MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that Ms./Mr. ……...…… Reg. No. …..…… Section: …… Roll No: ……... has satisfactorily completed the lab exercises prescribed for Software Engineering Lab [CSE 3161] of Third Year B. Tech. (Computer Science and Engineering) Degree at MIT, Manipal, in the academic year 2024-2025.

Date: …….....

Signature
Faculty in Charge

CONTENTS

COURSE OUTCOMES:

CO1: To understand the tools used for software modelling.

CO2: To demonstrate Software Development Life Cycle Models, through Case Studies..

CO3: To understand different UML diagrams requiredfor software design

CO4: To extend the code from the UML diagrams for further programming.

CO5: To know various testing tools and the testing methods and approaches to test a software.

Evaluation plan

- Internal Assessment Marks: 60 Marks
- End semester assessment: 40 Marks
  - ✓ Duration: 2 hours
  - ✓ Total marks: Write up: 15 Marks
    Execution: 25 Marks

INSTRUCTIONS TO THE STUDENTS

Pre- Lab Session Instructions

1. Students should carry the Class notes, Lab Manual and the required stationary to every lab session
2. Be in time and follow the Instructions from Lab Instructors.
3. Must Sign in the log register provided.
4. Make sure to occupy the allotted seat and answer the attendance.
5. Adhere to the rules and maintain the decorum.

In- Lab Session Instructions

- Follow the instructions on the allotted exercises given in Lab Manual.
- Show the program and results to the instructors on completion of experiments.

- On receiving approval from the instructor, copy the program and results in the Lab record.
- Prescribed textbooks and class notes can be kept ready for reference if required.

General Instructions for the exercises in Lab

- The programs should meet the following criteria:
    - Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs.
    - Use meaningful names for variables and procedures.
- Copying from others is strictly prohibited and would invite severe penalty during evaluation.
- The exercises for each week are divided under three sets:
    - Lab exercises – to be completed during lab hours
    - Additional Exercises – to be completed outside the lab or in the lab to enhance the skill
- In case a student misses a lab class, he/ she must ensure that the experiment is completed at students' end or in a repetition class (if available) with the permission of the faculty concerned but credit will be given only to one day's experiment(s).
- Questions for lab tests and examination are not necessarily limited to the questions in the manual, but may involve some variations and / or combinations of the questions.

THE STUDENTS SHOULD NOT...

1. Bring mobile phones or any other electronic gadgets to the lab.
2. Go out of the lab without permission.

LAB NO: 1 **Date:**

SOFTWARE LIFE CYCLE MODELS -I

Objectives:

- To understand the basics of Waterfall Models and its extension
- Ability to design Life Cycle models for different case studies

Prerequisites:

- Knowledge of the basics of different life cycle models

I. INTRODUCTION:

A software development life cycle (SDLC) model (also called software life cycle model and software development process model) describes the different activities that need to be carried out for the software to evolve in its life cycle.

a. Classical Waterfall Model

The classical waterfall model divides the life cycle into a set of phases as shown in Figure 1.1. It can be easily observed from this figure that the diagrammatic representation of the classical waterfall model resembles a multi-level waterfall.

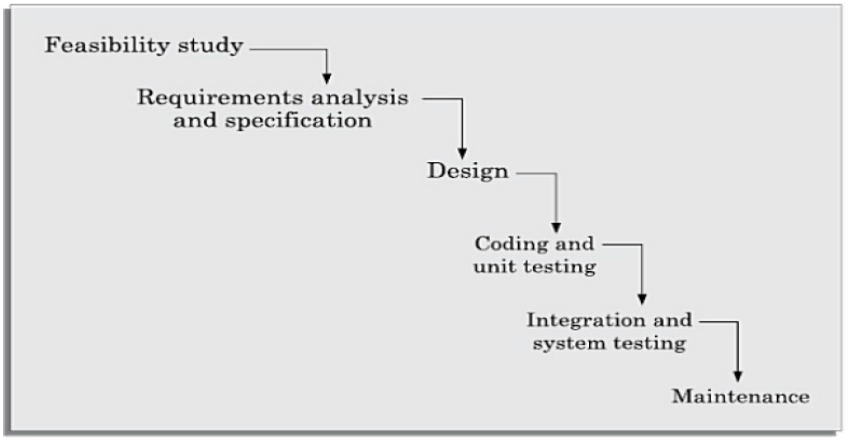


Fig 1.1 Classical Waterfall Model

b. Iterative Waterfall Model

The main change brought about by the iterative waterfall model to the classical waterfall model is in the form of providing feedback paths from every phase to its preceding phases.

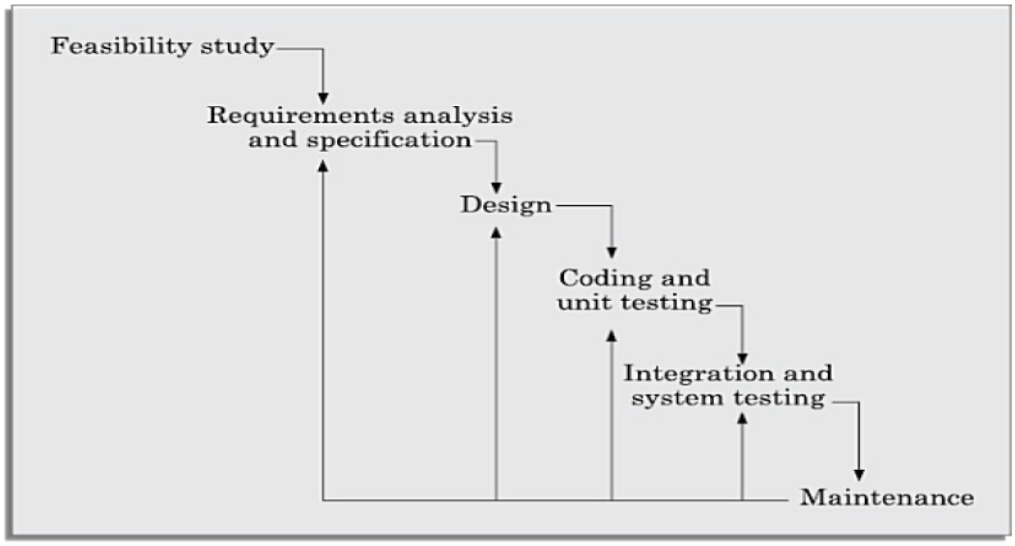


Fig 1.2 Iterative Waterfall Model

c. V Model

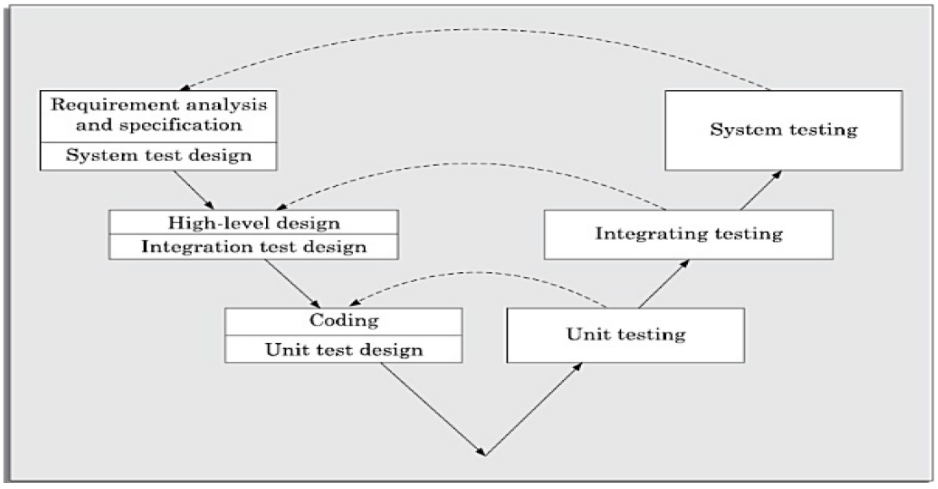


Fig 1.3 V Model

d. Prototyping Model

This model suggests building a working prototype of the system, before development of the actual software. A prototype is a toy and crude implementation of a system. It has limited functional capabilities, low reliability, or inefficient performance as compared to the actual software. A prototype can be built very quickly by using several shortcuts. The shortcuts usually involve developing inefficient, inaccurate, or dummy functions. The shortcut implementation of a function, for example, may produce the desired results by using a table look-up rather than by performing the actual computations.

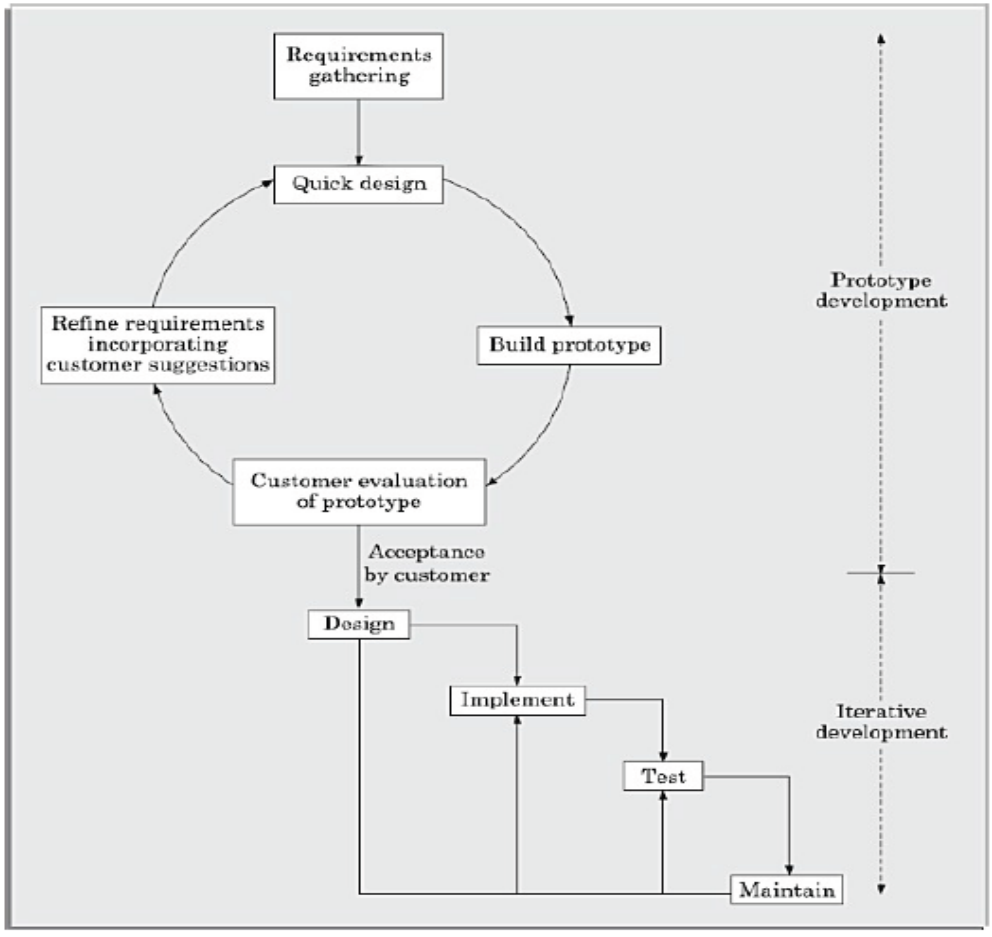


Fig 1.4 Prototype Model

e. Incremental Development Model

The incremental model has schematically been shown in Figure 1.5. After the requirements gathering and specification, the requirements are split into several versions. Starting with the core (version 1), in each successive increment, the next version is constructed using an iterative waterfall model of development and deployed at the customer site. After the last (shown as version n) has been developed and deployed at the client site, the full software is deployed.

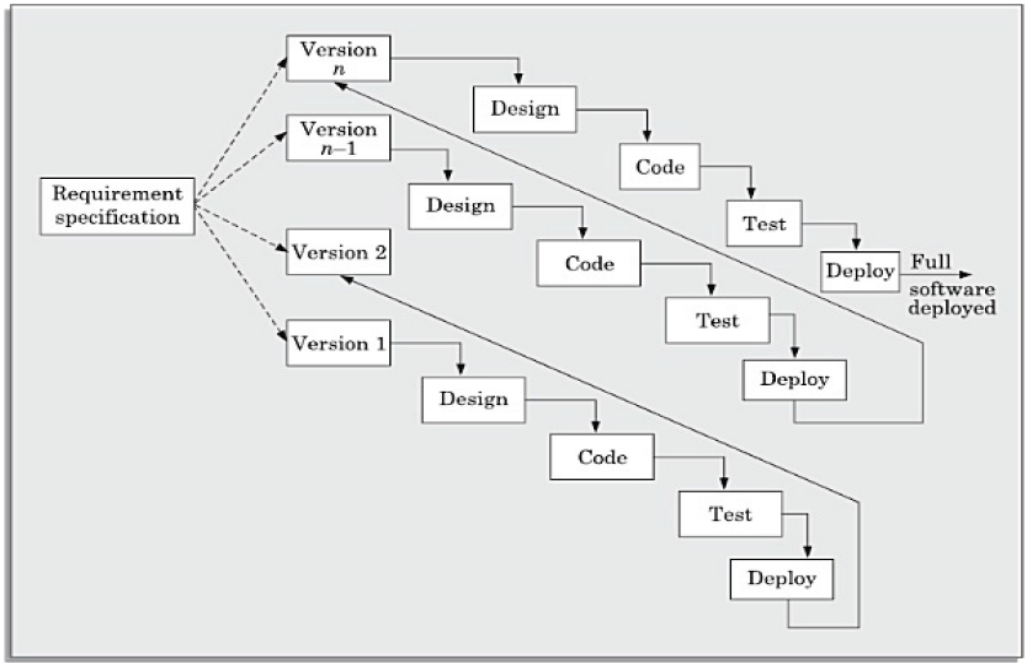


Fig 1.5 Incremental Development Model

f. Evolutionary Model

The principal idea behind the evolutionary life cycle model is conveyed by its name. In the incremental development model, complete requirements are first developed and the SRS document prepared. In contrast, in the evolutionary model, the requirements, plan, estimates, and solution evolve over the iterations, rather than fully defined and frozen in a major up-front specification effort before the development iterations begin. Such evolution is consistent with the pattern of

unpredictable feature discovery and feature changes that take place in new product development.

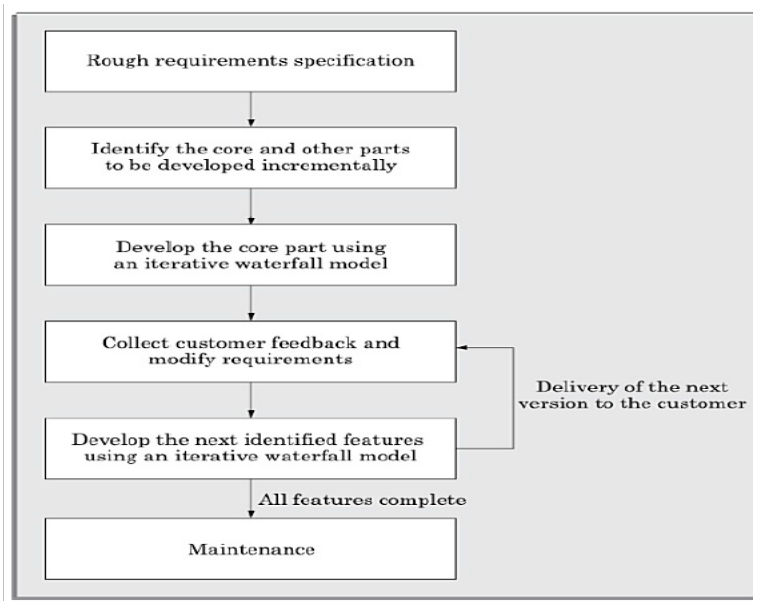


Fig 1.6 Evolutionary Model

II. LAB EXERCISES:

1. Design the waterfall model for the Hotel Automation Software [case study #2]
2. Design an iterative waterfall model for Time Management Software [case study # 1]
3. Design the V model for the Judiciary Information System [case study # 4]
4. Design a prototype model for the Road Repair and Tracking Software [case study #3]

III. ADDITIONAL EXERCISES:

1. Design the incremental model for Hotel Automation Software
2. Design an evolutionary model for Work Processing Software [case study #5]

LAB NO: 2 **Date:**

SOFTWARE LIFE CYCLE MODELS - II

Objectives:

- To understand the basics of RAD, AGILE and SPIRAL models
- Ability to design Life Cycle models for different case studies

Prerequisites:

- Knowledge of the basics of different life cycle models

1. INTRODUCTION

1.1 RAPID APPLICATION DEVELOPMENT (RAD)

This model has the features of both prototyping and evolutionary models. It deploys an evolutionary delivery model to obtain and incorporate the customer feedbacks on incrementally delivered versions. In this model prototypes are constructed, and incrementally the features. are developed and delivered to the customer. But unlike the prototyping model, the prototypes are not thrown away but are enhanced and used in the software construction

1.2 AGILE DEVELOPMENT MODEL

Agile model emphasises face-to-face communication over written documents. It is recommended that the development team size be deliberately kept small (5–9 people) to help the team members meaningfully engage in face-to-face communication and have collaborative work environment. It is implicit then that the agile model is suited to the development of small projects. However, if a large project is required to be developed using the agile model, it is likely that the collaborating teams might work at different locations. In this case, the different teams are needed to maintain as much daily contact as possible through video conferencing, telephone, email etc.

1.2.1 Extreme Programming Model

This model is based on a rather simple philosophy:" If something is known to be beneficial, why not put it to constant use?" Based on this principle, it puts forward several key practices that need to be practised to the extreme

Good practices that need to be practised to the extreme:

1. **Code review:** It is good since it helps detect and correct problems most efficiently. It suggests *pair programming* as the way to achieve continuous review.

2. **Testing:** Testing code helps to remove bugs and improves its reliability. XP suggests *test-driven development* (TDD) to continually write and execute test cases. In the TDD approach, test cases are written even before any code is written.

3. **Incremental development:** Incremental development is good, since it helps to get customer feedback, and extent of features delivered is a reliable indicator of progress. It suggests that the team should come up with new increments every few days.

4. **Simplicity:** Simplicity makes it easier to develop good quality code, as well as to test and debug it. Therefore, one should try to create the simplest code that makes the basic functionality being written to work

5. **Design:** Since having a good quality design is important to develop a good quality solution, everybody should design daily. This can be achieved through refactoring, whereby a working code is improved for efficiency and maintainability.

6. **Integration testing**: It is important since it helps identify the bugs at the interfaces of different functionalities. To this end, extreme programming suggests that the developers should achieve continuous integration, by building and performing integration testing several times a day.

1.2.2 Scrum Model

In the scrum model, a project is divided into small parts of work that can be incrementally developed and delivered over time boxes that are called sprints. The software therefore gets developed over a series of manageable chunks. In the scrum model, the team members assume three fundamental roles - software owner, scrum master, and team member. The software owner is responsible for communicating the customers vision of the software to the development team. The scrum master acts as a liaison between the software owner and the team, thereby facilitating the development work

1.3 Spiral Model

This model gets its name from the appearance of its diagrammatic representation that looks like a spiral with many loops (see Figure 1.3). The exact number of loops of the spiral is not fixed and can vary from project to project. The number of loops shown in Figure 1.3 is just an example. Each loop of the spiral is called a phase of the software process. The exact number of phases through which the product is developed can be varied by the project manager depending upon the project risks. A prominent feature of the spiral model is handling unforeseen risks that can show up much after the project has started. In this context, please recollect that the prototyping model can be used effectively only when the risks in a project can be identified upfront before the development work starts**.**

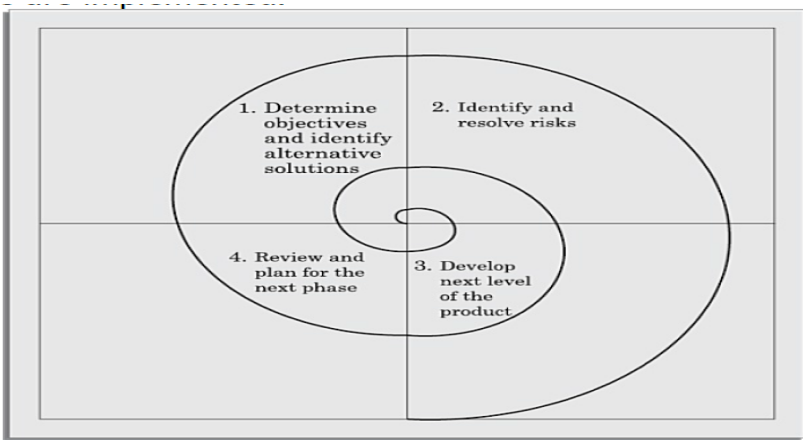


Fig 1.3 Spiral Model of Software Development

II. LAB EXERCISES:

1. Justify whether the RAD model can be applied for Transport Company Computerization Software. [case study # 7]. Discuss with key points
2. Justify whether Extreme Programming model can be applied for Supermarket Automation Software [case study #10]. Justify with key points.
3. Discuss whether the Scrum model can be applied for Graphics Editor Software [case study # 0]. Justify with key points.
4. Design the Spiral model for Judiciary Information System [case study # 4].

III. ADDITIONAL EXERCISES:

5. Discuss whether Extreme Programming model can be applied for Restaurant Automation System [case study #6]. Justify your answer with key points.
6. Design the spiral model for Bookshop Automation Software [case study # 0]

LAB NO: 3 **Date:**

SOFTWARE REQUIREMENTS SPECIFICATION

Objectives:

- Identify and state functional requirements
- Identify and state non-functional requirements
- Prepare SRS document for given project
- Identify ambiguities, inconsistencies and incompleteness from a requirements specification
- Prepare Axiomatic and Algebraic Specifications

Prerequisites:

- Knowledge of Requirements Gathering
- Knowledge of developing SRS
- Knowledge of preparing Axiomatic and Algebraic Specifications

I. INTRODUCTION:

Requirements:

Requirements are descriptions of the services that a software system must provide and the constraints under which it must operate. It is as a specification of what should be implemented. Requirements specify how the target system should behave. It specifies what to do, but not how to do. Requirements engineering refers to the process of understanding what a customer expects from the system to be developed, and to document them in a standard and easily readable and understandable format. This documentation will serve as reference for the subsequent design, implementation and verification of the system.
It is necessary and important that before we start planning, design and implementation of the software system for our client, we are clear about it's requirements. If we don't have a clear vision of what is to be developed and what all features are expected, there would be serious problems, and customer dissatisfaction as well.

- **Unambiguity or No Anomaly:** There should not be any ambiguity what a system to be developed should do. For example, in a Process Control Application, the following When the temperature becomes high, the heater should be switched off. Please note that words such as “high”, “low”, “good”, “bad” etc are indications of ambiguous requirements as these lack quantifications and can be subjectively interpreted. If the threshold above which the temperature can be considered to be high is not specified, then it can be interpreted differently by different developers.
- **Consistency:** Consider the following two requirements that were collected from two different stakeholders in a process control application development project.
    - The furnace should be switched-off when the temperature of the furnace rises above 500^0 C.
    - When the temperature of the furnace rises above 500^0 C, the water shower should be switched-on and the furnace should remain on.
- **Completeness:** A particular requirement for a system should specify what the system should do and also what it should not. Eg, If a student secures a *grade point average* (GPA) of less than 6, then the parents of the student must be intimated about the regrettable performance through a (postal) letter as well as through e-mail. However, on an examination of all requirements, it was found that there is no provision by which either the postal or e-mail address of the parents of the students can be entered into the system. The feature that would allow entering the e-mail ids and postal addresses of the parents of the students was missing, thereby making the requirements incomplete. Requirements should be complete.

Users of SRS Documents:

A number large number of different people need the SRS document for very different purposes. Some of the important categories of users of the SRS documents are:

1. Users, customers and marketing personnel
2. Software developers
3. Test engineers
4. User documentation writers
5. Project managers
6. Maintenance Engineers

Reasons for Developing an SRS Document

1. Forms an agreement between the customers and the developers
2. Reduces future reworks
3. Provides a basis for estimating costs and schedules
4. Provides a baseline for validation and verification
5. Facilitates future extension

Software Requirements Specification(SRS)

An SRS document should clearly document the following aspects of a software:

- Functional requirements
- Non-functional requirements
  - — Design and implementation constraints
  - — External interfaces required
  - — Other non-functional requirements
- Goals of implementation.

Functional requirements (FRs): The functional requirements capture the functionalities required by the users from the system. In the Office Work Software for CSE Department, If a student secures a *grade point average* (GPA) of less than 6, then the parents of the student must be intimated about the regrettable performance through a (postal) letter as well as through e-mail. However, on an examination of all requirements, it was found that there is no provision by which either the postal or e-mail address of the parents of the students can be entered into the system. The feature that would allow entering the e-mail ids and postal addresses of the parents of the students was missing, thereby making the requirements incomplete.

Non-functional requirements (NFRs): The non-functional requirements are non-negotiable obligations that must be supported by the software. The non-functional requirements capture those requirements of the customer that cannot be expressed as functions (i.e., accepting input data and producing output data). Non-functional requirements usually address aspects concerning external interfaces, user interfaces, maintainability, portability, usability, maximum number of concurrent users, timing, and throughput (transactions per second, etc.). The non-functional requirements can be critical in the sense that any failure by the developed software to achieve some minimum defined level in these requirements can be considered as a failure and make the software unacceptable by the customer.

The IEEE 830 standard recommend They are not directly related what functionalities are expected from the system. However, NFRs could typically define how the system should

behave under certain situations. For example, a NFR could say that the system should work with 128MB RAM. Under such condition, a NFR could be more critical than a FR. Non-functional requirements could be further classified into different types like:

- **Product requirements:** For example, a specification that the web application should use only plain HTML, and no frames
- **Performance requirements:** For example, the system should remain available 24x7
- **Organizational requirements:** The development process should comply to SEI CMM level 4

Functional Requirements:

Identifying Functional Requirements

Given a problem statement, the functional requirements could be identified by focusing on the following points:

- Identify the high level functional requirements simply from the conceptual understanding of the problem. For example, a Library Management System, apart from anything else, should be able to issue and return books.
- Identify the cases where an end user gets some meaningful work done by using the system. For example, in a digital library a user might use the "Search Book" functionality to obtain information about the books of his interest.
- If we consider the system as a black box, there would be some inputs to it, and some output in return. This black box defines the functionalities of the system. For example, to search for a book, user gives title of the book as input and get the book details and location as the output.
- Any high level requirement identified could have different sub-requirements. For example, "Issue Book" module could behave differently for different class of users, or for a particular user who has issued the book thrice consecutively.

Preparing Software Requirements Specifications(SRS):

The SRS should consist of the following:

1)Introduction

2)Overall description of organization of SRS document- It includes Product perspective, product features, user classes, operating environment, design & environment constraints and user documentation.

3)Functional Requirements for organization of SRS document- This section can classify the functionalities either based on the specific functionalities invoked by different users, or the functionalities that are available in different modes, etc., depending what may be appropriate.

1. User class 1
    (a) Functional requirement 1.1
    (b) Functional requirement 1.2
2. User class 2

(a) Functional requirement 2.1
(b) Functional requirement 2.2

4)External Interface Requirements-It includes User interfaces, hardware interfaces, software interfaces, communication interfaces.

5)Other non functional requirements for organization of SRS document: Other non functional requirements are performance requirements, safety requirements and Security requirements.

(Personal library software): It is proposed to develop a software that would be used by individuals to manage their personal collection of books. The following is an informal description of the requirements of this software as worked out by the marketing department.Develop the functional and non-functional requirements for the software.

A person can have up to a few hundreds of books. The details of all the books such as name of the book, year of publication, date of purchase, price, and publisher would be entered by the owner. A book should be assigned a unique serial number by the computer. This number would be written by the owner using a pen on the inside page of the book. Only a registered friend can be lent a book. While registering a friend, the following data would have to be supplied—name of the friend, his address, land line number, and mobile number. Whenever a book issue request is given, the name of the friend to whom the book is to be issued and the unique id of the book is entered. At this, the various books outstanding against the borrower along with the date borrowed are displayed for information of the owner. If the owner wishes to go ahead with the issue of the book, then the date of issue, the title of the book, and the unique identification number of the book are stored. When a friend returns a book, the date of return is stored and the book is removed from his borrowing list. Upon query, the software should display the name, address, and telephone numbers of each friend against whom books are outstanding along with the titles of the outstanding books and the date on which those were issued. The software should allow the owner to update the details of a friend such as his address, phone, telephone number, etc. It should be posssible for the owner to delete all the data pertaining to a friend who is no more active in using the library. The records should be stored using a free (public domain) data base management system. The software should run on both Windows and Unix machines.

Whenever the owner of the library software borrows a book from his friends, would enter the details regarding the title of the book, and the date borrowed and the friend from whom he borrowed it. Similarly, the return details of books would be entered. The software should be able to display all the books borrowed from various friends upon request by the owner.
It should be possible for any one to query about the availability of a particular book through a web browser from any location. The owner should be able to query the total number of books in the personal library, and the total amount he has invested in his library. It should also be possible for him to view the number of books borrowed and returned by any (or all) friend(s) over any specified time.

Example for SRS- <u>Personal Library System:</u>

1)Introduction:
Purpose or Goal- To Maintain a Personal Library Software
Project Scope- To develop a software that would be used by individuals to manage their personal collection of books. Since there are many functional requirements, the requirements have been organised into four sections: Manage own books, manage friends, manage borrowed books, and manage statistics. Now each section has less than 7 functional requirements, increasing the readability of the documents.
Environmental Characteristics-The records should be stored using a free(public domain) database management system. The software should run on both Windows and Unix machines. It should be possible for any one to query about the availability of a particular book through a web browser from any location.
2)Overall description of organization of SRS document-
Product perspective-The software is developed for an individual to manage his personal library.
Product features- The product features include Managing own books, manage friends, manage borrowed books, and manage statistics.
User classes- System Analyzer who provides the Problem Statement of the software, Individuals(owners) who lend books from their personal library & registered friends, who can borrow books from the personal library.
Operating environment- The software must run on both Windows & Unix operating systems. The library user can check the availability of a book through a web browser from any location.
Design & environment constraints- The records should be stored using a free(public domain) database management system
User documentation- The user manual depicts how an user can register, search & borrow books from the personal library & FAQ's on how he can efficiently use the system.

Functional requirements

The software needs to support three categories of functionalities as described below:

1. Manage own books

1.1 Register book

Description: To register a book in the personal library, the details of a book, such as name, year of publication, date of purchase, price and publisher are entered. This is stored in the database and a unique serial number is generated.

Input: Book details

Output: Unique serial number

R.1.2: Issue book

Description: A friend can be issued book only if he is registered. The various books outstanding against him along with the date borrowed are first displayed.

R.1.2.1: Display outstanding books

Description: First a friend's name and the serial number of the book to be issued are entered. Then the books outstanding against the friend should be displayed.

Input: Friend name

Output: List of outstanding books along with the date on which each was borrowed.

R.1.2.2: Confirm issue book

If the owner confirms, then the book should be issued to him and the relevant records should be updated.

Input: Owner confirmation for book issue. Output: Confirmation of book issue.

R.1.3: Query outstanding books

Description: Details of friends who have books outstanding against their name is displayed.

Input: User selection

Output: The display includes the name, address and telephone numbers of each friend against whom books are outstanding along with the titles of the outstanding books and the date on which those were issued.

R.1.4: Query book

Description: Any user should be able to query a particular book from anywhere using a web browser.

Input: Name of the book.

Output: Availability of the book and whether the book is issued out.

R.1.5: Return book

Description: Upon return of a book by a friend, the date of return is stored and the book is removed from the borrowing list of the concerned friend.

Input: Name of the book.

Output: Confirmation message.

2. Manage friend details

R.2.1: Register friend

Description: A friend must be registered before he can be issued books. After the registration data is entered correctly, the data should be stored and a confirmation message should be displayed.

Input: Friend details including name of the friend, address, land line number and mobile number.

Output: Confirmation of registration status.

R.2.2: Update friend details

Description: When a friend's registration information changes, the same must be updated in the computer.

R.2.2.1: Display current details

Input: Friend name.

Output: Currently stored details.

R2.2.2: Update friend details

Input: Changes needed.

Output: Updated details with confirmation of the changes.

R.3.3: Delete a friend record

Description: Delete records of inactive members.

Input: Friend name.

Output: Confirmation message.

3. Manage borrowed books

R.3.1: Register borrowed books

Description: The books borrowed by the user of the personal library are registered.

Input: Title of the book and the date borrowed.

Output: Confirmation of the registration status.

R.3.2: Deregister borrowed books

Description: A borrowed book is deregistered when it is returned.

Input: Book name.

Output: Confirmation of deregistration.

R.3.3: Display borrowed books

Description: The data about the books borrowed by the owner are displayed.

Input: User selection.

Output: List of books borrowed from other friends.

4. Manage statistics

R.4.1: Display book count

Description: The total number of books in the personal library should be displayed.

Input: User selection.
Output: Count of books.

R4.2: Display amount invested

Description: The total amount invested in the personal library is displayed.
Input: User selection.
Output: Total amount invested.

R.4.2: Display number of transactions Description: The total numbers of books issued and returned over a specific period by one (or all) friend(s) is displayed.
Input: Start of period and end of period.
Output: Total number of books issued and total number of books returned.

External interface requirements

User interfaces- The GUI consists of screen image of the home page. The home page consists of tabs to manage own books, manage friend details by the owner, manage borrowed books and statistics.

Hardware interfaces- A web browser that can run on both Windows and Unix operating systems. The TCP/IP protocol to be used for communication with the personal library software.

Software interfaces- A database management that stores the records using a free(Public domain) should be used. The operating system can be Windows or Unix systems.

Non-functional requirements

N.1: Database: A database management system that is available free of cost in the public domain should be used.

N.2: Platform: Both Windows and Unix versions of the software need to be developed.

N.3: Web-support: It should be possible to invoke the query book functionality from any place by using a web browser.

Axiomatic Specification:

In axiomatic specification of a system, first-order logic is used to write the pre- and post-conditions to specify the operations of the system in the form of axioms. The pre-conditions basically capture the conditions that must be satisfied before an operation can successfully be invoked. In essence, the pre-conditions capture the requirements on the input parameters of a function. The post-conditions are the conditions that must be satisfied when a function post-conditions are essentially constraints on the results produced for the function execution to be considered successful.

The following are the sequence of steps that can be followed to systematically develop the axiomatic specifications of a function:

Establish the range of input values over which the function should behave correctly. Establish the constraints on the input parameters as a predicate. Specify a predicate defining

the condition which must hold on the output of the function if it behaved properly. Establish the changes made to the function's input parameters after execution of the function. Pure mathematical functions do not change their input and therefore this type assertion is not necessary for pure functions. Combine all of the above into pre- and post-conditions of the function.

We now illustrate how simple abstract data types can be algebraically specified through two simple examples.

Example 1 Specify the pre- and post-conditions of a function that takes a real number as argument and returns half the input value if the input is less than or equal to 100, or else returns double the value.

f (x : real) : real

pre : $x \in R$

post : $\{(x \leq 100) \wedge (f(x) = x/2)\} \vee \{(x > 100) \wedge (f(x) = 2 * x)\}$

Example 4.2 Axiomatically specify a function named search which takes an integer array and an integer key value as its arguments and returns the index in the array where the key value is present.

search(X : intArray, key : integer) : integer

pre : $\exists i \in$ [X f irst...X last], X [i] = key

post : $\{(X^I$ [search(X, key)] = key) $\wedge$ (X = X□)$\}$

Please note that we have followed the convention that if a function changes any of its input parameters, and if that parameter is named X , then we refer to it after the function completes execution as X^I. One practical application of the axiomatic specification is in program documentation. Engineers developing code for a function specify the function by noting down the pre and post conditions of the function in the function header. Another application of the axiomatic specifications is in proving program properties by composing the pre and post-conditions of a number of functions.

Algebraic Specification:

In the algebraic specification technique, an object class or type is specified in terms of relationships existing between the operations defined on that type. Various notations of algebraic specifications have evolved, including those based on OBJ and Larch languages. Essentially, algebraic specifications define a system as a *heterogeneous algebra*. A heterogeneous algebra is a collection of different sets on which several operations are defined. Traditional algebras are homogeneous. A homogeneous algebra consists of a single set and several operations defined in this set; e.g. { I, +, -, *, / }. In contrast, alphabetic strings S together with operations of concatenation and length {S, I , con, len}, is not a homogeneous algebra, since the range of the length operation is the set of integers.

An algebraic specification is usually presented in four sections.

Types section: In this section, the sorts (or the data types) being used is specified.
Exception section: This section gives the names of the exceptional conditions that might occur when different operations are carried out. These exception conditions are used in the later sections of an algebraic specification.
Syntax section: This section defines the signatures of the interface procedures. The collection of sets that form input domain of an operator and the sort where the output is produced are called the *signature* of the operator. For example, PUSH takes a stack and an element as its input and returns a new stack that has been created.
Equations section: This section gives a set of *rewrite rules* (or equations) defining the meaning of the interface procedures in terms of each other. In general, this section is allowed to contain conditional expressions.
After having identified the required operators, it is helpful to classify them as either basic constructor operators, extra constructor operators, basic inspector operators, or extra inspection operators. The definition of these categories of operators is as follows:
Basic construction operators: These operators are used to create or modify entities of a type. The basic construction operators are essential to generate all possible element of the type being specified. For example, 'create' and 'append' are basic construction operators in Example 4.13.
Extra construction operators: These are the construction operators other than the basic construction operators. For example, the operator 'remove' in
Example 4.13 is an extra construction operator, because e ven without using 'remove' it is possible to generate all values of the type being specified.
Basic inspection operators: These operators evaluate attributes of a type without modifying them, e.g., eval, get, etc. Let S be the set of operators whose range is not the data type being specified—these are the inspection operators. The set of the basic operators S1 is a subset of S , such that each operator from S -S 1 can be expressed in terms of the operators from S 1.
Extra inspection operators: These are the inspection operators that are not basic inspectors.
A simple way to determine whether an operator is a constructor (basic or extra) or an inspector (basic or extra) is to check the syntax expression for the operator. If the type being specified appears on the right hand side of the expression then it is a constructor, otherwise it is an inspection operator. For example, in Example 4.13, create is a constructor because point appears on the right hand side of the expression and point is the data type being specified. But, xcoord is an inspection operator since it does not modify the point type.
A good rule of thumb while writing an algebraic specification, is to first establish which are the constructor (basic and extra) and inspection operators (basic and extra). Then write down an axiom for composition of each basic construction operator over each basic inspection operator and extra constructor operator. Also, write down an axiom for each of the extra

inspector in terms of any of the basic inspectors. Thus, if there are m1 basic constrctors, m2 extra constructors, n1 basic inspectors, and n2 extra inspectors, we should have m1 × (m2 + n1) + n2 axioms. However, it should be clearly noted that these m1 × (m2 + n1) + n2 axioms are the minimum required and many more axioms may be needed to make the specification
complete. Using a complete set of rewrite rules, it is possible to simplify an arbitrary sequence of operations on the interface procedures.
While developing the rewrite rules, different persons can come up with different sets of equations. However, while dev eloping the equations one has to be careful that the equations should be able to handle all meaningful composition of operators, and they should have the unique termination and finite termination properties. These two properties of the rewrite rules are discussed later in this section.
Example 4.13 Let us specify a data type point supporting the operations create, xcoord, ycoord, isequal; where the operations have their usual meaning.
Types:
defines point
uses boolean, integer
Syntax:
1. create : integer × integer → point
2. xcoord : point → integer
3. ycoord : point → integer
4. isequal : point × point → boolean
Equations:
1. xcoord(create(x, y)) = x
2. ycoord(create(x, y)) = y
3. isequal(create(x1, y1), create(x2, y2)) = ((x1 = x2)and(y1 = y2))
In this example, we have only one basic constructor (create), and three basic inspectors (xcoord, ycoord, and isequal). Therefore, we have only 3 equations.
The rewrite rules let you determine the meaning of any sequence of calls on the point type. Consider the following expression:
isequal (create (xcoord (create(2, 3)), 5),create (ycoord (create(2, 3)), 5)).
By applying the rewrite rule 1, you can simplify the given expression as
isequal (create (2, 5), create (ycoord(create(2, 3)), 5)).
By using rewrite rule 2, you can further simplify this as
isequal (create (2, 5),((create (3, 5)).
This is false by rewrite rule 3.

II. LAB EXERCISES:

1)Develop Software Requirements Specification for any two of the following Systems:

a) Passport automation System
b) Online Exam Registration
c) Stock Maintenance System
d) Online course reservation system
e) E-ticketing
f) Credit Card Processing

2) What do you understand by pre- and post-conditions of a function?
Write the pre- and post-conditions to axiomatically specify the function:
A function takes two floating point numbers representing the sides of a rectangle as input and returns the area of the corresponding rectangle as output.

3) Using the algebraic specification method, formally specify a **string** supporting the following operations:

- **append:** append a given string to another string
- **add:** add a character to a string
- **create:** create a new null string
- **isequal:** checks whether two strings are equal or not
- **isempty:** checks whether the string is null

III. ADDITIONAL EXERCISES:

4) Write the pre- and post-conditions to axiomatically specify the following functions:

(a)A function accepts three integers in the range of -100 and +100 and determines the largest of the three integers.

(b) A function takes an array of integers as input and finds the minimum value.

(c) A function named square-array creates a 10 element array where each all elements of the array, the value of any array element is square of its index.

(d) A function sort takes an integer array as its argument and sorts the input array in ascending order.

5) Using the algebraic specification method, formally specify an **array** of generic type elem. Assume that array supports the following operations:

· **create:** takes the array bounds as parameters and initializes the values of the array to undefined.

· **eval:** reveals the value of a specified element.

· **first:** returns the first bound of the array.

· **last:** returns the last bound of the array.

LAB NO: 4 and 5 **Date:**

FUCNTION ORIENTED AND STRUCTURED DESIGN

Objectives:

- To understand the basic principles of function-oriented design.
- To learn how to create context diagrams.
- To gain proficiency in designing data flow diagrams (DFD).
- To comprehend the construction and interpretation of structured charts.

Prerequisites:

- Creation of Software Requirement Specification
- Basic knowledge of Software Design

I. INTRODUCTION

Function-oriented design is a systematic approach to software development where the focus is on functions or procedures as the primary units of software structure. This methodology is rooted in the concept of breaking down a system into smaller, more manageable parts, each of which performs a specific function or task. This approach is particularly beneficial in creating systems that are modular, maintainable, and scalable.

Key Concepts:

1. **Modularity**: The system is divided into discrete modules, each responsible for a particular functionality.
2. **Hierarchy**: Functions are organized in a hierarchical manner, with higher-level functions calling lower-level functions.

3. **Decomposition**: The process of breaking down a complex system into simpler, smaller parts.
4. **Abstraction**: Function-oriented design uses abstraction to hide the implementation details of functions.

Approaches to Function-Oriented Design:

1. **Structured Analysis and Design (SA/SD)**: This approach involves a series of steps to analyse and design a system from a functional perspective. It typically starts with the creation of a context diagram to identify the system boundaries and interactions with external entities. The next step is to develop data flow diagrams (DFD) to represent the flow of data through the system. Finally, structured charts are used to depict the hierarchical organization of functions within the system.
2. **Top-Down Design**: In top-down design, the system is designed by starting from the highest level of abstraction and progressively breaking down into more detailed levels. The process begins with identifying the main function of the system and then decomposing it into sub-functions. Each sub-function is further broken down until the lowest level of detail is reached.
3. **Functional Decomposition**: Functional decomposition involves breaking down a complex function into simpler sub-functions. Each sub-function is designed to perform a specific task, and these tasks are then integrated to form the complete system. This approach helps in managing complexity by focusing on smaller, manageable functions.

Structured Analysis/Structured Design (SA/SD)

- Used to perform the high-level design of a software.
- SA/SD methodology involves carrying out two distinct activities: Structured analysis (SA) Structured design (SD)

- During structured analysis, the SRS document is transformed into a data flow diagram (DFD) model.
- During structured design, the DFD model is transformed into a structure chart.
- The purpose of Structured Analysis is to capture the detailed structure of the system as perceived by the user, whereas the purpose of Structured Design is to define the structure of the solution that is suitable for implementation in some programming language.

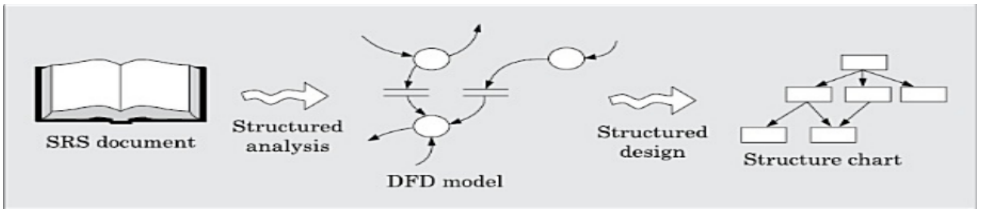


Fig 4.1: Structured analysis and structured design methodology

Data Flow Diagram (DFD)

The DFD (also known as the bubble chart) is a simple graphical formalism that can be used to represent a system in terms of the input data to the system, various processing carried out on those data, and the output data generated by the system. It maps out the input, processing, and output of data in a system, providing a clear picture of how data moves through different processes and data stores.

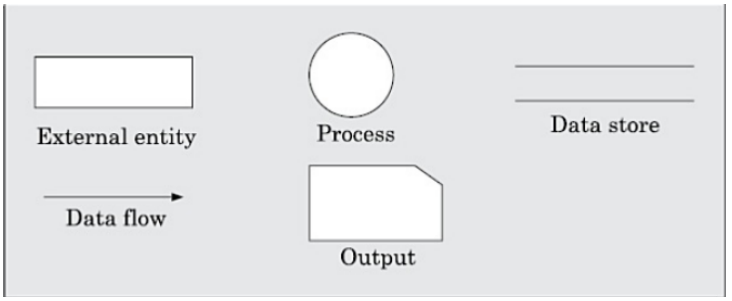


Fig 4.2: Symbols for constructing DFD

Levels of DFDs

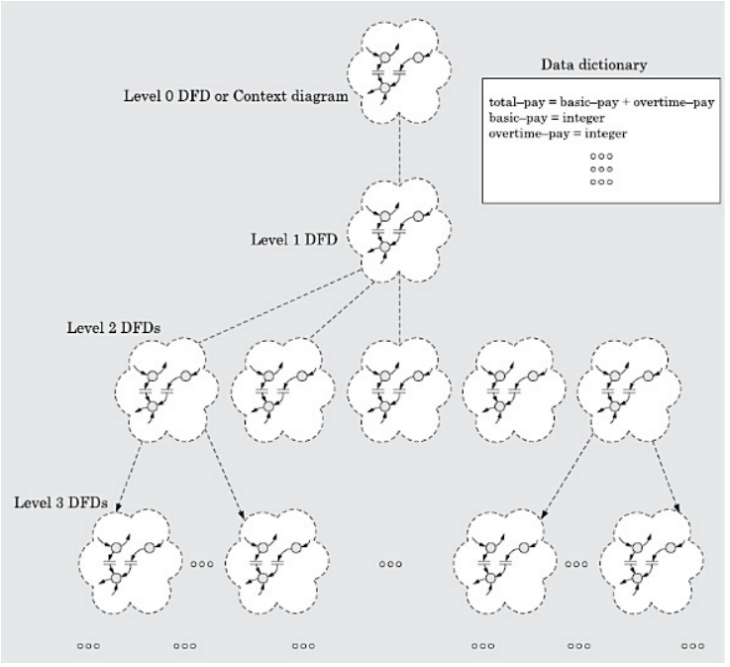


Fig 4.3: Levels of DFD

1. **Context Diagram (Level 0 DFD)**:
   - The highest level in a DFD, representing the entire system as a single process.
   - Shows the system's interactions with external entities and the internal process details are not shown.
   - The context diagram establishes the context in which the system operates; that is, who are the users, what data do they input to the system, and what data they received by the system.

2. **Level 1 DFD**:
   - Breaks down the main process from the context diagram into sub-processes.
   - Provides more detail on the data flows and data stores within the system.

3. **Level 2 and Lower-Level DFDs**:
    - Further decomposes the sub-processes from Level 1 into more detailed processes.
    - Each subsequent level provides more granular details of the system's operations.

Data Dictionary

A data dictionary is an essential component in the context of a Data Flow Diagram (DFD). It serves as a centralized repository of information about the data elements, data structures, and data flows within a system. The data dictionary provides detailed descriptions of all the data that is used or produced by the system, ensuring consistency, clarity, and understanding among all stakeholders involved in the system's development and maintenance.

Points to be noted while drawing DFD

- Context diagram should depict the system as a single bubble.
- 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.
- 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.
- DFD should represent only data flow, and it does not represent any control information.

Structured Design

The aim of structured design is to transform the results of the structured analysis (that i s,

the DFD model) into a structure chart. A structure chart represents the software architecture. The various modules making up the system, the module dependency (i.e. which module calls which other modules), and the parameters that are passed among the different modules. The structure chart representation can be easily implemented using some programming language. The basic building blocks using which structure charts are designed are as following:

- *Rectangular boxes*: A rectangular box represent a module.
- *Module invocation arrows*: An arrow connecting two modules implies that during program execution control is passed from one module to the other in the direction of the connecting arrow.
- *Data flow arrows*: These are small arrows appearing alongside the module invocation arrows. The data flow arrows are annotated with the corresponding data name.
- *Library modules*: A library module is usually represented by a rectangle with double edges. Libraries comprise the frequently called modules. Usually, when a module is invoked by many other modules, it is made into a library module.
- *Selection:* The diamond symbol represents the fact that one module of several modules connected with the diamond symbol is invoked depending on the outcome of the condition attached with the diamond symbol.
- *Repetition*: A loop around the control flow arrows denotes that the respective modules are invoked repeatedly.

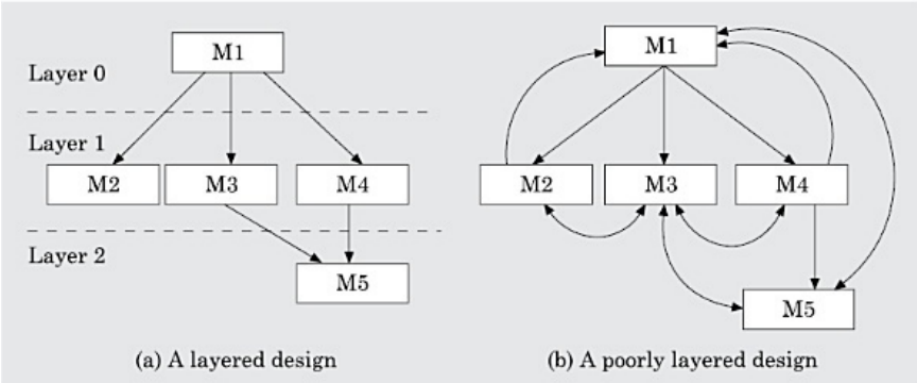


Fig 4.4: Examples of properly and poorly layered designs.

Transformation of a DFD Model into Structure Chart

Systematic techniques are available to transform the DFD representation of a problem into a module structure represented by as a structure chart. Structured design provides two strategies to guide transformation of a DFD into a structure chart:

- Transform analysis
- Transaction analysis

It starts with the level 1 DFD, transforming it into module representation using either the transform or transaction analysis and then proceed toward the lower level DFDs. At each level of transformation, it is important to first determine whether the transform or the transaction analysis is applicable to a particular DFD.

This is done by examining the data input to the diagram. The data input to the diagram can be easily spotted because they are represented by dangling arrows. If all the data flow into the diagram are processed in similar ways (i.e. if all the input data flow arrows are incident on the same bubble in the DFD) then transform analysis is applicable. Otherwise, transaction analysis is applicable. Normally, transform analysis is applicable only to very simple processing.

Transform analysis

Transform analysis identifies the primary functional components (modules) and the input and output data for these components. The first step in transform analysis is to divide the DFD into three types of parts:

- Input.
- Processing.
- Output

Transaction analysis

Transaction analysis is an alternative to transform analysis and is useful while designing transaction processing programs. As in transform analysis, first all data entering into the DFD need to be identified. A simple way to identify a transaction is the following.

- Check the input data.
- The number of bubbles on which the input data to the DFD are incident defines the number of transactions.
- However, some transactions may not require any input data. These transactions can be identified based on the experience gained from solving a large number of examples

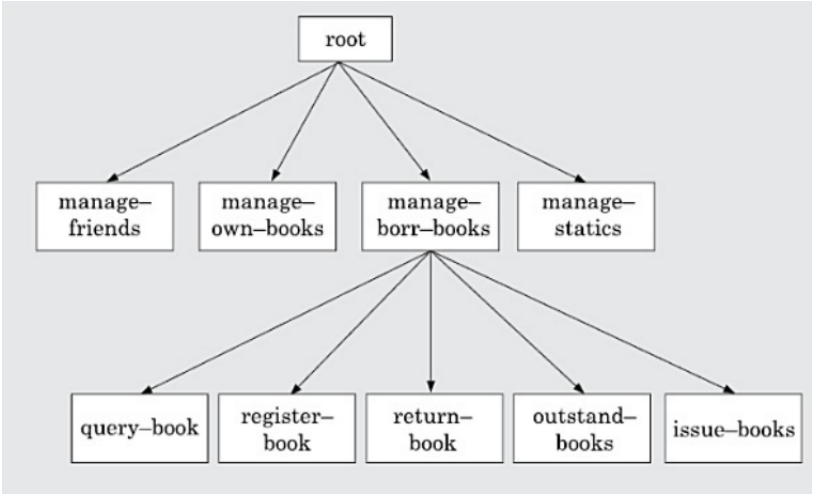


Fig 4.5: Structured chart example for Library Management Software

1.2 SOLVED EXERCISE:

Supermarket Prize Scheme: A super market needs to develop a software that would help it to automate a scheme that it plans to introduce to encourage regular customers. In this scheme, a customer would have first register by supplying his/her residence address, telephone number, and the driving license number. Each customer who registers for this scheme is assigned a unique customer number (CN) by the computer. A customer can present his CN to the check-out staff when he makes any purchase. In this case, the value of his purchase is credited against his CN. At the end of each year, the supermarket intends to award surprise gifts to 10 customers who make the highest total purchase over the year. Also, it intends to award a 22-caret gold coin to every customer whose purchase exceeded Rs. 10,000. The entries against the CN are reset on the last day of every year after the prize winners' lists are generated.

For the given problem scenario, Draw the context diagram and design level 1 and level 2 DFDs. Also give the data dictionary for the same.

Solution:

Level 0 DFD / Context diagram:

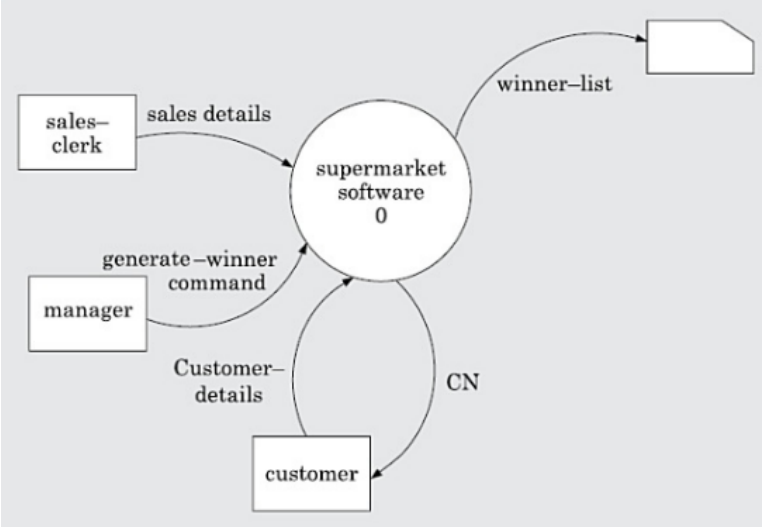


Fig 4.6: Level 0 DFD

Level 1 DFD:

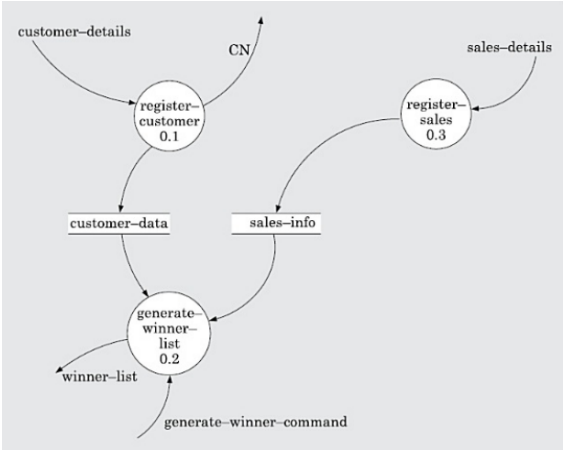


Fig 4.7: Level 1 DFD

Level 2 DFD:

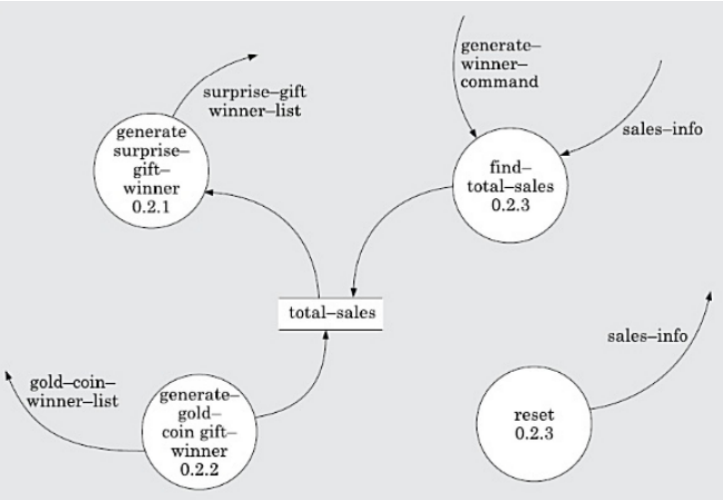


Fig 4.8: Level 2 DFD

Data dictionary for the DFD model

address: name+house#+street#+city+pin

sales-details: {item+amount}* + CN

CN: integer

customer-data: {address+CN}* sales-info: {sales-details}*

winner-list: surprise-gift-winner-list + gold-coin-winner-list

surprise-gift-winner-list: {address+CN}*

gold-coin-winner-list: {address+CN}*

gen-winner-command: command total-sales: {CN+integer}*

II. LAB EXERCISES

For the assigned case study,

1. Draw a context diagram by identifying the major external entities and the primary

interactions with the system.

2. Create a DFD and clearly show the main processes, data stores, and data flows involved.
3. Write a Data Dictionary.
4. Draw a structured chart.

LAB NO: 6 **Date:**

OBJECT ORIENTED DESIGN -I

Objectives:

- To understand the basics of Object Oriented Design
- To Draw Activity Diagram & Use Case Diagram

Prerequisites:

- Life Cycle Models & Function Oriented Design

Domain Modelling:

Domain modeling in Object-Oriented Analysis and Design (OOAD) is the process of systematically identifying, analyzing, and representing the essential concepts, behaviors, and relationships within a specific problem domain.

It involves translating real-world domain knowledge into software artifacts, such as classes, attributes, methods, and associations, to create a conceptual framework that accurately reflects the structure and dynamics of the domain.

1. Entities

Entities stand for the basic concepts or objects that cover the issue area. In most cases of word history, they are nouns distinguished by the attributes they have. For instance, in a banking space, the entities could relate to Customer, Account, Transaction, etc.

2. Relationships

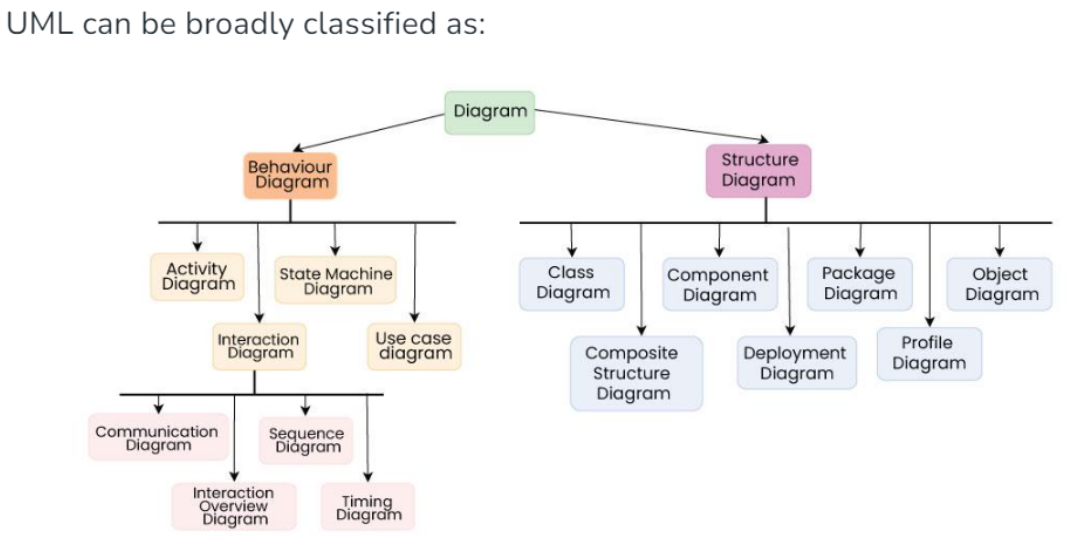
Relationships imply the connections and interactions of persons, things, places, and events. They determine which thing such as other people are subject to or with who they interact. Different types of relationships generally fall into one-to-one, one-to-many, or many-to-many categories.

3. Attributes

Attributes are the characterization or description of entities. They give long descriptions outlining objects and positions, and these are simply shown as data fields. Likewise, an instance of Customer entity class may include properties such as the name, address, phone number, and so on.

Unified Modeling Language (UML)

Unified Modeling Language (UML) is a standardized visual modeling language used in the field of software engineering to provide a general-purpose, developmental, and intuitive way to visualize the design of a system. UML helps in specifying, visualizing, constructing, and documenting the artifacts of software systems.

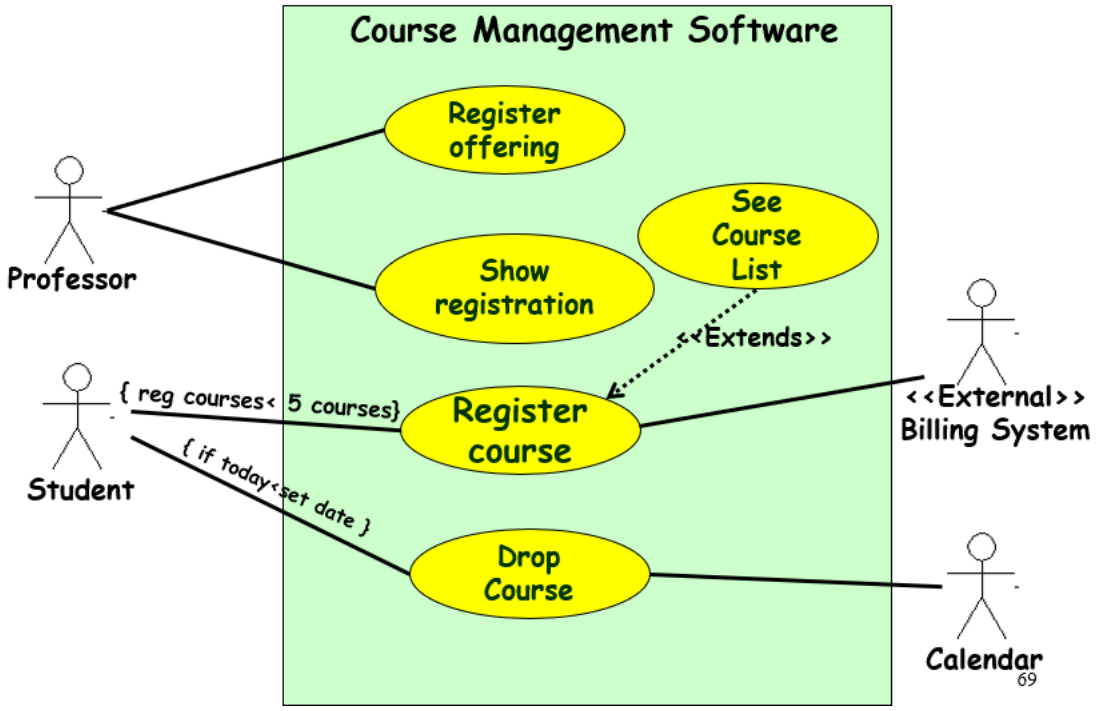


Techniques for Domain Modeling

1. Use Case Diagrams

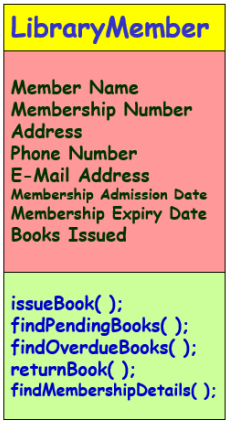
Use case diagrams are the way to illustrate functional requirements of a system by drawing the interactions of the actors (users or external systems) and the system itself. These use cases display the utilization and interconnection of various operations within the system, in order to establish its full function.

Example of use case diagram for course management software:



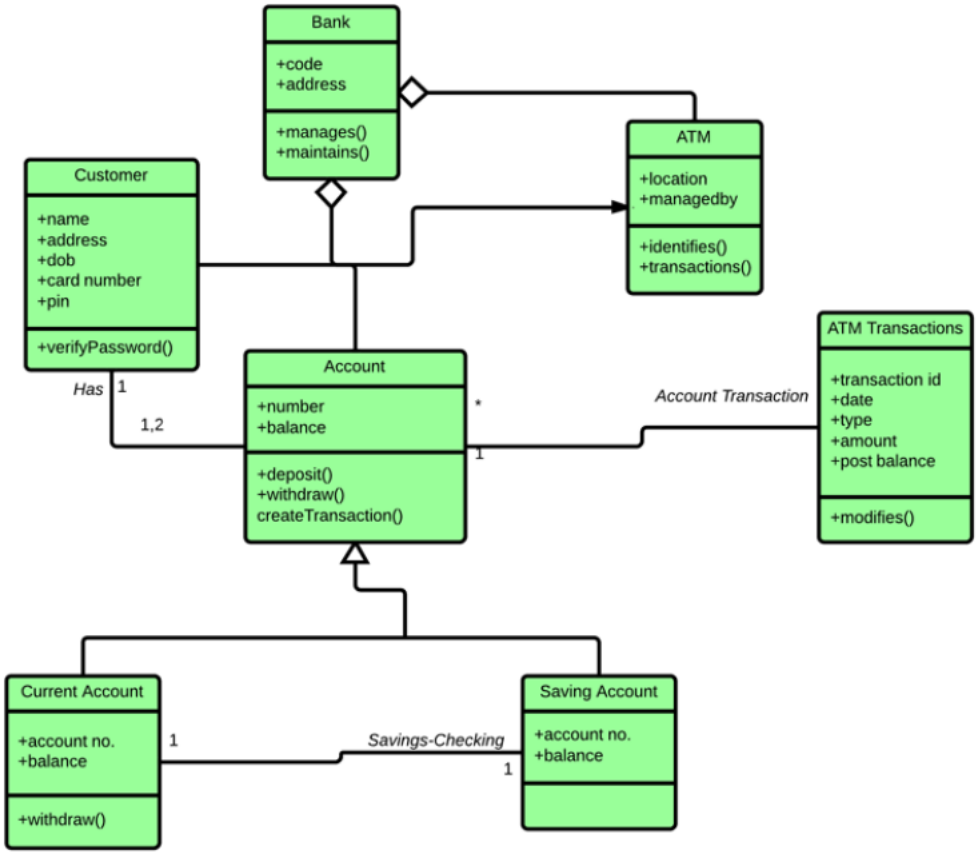
2. Class Diagrams

Class diagrams symbolize the "static structure" of the system by presenting a class, attributes, methods, and their relations. They are the reference hub for the data flow and behaviour in the system, including inheritance, connections, and complexity.



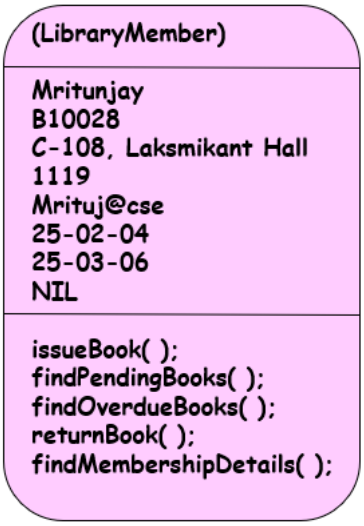
ATM System:

ATMs are deceptively simple: although customers only need to press a few buttons to receive cash, there are many layers of security that a safe and effective ATM must pass through to prevent fraud and provide value for banking customers. The various human and inanimate parts of an ATM system are illustrated by this easy-to-read diagram—every class has its title, and the attributes are listed beneath.



3. Object Diagrams

Object diagrams are a visual representation that illustrates the instances of classes and their relationships within a system at a specific point in time. They display objects, their attributes, and the links between them, providing a snapshot of the system's structure during execution.



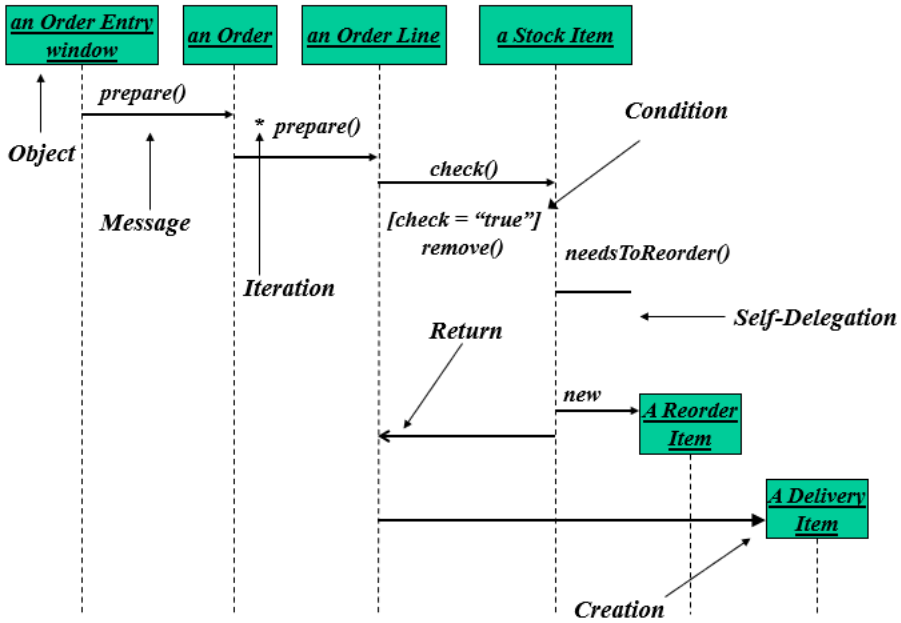
4. Interaction Diagrams

The interaction diagrams, these are sequence diagrams and collaboration diagrams, they are used for indicating how the dynamic behaviour of system works. They illustrate the way in which objects unite with each other and act throughout time to accomplish a certain job or situation.

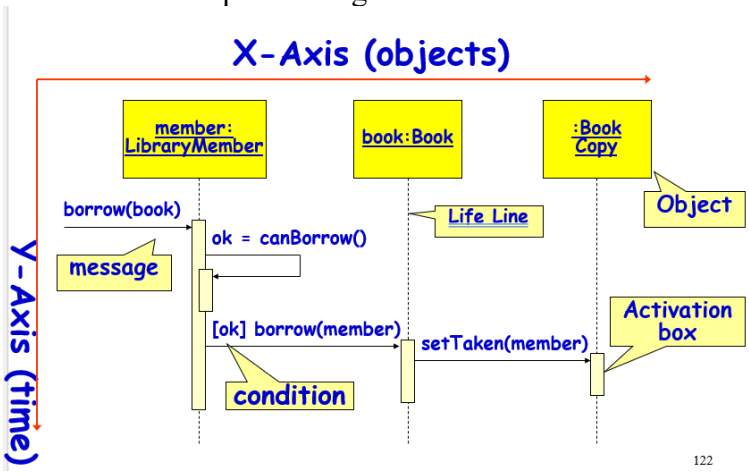
Sequence Diagrams:

Captures how objects interact with each other. Shows interaction among objects as a two-dimensional chart. Objects are shown as boxes at top. If object created during execution, then shown at appropriate place in time line. Object existence is shown as dashed lines (lifeline) and object activeness, shown as a rectangle on lifeline.

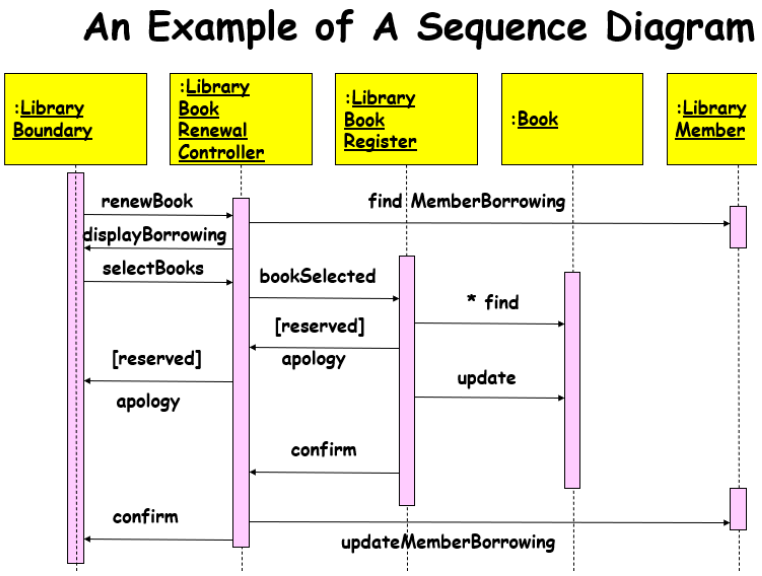
Messages are shown as arrows. Each message labelled with corresponding message name. Each message can be labelled with some control information.



Elements of a sequence diagram:



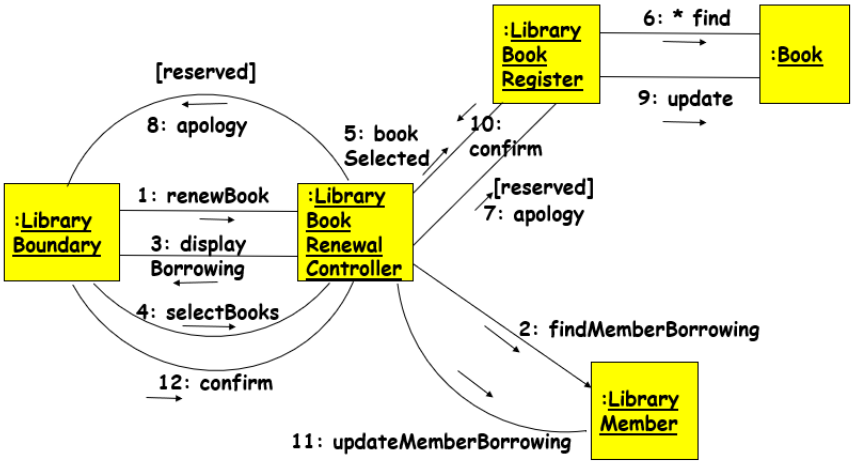
Example of a sequence diagram for the renew book case:



Collaboration Diagram:

Shows both structural and behavioural aspects. Objects are collaborator, shown as boxes along with links among them indicating association. Messages between objects shown as labelled arrow placed near links. Messages are prefixed with sequence numbers to show relative sequencing.

Example Collaboration Diagram for the renew book use case:

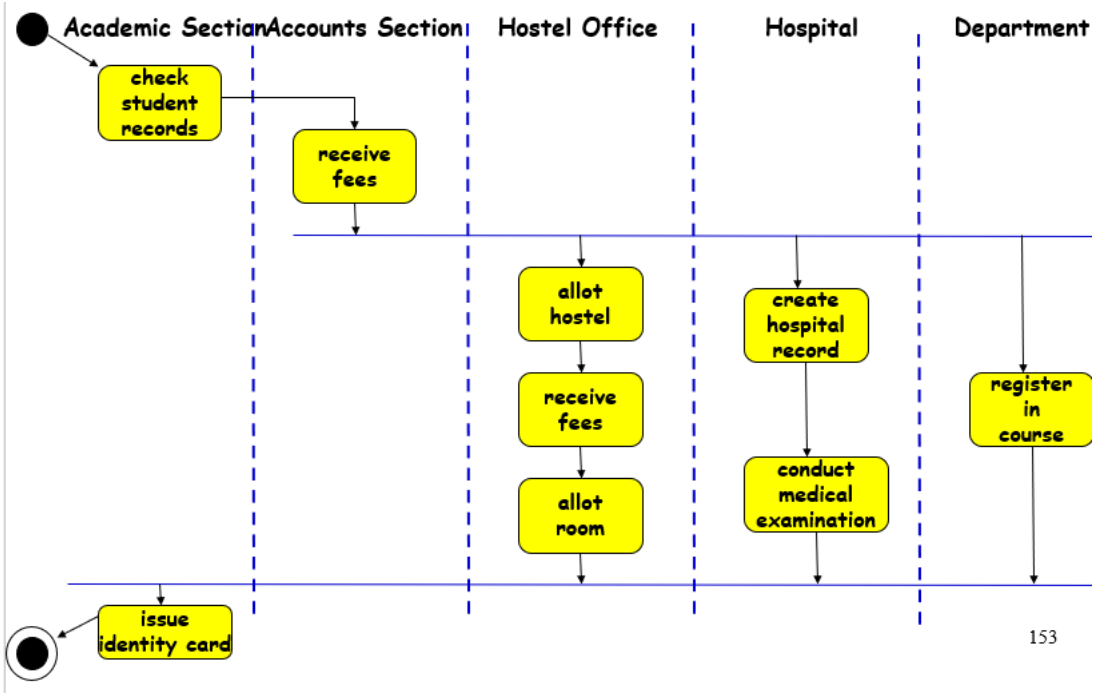


5. Activity diagram:

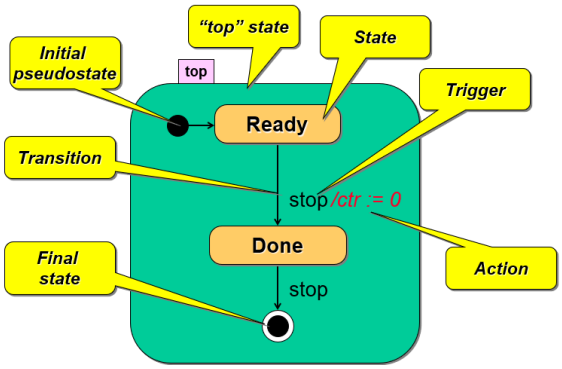
Aan activity diagram is a simple and intuitive illustration of what happens in a workflow, what activities can be done in parallel, and whether there are alternative paths through the workflow.

Often used to represent various activities or chunks of processing and their sequence of activation. An activity is a state with an internal action and has one or more outgoing transitions, e.g. fillOrder.

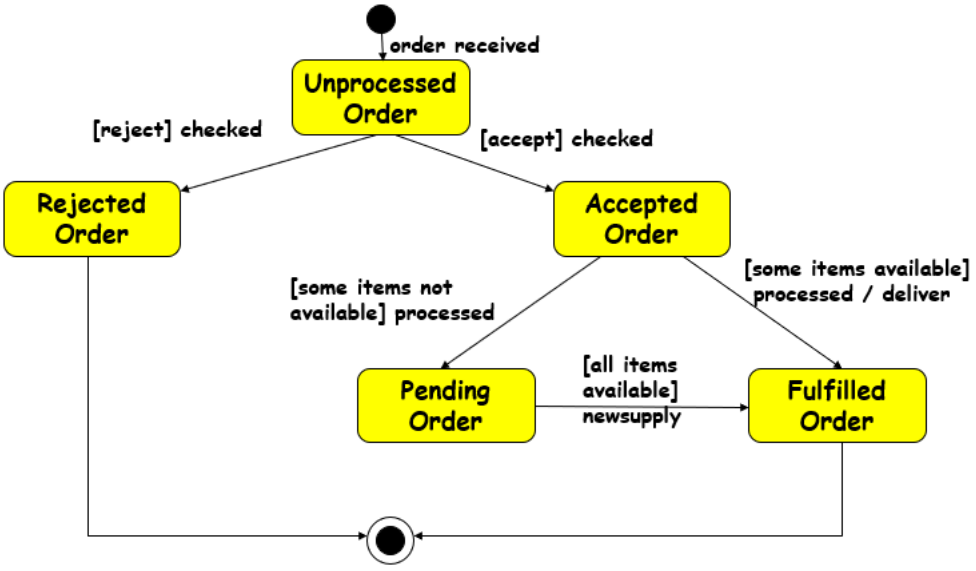
Example activity diagram for student admission process:



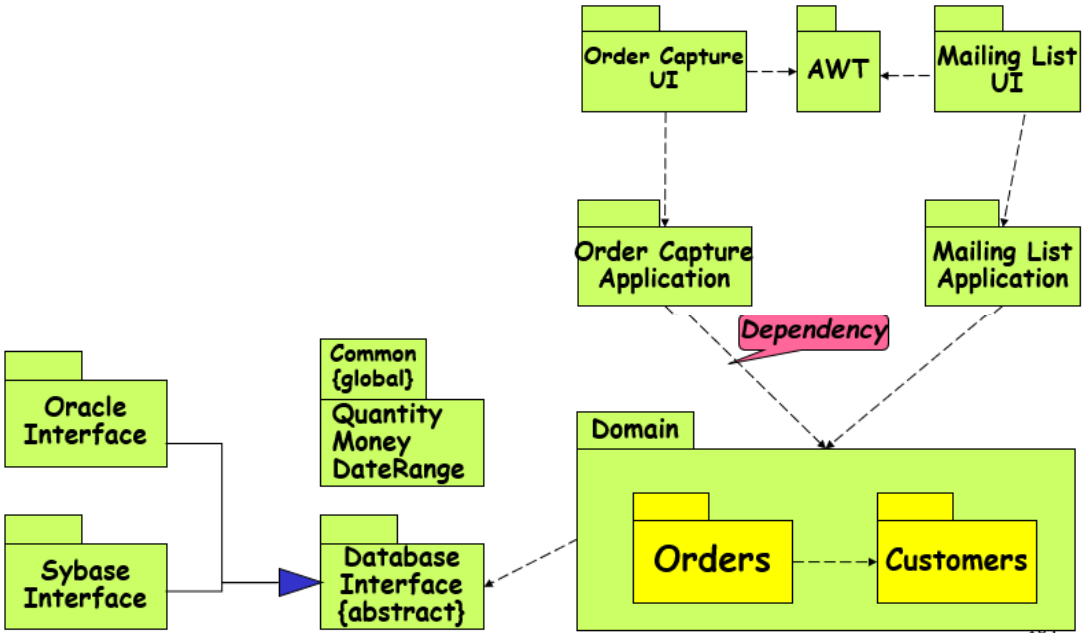
5. State chart diagram:
Used to model how the state of an object changes in its life time. Good at describing how the behavior of an object changes across several use case executions.
Basic UML state chart diagram:



Example of state chart diagram for an order object:

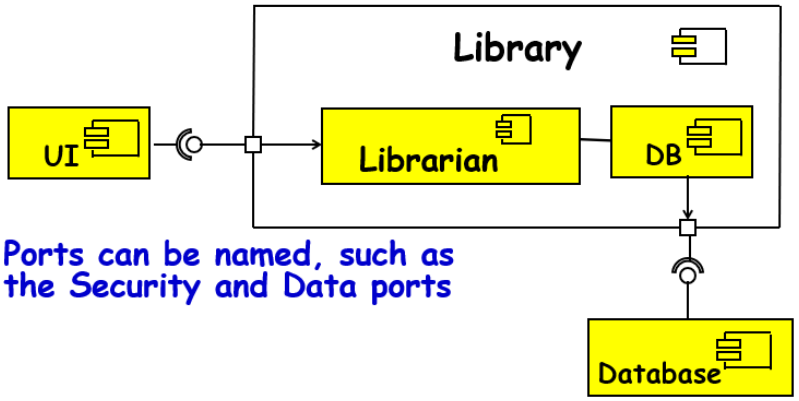


6. Package diagram:
A package is a grouping of several classes. Package diagrams show module dependencies. Useful for large projects with multiple binary files.



7. Component diagram:
Component is a piece of software that can be independently purchased and upgraded, and can integrate seamlessly into customers' existing software. Component diagram captures physical structure of the implementation in terms of various components of the system.

Example for component diagram:

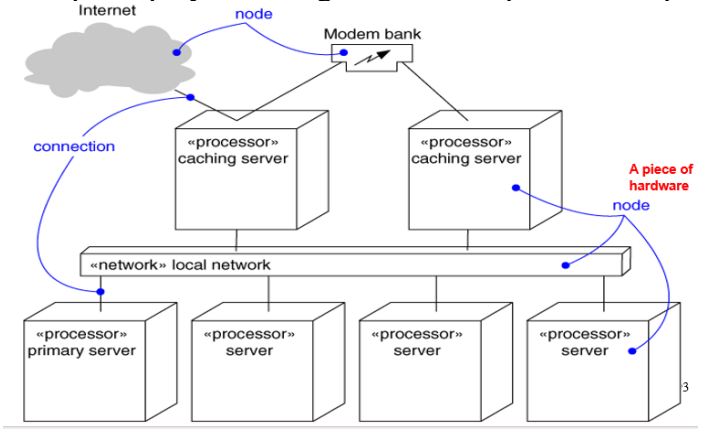


8. Deployment diagram:

The deployment diagram shows, how a software system will be physically deployed in the hardware environment. The main purpose is to s how which component will execute on which hardware component and how they will communicate with each other.

Node is ssomething that can host an artifact. A node represents a physical machine. To model a node, draw a three-dimensional cube with the name of the node at the top. Physical nodes should be labelled with the stereotype device. An artifact is usually a component and usually Files, assemblies, DLL, or scripts.

Example deployment diagram which captures the topology of a system's hardware:



Instructions:
You need to submit the following for the problem assigned to you:
- Functional requirements specification
- UML models: Use case diagrams, Class diagrams, Interaction diagram.

Prob No.1: Graphics Editor:
Those who are not familiar with any graphics editor, please look at the Graphics Drawing features available in either MS-Word or Powerpoint software. You can also examine any other Graphical Drawing package accessible to you. An understanding of the standard features of a Graphics editor should support the following features:

- The graphics editor should support creating several types of geometric objects such as circles, ellipses, rectangles, lines text and polygons.
- Any created object can be selected y clicking a mouse button on the object. A selected object should be shown in a highlighted color.
- A selected object can be edited, i.e. its associated characteristics such as its geometric shape, location, color, fill style, line width, line style, etc. can be changed. For texts, the text content can be changed.
- A selected object can be copied, moved, or deleted.
- The graphics editor should allow user to save his created drawing on the disk under a name he would specify. The graphics editor should also support loading previously created drawings from the disk.
- The user should be able to define any rectangular area on the screen to be zoomed to fill the entire screen.
- A fit screen function makes the entire drawing fit the screen by automatically adjusting the zoom and pan values.
- A pan function should allow the displayed drawing to be panel along any direction by a specified amount.
- The graphics editor should support grouping. A group is simply a set of drawing objects including other groups which when grouped behave as a single entity. This feature is especially useful when you wish to manipulate several entities in the same way. A drawing object can be a direct member of at most one group. It should be possible to perform several editing operations on a group such as move, delete, and copy.
- A set of 10 clip boards should be provided to which one can copy various types of selected entities (including groups) for future use in pasting these at different places when required.

Prob No.2: Time Management Software:
We need to develop the following Time Management Software for a company:
The company needs to develop a time management system for its executives. The software should let the executives register their daily appointment schedules. The information to be stored includes person(s) with whom meeting is arranged, venue, the time and duration of the meeting, and the purpose (e.g. for a specific project work). When a meeting involving many executives, and arrange a meeting (i.e. make relevant entries in the diaries of all the concerned executives) at that time. It should also inform the concerned executives about the scheduled meeting through e-mail. If no common slot is available, TMS should help the secretary to rearrange the appointments of the executives in consultation with the concerned executives for making room for a common slot. To help the executives check their schedules for a particular day the system should have a very easy-to-use graphical interface. Since the executives and the secretaries have their own desktop computers, the time management software should be able to serve several remote request simultaneously. Many of the executives are relative novices in computer usage. Everyday morning the time management software should e-mail every executive his appointments for the day. Besides registering their appointments and meeting, the executives might mark period for which they plan to be on leave. Also, executives might plan out the important jobs they need to do on any day at different hours and post it in their daily list of engagements. Other features to be supported by the TMS are the following: TMS should be able to provide several types of statistics such as which executive spent how much time on meetings. For which project how many meetings were organized for what duration and how many man-hours were devoted to it. Also, it should be able to display on the whole during any given period of time what fraction of time on the average each executive spent on meetings.

Prob No.3: Hotel Automation Software:
we need to develop the following software for automating the activities of a 5-star hotel.
Hotel Automation Software: A hotel has a certain number of rooms. Each room can be either signle bed or double bed type and may be AC or Non-AC type. The rooms have different rates depending on whether they are of single or double, AC or Non-AC types. The room tariff however may vary during different parts of the year depending up on the occupancy rate. For this, the computer should be able to display the average occupancy rate for a given month so that the manager can revise the room tariff either upwards or downwards by a certain percentage.
Guests can reserve rooms in advance or can reserve rooms on the spot depending upon availability of rooms. The receptionist would enter data pertaining to guests such as their arrival time, advance paid, approximate duration of stay, and the type of the room required.

Depending on this data and subject to the available of a suitable room, the computer would allot a room number to the guest and assign a unique token number to each guest. If the guest cannot be accommodated, the computer generates an apology message. The hotel catering services manager would input the quantity and type of food items as and when consumed by the guest, thetoken number of the guest, and the corresponding date and time. When a customer prepares to check-out, the hotel automation software should generate the entire bill for the customer and also print the alance amount payable by him. Frequent guests should be issued an identity number which helps them to get special discounts on their bills.

Prob No.4: Road Repair and Tracking Software (RRTS) to be developed for automatic various book keeping activities associated with the road repairing task of the Public Works Department of the Corporation of large city. Road Repair and Tracking System (RRTS): A city corporation has branch offices at different suburbs of the city. Residents raise repair requests for different roads of the city. These would be entered into his computer system by a clerk. Soon after a repair request is raised, a supervisor visits the road and studies the severity of road condition. Depending on the severity of the road condition and the type of the locality (e.g., commercial area, busy area, relatively deserted area, etc.), he determines the priority for carrying out work, the types and number of machine required, and the number and types of personnel required. Based on this data, the computer system should schedule the repair of the road depending up on the priority of the repair work and subject to the availability of raw material, machines, and personnel. This schedule report is used by the supervisor to direct different repair work. The manpower and machine that are available are entered by the city corporation administrator. He can change these data any time. Of course, any change to the available manpower and machine would require a reschedule of the project. The mayor of the city can request for various road repair statistics such as the number and type of repairs carried out over a period of time and the repair work outstanding at any point of time and the utilization statistics of the repair manpower and machine over any period of time.

Prob No.5: Judiciary Information System (JIS) software:
The attorney general's office has requested us to develop a Judiciary Information System (JIS), to help handle court cases and also to make the past court cases easily accessible to the lawyers and judges. For each court case, the name of the defendant, defendant's address, the crime type (e.g. theft, arson, etc.), when committed (date), where committed (location), name of the arresting officer, and the date of the arrest are entered by the court register. Each court case is identified by a unique case identification number (CIN) which is generated by the computer. The registrar assigns a date of hearing for each case. For this the registrar expects the computer to display the vacant slots on any working day during which the case

can be scheduled. Each time a case is adjourned, the reason for adjournment is entered by the registrar and he assigns a new hearing date. If hearing takes place on any day for a case, the registrar enters the summary of the court proceedings and assigns a new hearing date. Also, on completion of a court case, the summary of the judgment is recorded and the case is closed but the details of the case are maintained for future reference. Other date maintained about a case includes the name of the presiding judge, the public prosecutor, the starting date, and the expected completion date of a trial. The judges should be able to browse through the old cases for guidance on their judgment. The lawyers should also be permitted to browse old cases, but should be charged for each old case they browse. Using the JIS software, the Registrar of the court should be able to query the following:

(a) The currently pending court cases.

In response to this, query, the computer should print out the pending cases sorted by CIN. For each pending case, the following data should be listed: the date in which the case started the defendant's name, address, crime details, the lawyer's name, the public prosecutor's name, and the attending judge's name.

(b) The cases that have been resolved over any given period.

The output in this case should be chronologically list the starting date of the case, the CI, the date on which the judgment was delivered, the name of the attending judge, and the judgment summary.

(c) The cases that are coming up for hearing on a particular date.

(d) The status of any particular case (cases are identified by CIN),

The lawyers and the judges need to refer to the past court cases. The lawyers need to refer these to prepare for their line of arguments. The judges need to refer the past court cases to examine the lines of judgments given previously to similar cases. It should be possible to search for the history of past court cases by entering key words. However, the lawyers should be charged for each time they see the details of a court case to recover some of the computerization costs. For this purpose, it is necessary to provide separate login accounts to the JIS software and they trace of how many court cases each lawyer's views. The registrar should be able to create login accounts for the different users (i.e judges, lawyers, etc.) and should be able to delete these accounts.

Prob No.6: Word Processing Software:

- The word processing software should be able to read text from an ACII file or HTML and store the formatted text as HTML files in the disk.
- The word processing software should ask the user about the number of characters in an output line the formatted text. The user should be allowed to select any number between 1 and 132.
- The word processing software should process the input text in the following way:

- Each output line is to contain exactly the number of characters specified by the user (including blanks).
- The word processing software is to both left and right justify the text so that there are no blanks at the left-and right- hand ends of lines except the first and possibly the last lines of paragraphs. The word processing software should do this by inserting extra blanks between words.
- The input text from the ASCII tile should consist of words separated by one or more blanks and a special character PP, which denotes the end of a paragraph and the beginning of another.
- The first line of each paragraph should be indented by five spaces and should be right justified.
- The last line of each paragraph should be left justified.

- The user should be able to browse through the document and add, modify or delete words. He/she should also be able to mark any word as bold, italic, superscript or subscript.
- The user can request to see the number of characters, words, lines, and paragraphs used in the documents.
- The user should be able to save his document under a name specified by him.

Prob No.7: Restaurant Automation System (RAS): a restaurant owner wants to computerize his order processing, billing and accounting activities. He also expects the computer to generate statistical report about sales of different items. A major goal of this computerization is to make supply ordering more accurate so that the problems of excess inventory is avoided as well as the problem of non-availability of ingredients required to satisfy order for some popular items is maintained. The computer should maintain the prices of all the items and also support changing the prices by the manager. Whenever any item is sold, the sales clerk would enter the item code and the quantity sold. The computer should generate bills whenever food items are sold. Whenever ingredients are issued for preparation of food items, the data is to be entered into the computer. Purchase orders are generated on a daily basis, whenever the stock for any ingredient falls below a threshold values. The computer should calculate the threshold value for each item based on the average consumption of this ingredient for the past three days and assuming that a minimum of two days stock must be maintained for all ingredients. Whenever the ordered ingredients arrive, the invoice data regarding the quantity and price is entered. If sufficient cash balance is available, the computer should print cheques immediately against invoice. Monthly sales receipt and expenses data should be generated whenever the manager would request to see them.

Prob No.8: transport company computerization (TCC) software: A transport company s=wishes to computerize various book keeping activities associated with its operations.

- A transport company owns a number of trucks.
- The transport company has its head office located at the capital and has branch office at several other cities.
- The transport company received consignments of various sizes at (measured in cubic meters) its different offices to be forwarded to different branch offices across the country.
- Once the consignment arrives at the office of the transport company, the details of the volume, destination address, sender address, etc. are entered into the computer. The computer would compute the transport charge depending upon the volume of the consignment and the distribution and would issue a bill for the consignment.
- Once the volume of any particular destination becomes 500 cubic meters, the computerization system should automatically allot the next available truck.
- A truck stays with the branch office until the branch office has enough cargo to load the truck fully.
- The manager should be able to view the status of different trucks at any time.
- The manager should be able to view truck usage over a given period of time.
- When a truck is available and the required consignment is available for dispatch, the computer system should print the details of the consignment number, volume, sender's name and address, and the receiver's name and address to be forwarded along with the truck.
- The manager of the transport company can query the status of any particular consignment and the details of volume of consignments handled to any particular destination and the corresponding revenue generated.
- The manager should also be able to view the average waiting period for different consignments. This statistics is important for him since he normally orders new trucks when the average waiting period for consignments becomes high due to non-availability of trucks. Also, the manager would like to see the average idle time of the truck in the branch for a given period for future planning.

Prob No.9: We need to develop the following simulation software:
A factory has different categories of machines such as lathe machines, turning machines, drilling machines soldering machines, etc. the factory can have different number of machines from each category such as 200 lathe machines, 50 drilling machines, etc. These machines require frequent adjustments and repair. Each category of machine falls uniformly after continuous operation and the failure profile of the different categories of machines is given

by its mean time to failure (MTTF). A certain number of adjusters are employed to keep the machine running. The adjusters have expertise in maintaining different categories of machines. An adjuster may be expert in maintaining more than one type of machine. A service manager coordinates the activities of the adjusters. The service manager maintains a queue of inoperative machine. If there are machines waiting to be repaired, the service manager assigns the machine at the front of the queue to the next available adjuster. Likewise, when some adjusters are not busy, the service manager maintains a queue of idle adjusters and assigns the adjuster at the front of the queue to the next machine that breaks down.
At any given time, one of the two queues will be empty. Thus, the service manager needs to maintain only a single queue, which when it is not empty contains only machines or only adjusters. The factory management wishes to get as much as possible out of the machines is up and running and the adjuster utilization—the percentage of time an adjuster is busy. The goal of our simulation is then to see how the average machine and adjuster utilizing depend on such factory as the number of machines, the number of adjusters, the reliability of the machines in terms of mean time to failure (MTTF). This software would be used by different factories to determine the optimum number of adjusters that they should employ.

Prob No.10: Software component cataloguing software: The software component cataloguing software consists of a software components catalogue and various functions defined on this components catalogue. The software components catalogue should hold details of the components which are potentially reusable. The reusable components can be either design or code. The design might have been constructed using different design notations such as UML, ERD, structured design, etc. Similarly, the code might hae been written using different programming languages. A cataloguer may enter components in the catalogue, may delete the components from the catalogue, and may associate reuse information with a catalogue component in the form of a set of key words. A user of the catalogue may query about the availability of a component using certain key words to describe the component. In order to help manage the component catalogue (i.e., periodically purge the unused components) the cataloguing software should maintain information such as how many times a component has been used, and how many times the component has come up in a query but not used. Since the number of components usually tend to be very high, it is desirable to be able to classify the different types of components hierarchically. A user should be able to browse the components in each category.

Prob No.11: supermarket automation software (SAS): The manager of supermarket wants us to develop an automation software. The supermarket stocks a set of items. Customers pick up their desired items from the different counters in required quantities. The customers

present these items to the sales clerk. The sales clerk passes the items over a bar code reader and an automatic weighing scale and the data regarding the item type and the quantity get registered.

- SAS should be the end of a sales transaction prime is bill containing the serial number of the sales transaction, the name of the item, code number, quantity, unit price, and item price. the bill should indicate the total amount payable.
- SAS should maintain the inventory of the various items of the supermarket. The manager upon query should be able to see the inventory details. in order to support inventory management, the inventory of an item should be decreased whenever an items is sold. SAS should also support an option by which an employee can update the inventory when new supply arrives.
- SAS should support printing the sales statistics for every item the supermarket deals with for any particular day or any particular period. The sales statistics should indicate the quantity of an item sold, the price realized, and the profit.
- The manager of the supermarket should be able to change the price at which an item is sold as the prices of the different items vary on a day-to-day basis.

Prob No.12: We need to develop software for automating various activities of a small book shop. From the discussion with the owner of the book shop, the following user requirements have been arrived at:

Book-shop Automation Software (BAS)

BAS should help the customers query whether a books is in stock. The users can query the availability of a book either by using the book title or by using the name of the author. If the book is not currently being sold by the book-shop, then the customer is asked to enter full details of the book for procurement of the book in future. If a book is in stock, the query for the book is used to increment a request field for the book. The manager can periodically view the request field of the books to arrive at a rough estimate regarding the current demand for different books. BAS should maintain the price of various books. As soon as a customer selected a book for purchase, the sales clerk would enter the ISBN number of the book. BAS should up to the stock and generate the sales receipt for the book. BAS should update the stock and generate the sales receipt for the book. BAS should generate sales statistics (viz., book name, publisher, ISBN number, number of copies sold, and the sales revenue) for any period. The sales statistics will help the owner to the exact business done over any period of time and also to determine inventory level required for various books. The inventory level required for a book is equal to the number of copies of the book sold over a period of two weeks multiplied by the available number of days it takes to procure the book from its publisher. Every day the book shop owner would give a command for the BAS to print the books which have fallen below the threshold

Prob No.13: The local newspaper and magazine delivery agency has asked us to develop a software for him to automate various clerical activities associated with its business.
Newspaper Agency Automation Software:

- This software is to be used by the manager of the news agency and his delivery persons.
- For each delivery person, the system must print each day the publications to be delivered to each address. The addresses should be generated in consecutive order as far as possible so that the commutation of the delivery person is minimal.
- Customers usually subscribe one or more news papers and magazines. They are allowed to change their subscription list by giving one week's advance notice.
- For each delivery person, the system must print each day the publication to be delivered to each address.
- The system should also print for the news agent the information regarding who received what publications and summary information of the current month.
- At the beginning of every month bills are printed by the system to be delivered to the customers. These bills should be computed by the system automatically and should include the publication type, the number of copies delivered during the month, and the cost for these.
- The customers may ask for stopping the deliveries to them for certain periods when they go out of station.
- Customers may request to subscribe new newspapers/magazines, modify their subscription list, or stop their subscription altogether.
- Customers usually pay their monthly dues either by cheques or cash. Once the cheque number or cash received is entered in the system, receipt for the customer should be printed.
- If any customer has any outstanding due for more than one month, a polite reminder message is printed for him and his subscription is discontinued if his dues remain outstanding for period of more than two months.
- The software should compute an print out the amount payable to each delivery boy. Each delivery boy gets 2.5% of the value of the publications delivered by him.

Prob No.14: Department offices in different universities do a lot of book-keeping activities and it is necessary to develop a software to automate these activities.
University Department Information System:

- Various details regarding each student such as his name, address, course registered, etc. are entered at the time he/she takes admission.

- At the beginning of every semester, students register for courses. The information system should allow the department secretary to enter data regarding student registrations. When the secretary enters the roll number of each student, the computer system should bring up a form for the corresponding student and should keep track of courses he has already completed and the courses he has back-log, etc.
- At the end of the semester, the instructors leave their grading information at the office which the secretary enters into the computer. The information system should be able to compute the grade point average for each student for the semester and his cumulative grade point average (OGPA) and print the grade sheet for each student.
- The information system should also keep track of inventories of the Department, such as equipments, their location, furniture, etc.
- The Department gets an yearly grant from the University and the Department spends it in buying equipments, books, stationery items, etc. Also, in addition to the annual grant that the Department gets from the University, it gets funds from different consultancy service it provides to different organizations. It is necessary that the Department information system keeps track of the Department accounts.
- The information system should also keep track of information such as the research projects running in the Department, publications by the faculties etc. these information are keyed in by the secretary of the Department.
- The information system should support querying the up-to-date details about every student by inputting his roll number. It should also support querying the details of the cash book account. The output of this query should include the income, expenditure and balance.

Prob No.15: Medicine Shop Software (MSS):

A retail medicine shop deals with a large number of medicines procured from various manufacturers. The shop owner maintains different medicines in wall mounted and numbered racks.

Thus, one important problem the shop owner faces is to be able to order items as soon as the number of items in the inventory reduces below a threshold value. The shop owner wants to maintain medicines to be able to sustain selling for about one week. To calculate the threshold value for each item, the software must be able to calculate the average number of medicines sales for one week for each part.

At the end of each day, the shop owner would request the computer to generate the items to be ordered. The computer should print out the medicine description, the quantity required, and the address of the vendor supplying the medicine.

At the end of every day the shop owner would give a command to generate the list of medicines which have expired. It should also prepare a vendor-wise list of the expired items

so that the shop owner can ask the vendor to replace these items. Currently, this activity alone takes a tremendous amount of labour on the part of the shop owner and is a major motivator for the automation endeavor.
Whenever any sales occur, the shop owner would enter the code number of each medicine and the corresponding quantity sold. The MSS should print out the cash receipt.
The computer should also generate the revenue for each day and at the end of the month, the computer should generate a graph showing the total sales for each day of the month and also these figures for any given medicine.

I. EXERCISES

1) Create UML Use case diagram and Activity Diagram for the 3 given systems. Write the use case template & Activity diagram for any 2 use cases of the system.

II. AADITIONAL EXERCISES

2)Write the use case template & activity diagram for all use cases in the system.

LAB NO: 7 **Date:**

OBJECT ORIENTED DESIGN -II

Objectives:

- To create UML Class, Object, Sequence & Collaboration Diagrams

Prerequisites:

- UML & OOD Principles, Modelling in UML
- Life Cycle Models & Function Oriented Design

I. EXERCISES

1)Create UML Class diagram & Object Diagram for the 2 given systems. Depict all the attributes, methods, relationships & multiplicities in the class diagram.

2)Draw Sequence diagram & Collaboration diagram for any 2 use cases of the system.

II. ADDITIONAL EXERCISES

3) Draw Sequence diagram & Collaboration diagram for all the use cases of the system.

LAB NO: 8 **Date:**

OBJECT ORIENTED DESIGN -III

Objectives:

- To create UML Statechart diagram, Component & Package Diagrams

Prerequisites:

- UML basics
- Life Cycle Models & Function Oriented Design

I. EXERCISES

1)Create UML Statechart diagram for given 2 systems.

2)Depict Component & Package diagrams for the above systems.

II. ADDITIONAL EXERCISES

3)Draw the Deployment diagram for the above systems

LAB NO: 9 **Date:**

OBJECT ORIENTED DESIGN -IV

Objectives:

- To create Code from UML Diagrams

Prerequisites:

- UML Class diagram & sequence diagram
- Life Cycle Models & Function Oriented Design

I. EXERCISES

1)Create code from UML Class diagram for given 2 systems.

2) Create code from UML sequence diagram for given 2 systems.

II. ADDITIONAL EXERCISES

3)Write the object oriented code manually by examining the problem statement.

LAB NO:10 **Date:**

Objectives:

- To create Graphical User Interface

Prerequisites:

- Defining Problem Statement
- Interfaces & their representation

I. EXERCISES

1)Develop the GUI for 2 given systems.

2) Estimate the KLOC & the duration of the Project in person months by examining the interfaces for the above systems.

II. ADDITIONAL EXERCISES

3)Develop GUI for any other system.

LAB NO:11 & 12 **Date:**

MINI PROJECT

Guidelines:

Students need to submit the Final Report and demonstrate all the Software engineering concepts discussed in the lab. Students can choose any problem statement not done in the lab.

Final Report Format:

- Title, Team members
- Problem statement
- Create USe case, Class, Collaboration, Activity, Statechart, Package & Deployment diagrams for the Project.
- Generate code from UML diagrams manually.
- Develop GUI for the system.
- Estimate the KLOC from interfaces of the GUI & thus calculate the effort in person months required to complete the project.