

SOFTWARE ENGINEERING

AMBEDKAR INSTITUTE OF TECHNOLOGY | Shakarpur, Delhi -92

INDEX

S. No.	Topic	Page No.
1	What do you understand from Software Engineering? Describe Waterfall Model?	1-3
2	Describe Spiral Model? What are its advantages and disadvantages?	4-6
3	Describe Prototype Model? What are its advantages and disadvantages?	7-8
4	Compare the various SE process models along with its advantages and disadvantages.	9-11
5	What do you understand from Requirements? What are characteristics of Requirements? Describe the different types of Requirements.	12-13
6	Identify Functional and Non-Functional Requirements from the given problem statement.	14-14
7	Identify Functional and Non-Functional Requirements from the given problem Statement.	15-15
8	What are the various Project Estimation Techniques? Explain COCOMO. What are various types of COCOMO? Explain Basic COCOMO model with respect to Organic, Semi-detached and Embedded.	16-18
9	Using basic COCOMO model, find out the EFFORT, TIME FOR DEVELOPMENT, and NUMBER OF DEVELOPERS REQUIRED if the project is Organic and the project size is 200KLOC.	19-19
10	Using basic COCOMO model, find out the EFFORT, TIME FOR DEVELOPMENT, and NUMBER OF DEVELOPERS REQUIRED if the project is embedded and the project size is 350KLOC.	20-20
11	What do you understand from Entity-Relationship Model? Explain Entity Set and Relationship Set. What do you mean by Attributes and Keys?	21-27
12	Draw an ER Diagram for Library Management System which includes book_info, Staff_info, issue_of_book, return_of_book, fine_calculation.	28-28
13	What do you understand from Data Flow Diagrams? Describe Graphical notations for Data flow Diagrams. Explain the Symbols used in DFD.	30-34
14	Make a DFD for Library Management System.	35-36

Practical-1

Aim: What do you understand from Software Engineering? Describe Waterfall Model?

Software Engineering:

Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called **software product**.

Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods.

Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

Fritz Bauer, a German computer scientist, defines software engineering as “Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and work efficiently on real machines.”

Waterfall Model:

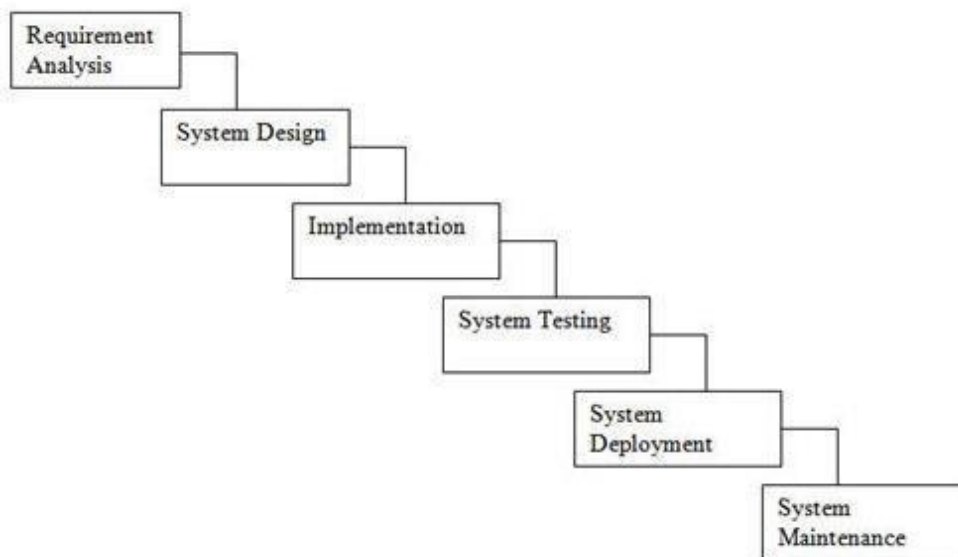
Introduction:

Waterfall model is an example of Sequential model. In this model, the software development activity is divided into different phases and each phase consists of series of tasks and has different objectives.

Waterfall model is the pioneer of the SDLC processes. In fact, it was the first model which was widely used in the software industry. It is divided into phases and output of one phase becomes input of the next phase. It is mandatory for a phase to be completed before the next phase starts. In short, there is no overlapping in Waterfall model

In waterfall, development of one phase starts only when the previous phase is complete. Because of this nature, each phase of waterfall model is quite precise well defined. Since the phase's falls from higher level to lower level, like a water fall, It's named as waterfall model.

Pictorial representation of waterfall model:



The activities involved in different phases are as follows:

S.No	Phase	Activities Performed	Deliverables
1	Requirement Analysis	1. Capture all the requirements. 2. Do brainstorming and walkthrough to understand the requirements. 3. Do the requirements feasibility test to ensure that the requirements are testable or not.	RUD (Requirements Understanding Document)
2	System Design	1. As per the requirements, create the design 2. Capture the hardware / software requirements. 3. Document the designs	HLD (High Level Design document) LLD (Low level design document)
3	Implementation	1. As per the design create the programs / code 2. Integrate the codes for the next phase. 3. Unit testing of the code	Programs Unit test cases and results
4	System Testing	1. Integrate the unit tested code and test it to make sure if it works as expected. 2. Perform all the testing activities (Functional and non-functional) to make sure that the system meets the requirements. 3. In case of any anomaly, report it. 4. Track your progress on testing through tools like traceability metrics, ALM 5. Report your testing activities.	Test cases Test reports Defect reports Updated matrices.
5	System Deployment	1. Make sure that the environment is up 2. Make sure that the test exit criteria are met. 3. Deploy the application in the respective environment. 4. Perform a sanity check in the environment after the application is deployed to ensure the application does not break.	User Manual Environment definition / specification
6	System maintenance	1. Make sure that the application is up and running in the respective environment. 2. In case user encounters and defect, make sure to note and fix the issues faced. 3. In case any issue is fixed; the updated code is deployed in the environment. 4.The application is always enhanced to incorporate more features, update the environment with the latest features	User Manual List of production tickets List of new features implemented.

When to use SDLC Waterfall Model?

SDLC Waterfall model is used when

- Requirements are stable and not changed frequently.
- Application is small.
- There is no requirement which is not understood or not very clear.
- The environment is stable
- The tools and technology used is stable and is not dynamic
- Resources are well trained and are available.

Advantages and Disadvantages of waterfall model:

Advantages of using Waterfall model are as follows:

- Simple and easy to understand and use.
- For smaller projects, waterfall model works well and yield the appropriate results.
- Since the phases are rigid and precise, one phase is done one at a time, it is easy to maintain.
- The entry and exit criteria are well defined, so it easy and systematic to proceed with quality.
- Results are well documented.

Disadvantages of using Waterfall model:

- Cannot adopt the changes in requirements
- It becomes very difficult to move back to the phase. For example, if the application has now moved to testing stage and there is a change in requirement, it becomes difficult to go back and change it.
- Delivery of the final product is late as there is no prototype which is demonstrated intermediately.
- For bigger and complex projects, this model is not good as risk factor is higher.
- Not suitable for the projects where requirements are changed frequently.
- Does not work for long and ongoing projects.
- Since the testing is done at later stage, it does not allow identifying the challenges and risks in the earlier phase so the risk mitigation strategy is difficult to prepare.

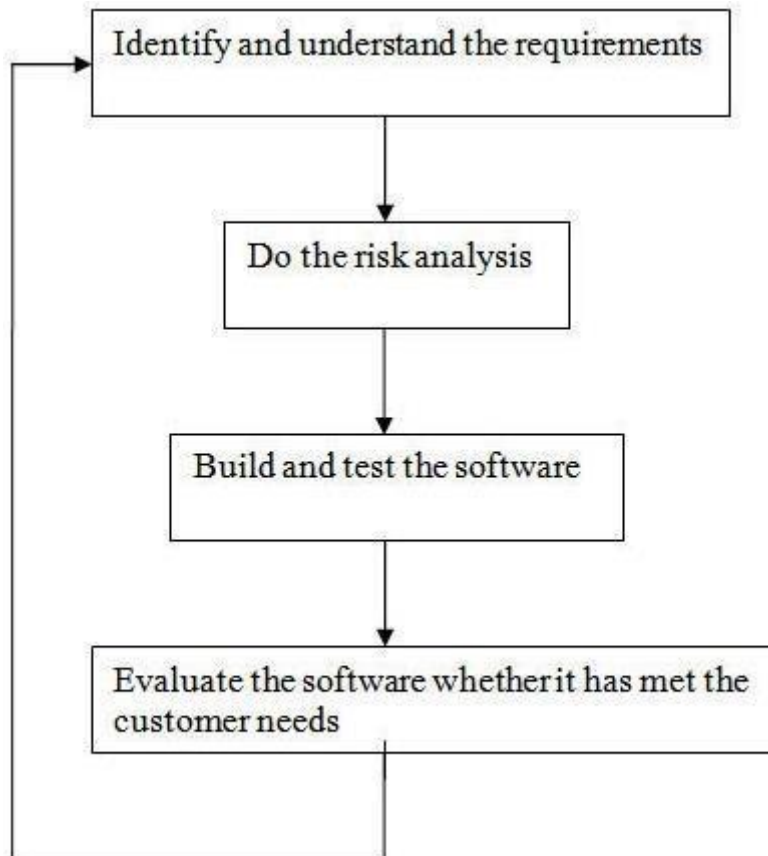
Practical-2

Aim: Describe Spiral Model? What are its advantages and disadvantages?

Introduction:

Spiral model is a combination of sequential and prototype model. This model is best used for large projects which involves continuous enhancements. There are specific activities which are done in one iteration (spiral) where the output is a small prototype of the large software. The same activities are then repeated for all the spirals till the entire software is build.

To explain in simpler terms, the steps involved in spiral model are:



A spiral model has 4 phases described below:

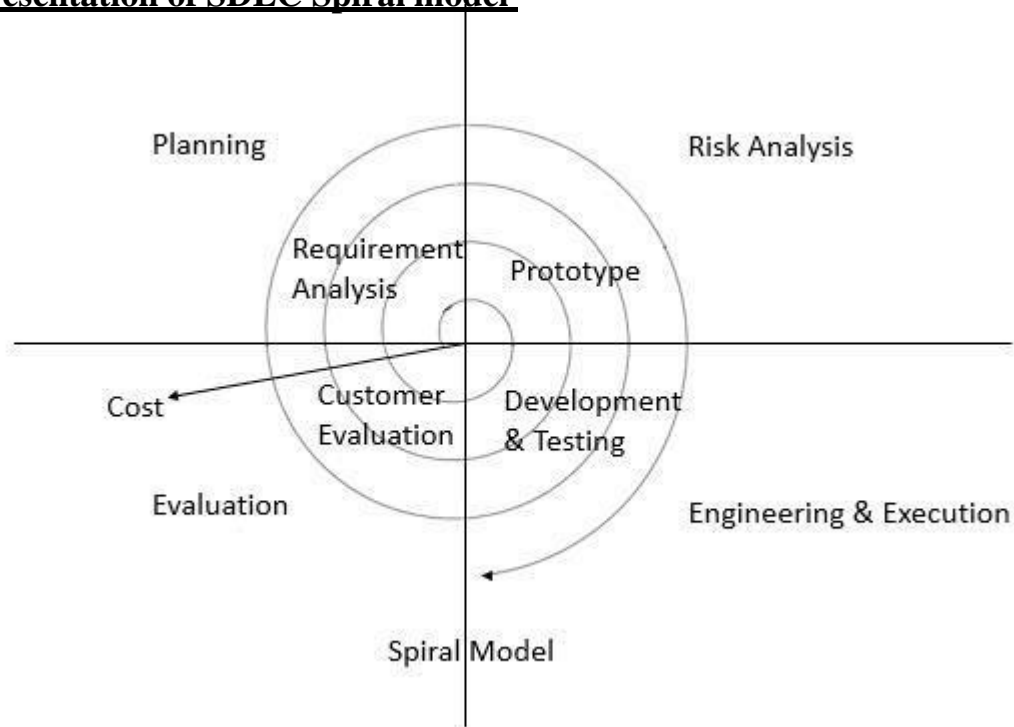
1. Planning phase
2. Risk analysis phase
3. Engineering phase
4. Evaluation phase.

Activities which are performed in the spiral model phases are shown below:

Phase Name	Activities performed	Deliverables / Output
Planning	-Requirements are studied and gathered. - Feasibility study - Reviews and walkthroughs to streamline the requirements	Requirements understanding document Finalized list of requirements.

Phase Name	Activities performed	Deliverables / Output
Risk Analysis	Requirements are studied and brain storming sessions are done to identify the potential risks Once the risks are identified, risk mitigation strategy is planned and finalized	Document which highlights all the risks and its mitigation plans.
Engineering	Actual development and testing of the software takes place in this phase	Code Test cases and test results Test summary report and defect report.
Evaluation	Customers evaluate the software and provide their feedback and approval	Features implemented document

Pictorial representation of SDLC Spiral model



Testing and development starts from planning phase and carries up to evaluation phase. All the requirements are collected in the planning phase itself. In the risk analysis phase, we assume all the risks could be occurred during testing and development. In engineering and execution phase we start executing the test cases which are planned and identified and finally we move to the evaluation phase where we review the progress of the project. The reason of success of Spiral Model is that analysis and engineering both carried out in each phase of the project.

When to Use Spiral model?

Spiral is used in the following scenarios:

- When the project is large.
- Where the software needs continuous risk evaluation.
- Requirements are a bit complicated and require continuous clarification.
- Software requires significant changes.

Where enough time frame is there to get end user feedback.

Where releases are required to be frequent.

Advantages and Disadvantages of using Spiral Model:

Advantages of using Spiral model are as follows:

Development is fast

Larger projects / software are created and handled in a strategic way

Risk evaluation is proper.

Control towards all the phases of development.

More and more features are added in a systematic way.

Software is produced early.

Has room for customer feedback and the changes are implemented faster.

Disadvantages of Spiral model are as follows:

Risk analysis is important phase so requires expert people.

Is not beneficial for smaller projects.

Spiral may go infinitely.

Documentation is more as it has intermediate phases.

It is costly for smaller projects.

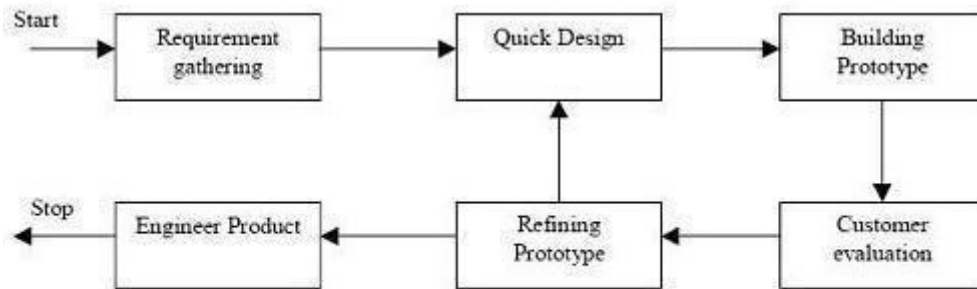
Practical-3

Aim: Describe Prototype Model? What are its advantages and disadvantages?

Introduction:

The basic idea here is that instead of freezing the requirements before a design or coding can proceed, a throwaway prototype is built to understand the requirements. This prototype is developed based on the currently known requirements. By using this prototype, the client can get an “actual feel” of the system, since the interactions with prototype can enable the client to better understand the requirements of the desired system. Prototyping is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determining the requirements. The prototype are usually not complete systems and many of the details are not built in the prototype. The goal is to provide a system with overall functionality.

Diagram of Prototype model:



Prototyping Model

When to use Prototype model:

- Prototype model should be used when the desired system needs to have a lot of interaction with the end users.
- Typically, online systems, web interfaces have a very high amount of interaction with end users, are best suited for Prototype model. It might take a while for a system to be built that allows ease of use and needs minimal training for the end user.
- Prototyping ensures that the end users constantly work with the system and provide a feedback which is incorporated in the prototype to result in a useable system. They are excellent for designing good human computer interface systems.

Advantages of Prototype model:

- Users are actively involved in the development.
- Since in this methodology a working model of the system is provided, the users get a better understanding of the system being developed.
- Errors can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily.
- Confusing or difficult functions can be identified

Disadvantages of Prototype model:

- Leads to implementing and then repairing way of building systems.
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
- Incomplete application may cause application not to be used as the full system was designed.
- Incomplete or inadequate problem analysis.

Practical-4

Aim: Compare the various SE process models along with its advantages and disadvantages.

SDLC Models

A software project, regardless of whether it is large or small, goes through certain defined stages, which together, are known as the Software Development Life Cycle (SDLC).

There are five phases that are the part of the SDLC. These phases are:

- Systems Investigation – Identify problems or opportunities.
- Systems Analysis – How can we solve the problem?
- Systems Design – Select and plan the best solution.
- Systems Implementation – Place solution into effect.
- Systems Maintenance and Review – Evaluate the results of the solution.

SDLC models are created based on the various phases of the SDLC, the order in which they occur and the interaction between them. The output generated by each phase serves as the input for the next.

- 1) Waterfall Model
- 2) Prototyping Model
- 3) Incremental Model
- 4) Rapid Application development (RAD Model)
- 5) Spiral Model
- 6) Extreme Programming Model

Prototyping Model

Strength	Weakness
<ul style="list-style-type: none">• Users are actively involved in the development• It provides a better system to users, as users have natural tendency to change their mind in specifying requirements.• Interaction with the prototype stimulates awareness of additional needed functionality• Errors can be detected much earlier as the system is mode side by side.• Quicker user feedback is available leading to better solutions.	<ul style="list-style-type: none">• Leads to implementing and then repairing way of building systems.• Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.• Tendency to abandon structured program development for “code-and-fix” development.• Overall maintainability may be overlooked.• The customer may want the prototype delivered.• Process may continue forever (scope creep)

Incremental model

STRENGTH	WEAKNESS
<ul style="list-style-type: none">• Generates working software quickly and early during the software life cycle.• More flexible - less costly to change scope and requirements.• Easier to test and debug during a smaller iteration.• Easier to manage risk because risky pieces are identified and handled during its iteration.• Lowers initial delivery cost• Risk of changing requirements is reduced	<ul style="list-style-type: none">• Problems may arise pertaining to system architecture because not all requirements are gathered up front for the entire software life cycle.• Requires early definition of a complete and fully functional system to allow for the definition of increments.• Each phase of an iteration is rigid and do not overlap each other.• Total cost of the complete system is not lower.• Requires good planning and design

Waterfall Model

Strength	Weakness
<ul style="list-style-type: none"> • Simple and easy to use. • Easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process. • Milestones are better understood • Sets requirements stability • Phases are processed and completed one at a time. • Works well for smaller projects where requirements are very well understood. • Reinforces good habits: define-before- design, design-before-code. • Works well when quality is more important than cost or schedule. 	<ul style="list-style-type: none"> • No working software is produced until late during the life cycle. • High amounts of risk and uncertainty. • Idealized, doesn't match reality well. • Software is delivered late in project, delays discovery of serious errors. • After project requirements are gathered in the first phase, there is no formal way to make changes to the project as requirements change or more information becomes available to the project team. • It might not a good model for complex projects or projects that take more than a few months to complete.

RAD Model

STRENGTH	WEAKNESS
<ul style="list-style-type: none"> • The time required to develop the software is drastically reduced due to a reduced requirement analysis. • All the software prototypes produced can be kept in a repository for future use. • Project cost controls are easier to implement and manage as well. • It is a big cost saver in terms of project budget as well as project time and cost due to reusability of the prototypes. • Helps in saving time required for testing. • The project management requirements are collected in a dynamic manner. If there are any additional requirements, these are then included in the next prototype built. • It promotes better documentation through written test cases. 	<ul style="list-style-type: none"> • This method may not be useful for large, unique or highly complex projects. • This method cannot be a success if the team is not sufficiently motivated and nor is unable to work cohesively together. • There are times when the team ignores necessary quality parameters such as consistency, reliability and standardization. Hence this can make project quality management hard to implement during the project management life cycle . • It should not be used for new cutting edge functionality which has not been developed before. • Success depends on the extremely high technical skills of the developers.

Extreme Programming Model

Strength	Weakness
<ul style="list-style-type: none"> • Lightweight methods suit small-medium size projects. • Produces good team cohesion. • Emphasizes final product. • Iterative. • Test based approach to requirements and quality assurance. 	<ul style="list-style-type: none"> • Difficult to scale up to large projects where documentation is essential. • Needs experience and skill if not to degenerate into code-and-fix. • Programming pairs is costly. • Test case construction is a difficult and specialized skill.

Spiral Model

STRENGTH	WEAKNESS
<ul style="list-style-type: none"> • High amount of risk analysis. • Good for large and mission critical projects • Software is produced early in the software life cycle. • Provides early indication of insurmountable risks, without much cost. • Users see the system early because of rapid prototyping tools. • Critical high-risk functions are developed first • The design does not have to be perfect • Early and frequent feedback from users • Cumulative costs assessed frequently 	<ul style="list-style-type: none"> • Can be a costly model to use. • Risk analysis requires highly specific expertise. • Project's success is highly dependent on the risk analysis phase. • Doesn't work well for smaller projects.

Tabular Comparison of SDLC Models

Features	Waterfall	Prototyping	Incremental	RAD	Spiral	XP
Requirements Specification	Beginning	Frequently changed	Beginning	Timebox released	Beginning	Defined incrementally
Cost	Low	High	Low	Low	Expensive	Low
Guarantee Of Success	Less	Good	High	Good	High	High
Simplicity	Simple	Simple	Intermediate	Very Simple	Intermediate	Simple
Overlapping Phases	No overlapping	Yes	No	No	Yes	No
Risk Involvement	High	Low	Easily manage	Very Low	Low	Low
Expertise Required	High	Medium	High	Medium	High	High
Changes Incorporated	Difficult	Easy	Easy	Easy	Easy	Easy
User Involvement	At Beginning	High	Intermediate	High	High	High
Flexibility	Rigid	Highly flexible	Less Flexible	High	Flexible	Flexible
Maintenance	Least Glamorous	Routine Maintenance	Promoted maintainability	Easily Maintained	Typical	Easily Maintained
Integrity & Security	Least	Weak	Robust	Vital	High	Robust
Reusability	Limited	Weak	Yes	High	High	High
Documentation & Training Required	Vital	Weak	Yes	Yes	Yes	Yes
Time Frame	Long	Short	Very Long	Short	Long	Short

Practical-5

Aim: What do you understand from Requirements? What are characteristics of Requirements? Describe the different types of Requirements.

Requirements:

Somerville defines "requirement" 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.

Characteristics of Requirements

Requirements gathered for any new system to be developed should exhibit the following three properties:

Unambiguity: There should not be any ambiguity what a system to be developed should do. For example, consider you are developing a web application for your client. The client requires that enough number of people should be able to access the application simultaneously. What's the "enough number of people"? That could mean 10 to you, but, perhaps, 100 to the client. There's an ambiguity.

Consistency: To illustrate this, consider the automation of a nuclear plant. Suppose one of the clients say that if the radiation level inside the plant exceeds R1, all reactors should be shut down. However, another person from the client side suggests that the threshold radiation level should be R2. Thus, there is an inconsistency between the two end users regarding what they consider as threshold level of radiation.

Completeness: A particular requirement for a system should specify what the system should do and also what it should not. For example, consider a software to be developed for ATM. If a customer enters an amount greater than the maximum permissible withdrawal amount, the ATM should display an error message, and it should not dispense any cash.

Traceable: The requirement meets all or part of a business need as stated by stakeholders and authoritatively documented.

Specify Importance: Many requirements represent a stakeholder-defined characteristic the absence of which will result in a major or even fatal deficiency. Others represent features that may be implemented if time and budget permits. The requirement must specify a level of importance.

Verifiable: The implementation of the requirement can be determined through basic possible methods: inspection, demonstration, test (instrumented) or analysis (to include validated modelling & simulation).

Categorization of Requirements

Based on the target audience or subject matter, requirements can be classified into different types, as stated below:

User requirements: They are written in natural language so that both customers can verify their requirements have been correctly identified

System requirements: They are written involving technical terms and/or specifications, and are meant for the development or testing teams

Requirements can be classified into two groups based on what they describe:

Functional requirements (FRs): These describe the functionality of a system -- how a system should react to a particular set of inputs and what should be the corresponding output.

Non-functional requirements (NFRs): 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

Practical-6

Aim: Identify *Functional and Non-Functional Requirements* from the given problem statement.

Consider the problem statement for an “Online Polling System” to be developed: “Internet has led to discussion of e-democracy and online voting. Many people think that the internet could replace representative democracy, enabling everyone to vote on everything and anything by online voting. Online voting could reduce cost and make voting more convenient. This type of voting can be done for e-democracy, or it may be used for finalizing a solution, if many alternatives are present. Online voting make’s use of authentication, hence it needs security, and the system must be able to address obtaining, marking, delivering and counting ballots via computer. Advantage of online voting is it could increase voter turnout because of convenience, and it helps to reduce fraud voting.”

Functional Requirements:

User Login: a user has to login in order to vote. User can login to see candidate details.

Publish Manifesto: A candidate can login and publish his election manifesto.

Vote: User can cast his vote in favour of specific candidate.

Count votes: System must be able to count votes per candidate.

Publish results: Admin can publish results.

Prevent fraud voting: Only valid user can participate in polling.

There could be others like email notifications, error handling and so on.

Non-Functional Requirements:

The system must remain accessible to thousands of users at a time.

Practical-7

Aim: Identify *Functional and Non-Functional Requirements* from the given problem Statement.

Consider the problem statement for an “Online Auction System” to be developed:

“New user can register to the system through an online process. By registering a user agrees to abide by different pre-defined terms and conditions as specified by the system. Any registered user can access the different features of the system authorized to him/her, after he authenticates himself through the login screen. An authenticated user can put items in the system for auction. Authenticated users can place bid for an item. Once the auction is over, the item will be sold to the user placing the maximum bid. Payments are to be made by third party services, which, of course, is guaranteed to be secure. The user selling the item will be responsible for its shipping. If the seller thinks he’s getting a good price, he can, however, sell the item at any point of time to the maximum bidder available.”

Functional Requirements:

User Registration: New user can register.

Terms and Conditions: Before registering, user have to agree with pre-defined terms and conditions.

User Login: Registered user can login. User can put items in the system for auction.

Bids: For auction, user can place bids.

Buyer: User which placed maximum bids will be buyer.

Online Payment: Third party services will be used for payment.

Seller can change buyer at any time.

There can be others like email verification, etc.

Non-Functional Requirements:

The system must remain accessible to thousands of users at a time.

Attractive GUI.

Practical-8

Aim: What are the various Project Estimation Techniques? Explain COCOMO. What are various types of COCOMO? Explain Basic COCOMO model with respect to Organic, Semi-detached and Embedded.

Project estimation may involve the following:

Software size estimation: Software size may be estimated either in terms of KLOC (Kilo Line of Code) or by calculating number of function points in the software. Lines of code depend upon coding practices and Function points vary according to the user or software requirement.

Effort estimation: The managers estimate efforts in terms of personnel requirement and man-hour required to produce the software. For effort estimation software size should be known. This can either be derived by managers' experience, organization's historical data or software size can be converted into efforts by using some standard formulae.

Time estimation: Once size and efforts are estimated, the time required to produce the software can be estimated. Efforts required is segregated into sub categories as per the requirement specifications and interdependency of various components of software. Software tasks are divided into smaller tasks, activities or events by Work Breakthrough Structure (WBS). The tasks are scheduled on day-to-day basis or in calendar months. The sum of time required to complete all tasks in hours or days is the total time invested to complete the project.

Cost estimation: This might be considered as the most difficult of all because it depends on more elements than any of the previous ones. For estimating project cost, it is required to consider -

- o Size of software
- o Software quality
- o Hardware

- o Additional software or tools, licenses etc.
- o Skilled personnel with task-specific skills

- o Travel involved
- o Communication

- o Training and support

Project Estimation Techniques

We discussed various parameters involving project estimation such as size, effort, time and cost.

Project manager can estimate the listed factors using two broadly recognized techniques –

Decomposition Technique: This technique assumes the software as a product of various compositions. There are two main models -

- ▣ **Line of Code:** Estimation is done on behalf of number of line of codes in the software product.
- ▣ **Function Points:** Estimation is done on behalf of number of function points in the software product.

Empirical Estimation Technique: This technique uses empirically derived formulae to make estimation. These formulae are based on LOC or FPs.

❓ **Putnam Model:** This model is made by Lawrence H. Putnam, which is based on Norden's frequency distribution (Rayleigh curve). Putnam model maps time and efforts required with software size.

❓ **COCOMO:** COCOMO stands for CONstructive COSt MOdel, developed by Barry W. Boehm. It divides the software product into three categories of software: organic, semi-detached and embedded.

Cost Constructive Model (COCOMO):

COCOMO (Constructive Cost Estimation Model) was proposed by Boehm [1981]. According to Boehm, software cost estimation should be done through three stages: Basic COCOMO, Intermediate COCOMO, and Complete COCOMO.

Types of COCOMO:

There are three types of COCOMO:

Basic COCOMO: The Basic COCOMO model is a static, single-valued model that computes software development effort (and cost) as a function of program size expressed in estimated lines of code (LOC).

Intermediate COCOMO: The Intermediate COCOMO model computes software development effort as a function of program size and a set of "cost drivers" that include subjective assessments of product, hardware, personnel and project attributes.

Detailed COCOMO model: The Detailed COCOMO model incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step (analysis, design, etc.) of the software engineering process.

COCOMO can be applied to:

- Organic
- Semi detached
- Embedded

Difference between Organic, Semi-detached and Embedded software:

<i>Mode</i>	<i>Project size</i>	<i>Nature of Project</i>	<i>Innovation</i>	<i>Deadline of the project</i>	<i>Development Environment</i>
Organic	Typically 2-50 KLOC	Small size project, experienced developers in the familiar environment. For example, pay roll, inventory projects etc.	Little	Not tight	Familiar & In house
Semi detached	Typically 50-300 KLOC	Medium size project, Medium size team, Average previous experience on similar project. For example: Utility systems like compilers, database systems, editors etc.	Medium	Medium	Medium
Embedded	Typically over 300 KLOC	Large project, Real time systems, Complex interfaces, Very little previous experience. For example: ATMs, Air Traffic Control etc.	Significant	Tight	Complex Hardware/ customer Interfaces required

Basic COCOMO Model:

The Basic COCOMO equations take the form:

$$E = a_b (KLOC)^{b_b}$$

$$D = c_b (E)^{d_b}$$

where E is the effort applied in person-months, D is the development time in chronological months and KLOC is the estimated number of delivered lines of code for the project (express in thousands). The coefficients a_b and c_b and the exponents b_b and d_b are given in following table:

Software Project	a_b	b_b	c_b	d_b
organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
embedded	3.6	1.20	2.5	0.32

Practical-9

AIM: Using basic COCOMO model, find out the **EFFORT, TIME FOR DEVELOPMENT, NUMBER OF DEVELOPERS REQUIRED** if the project is Organic and the project size is 200KLOC.

Solution:

$$\text{Effort} = a_b(\text{KLOC})^b$$

where $a_b=2.4$ and $b_b=1.05$

$$\text{Effort} = 2.4(20)^{1.05} = 96 \text{ persons-month (approx.)}$$

$$\text{Development Time} = c_b E^d$$

where $c_b=2.5$ and $d_b=0.38$

$$\text{Development time} = 2.5(96)^{0.38} = 12 \text{ months (approx.)}$$

$$\text{No. of developers required} = E/D = 20 \text{ persons (approx.)}$$

Practical-10

AIM: Using basic COCOMO model, find out the **EFFORT, TIME FOR DEVELOPMENT, NUMBER OF DEVELOPERS REQUIRED** if the project is embedded and the project size is 350KLOC.

Solution:

$$\text{Effort} = a_b(\text{KLOC})^b$$

where $a_b=3.6$ and $b_b=1.2$

$$\text{Effort} = 3.6(20)^{1.2} = 8575 \text{ persons-month (approx.)}$$

$$\text{Development Time} = c_b E^d$$

where $c_b=2.5$ and $d_b=0.32$

$$\text{Development time} = 2.5(8575)^{0.32} = 53.5 \text{ months (approx.)}$$

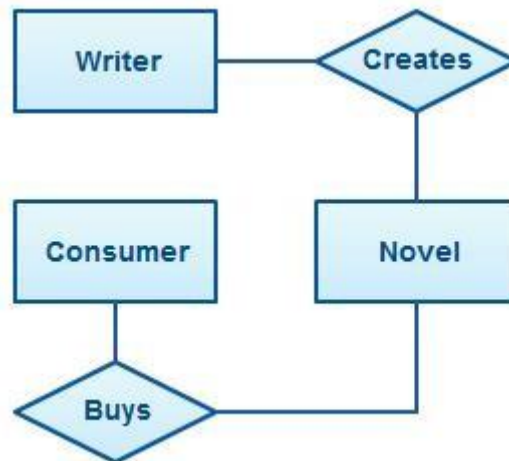
$$\text{No. of persons required} = E/D = 60 \text{ persons (approx.)}$$

PRACTICAL NO :-11

Aim:-What do you understand from Entity-Relationship Model. Explain Entity Set and Relationship Set. What do you mean by Attributes and Keys?

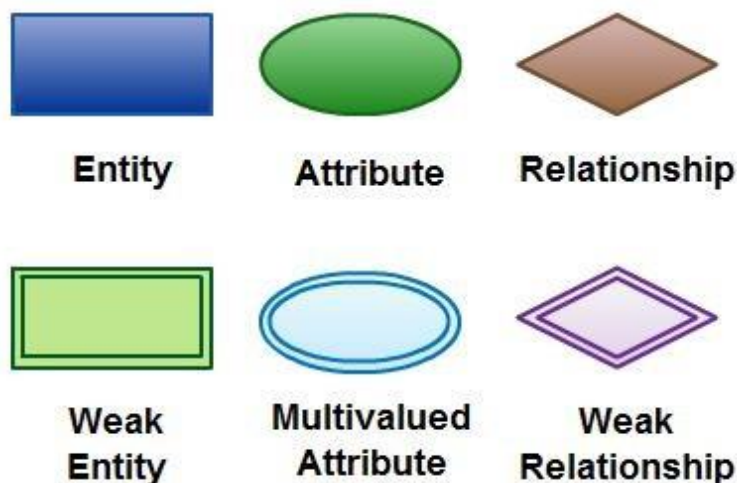
Theory:-

An **Entity Relationship Diagram (ERD)** is a visual representation of different data using conventions that describe how these data are related to each other. For example, the elements writer, novel, and consumer may be described using ER diagrams this way:



In the diagram, the elements inside rectangles are called entities while the items inside diamonds denote the relationships between entities. This ER diagram tutorial for beginners covers most things related to ER diagram, for quick navigation use the links below.

ER Diagram Symbols and Notations:-



There are three basic elements in an ER Diagram: entity, attribute, relationship. There are more elements which are based on the main elements. They are weak entity, multivalued attribute, derived attribute, weak relationship and recursive relationship. Cardinality and ordinality are two other notations used in ER diagrams to further define relationships.

Entity:-

An entity can be a person, place, event, or object that is relevant to a given system. For example, a school system may include students, teachers, major courses, subjects, fees, and other items. Entities are represented in ER diagrams by a rectangle and named using singular nouns.

Weak Entity:-

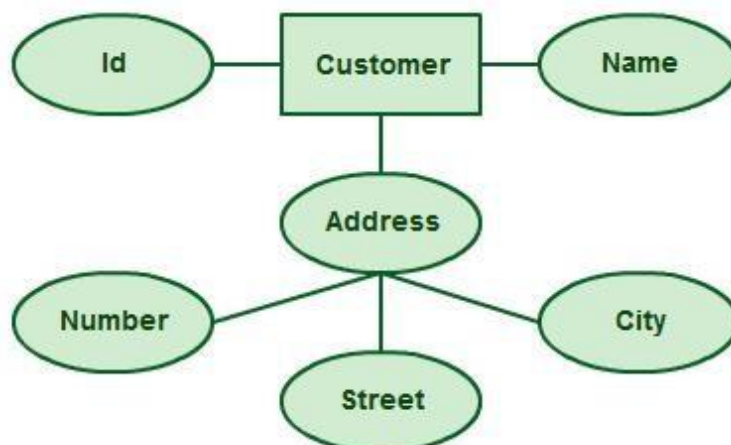
A weak entity is an entity that depends on the existence of another entity. In more technical terms it can be defined as an entity that cannot be identified by its own attributes. It uses a foreign key combined with its attributes to form the primary key. An entity like order item is a good example for this. The order item will be meaningless without an order so it depends on the existence of order.



Weak Entity Example in ER diagrams

Attribute:-

An attribute is a property, trait, or characteristic of an entity, relationship, or another attribute. For example, the attribute Inventory Item Name is an attribute of the entity Inventory Item. An entity can have as many attributes as necessary. Meanwhile, attributes can also have their own specific attributes. For example, the attribute “customer address” can have the attributes number, street, city, and state. These are called composite attributes. Note that some top level ER diagrams do not show attributes for the sake of simplicity. In those that do, however, attributes are represented by oval shapes.



Attributes in ER diagrams, note that an attribute can have its own attributes (composite attribute)

Multivalued Attribute:-

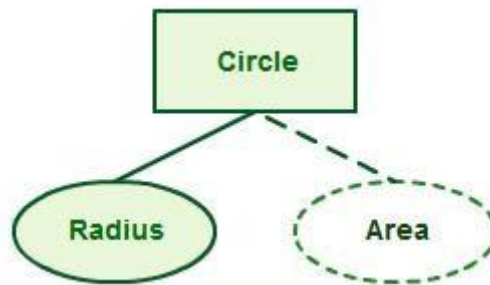
If an attribute can have more than one value it is called an multivalued attribute. It is important to note that this is different to an attribute having its own attributes. For example a teacher entity can have multiple subject values.



Example of a multivalued attribute

Derived Attribute:-

An attribute based on another attribute. This is found rarely in ER diagrams. For example for a circle the area can be derived from the radius.



Derived Attribute in ER diagrams

Relationship:-

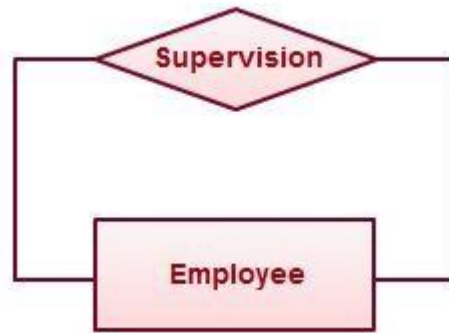
A relationship describes how entities interact. For example, the entity “carpenter” may be related to the entity “table” by the relationship “builds” or “makes”. Relationships are represented by diamond shapes and are labeled using verbs.



Using Relationships in Entity Relationship Diagrams

Recursive Relationship:-

If the same entity participates more than once in a relationship it is known as a recursive relationship. In the below example an employee can be a supervisor and be supervised, so there is a recursive relationship.



Example of a recursive relationship in ER diagrams

Cardinality and Ordinality:-

These two further defines relationships between entities by placing the relationship in the context of numbers. In an email system, for example, one account can have multiple contacts. The relationship in this case follows a “one to many” model. There are number of notations used to present cardinality in ER diagrams. Chen, UML, Crow’s foot, Bachman are some of the popular notations. [Creately](#) supports Chen, UML and Crow’s foot notations. The following example uses UML to show cardinality.



Cardinality in ER diagrams using UML notation

Most database management systems allow you to have more than one key so that you can sort records in different ways. One of the keys is designated the primary key, and must hold a unique value for each record. A key field that identifies records in a different table is called a foreign key.

KEYS in ER Model: Primary Key, Candidate Key, Super Key:-

Super Keys :- Super key stands for superset of a key.

A Super Key is a set of one or more attributes that are taken collectively and can identify all other attributes uniquely.

Candidate Keys:-

Candidate Keys are super keys for which no proper subset is a super key. In other words candidate keys are minimal super keys.

Primary Key:-

It is a candidate key that is chosen by the database designer to identify entities within an entity set. Primary key is the minimal super key. In the ER diagram primary key is represented by underlining the primary key attribute. Ideally a primary key is composed of only a single attribute. But it is possible to have a primary key composed of more than one attribute.

Composite Key:-

Composite key consists of more than one attributes.

Example: Consider a Relation or Table R1. Let A,B,C,D,E are the attributes of this relation.

$R(A,B,C,D,E)$

$A \rightarrow BCDE$ This means the attribute 'A' uniquely determines the other attributes B,C,D,E.

$BC \rightarrow ADE$ This means the attributes 'BC' jointly determine all the other attributes A,D,E in the relation.

Primary Key :A

Candidate Keys :A, BC

Super Keys : A,BC,ABC,AD

ABC,AD are not Candidate Keys since both are not minimal super keys.

PRACTICAL NO :-12

Aim:-Draw an ER Diagram for Library Management System which includes book_info, Staff_info, issue_of_book, return_of_book , fine_calculation.

Theory:-

In the following Library Management System, the following entities and attributes can be identified as following:

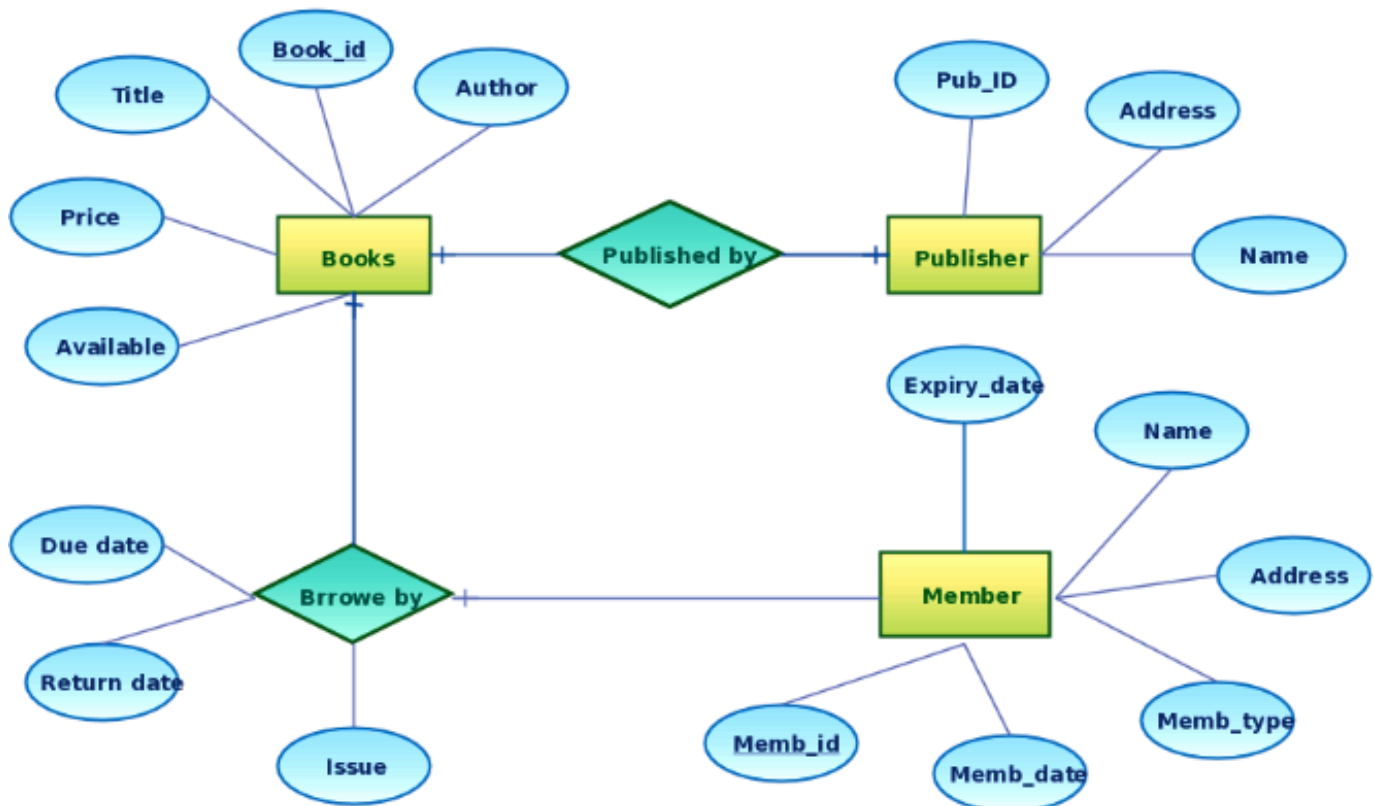
Book_Info:- The set of all the books in the library, each book has a book_id,title,author, and availability.

Member:- The set of all library members,the member is described by the attributes member_id, issue_date,name,street,city,zip_code etc.

Publisher:- The set of all publishers of the book, attributes are Pub_id,name,city, zip_code.

Supplier/borrow_by:- The set of all suppliers of the books, attributes of entity are sup_id, name, street , Zip_Code.

E-R Diagram for Library Management System



PRACTICAL NO :-13

Aim:- What do you understand from Data Flow Diagrams? Describe Graphical notations for Data flow Diagrams. Explain the Symbols used in DFD.

Theory:-

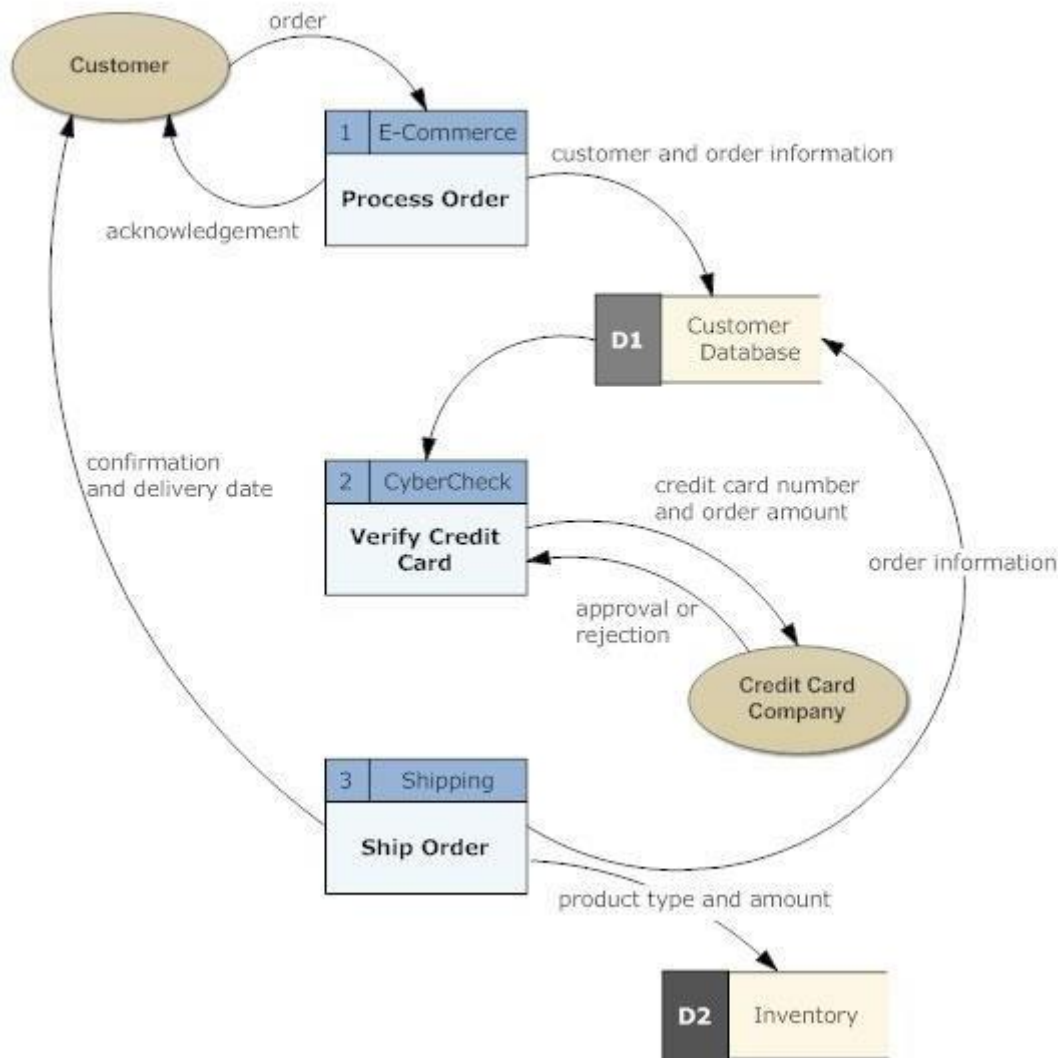
Data Flow Diagram:-

A data flow diagram (DFD) illustrates how data is processed by a system in terms of inputs and outputs. As its name indicates its focus is on the flow of information, where data comes from, where it goes and how it gets stored.

"A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. Often they are a preliminary step used to create an overview of the system which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design). A DFD shows what kinds of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of processes, or information about whether processes will operate in sequence or in parallel (which is shown on a flowchart)." [Data flow diagram. Wikipedia]

The Data Flow Diagrams solution from the Software Development area of ConceptDraw Solution Park provides three vector stencils libraries for drawing DFD using the ConceptDraw PRO diagramming and vector drawing software.

Data Flow Diagram - Online Order System



History of Data Flow Diagrams:-

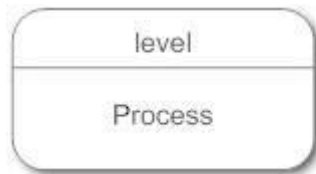
Data flow diagrams became popular in the 1970s in software development. They were first described in a classic text about *Structured Design* written by Larry Constantine and Ed Yourdon. Yourdon & Coad's Object Oriented Analysis and Design (OOA/OOD) was a way of visualizing software systems before UML diagrams.

Data Flow Diagrams Notations:-

There are essentially two different types of notations for data flow diagrams (Yourdon & Coad or Gane & Sarson) defining different visual representations for processes, data stores, data flow and external entities. Yourdon and Coad type data flow diagrams are usually used for system analysis and design, while Gane and Sarson type DFDs are more common for visualizing information systems.

Visually, the biggest difference between the two ways of drawing data flow diagrams is how processes look. In the Yourdon and Coad way, processes are depicted as circles, while in the Gane and Sarson diagram the processes are squares with rounded corners.

Process Notations. A process transforms incoming data flow into outgoing data flow.



Gane & Sarson

Datastore Notations. Datastores are repositories of data in the system. They are sometimes also referred to as files.

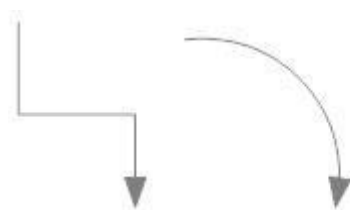


Yourdon & Coad





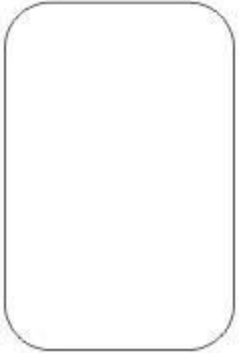
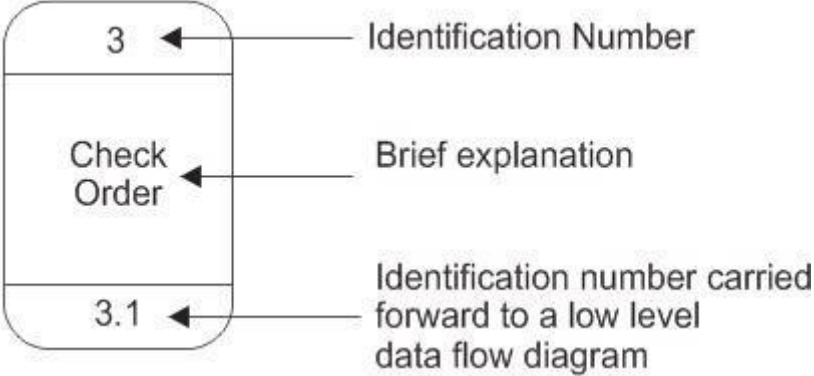




Gane & Sarson

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



Data flow

External Entity Notations. External entities are objects outside the system, with which the system communicates. External entities are sources and destinations of the system's inputs and outputs.

Symbol	Meaning	Example
	<p>An entity. A source of data or a destination for data.</p>	
	<p>A process or task that is performed by the system.</p>	
	<p>A data store, a place where data is held between processes.</p>	
	<p>A data flow.</p>	

PRACTICAL NO :-14

Aim:- Make a DFD for Library Management System.

Theory:-

