane Guidelines
7. Action-Object Guideline

Experiment-7

AIM: To draw the Collaboration diagram.

REQUIREMENTS-

Hardware Interfaces

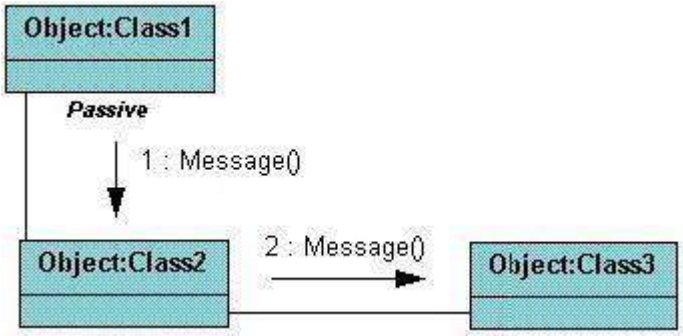
- Pentium(R)4 CPU 2.26GHz, 128 MBRAM
- Screenresolutionofatleast800x600requiredforproper and complete viewing of screens. Higher resolution would not be a problem.
- CD ROM Driver

Software Interfaces

- Any window-based operating system(Windows 95/98/2000/XP/NT)
- WordPad or Microsoft Word
- Star UML

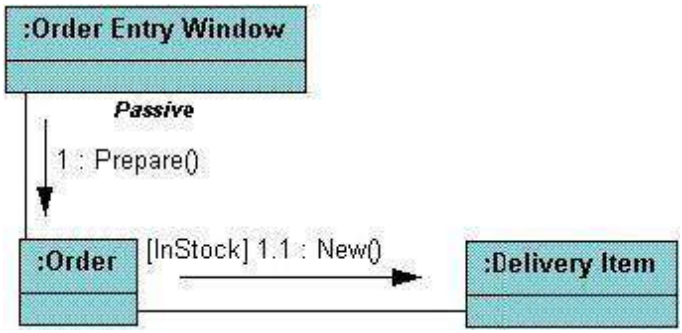
THEORY:

Collaboration diagrams are also relatively easy to draw. They show the relationship between objects and the order of messages passed between them. The objects are listed as icons and arrows indicate the messages being passed between them. The numbers next to the messages are called sequence numbers. As the name suggests, they show the sequence of the messages as they are passed between the objects. There are many acceptable sequence numbering schemes in UML. A simple 1, 2, 3... Format can be used, as the example below shows, or for more detailed and complex diagrams a 1, 1.1, 1.2, 1.2.1... Scheme can be used.



The example below shows a simple collaboration diagram for the placing an order use case. This time the names ofthe objects appear after the colon, such as: Order Entry Window

following the object Name: class Name naming convention. This time the class name is shown to demonstrate that all of objects of that class will behave the same way.



Experiment-8

AIM: Draw the Sequence Diagram.

REQUIREMENTS:

Hardware Interfaces

- Pentium(R)4 CPU 2.26GHz, 128 MBRAM
- Screenresolutionofatleast800x600requiredforproper and complete viewing of screens. Higher resolution would not be a problem.
- CD ROM Driver

Software Interfaces

- Any window-based operating system(Windows 95/98/2000/XP/NT)
- WordPad or Microsoft Word
- Star UML

Theory:

UML sequence diagrams model the flow of logic within the system in a visual manner, enabling the user both to document and validate the logic, and are commonly used for both analysis and design purposes. Sequence diagrams are the most popular UML artifact for dynamic modeling, which focuses on identifying the behavior within your system. Sequence diagrams, along with class diagrams and physicaldata models are the most important design-level models for modern application development.

Sequence diagrams are typically used to model:

1. **Usage scenarios**. A usage scenario is a description of a potential way the system is

used. The logic of a usage scenario may be part of a use case, perhaps an alternate course. It may also be one entire pass through a use case, such as the logic described by the basic course of action or a portion of the basic course of action, plus one or more alternate scenarios. The logic of a usage scenario may also be a pass through the logic contained in several use cases. For example, a student enrolls in the university, and then immediately enrolls in three seminars.

2. **The logic of methods**. Sequence diagrams can be used to explore the logic of a complex operation, function, or procedure. One way to think of sequence diagrams, particularly highly detailed diagrams, is as visual object code.
3. **The logic of services**. A service is effectively a high-level method, often one that can be invoked by a wide variety of clients. This includes web-services as well as business transactions implemented by a variety of technologies such as CICS/COBOL or CORBA-compliant object request brokers (ORBs).

Experiment-9

AIM: To draw class diagram for any project.

REQUIREMENTS:

Hardware Interfaces

- Pentium(R)4 CPU 2.26GHz, 128 MBRAM
- Screenresolutionofatleast800x600requiredforproper and complete viewing of screens. Higher resolution would not be a problem.
- CD ROM Driver

Software Interfaces

- Any window-based operating system(Windows 95/98/2000/XP/NT)
- WordPad or Microsoft Word
- Star UML

THEORY:

A **class diagram** is a type of **static structure diagram** that describes the structure of a systemby showing the system's classes, their attributes, and the relationships between the classes.

Class diagrams show the classes of the system, their inter-relationships, and the operations andattributes of the classes. Class diagrams are typically used, although not all at once, to:

- Explore domain concepts in the form of a domain model
- Analyze requirements in the form of a conceptual/analysis model
- Depict the detailed design of object-oriented or object-based software

A class model is comprised of one or more class diagrams and the supporting specifications thatdescribe model elements including classes, relationships between classes, and interfaces. There are guidelines

1. General issues
2. Classes
3. Interfaces
4. Relationships
5. Inheritance
6. Aggregation and Composition

GENERAL GUIDELINES

Because class diagrams are used for a variety of purposes – from understanding requirements todescribing your detailed design – it is needed to apply a different style in each circumstance.

This section describes style guidelines pertaining to different types of class diagrams.

A class is effectively a template from which objects are created (instantiated). Classes define attributes, information that is pertinent to their instances, and operations, functionality that the objects support. Classes will also realize interfaces (more on this later).

Class diagrams are widely used to describe the types of objects in a system and their relationships. Class diagrams model class structure and contents using design elements such as classes, packages and objects. Class diagrams describe three different perspectives when designing a system, conceptual, specification, and implementation. These perspectives becomeevident as the diagram is created and help solidify the design.

INTERFACES

An interface is a collection of operation signature and/or attribute definitions that ideally defines a cohesive set of behaviors. **I**nterface a class or component must implement the operations and attributes defined by the interface. Any given class or component may implement zero or more interfaces and one or more classes or components can implement the same interface.

RELATIONSHIPS

A relationship is a general term covering the specific types of logical connections found on a class and object diagram.
Class diagrams also display relationships such as containment, inheritance, associations and others.

The association relationship is the most common relationship in a class diagram. The associationshows the relationship between instances of classes.

Another common relationship in class diagrams is a generalization. A generalization is used when two classes are similar, but have some differences.

Aggregation is a variant of the "has a" or assocation relationship; composition is more specificthan aggregation.
gregation occurs when a class is a collection or container of other classes, but where the contained classes do not have a strong **life cycle dependency** on the container--essentially, if thecontainer is destroyed, its contents are not.

Association are semantic connection between classes. When an association connects two classes

, each class can send messages to other in a sequence or a collaboration diagram . Associationscan be bidirectional or unidirectional.

Dependencies connect two clases . Dependencies are always unidirectional and show that oneclass, depends on the definitions in another class .



Experiment-10

AIM: Estimation of Project Metrics.

REQUIREMENTS:

Hardware Interfaces

- Pentium(R)4 CPU 2.26GHz, 128 MBRAM
- Screenresolutionofatleast800x600requiredforproper and complete viewing of screens. Higher resolution would not be a problem.
- CD ROM Driver

Software Interfaces

- Any window-based operating system(Windows 95/98/2000/XP/NT)
- Word Pad or Microsoft Word

THEORY

Project Estimation Techniques

A software project is not just about writing a few hundred lines of source code to achieve a particular objective. The scope of a software project is comparatively quite large, and such a project could take several years to complete. However, the phrase "quite large" could only give some (possibly vague) qualitative information. As in any other science and engineering discipline, one would be interested to measure how complex a project is. One of the major activities of the project planning phase, therefore, is to estimate various project parameters in order to take proper decisions. Some important project parameters that are estimated include:

Project size: What would be the size of the code written say, in number of lines, files, modules?

Cost: How much would it cost to develop a software? A software may be just pieces of code, but one has to pay to the managers, developers, and other project personnel.

Duration: How long would it be before the software is delivered to the clients?

Effort: How much effort from the team members would be required to create the software?

In this experiment we will focus on two methods for estimating project metrics: COCOMO and Halstead's method.

COCOMO

COCOMO (Constructive Cost Model) was proposed by Boehm. According to him, there could be three categories of software projects: organic, semidetached, and embedded. The classification is done considering the characteristics of the software, the development team and environment. These product classes typically correspond to application, utility and system programs, respectively. Data processing programs could be considered as application programs. Compilers, linkers, are examples of utility programs. Operating systems, real-time system programs are examples of system programs. One could easily apprehend that it would take much more time and effort to develop an OS than an attendance management system.

The concept of organic, semidetached, and embedded systems are described below.

Organic: A development project is said to be of organic type, if The project deals with developing a well understood application The development team is small.

The team members have prior experience in working with similar types of projects:

Semidetached: A development project can be categorized as semidetached type, if The team consists of some experienced as well as inexperienced staff Team members may have some experience on the type of system to be developed.

Embedded: Embedded type of development project are those, which Aims to develop a software strongly related to machine hardware Team size is usually large

Boehm suggested that estimation of project parameters should be done through three stages: Basic COCOMO, Intermediate COCOMO, and Complete COCOMO.

Basic COCOMO Model

The basic COCOMO model helps to obtain a rough estimate of the project parameters. It estimates effort and time required for development in the following way:

$$\textbf{Effort} = a * (KDSI)b \ PM$$

$$\textbf{Tdev} = 2.5 * (Effort)c \ Months$$

Where:

KDSI is the estimated size of the software expressed in Kilo Delivered Source Instructions

a, b, c are constants determined by the category of software project

Effort denotes the total effort required for the software development, expressed in person months (PMs).

Internal Marks Assessment Rubrics

	Rubrics	High	Medium	Low	
Lab Continuous Evaluation (20 Marks)	R1: Experiment Performance (5 Marks)	Executed the input with desired output (5 Marks)	Partially executed (4 Marks)	Executed the input with no desired output (0 Marks)	**Every Lab Session**
	R2 : Record (5 Marks)	Completed records was submitted (5 Marks)	Record was Submitted but having minor mistake (4 Marks)	Record was submitted but having major mistake or not submitted at all (0-2 Marks)	
	R3 : Experiment Result (5 Marks)	Attended almost every lab session with 1 or 2 lab missed (5 Marks)	Attended lab sessions but missed 3 or 4 (4 marks)	Not regular at all in Labs more than 4 misses (0-3 Marks)	
	R4 : Viva Voice (5 questions 1 Marks each i.e. 5 Marks)				