

GANPAT UNIVERSITY

U. V. PATEL COLLEGE OF ENGINEERING



2CEIT502 - SOFTWARE ENGINEERING

LAB MANUAL

July-Dec 2023

5th Semester Computer Engineering & Information Technology



CERTIFICATE



**Ganpat
University**

॥ विद्यया समाजोत्कर्षः ॥

**U.V. Patel
College of
Engineering**



25 Years excellence in innovative technical education in shaping engineers

This is to certify that Mr. / Ms. _____

Enrolment No. _____ of B. Tech Semester V

_____ branch has satisfactorily

completed the term work in **Software Engineering (2CEIT502)**

within four walls of Ganpat University – U. V. Patel College of

Engineering, Ganpat Vidyanagar, Kherva for the academic year

202__/202__ as prescribed in curriculum.

Date of Submission: ____/____/202__

Subject Faculty

Head of the Department

INDEX

List of Laboratory Experiments

#	Experiment Aim	Course Outcome	Page No	Start Date	End Date	Sign	Grade / Marks
1	To study various Software development life cycle (SDLC) models prepare a report of their analysis with answering given questions.	CO1	1				
2	To prepare a problem statement and Requirement Elicitation techniques.	CO3	3				
3	To create a static view (structure diagrams) of a software design on chosen definition.	CO2	8				
4	To create a dynamic view (behavioral diagrams) of a software design on chosen definition.	CO2	14				
5	Identify the design principle that is being violated in relation to the given scenario.	CO3	23				
6	To study about various Project Management Software (tools) available in the market.	CO4	24				
7	Prepare Software requirement specification (SRS) document.	CO3	27				
8	To study about Structure System analysis and Design method (SSADM).	CO2	29				
9	To learn estimating techniques to estimate project parameters on a given case study using SE Vlabs tool.	CO4	33				
10	To prepare the test suite for a given case study on SE Vlab tool.	CO5	37				

Assignments

1							
2							

Instructions for the students:

- It is expected from the students to perform the practical number 2,3, 4 and 7 in **group**. Your group should be comprising with the same members with whom you are associated in your capstone project-1. Rest of the all practicals (except the above said) are to be performed **individually** only.
- The students are advised to use left side blank pages to write the answers of practicals tasks if you feel the space is inadequate for particular question

Practical – 1

Aim: To study various Software development life cycle (SDLC) models prepare a report of their analysis with answering given questions.

Task-1: Watch these videos on Youtube first:

What is SDLC? (4:54) <https://www.youtube.com/watch?v=4xCldpbDZ10>

Waterfall model: (6:08) https://www.youtube.com/watch?v=Y_A0E1ToC_I

Task-2 : Attempt the questions given below:

1. Write one example of a software project that would be amenable to the classical waterfall model.

2. What is the difference between incremental process model and evolutionary process model?

3. Which model is more suitable for your capstone project-1? Why?

4. State the parameters to choose suitable life cycle model for any system.

-
5. Which life cycle model is/are widely used in software industry nowadays?
-
-
-

6. Give the comparison of all life cycle models based upon characteristics of different SDLC models, fill the details with following options: **Poor, Good, Excellent**

Sr. No	Model / Feature	Waterfall model	Prototype Model	Spiral Model	Iterative or Incremental Model
1	Unclear User requirements				
2	Unfamiliar technology				
3	Complex System				
4	Short-time schedule				
5	Strong project management				
6	Cost limitation				
7	Documentation				
8	Component reusability				

Practical – 2

Aim: To prepare a problem statement and Requirement Elicitation techniques.

Theory:

The first phase of requirements engineering begins with requirements elicitation i.e. gathering of information about requirements. Here, requirements are identified with the help of customer and existing system processes. So from here begins the preparation of problem statement. So, basically a problem statement describes “*what needs to be done without describing how*”.

Note: The questions given here are supposed to be ask by students to themselves. Because asking such questions will get them the idea of software to be developed. Make sure that you are not supposed to write description of your problem by providing the answers of these questions. Think of the answers, write the points in some temporary space, discuss them with your partner and then compile the notes in paragraphed manner.

1. What actually your system is?
2. Why you want to develop this system? Who will be benefitted by making the use of that software? Purpose for the development.
3. Who will be the targeted audience/users of your software solution? Who will be your users and stake holders of the system? Types of users (not number of users)? Roles and responsibility of the users? Objectives of the development.
4. What set of functionality you are supposed to provide? Categorize the functionality by types of users?
5. Detailed discussion on features... How the features will work? What inputs and outputs are required for each functionality/features?

For Example:

For Simple User	For Manager (Production, sales, marketing etc.)	For Board of governors, directors etc.	For Employees of the company
Feature 1:	Feature 1:	Feature 1:	Feature 1:
Feature 2:	Feature 2:	Feature 2:	Feature 2:
Feature 3:	Feature 3:	Feature 3:	Feature 3:
...
Feature n:	Feature n:	Feature n:	Feature n:

6. What are basic hardware/software requirements of the software? (Recommended and Minimum)

It might be possible that all above listed questions may / may not be relevant for the problem you decided. So in that case you can safely ignore (some of) such questions.

**Task-1: Write your one-page (subjective) problem description in paragraphed manner:
(Hint: Number of relevant questions = number of paragraph)**

Task-2: Draw the prescribed format of the table here (below provided space) and write the features/functionalities you have thought of for each of user(s)/Stakeholder(s).

Task-3 To identify the various elicitation techniques and their usage for the Banking case study.

Note: Requirement elicitation is the process of seeking, discovering, acquiring and elaborating requirements. This includes learning and understanding the needs of the users. This activity is communication centric and iterative in nature. The techniques used here are important to get stakeholder consensus on the requirements.

Case study

KHL is a leading global bank that provides standard banking services to its customers spanning across the globe. The head office is located in London and the bank has a presence in more than 20 countries with a client base of nearly 500,000. Tuning with time and ever increasing clients and transactions, the bank has specialized branches for specific customer segments like consumer, corporate and the SMEs. KHL Bank aims to be a one stop shop for its customers to address their changing financial needs. KHL bank offers various banking products and services across its customer segments including Core Banking and Wealth Management amongst other services. KHL Bank is well known among its clients for world-class processes and speed of execution of transactions as part of core banking. Currently, KHL bank has made a proposal for investing around \$200 million in setting-up 24x7 banking support facilities for the customers. The bank has decided to leverage IT for automating several business processes including:

- 1) Managing Accounts
- 2) Transaction Management

The aim of this proposed banking system is to create a paperless bank thereby moving towards e-banking. FinSoft, a newly established software company has the vision of providing software solutions in the financial sector. Managing Director (MD) of KHL bank has approached FinSoft for the computerization of the bank so that there is no more manual way of doing transactions in any of its branches. As part of automation, the KHL bank users are to be provided with ATM facility, e-banking facility over internet and phone banking facility over land lines and cellular networks. FinSoft is doing such a project for the first Time. Requirements development team in FinSoft has planned for carrying out the requirement elicitation for this project.

In the context of the case study, for the following scenarios identify the most appropriate requirements elicitation techniques: **Brainstorming, Workshops, Questionnaire, Task Analysis, Observation, Prototyping, Scenario identification**

Scenario Requirement elicitation technique

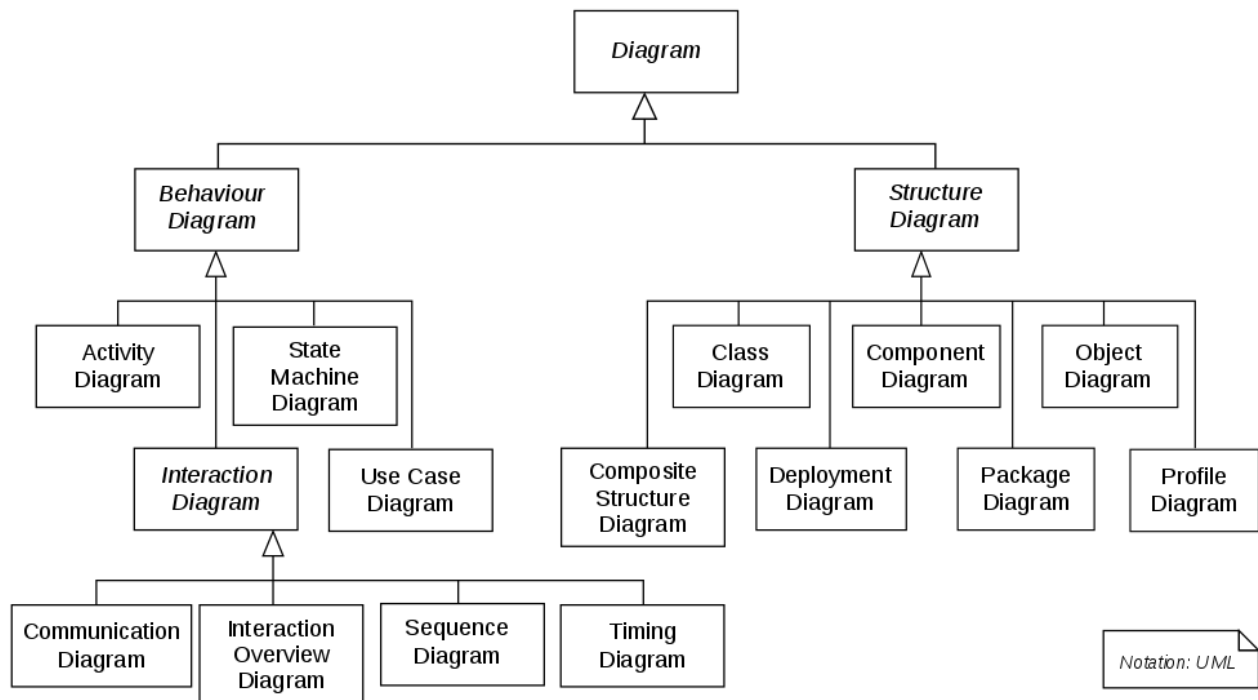
#	Scenario	Technique to be used (First box) Justification (in next cell)
1	Interrogative conversation with Managers, Cashiers, Clerks and other Staff for arriving at the requirement for automating transactions.	
2	Formal and planned requirement discussion in a conference room conducted among managers of diversified branches facilitated by anchor.	
3	Survey form circulated among the users (account holders) who visit the bank, to ease their interactions with bank	
4	Analysis for understanding mode of transactions-Checks, Cash, DD, MT, Gold, etc.	
5	Ethnographers deployed for understanding the users interactions with bank officials.	
6	UI design of e-banking portal, ATM, Computer Systems	
7	Understanding the process involved in each transaction like withdraw, deposit, fund transfer etc.	

Practical-3

Aim: To create a static view (structure diagrams) of a software design on chosen definition.

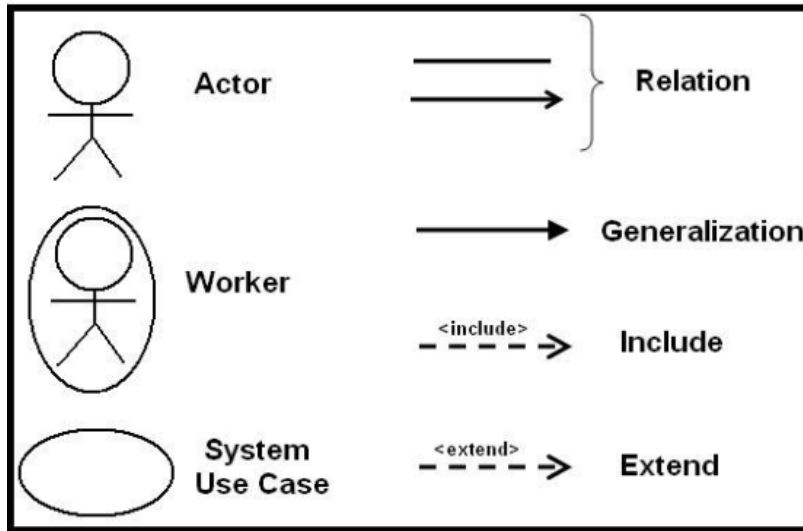
Theory: The **Unified Modeling Language (UML)** is a general-purpose, developmental, modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system.

The creation of UML was originally motivated by the desire to standardize the disparate notational systems and approaches to software design. It was developed at Rational Software in 1994–1995, with further development led by them through 1996.

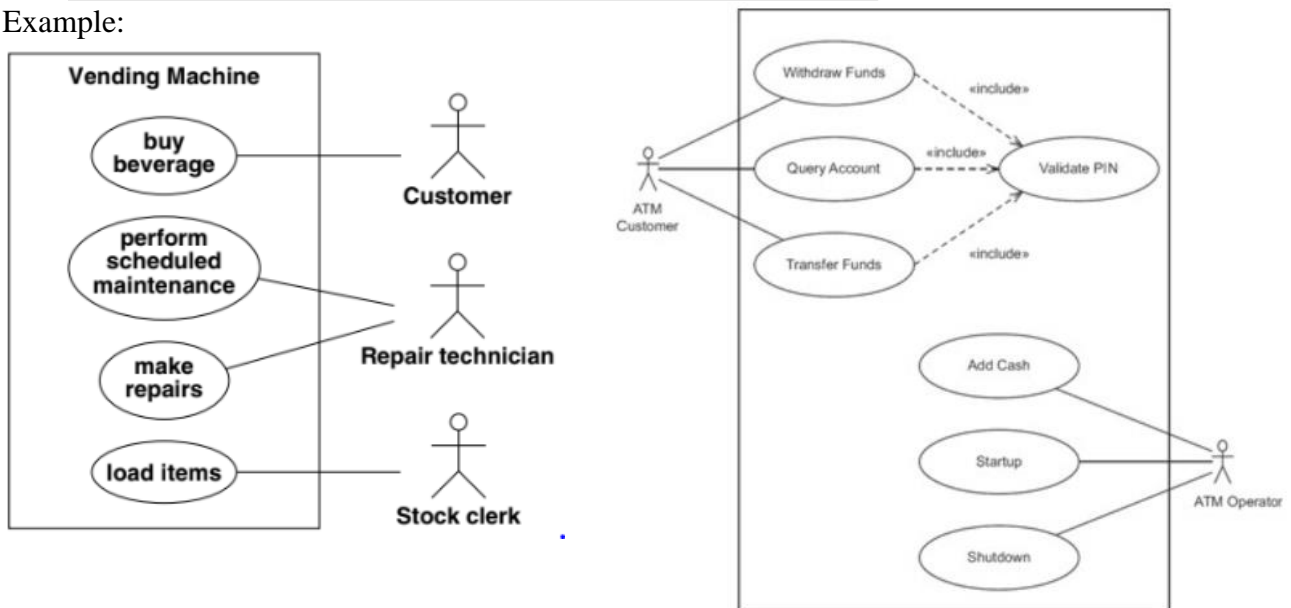


The students are advised here to prepare 5 diagrams for their identified definition.

- 1) Use case diagram
- 2) Class diagram
- 3) Sequence diagram
- 4) Activity diagram
- 5) State diagram.

Notations:**1 Use case diagram**

Example:

**Steps to draw a use case diagram:**

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

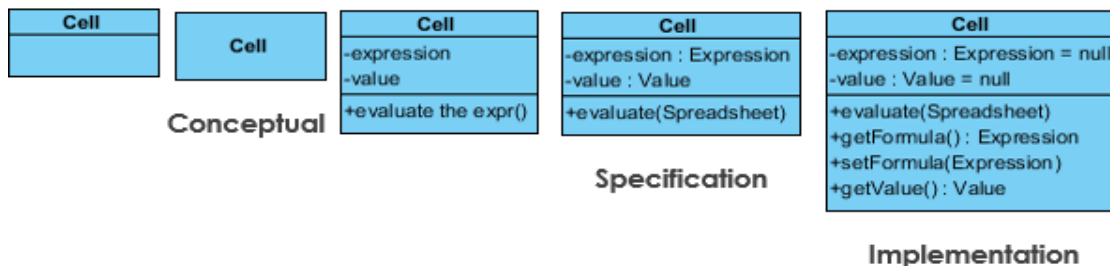
1. Identify the Actors (role of users) of the system.
2. For each category of users, identify all roles played by the users relevant to the system.
3. Identify what are the users required the system to be performed to achieve these goals.
4. Create use cases for every goal.
5. Structure the use cases.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

Guidelines for use case models:

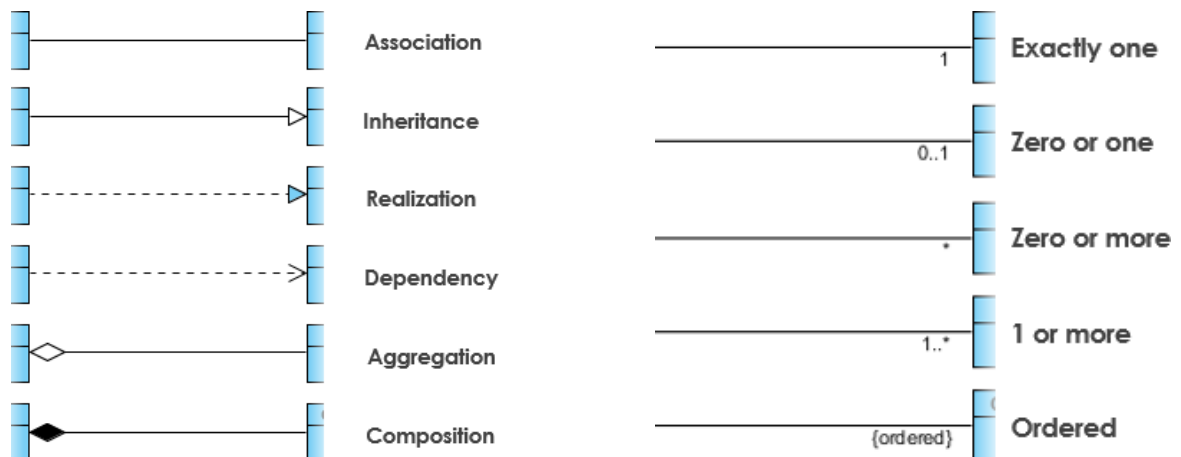
- (1). **First determine the system boundary.** It is impossible to identify use cases or actors if the system boundary is unclear.
- (2). **Ensure that actors are focused.** Each actor should have a single, coherent purpose. If a real-world object embodies multiple purposes, capture them with separate actors. For example, the owner of a personal computer may install software, set up a database, and send email. These functions differ greatly in their impact on the computer system and the potential for system damage. They might be broken into three actors: system administrator, database administrator, and computer user. Remember that an actor is defined with respect to a system, not as a free-standing concept.
- (3). **Each use case must provide value to users.** A use case should represent a complete transaction that provides value to users and should not be defined too narrowly. For example, dial a telephone number is not a good use case for a telephone system. It does not represent a complete transaction of value by itself; it is merely part of the use case make telephone call. The latter use case involves placing the call, talking, and terminating the call. By dealing with complete use cases, we focus on the purpose of the functionality provided by the system, rather than jumping into implementation decisions. The details come later. Often there is more than one way to implement desired functionality.
- (4). **Relate use cases and actors.** Every use case should have at least one actor, and every actor should participate in at least one use case. A use case may involve several actors, and an actor may participate in several use cases.
- (5). **Remember that use cases are informal.** It is important not to be obsessed by formalism in specifying use cases. They are not intended as a formal mechanism but as a way to identify and organize system functionality from a user-centered point of view. It is acceptable if use cases are a bit loose at first. Detail can come later as use cases are expanded and mapped into implementations.
- (6). **Use cases can be structured.** For many applications, the individual use cases are completely distinct. For large systems, use cases can be built out of smaller fragments using relationships.

2 Class Diagram

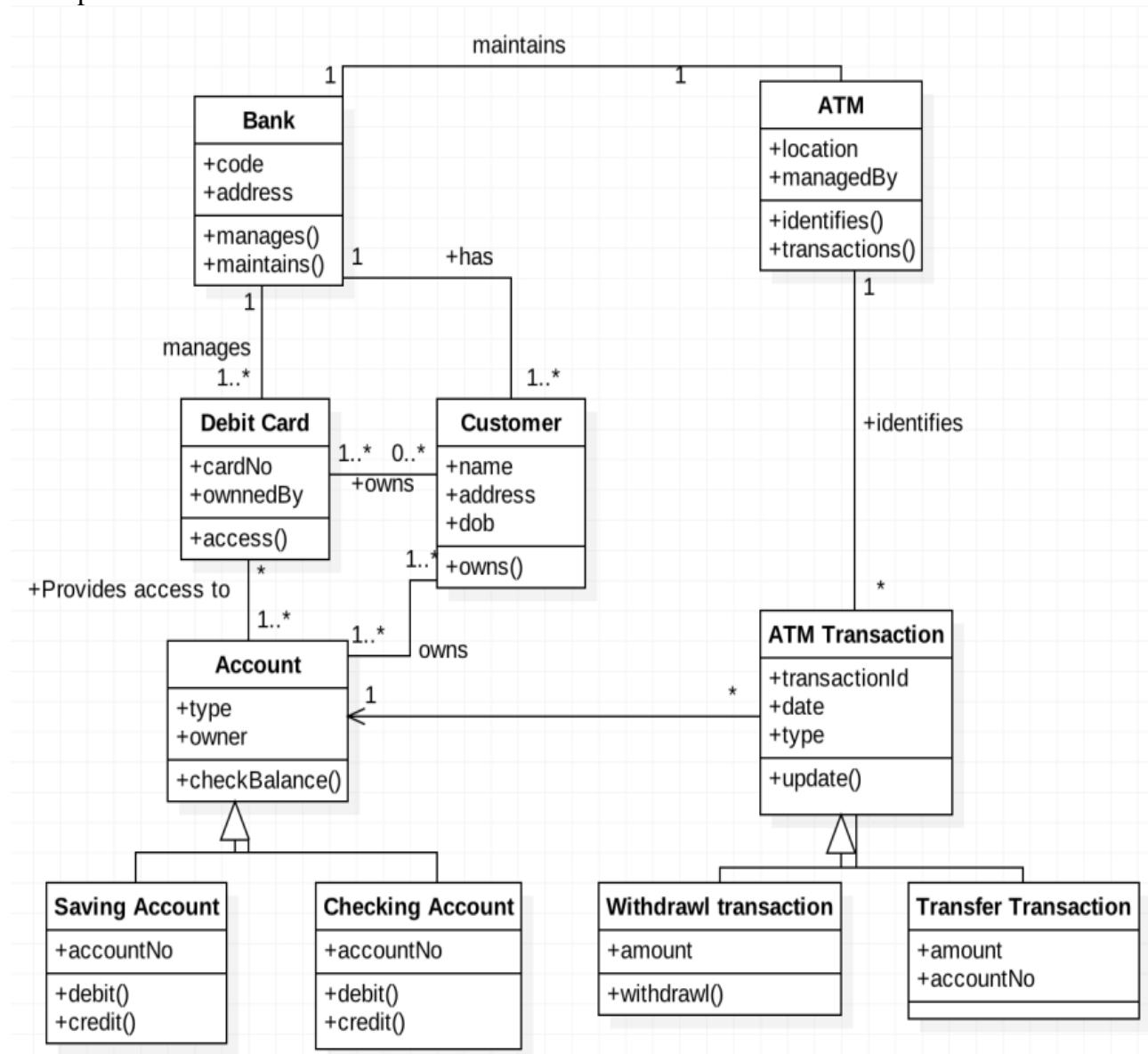
- (a) Different notations of class:



- (b) Relationships and cardinality:



Example:



(1) Space to draw your **use case diagram** (by pencil only):

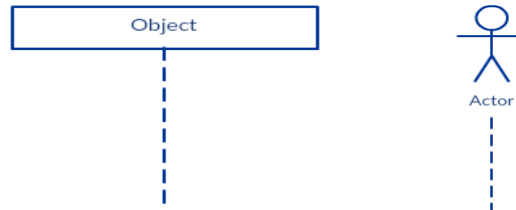
(2) Space to draw your **class diagram** (by pencil only):

Practical-4

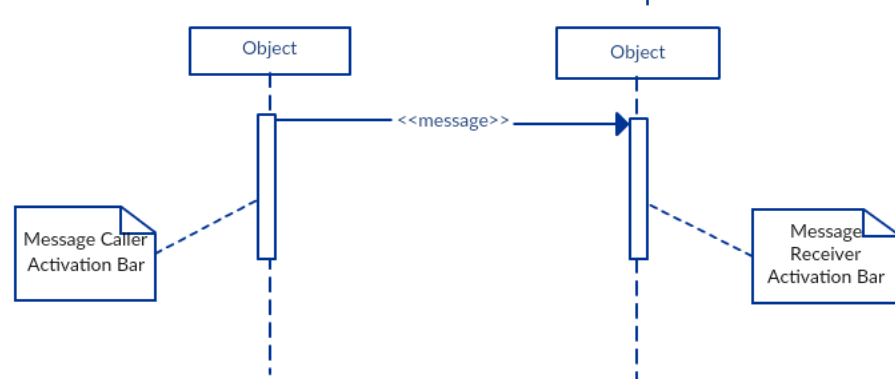
Aim: To create a dynamic view (behavioral diagrams) of a software design on chosen definition.

3 Sequence Diagram

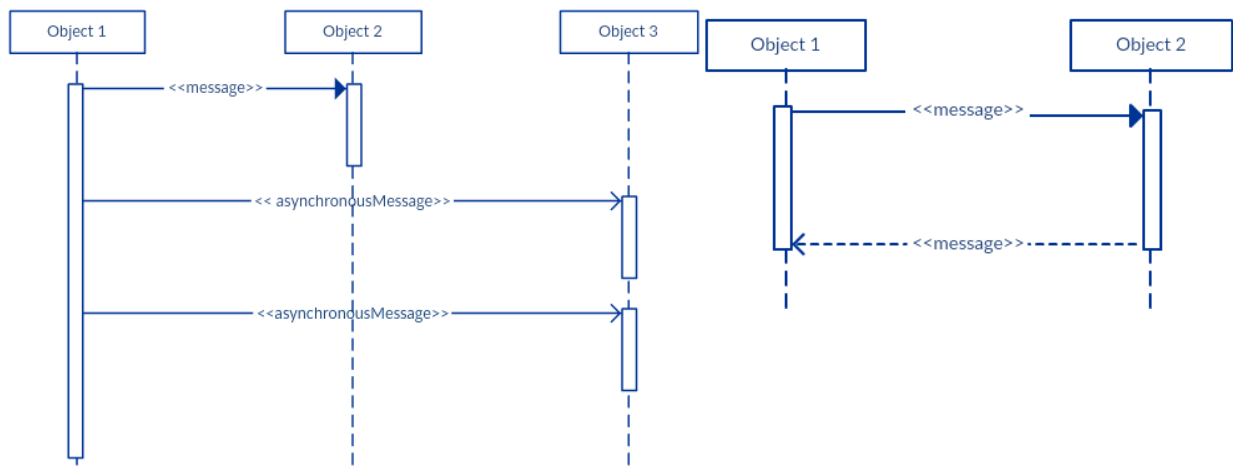
(a) Lifeline and actor Notations:



(b) Activation Bars:

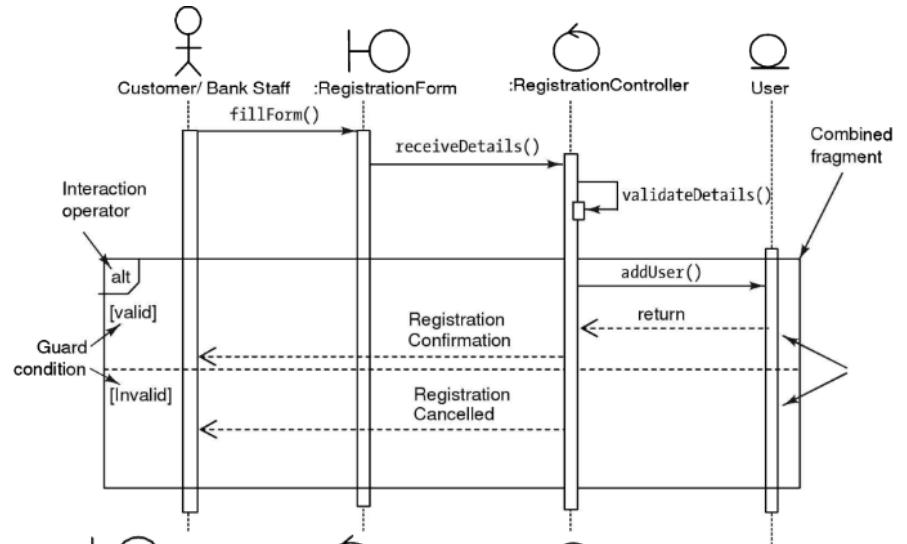


(c) Communication (messaging):

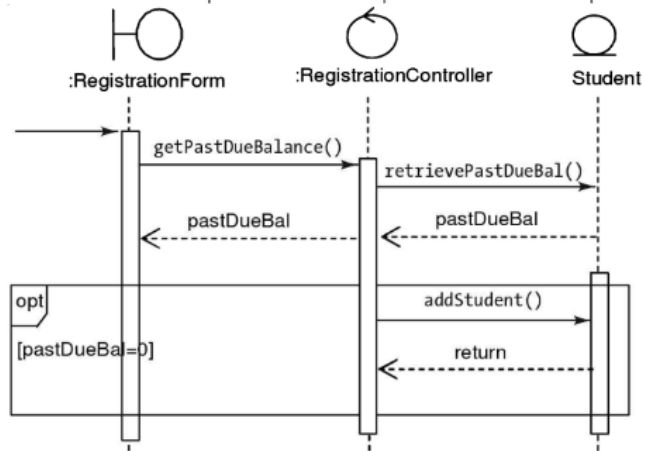


(d) Combined fragments:

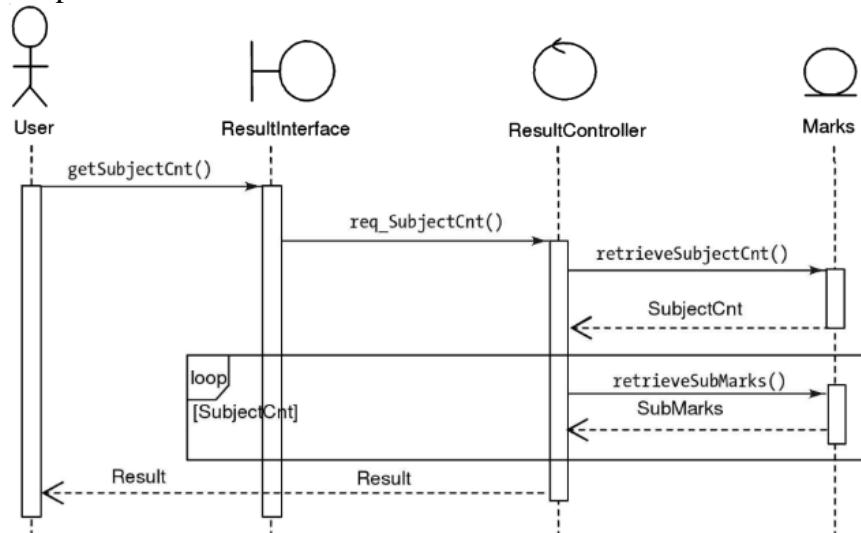
(1) Alternatives:



(2) Options:



(3) Loops:



4 Activity Diagram

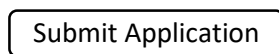
(a) Initial state:



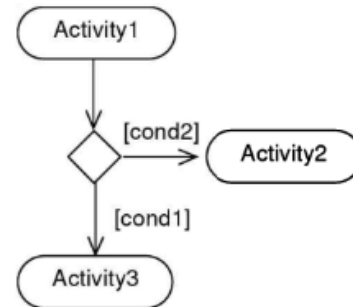
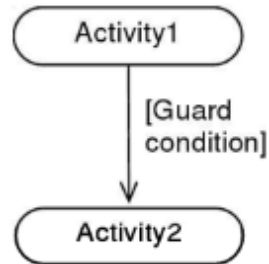
(b) Final State:



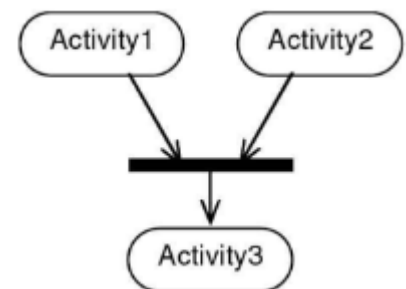
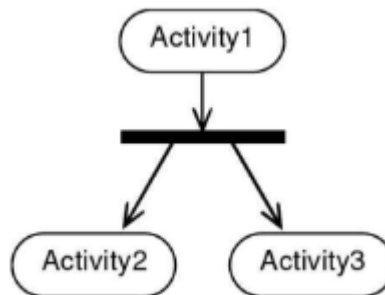
(c) Action / Activity:



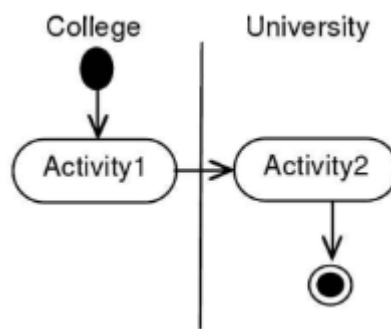
(d) Transition and decision:



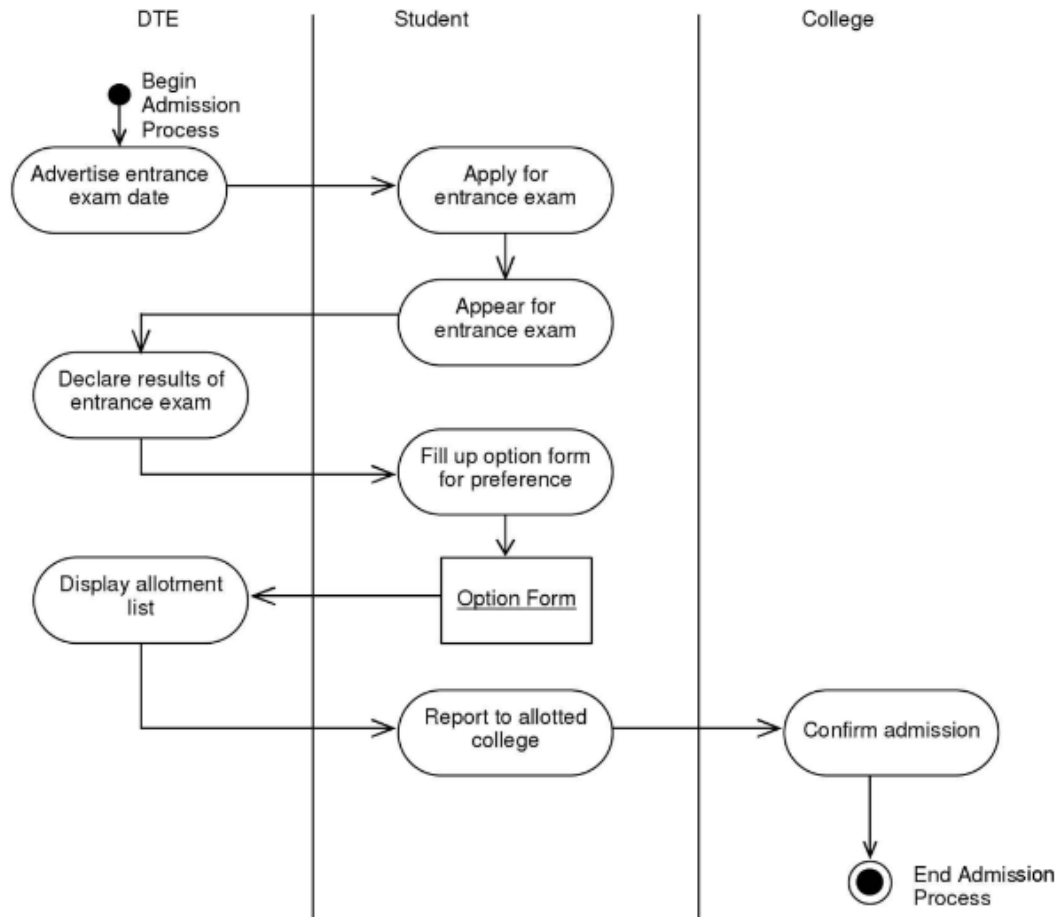
(e) Synchronization: Fork and Join



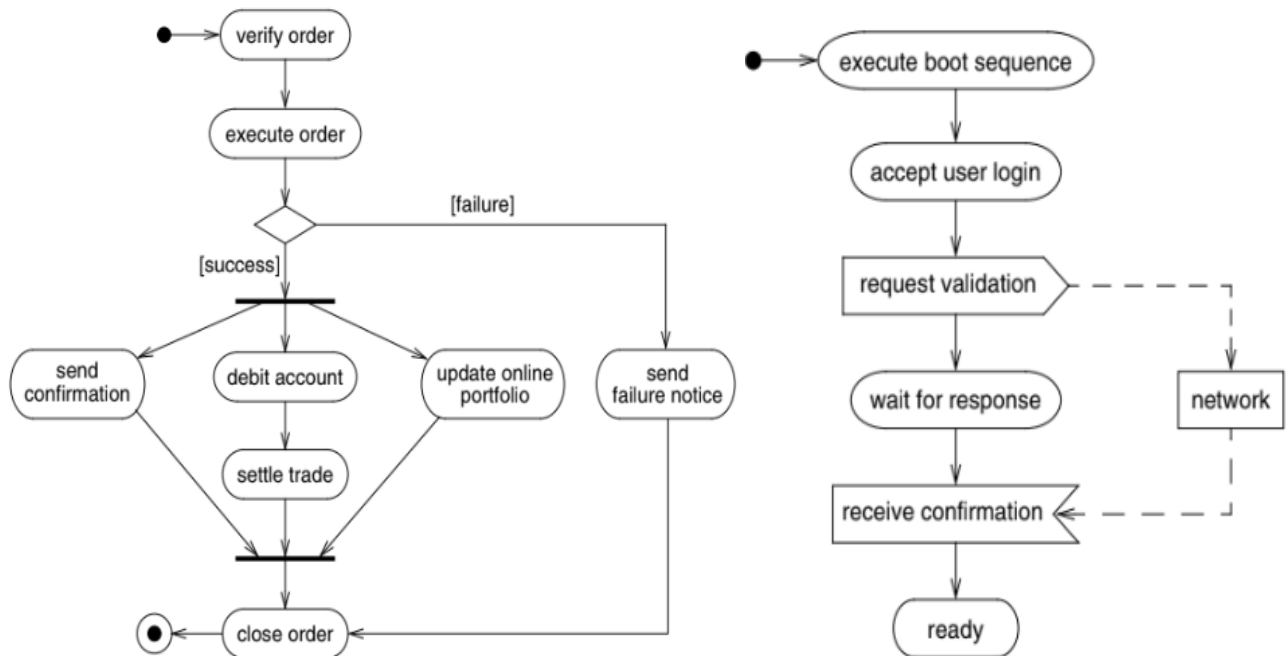
(f) Swimlanes:



Example (Swimlane activity diagram):



Example:



5 State diagram

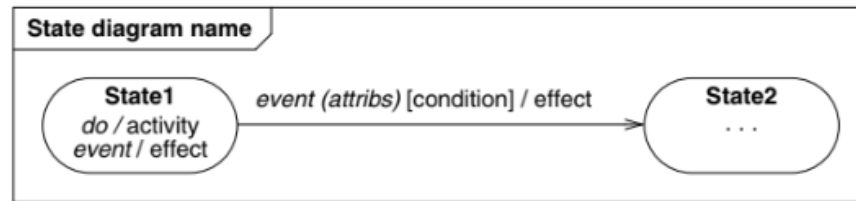
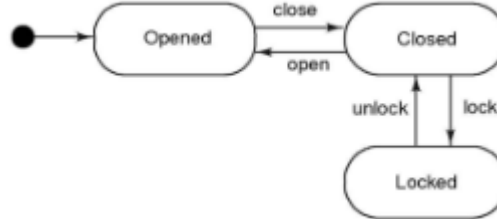
(a) Initial state:



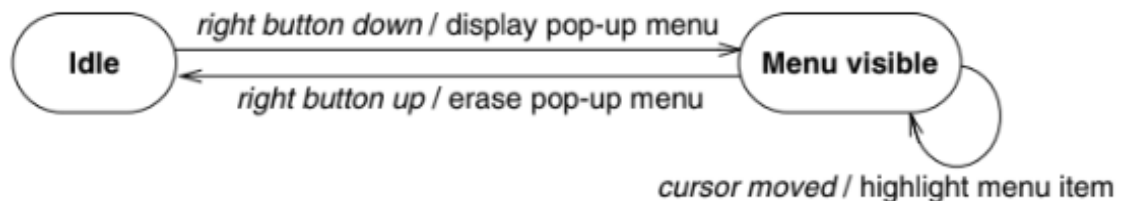
(b) Final State:



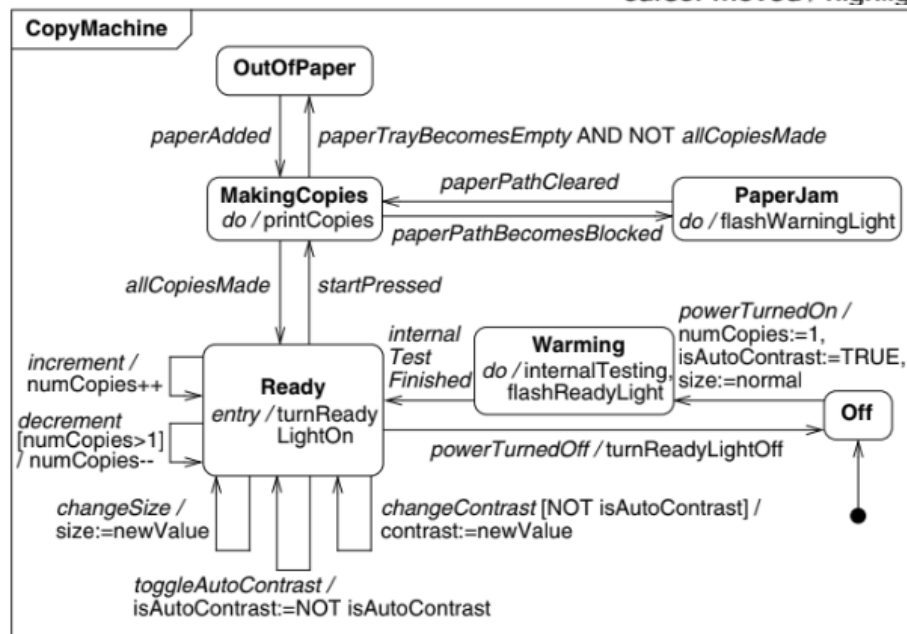
(c) State and transition:

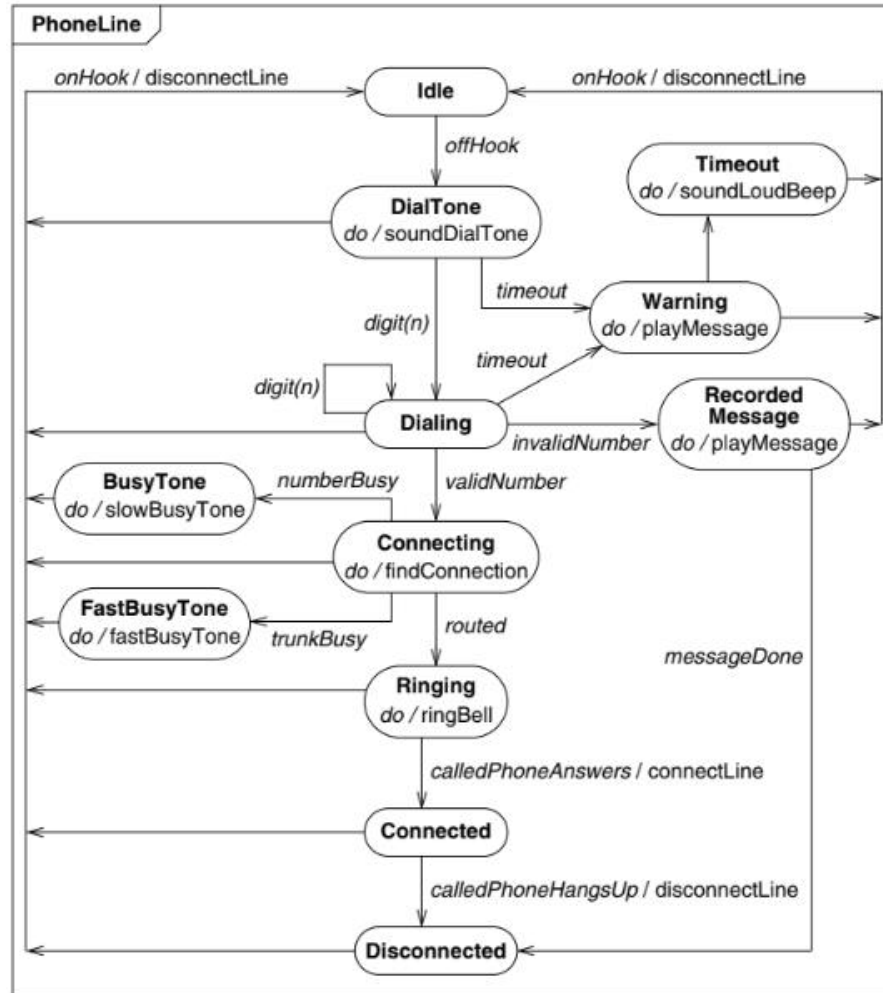


(d) Activity effects:



Examples:





(1) Space to draw (any two use cases from use case diagram) **Sequence diagram** (by pencil only):

(2) Space to draw **activity diagram** (by pencil only):

(3) Space to draw **State diagram** (by pencil only):

Practical-5

Aim: Identify the design principle that is being violated in relation to the given scenario.

Objective: Identify the design principle that is being violated in relation to the given scenario.

Note: A good object oriented design not only meets the specified requirements but also addresses implicit requirements. There are five design principles which address most of the implicit requirements:

1. **Abstraction:** Focus on solving a problem by considering the relevant details and ignoring the irrelevant.
2. **Encapsulation:** Wrapping the internal details, thereby making these details inaccessible. Encapsulation separates interface and implementation, specifying only the public interface to the clients, hiding the details of Implementation.
3. **Decomposition and Modularization:** Dividing the problem into smaller, independent, interactive subtasks for placing different functionalities in different components.
4. **Coupling & Cohesion:** Coupling is the degree to which modules are dependent on each other. Cohesion is the degree to which a module has a single, well defined task or responsibility. A good design is one with loose coupling and strong cohesion.
5. **Sufficiency, Completeness and Primitiveness:** Design should ensure the completeness and sufficiency with respect to the given specifications in a very simple way as possible.

Problem: Which of the following design principle(s) have been violated in the following scenarios?

#	Scenario	Principle Violated
1	Important information of a module is directly accessible by other modules	
2	Too many global variables in the program after implementing the design	
3	Code breaks in unexpected places	
4	Unfulfilled requirements in the code after the design has been implemented	
5	Cyclic dependency among classes	
6	Huge class doing too many unrelated operations	
7	Several un-related functionalities/tasks are carried out by a single module	
8	All data of all classes in public	
9	Design resulting in spaghetti code	
10	An algorithm documented as part of design is not understandable by the programmers	

Practical-6

Aim: To study about various Project Management Software (tools) available in the market.

Note:

- List out at least 10 tools and study 2 tools in detail with exploring their functionalities and differences.
- Submit a Report of your study.

Task-1: Write the best of 10 PMS software tools:

- (1)
- (2)
- (3)
- (4)
- (5)
- (6)
- (7)
- (8)
- (9)
- (10)

Task-2: Choose any two tools of your choice, download and install it to your machine. Use it and prepare a report containing its functionalities /features and every major details of the software.

Practical-7

Aim: Prepare a Software requirement specification (SRS) document for the chosen project definition.

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.

What generally an SRS should contain?

1.	Introduction
1.1	Purpose
1.2	Document convention
1.3	Intended Audience
1.4	Additional Information
1.5	Contact info. / 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
4.1.1	Description and priorities
4.1.2	Action / Result
4.1.3	Functional Requirements
4.2	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
App. A	Terminology / Glossary / Definitions list
App. B	To be continued.

Another template proposed by Roger pressman:



Software Requirements Specification Template

A software requirements specification (SRS) is a document that is created when a detailed description of all aspects of the software to be built must be specified before the project is to commence. It is important to note that a formal SRS is not always written. In fact, there are many instances in which effort expended on an SRS might be better spent in other software engineering activities. However, when software is to be developed by a third party, when a lack of specification would create severe business issues, or when a system is extremely complex or business critical, an SRS may be justified.

Karl Wiegers [Wie03] of Process Impact Inc. has developed a worthwhile template (available at www.processimpact.com/process_assets/srs_template.doc) that can serve as a guideline for those who must create a complete SRS. A topic outline follows:

Table of Contents

Revision History

1. Introduction

- 1.1 Purpose
- 1.2 Document Conventions
- 1.3 Intended Audience and Reading Suggestions
- 1.4 Project Scope
- 1.5 References

2. Overall Description

- 2.1 Product Perspective

- 2.2 Product Features
- 2.3 User Classes and Characteristics
- 2.4 Operating Environment
- 2.5 Design and Implementation Constraints
- 2.6 User Documentation
- 2.7 Assumptions and Dependencies

3. System Features

- 3.1 System Feature 1
- 3.2 System Feature 2 (and so on)

4. External Interface Requirements

- 4.1 User Interfaces
- 4.2 Hardware Interfaces
- 4.3 Software Interfaces
- 4.4 Communications Interfaces

5. Other Nonfunctional Requirements

- 5.1 Performance Requirements
- 5.2 Safety Requirements
- 5.3 Security Requirements
- 5.4 Software Quality Attributes

6. Other Requirements

Appendix A: Glossary

Appendix B: Analysis Models

Appendix C: Issues List

A detailed description of each SRS topic can be obtained by downloading the SRS template at the URL noted earlier in this sidebar.

INFO

Instructions for students:

1. The students are advised to ask the lab faculty to provide some sample SRS documents for reference.
2. Kindly go through the sample SRS document, read and inspect each line.
3. Once you develop the confidence after referring the samples. Start to develop the SRS for your chosen definition with your group members.
4. Prepare a report with minimum 20 number of pages. Do spiral bind of the printed report and don't forget to produce this lab manual plus SRS report at the time of submission viva and practical exam.

Practical-8

Aim: To study about Structure System analysis and Design method (SSADM).

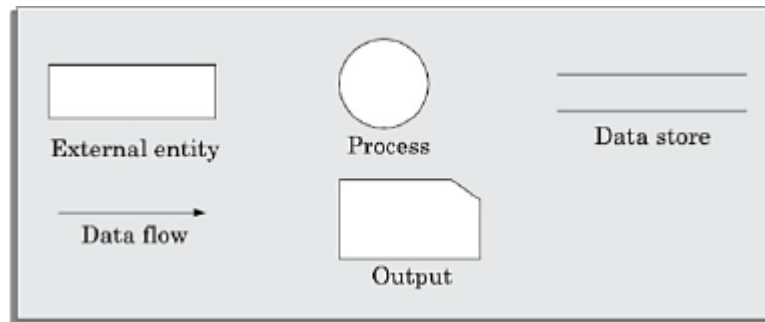
Objective: To get familiar with Structure Approach Method and learn various Techniques of SSADM to model a System.

References: Software Engineering by Roger Pressman, Software Engineering by K.K. Aggarwal, Software Engineering by Rajib mall.

Prerequisite: Knowledge of Data Modeling, Functional Modeling.

Summary: Structured Systems Analysis and Design Methodology (SSADM) is a systems approach to the analysis and design of information systems. SSADM uses a combination of three techniques: Logical Data Modeling, Data Flow Modeling, and Entity Behavior Modeling. System analysis is major focus on “**what**” the system is accomplished, not **how**. SSADM uses various Modeling tools to represent the data flows, processing and data stores like data flow diagrams, data dictionary, process specification, entity-relationship diagrams.

DFD Notations:



Practical: Draw the context level (0-level), 1-level and 2-level DFD for any two definitions.

Problem (pick any two):

- 1) Online Library Management System (LMS).
- 2) Online Airline Reservation System (ARS).
- 3) Online Railway Reservation System (RRS).
- 4) Online Video Library Management System (VLMS).

Guidelines to prepare a DFD:

- 1) Many beginners commit the mistake of drawing more than one bubble in the context diagram. Context diagram should depict the system as a single bubble.
- 2) Many beginners create DFD models in which external entities appearing at all levels of DFDs. All external entities interacting with the system should be represented only in the context diagram. The external entities should not appear in the DFDs at any other level.
- 3) It is a common oversight to have either too few or too many bubbles in a DFD. Only three to seven bubbles per diagram should be allowed. This also means that each bubble in a DFD should be decomposed three to seven bubbles in the next level.
- 4) Many beginners leave the DFDs at the different levels of a DFD model unbalanced.

- 5) A common mistake committed by many beginners while developing a DFD model is attempting to represent control information in a DFD.

Common mistakes:

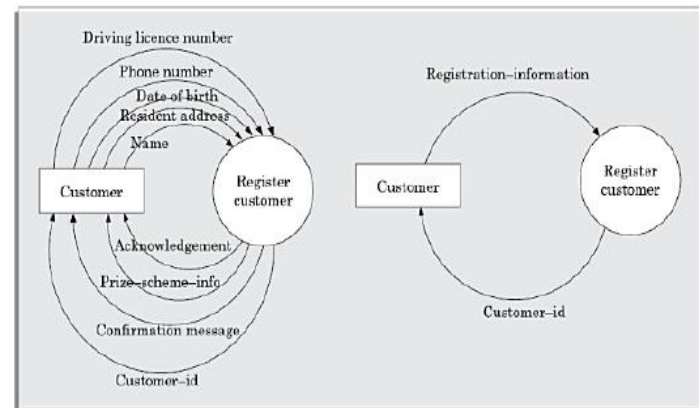


Figure 6.7: Illustration of how to avoid data cluttering.

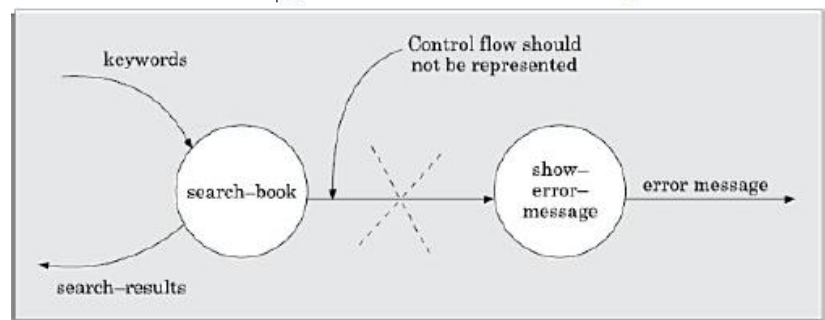
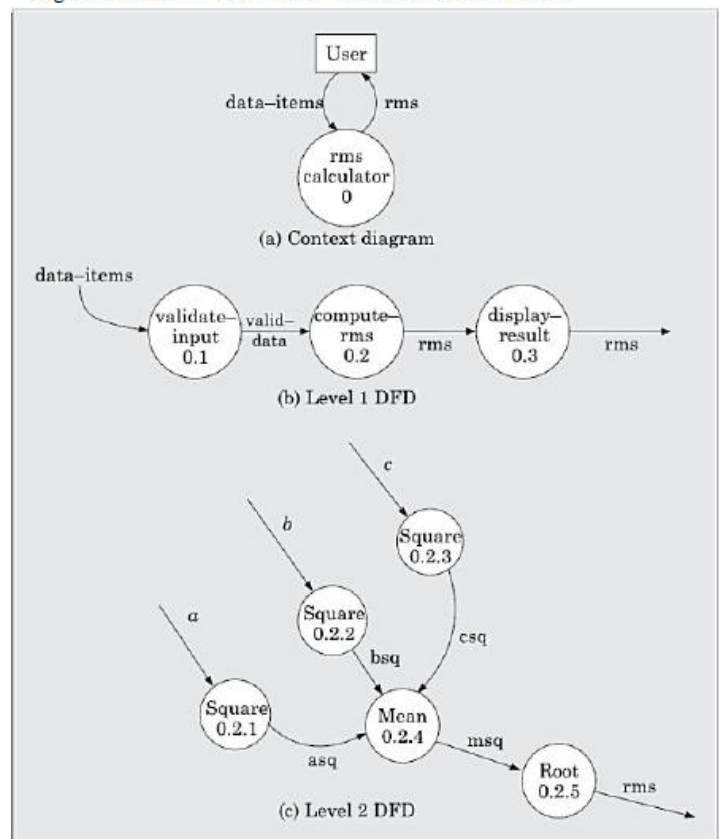


Figure 6.6: It is incorrect to show control information on a DFD.

Example:

A software system called RMS calculating software would read three integral numbers from the user in the range of -1000 and +1000 and would determine the root mean square (RMS) of the three input numbers and display it. In this example, The system accepts three integers from the user and returns the result to him (context-level). To draw the level 1 DFD, from a cursory analysis of the problem description, we can see that there are four basic functions that the system needs to perform—accept the input numbers from the user, validate the numbers, calculate the root mean square of the input numbers and, then display the result. After representing these four functions in level-1 we observe that the calculation of root mean square essentially consists of the functions—calculate the squares of the input numbers, calculate the mean, and finally calculate the root. This decomposition is shown in the level 2 DFD.



Draw the DFD (level-0, level-1, level-2) for first chosen definition here:

Draw the DFD (level-0, level-1, level-2) for second chosen definition here:

Practical-9

Aim: To learn estimating techniques to estimate project parameters on a given case study using SE Vlabs tool.

Case Study: The SE Vlabs Institute has been recently setup to provide state-of-the-art research facilities in the field of Software Engineering. Apart from research scholars (students) and professors, it also includes quite a large number of employees who work on different projects undertaken by the institution.

As the size and capacity of the institute is increasing with the time, it has been proposed to develop a Library Information System (LIS) for the benefit of students and employees of the institute. LIS will enable the members to borrow a book (or return it) with ease while sitting at his desk/chamber. The system also enables a member to extend the date of his borrowing if no other booking for that particular book has been made. For the library staff, this system aids them to easily handle day-to-day book transactions. The librarian, who has administrative privileges and complete control over the system, can enter a new record into the system when a new book has been purchased, or remove a record in case any book is taken off the shelf. Any non-member is free to use this system to browse/search books online. However, issuing or returning books is restricted to valid users (members) of LIS only.

The final deliverable would be a web application (using the recent HTML 5), which should run only within the institute LAN. Although this reduces security risk of the software to a large extent, care should be taken no confidential information (eg., passwords) is stored in plain text.

The SE Vlabs Institute has a IT management team of its own. This team has been given the task to execute the Library Information System project. The team consists of a few experts from industry, and a batch of highly qualified engineers experienced with design and implementation of information systems. It is planned that the current project will be undertaken by a small team consisting of one expert and few engineers. Actual team composition would be determined in a later stage.

Using COCOMO and based on the team size (small) and experience (high), the concerned project could be categorized as "organic". The experts, based on their prior experience, suggested that the project size could roughly be around 10 KLOC. This would serve as the basis for estimation of different project parameters using basic COCOMO, as shown below:

$$\text{Effort} = a * (\text{KLOC})^b \text{ PM}$$

$$T_{\text{dev}} = 2.5 * (\text{Effort})^c \text{ Months}$$

For organic category of project, the values of a, b, c are 2.4, 1.05, 0.38 respectively. So, the projected effort required for this project becomes

$$\begin{aligned} \text{Effort} &= 2.4 * (10)^{1.05} \text{ PM} \\ &= 27 \text{ PM (approx.)} \end{aligned}$$

So, around 27 person-months are required to complete this project. With this calculated value for effort we can also approximate the development time required:

$$T_{\text{dev}} = 2.5 * (27)^{0.38} \text{ Months}$$

$$= 8.7 \text{ Months (approx.)}$$

So, the project is supposed to be complete by nine months. However, estimations using basic COCOMO are largely idealistic. Let us refine them using intermediate COCOMO. Before doing so we determine the Effort Adjustment Factor (EAF) by assigning appropriate weight to each of the following attributes.

Cost drivers for Intermediate COCOMO (Source: http://en.wikipedia.org/wiki/COCOMO)						
Cost Drivers	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Size of application database		0.94	1.00	1.08	1.16	
Complexity of the product	0.70	0.85	1.00	1.15	1.30	1.65
Hardware attributes						
Run-time performance constraints			1.00	1.11	1.30	1.66
Memory constraints			1.00	1.06	1.21	1.56
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30	
Required turnabout time		0.87	1.00	1.07	1.15	
Personnel attributes						
Analyst capability	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Software engineer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project attributes						
Application of software engineering methods	1.24	1.10	1.00	0.91	0.82	
Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

The cells with yellow backgrounds highlight our choice of weight for each of the cost drivers. EAF is determined by multiplying all the chosen weights. So, we get

$$\text{EAF} = 0.53 \text{ (approx.)}$$

Using this EAF value we refine our estimates from basic COCOMO as shown below:

$$\begin{aligned}
 \text{Effort}_{\text{corrected}} &= \text{Effort} * \text{EAF} \\
 &= 27 * 0.53 \\
 &= 15 \text{ PM (approx)} \\
 \text{Tdev}_{\text{corrected}} &= 2.5 * (\text{Effort}_{\text{corrected}})^{0.38} \\
 &= 2.5 * (15)^{0.38} \\
 &= 7 \text{ months (approx)}
 \end{aligned}$$

After refining our estimates, it seems that seven months would likely be enough for completion of this project. This is still a rough estimate since we have not taken the underlying components of the software into consideration. Complete COCOMO model considers such parameters to give a more realistic estimate.

Task-1: Online Self Evaluation Exercise (http://vlabs.iitkgp.ernet.in/se/2/self_evaluation/):**1. According to the COCOMO model, a project can be categorized into**

- A. 3 Types
- B. 5 Types
- C. 6 Types
- D. No such categorization

2. In Intermediate COCOMO model, Effort Adjustment Factor (EAF) is derived from the effort multipliers by

- A. Adding items
- B. Multiplying items
- C. Taking their weighted average
- D. Considering their maximum

3. Project metrics are estimated during which phase?

- A. Feasibility study
- B. Planning
- C. Design
- D. Development

4. According to Halsetad's metrics, program length is given by the:

- A. Sum of total number of operators and operands
- B. Sum of number of unique operators and operands
- C. Total number of operators
- D. Total number of operands

5. Complete COCOMO considers a software as a



- A. Homogeneous system
- B. Heterogeneous system

6. Consider you are developing a web application, which would make use of a lot of web services provided by Facebook, Google, Flickr. Would it be wise to make estimates for this project using COCOMO?

- A. Yes, of course
- B. Not at all

Perform following lab Exercise online @ <http://vlabs.iitkgp.ernet.in/se/2/exercise/>:

Select 1, 2 and 3 exercise one by one from the drop menu given on website.

[Home](#) [Credits](#) [Feedback](#) [Advanced Network Technologies Virtual Lab](#) [Virtual Labs](#)

Estimation of Project Metrics

Introduction

Theory

Simulation

Case Study

Self-evaluation

Procedure

Exercises

References

Select Exercise #

Submit

Exercise-1:

Project Type	a	b	c
Organic	2.4	1.05	0.38
Project size (in KDSI)	<input style="width: 100%;" type="text"/>		
Effort (in PM)	<input style="width: 100%;" type="text"/>		
T _{dev} (in month)	<input style="width: 100%;" type="text"/>		
Effort Adjustment Factor (EAF)	<input style="width: 100%;" type="text"/>		
Effort _{corrected} (in PM)	<input style="width: 100%;" type="text"/>		
T _{dev} _{corrected} (in month)	<input style="width: 100%;" type="text"/>		
# of developers	<input style="width: 100%;" type="text"/>		
Calculate			

Exercise-2:

Identify the unique operators and operands from the following snippet of code:

```

01 int
02 main(int argc, char **argv)
03 {
04     int x = 10;
05     int y = 20;
06     int sum;
07
08     sum = x + y;
09
10     printf("Sum of %d and %d is: %d\n", x, y, sum);
11
12     return 0;
13 }
```

Operators

☐ int ☐ main ☐ argc ☐ char ☐ * ☐ argv

☐ {} ☐ {}

☐ x ☐ = ☐ 10 ☐ ; ☐ y ☐ 20 ☐ sum ☐ +

☐ printf ☐ "Sum of %d and %d is: %d" ☐ ,

☐ return ☐ 0

Operands

☐ int ☐ main ☐ argc ☐ char ☐ * ☐ argv

☐ {} ☐ {}

☐ x ☐ = ☐ 10 ☐ ; ☐ y ☐ 20 ☐ sum ☐ +

☐ printf ☐ "Sum of %d and %d is: %d" ☐ ,

☐ return ☐ 0

Check

Result

Exercise-3:

```

01 int
02 main(int argc, char **argv)
03 {
04     int radius = 12.34;
05
06     printf("Area of the circle with radius %f is: %f\n", radius, area(radius));
07
08     return 0;
09 }
10
11 float
12 area(float r) {
13     return 22 * r * r / 7;
14 }
```

Parameter	Value
Total # of operators	<input style="width: 100%;" type="text"/>
Total # of operands	<input style="width: 100%;" type="text"/>
Total # of unique operators	<input style="width: 100%;" type="text"/>
Total # of unique operands	<input style="width: 100%;" type="text"/>
Program length	<input style="width: 100%;" type="text"/>
Program vocabulary	<input style="width: 100%;" type="text"/>
Volume	<input style="width: 100%;" type="text"/>
Difficulty	<input style="width: 100%;" type="text"/>
Effort	<input style="width: 100%;" type="text"/>
Time to implement (in seconds)	<input style="width: 100%;" type="text"/>
Calculate	

Calculate

Result

Result

Practical-10

Aim: To prepare the test suite for a given case study on SE Vlab tool.

Case study: A Library Information System for SE VLabs Institute

The SE VLabs Institute has been recently setup to provide state-of-the-art research facilities in the field of Software Engineering. Apart from research scholars (students) and professors, it also includes quite a large number of employees who work on different projects undertaken by the institution. As the size and capacity of the institute is increasing with the time, it has been proposed to develop a Library Information System (LIS) for the benefit of students and employees of the institute. LIS will enable the members to borrow a book (or return it) with ease while sitting at his desk/chamber. The system also enables a member to extend the date of his borrowing if no other booking for that particular book has been made. For the library staff, this system aids them to easily handle day-to-day book transactions. The librarian, who has administrative privileges and complete control over the system, can enter a new record into the system when a new book has been purchased, or remove a record in case any book is taken off the shelf. Any non-member is free to use this system to browse/search books online. However, issuing or returning books is restricted to valid users (members) of LIS only.

The final deliverable would a web application (using the recent HTML-5), which should run only within the institute LAN. Although this reduces security risk of the software to a large extent, care should be taken no confidential information (e.g., passwords) is stored in plain text.

Test case preparation could begin right after requirements identification stage. It is desirable (and advisable) to create a Requirements Traceability Matrix (RTM) showing a mapping from individual requirement to test case(s). A simplified form of the RTM is shown in table 1 (the numbers shown in this table are arbitrary; not specific to LIS).

Requirement #	Test Case #
R1	TC1
R2	TC2, TC3, TC4
R3	TC5
R4	TC6

[Table 1: A simplified mapping from requirements to test cases]

Table 1 states which test case should help us to verify that a specified feature has been implemented and working correctly. For instance, if test case # TC6 fails, that would indicate requirement # R4 has not fully realized yet. Note that it is possible that a particular requirement might need multiple test cases to verify whether it has been implemented correctly.

Table 1 states which test case should help us to verify that a specified feature has been implemented and working correctly. For instance, if test case # TC6 fails, that would indicate

requirement # R4 has not fully realized yet. Note that it is possible that a particular requirement might need multiple test cases to verify whether it has been implemented correctly.

To be specific to our problem, let us see how we can design test cases to verify the "User Login" feature. The simplest scenario is when both username and password have been typed in correctly. The outcome will be that the user could then avail all features of LIS. However, there could be multiple unsuccessful conditions:

- (1) User ID is wrong
- (2) Password is wrong
- (3) User ID & password are wrong
- (4) Wrong password given twice consecutively
- (5) Wrong password given thrice consecutively
- (6) Wrong password given thrice consecutively, and security question answered *correctly*
- (7) Wrong password given thrice consecutively, and security question answered *incorrectly*

Task: Prepare table-2 A test suite to verify the user login feature.

#	Test Suite 1
Title	Verify 'user login' functionality
Description:	To test the different scenarios that might arise while a user is trying to login
Test Case 1 (Sample)	Summary: Verify that the user is already registered with the LIS is able to login with correct user ID and password.
	Dependency:
	Pre-condition: Employee ID 149405 is a registered user of LIS; user's password is this_is_password
	Post-condition: User is logged in
	Execution Steps: 1. Type in employee ID as 149405 2. Type in password this_is_password 3. Click on the 'Login' button
	Expected Output: "Home" page for the user is displayed
Test Case 2 (Sample)	Summary: Verify that an unregistered user of LIS is unable to login
	Dependency:
	Pre-condition: Employee ID 149405xx is not a registered user of LIS.
	Post-condition: User is not logged in.
	Execution Steps: 1. Type in employee ID as 149405xx 2. Type in password whatever 3. Click on the 'Login' button
	Expected Output: The "Login" dialog is shown with a "Login failed! Check your user ID and password" message
Test Case 3	Summary: Verify that user already registered with the LIS is unable to login with incorrect password
	Dependency:
	Pre-condition:

	Post-condition:	
	Execution Steps:	
	Expected Output:	
Test Case 4	Summary:	Verify that user already registered with the LIS is unable to login with incorrect password given twice consecutively
	Dependency:	TC3
	Pre-condition:	
	Post-condition:	
	Execution Steps:	
	Expected Output:	
Test Case 5	Summary:	Verify that user already registered with the LIS is unable to login with incorrect password given thrice consecutively
	Dependency:	TC4
	Pre-condition:	
	Post-condition:	
	Execution Steps:	

	Expected Output:	
Test Case 6	Summary:	Verify that a registered user can login after three consecutive failures by correctly answering the security question
	Dependency:	TC5
	Pre-condition:	
	Post-condition:	
	Execution Steps:	
	Expected Output:	
Test Case 7	Summary:	Verify that a registered user's account is blocked after three consecutive failures and answering the security question incorrectly
	Dependency:	
	Pre-condition:	
	Post-condition:	
	Execution Steps:	
	Expected Output:	