GOVT. POLYTECHNIC, NAGPUR

CERTIFICATE

This is to certify that Shri / Ku._____
_____

Enrollment No. _____

Term_____

Programme_____ has completed

term work / practicals in the Subject _____

Course code _____during the academic

year 202 - 202

Date: _____

Course Incharge H.O.D.

GOVT. POLYTECHNIC, NAGPUR

Programme: -AIML II Year	Term:-Even 24-25
Course Code:-AI205H	Course:-Software Engineering

S. No.	(Title of Experiment)	Page No.	Date of Per	Date of Sub	C.A. Grade/ Marks	Sign.
1.	Identify suitable software development model for the various scenario.					
2.	Identify number functional and non - functional requirements for various scenario.					
3.	Prepare Software Requirement Specification (SRS) document for the project management system.					
4.	Develop USE case diagrams for various scenario.					
5.	Develop E-R (Entity Relationship) diagram.					
6.	Develop DFD (Data Flow Diagram) .					
7.	Develop a Sequence diagram for various problem statement.					

8.	Develop project scheduling for system using GANTT chart and PERT chart.					
9.	Apply software modelling and data design concepts in software development.					
10.	Estimate time, size and cost for software product.					
11.	Estimate risk and Perform RMMM (Risk Monitoring Mitigation and Management).					
12.	Estimate scope and feasibility of a various systemin terms of technical aspects.					
13.	Estimate risk and impact factor for a various system.					
14.	Identify Software Quality Assurance techniques performed during different phases of a project for various system.					
15.	Perform Mini project, considering any one from various concepts for software engineering.					

Practical NO: 01

Aim: Apply suitable software development model for the given scenario.

Theory: o Software Development

Process:

A software development process is the process of dividing software development work into distinct phases to improve design, product management, and project management. It is also known as a software development life cycle. Most modern development processes can be vaguely described as agile. Other methodologies include waterfall, prototyping, iterative and incremental development, spiral development and rapid application development.

- Types of Development Models:
  1. Waterfall Model.
  2. Incremental Model.
  3. RAD Model.
  4. Prototyping.
  5. Spiral Model.

- Waterfall Model:
  1. The Waterfall Model is earliest SDLC approach.
  2. It is also known as Classic Life Cycle Model.
  3. In this Model, each phase must be completed before the next phase can begin.

- Features:
  1. It is good to use when technology is well understood.
  2. The project is short and cost is low.
  3. Risk is zero or simple and easy.

l . Advantages:

l. It is simple and easy.

2. Easy to manage.

3. It is good for low budget small project.

2. Disadvantages:
   l . It is not good for complex and object-oriented programming.
   2. It is a poor model for long and ongoing project.

3. Diagram:

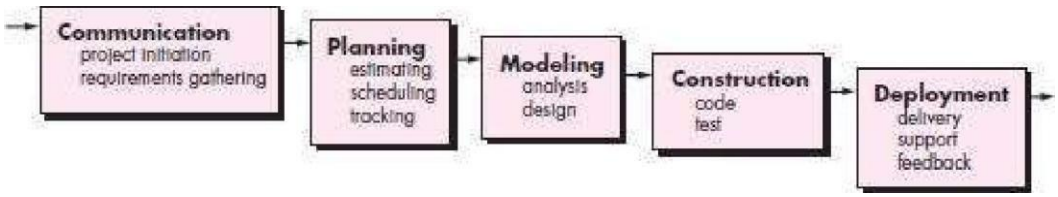


Fig. 1.1 Waterfall Model

Incremental Model:

Incremental model is a process of Software Development where requirements are broken down into multiple modules of Software Development Cycle. Increment means to add something. Incremental development is done in steps or in increments from communication, planning, modeling, construtions and deployment. Each iterations passes through all 5 phases . And each subsequent release of the system adds function to the previous relese until all designed fuctionality has been implemeted.

4. Features :
   1. Incremental is the combination of Linear and Prototype.
   2. Incremental Delivery.
   3. Priority to user's highly recommended requirements.
   4. Involvement of Users.
   5. Lower risk.
   6. Highest Priority System means in each increment they can easily find errors.
   7. At small requirements we can start our model.
   8. After every cycle the product is given to the customer.

5. Advantages:
    1. Work with small size team.
    2. Initial product delivery is faster.
    3. Customer response or feedback is considered.
    4. It is easier to test and Debug during a smaller iteration.
    5. The model is more flexible.
    6. Easier to manage risk.

6. Disadvantages:
    1. Actual cost will be more than Estimated cost.
    2. Needs good planning and design.

7. Diagram:

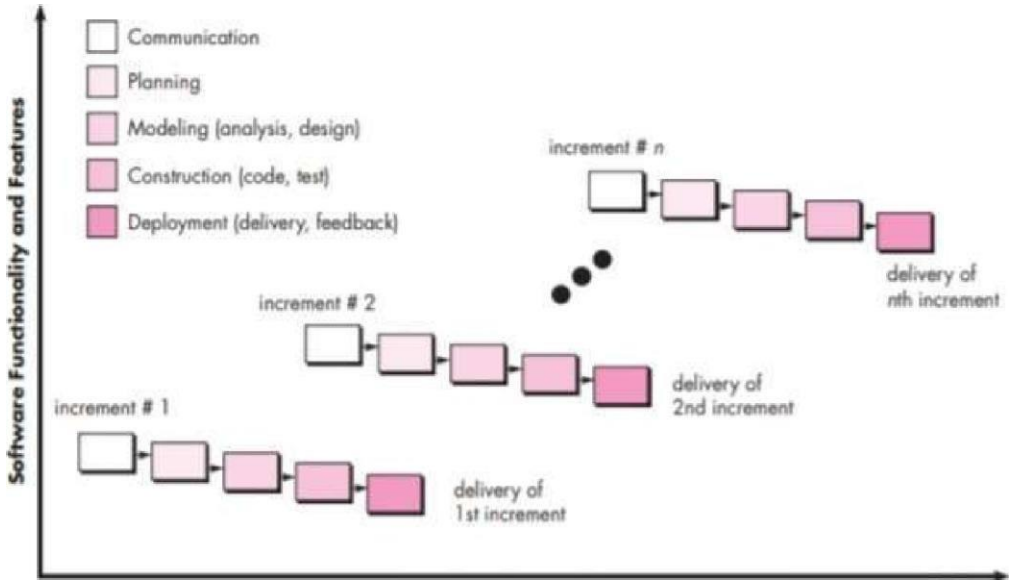


Fig. I .2 Incremental Model.

● RAD Model:

RAD or Rapid Application Development process is an adoption of the waterfall model; it targets at developing software in a short span of time. RAD follow the iterative.

● SDLC RAD model has following phases

1 . Business Modelling.
2 Data Modelling.
3 Process Modelling.
4 Application Generation.
5 Testing and Turnover.

It focuses on input-output source and destination of the information. It emphasizes on delivering proj ects in small pieces; the larger projects are divided into a series of smaller projects. The main features of RAD model are that it focuses on the reuse of templates, tools, processes, and code.

8. Different phases of RAD model include:

- Business Modelling:

  On basis of the flow of information and distribution between various business channels, the product is designed.

- Data Modelling:

  The information collected from business modelling is refined into a set of data objects that are significant for the business.

- Process Modelling

  The data object that is declared in the data modelling phase is transformed to achieve the information flow necessary to implement a business function.

- Application Generation

  Automated tools are used for the construction of the software, to convert process and data models into prototypes.

- Testing and Turnover

  As prototypes are individually tested during every iteration, the overall testing time is reduced in RAD.

9. When to use RAD Methodology?

1. When a system needs to be produced in a short span of time (2-3 months).
2. When the requirements are known.
3. When the user will be involved all through the life cycle.
4. When technical risk is less.

5. When there is a necessity to create a system that can be modularized in 2-3 months of time.
6. When a budget is high enough to afford designers for modelling along with the cost of automated tools for code generation.

10. Advantages:

1. Flexible and adaptable to changes.
2. It is useful when you have to reduce the overall project risk.
3. It is adaptable and flexible to changes.
4. It is easier to transfer deliverables as scripts, high-level abstractions and intermediate codes are used.
5. Due to code generators and code reuse, there is a reduction of manual coding.
6. Each phase in RAD delivers highest priority functionality to client.
7. With less people, productivity can be increased in short time.

11. Disadvantages:

1. It can't be used for smaller projects.
2. Not all application is compatible with RAD.
3. When technical risk is high, it is not suitable.
4. If developers are not committed to delivering software on time, RAD projects can fail.
5. Reduced features due to time boxing, where features are pushed to a later version to finish a release in short period.
6. Reduced scalability occurs because a RAD developed application begins as a prototype and evolves into a finished application.
7. Requires highly skilled designers or developers.

12. Diagram:

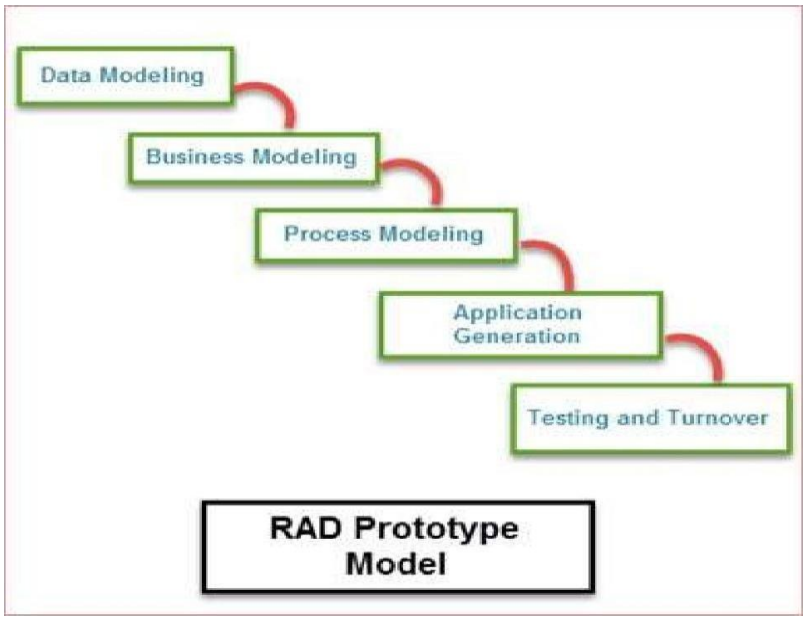


Fig.1.3 RAD Model. •

Prototyping Model:

In this model, a prototype of an end product is first developed, tested and defined as per customer's feedback repeatedly till the final expectable prototype is achieved. The intension behind creating this model is to get actual requirements more deeply from the user.

13. Process of Prototyping:

1. Initial requirement identification.
2. Prototype development.
3. Review.
4. Revise.

14. Advantages:

l . The customer gets to see the partial product early in the life cycle.

2. Missing functionality can be easily figured out.
3. Flexibility in design.
4. New requirements can be easily accommodated.

15.Disadvantages:

1. Costly with respect to time as well as money.
2. Poor documentation due to continuously change in requirements.
3. After seeing an early prototype, the customer demands the actual product will be delivered soon.

16.Diagram:

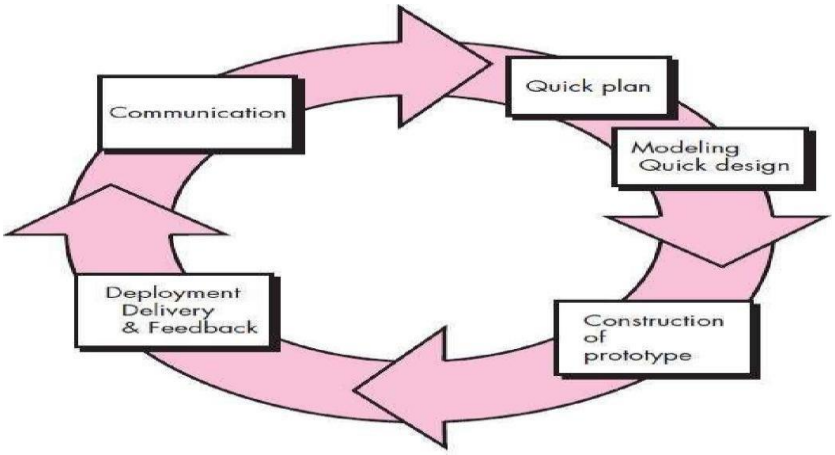


Fig. I .4 Prototype Model.

• Spiral Model:

The Spiral model was proposed by Barry Boehm. The Spiral model is the combination of Waterfall model and Incremental model. The Spiral model is divided into 4 task regions. Each task region begins with the design goal and ends with the client reviewing. Software is developed in the series of incremental release. The task region of Spiral model is: l . Concept Development.

2. System Development.
3. System Enhancement.
4. System Maintenance. 16. Features: l . When the release is frequent this model is used.
2. Used for large projects.
3. Compatible even if requirements are complex and unclear.
4. Changes can be done at any time.

17. Advantages: l . Additional functions or changes can be done at later stage.

2. Cost estimation becomes easy.
3. There is always space for customer's feedback.

18. Disadvantages:

l . Documentation is more as it has intermediate phases.
2. It is not advisable for smaller projects, as it may cost them a lot.

20. Diagram:

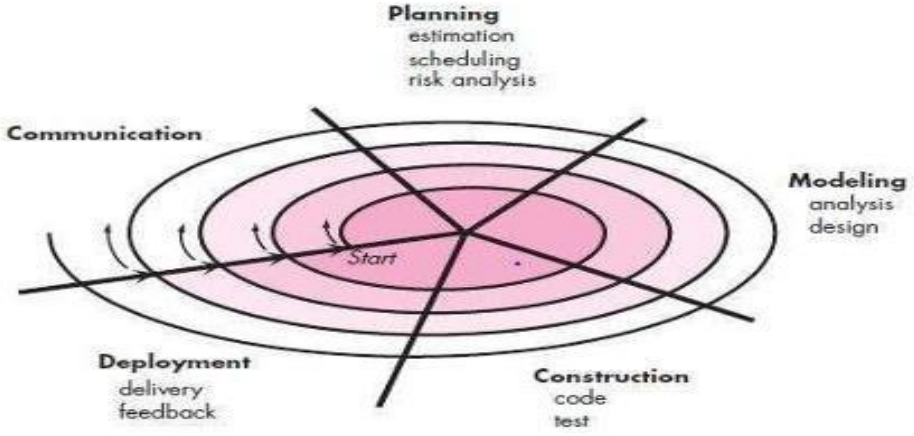


Fig. 1.5 Spiral Model.

● Scenario:

Questions:

1. Which model will you use?

2. Why this model? Elaborate.

- Conclusion:

(10)	(20)	(10)	(10)	Total (50)

Practical NO: 02

Aim: Classify above identified requirement into functional and non-functional requirements.

Theory:

- What is SRS?

  SRS stands for Software Requirement Specification. Software requirement is a functional or non-functional need to be implemented in the system. Functional means providing particular service to the user. For example, in context to banking application the functional requirement will be when customer select "View Balance" they must be able to look at their latest account balance. Software requirement can also be a non-functional, it can be a performance requirement. For example, a non-functional requirement is where every page of the system should be visible to the users within 5 seconds. So, basically Software requirement is a Functional or D Nonfunctional need that has to be implemented into the system. Software requirement is usually expressed as a statement.

- Functional Requirement:

  Functional requirements are the desired operations of program, or system as defined in software development and systems engineering. It describes the functions a software must perform. A function is nothing but inputs, its behaviour, and outputs. It can be a calculation, data manipulation, business process, user interaction, or any other functionality which defines what function a system is likely to perform. Functional Requirements are also called Functional Specification.

- Non-Functional Requirement:

  Non-functional requirements describe how the system works. A non-functional requirement defines the quality attribute of a

software system. They represent a set of standards used to judge the specific operation of system. A non-functional requirement is essential to ensure the usability and effectiveness of the entire software system. Non-functional requirements are often called Quality attributes of a system. Example, how fast does the website load?

- Scenario:

- Questions:

1. What are the Functional Requirements for above scenario?

2. What are Model: n-functional Requirements for above scenario?

3. What are Benefits of Non-functional Requirements?

4. Differentiate between Functional Requirements and Nonfunctional Requirements?

- Conclusion:

(10)	(20)	(10)	(10)	TOTAL

EXPERIMENT NO: 03

Aim: Prepare Software Requirement Specification (SRS) Document for the project management system.

Theory:

- **What is a Software Requirements Specification**?

A software requirements specification is a document which is used as a communication medium between the customer and the supplier. When the software requirement specification is completed and is accepted by all parties, the end of the requirements engineering phase has been reached. This is not to say, that after the acceptance phase, any of the requirements cannot be changed, but the changes must be tightly controlled. The software requirement specification should be edited by both the customer and the supplier, as initially neither has both the knowledge of what is required (the supplier) and what is feasible (the customer).

- **Why is a Software Requirement Specification Required?**

A software requirements specification has a number of purposes and contexts in which it is used. This can range from a company publishing a software requirement specification to companies for competitive tendering, or a company writing their own software requirement specification in response to a user requirement document. In the first case, the author of the document has to write the document in such a way that it is general enough as to allow a number of different suppliers to propose solutions, but at the same time containing any constraints which must be applied. In the second instance, the software requirement specification is used to capture the user's requirements and if any, highlight any inconsistencies and conflicting requirements and define system and acceptance testing activities.

A software requirement specification in its most basic form is a formal document used in communicating the software requirements between the customer and the developer. With this in mind then the minimum amount of information that the software requirement specification should contain is a list of requirements which has been

agreed by both parties. The requirements, to fully satisfy the user should have the However the requirements will only give a narrow view of the system, so more information is required to place the system into a context which defines the purpose of the system, an overview of the systems functions and the type of user that the system will have. This additional information will aid the developer in creating a software system which will be aimed at the user's ability and the client's function.

- **Types of Requirements**

Whilst requirements are being collated and analysed, they are segregated into type categories. The European Space Agency defined possible categories as

- Functional requirements,
- Performance requirements,
- Interface requirements,
- Operational requirements,
- Resource requirements,
- Verification requirements,
- Acceptance testing requirements,
- Documentation requirements,
- Quality requirements,
- Safety requirements,
- Reliability requirements and
- Maintainability requirements

- **Functional Requirements**

Functional or behavioural requirements are a sub-set of the overall system requirements. These requirements are used to consider trade-offs, system behaviour, redundancy and human aspects. Trade-offs may be

between hardware and software issues, weighing up the benefits of each. Behavioural requirements, as well as describing how the system will operate under normal operation should also consider the consequences and response due to software failure or invalid inputs to the system.

- **Performance Requirements**

All performance requirements must have a value which is measurable and quantitative, not a value which is perceptive. Performance requirements are stated in measurable values, such as rate, frequency, speeds and levels. The values specified must also be in some recognised unit, for example metres, centimetre square, BAR, kilometres per hour, etc. The performance values are based either on values extracted from the system specification, or on an estimated value.

- **Interface Requirements**

Interface requirements, at this stage are handled separately, with hardware requirements being derived separately from the software requirements. Software interfaces include dealing with an existing software system, or any interface standard that has been requested. Hardware requirements, unlike software give room for trade-offs if they are not fully defined, however all assumptions should be defined and carefully documented.

- **Operational Requirements**

Operational requirements give an "in the field" view to the specification, detailing such things as:

- how the system will operate,
- what is the operator syntax?
- how the system will communicate with the operators,
- how many operators are required and their qualification?
- what tasks will each operator be required to perform?
- what assistance/help is provided by the system,
- any error messages and how they are displayed, and
- what the screen layout looks like.

- **Resource Requirements**

Resource requirements divulge the design constraints relating to the utilisation of the system hardware. Software restrictions may be placed on only using specific, certified, standard compilers and databases. Hardware restrictions include amount, percentage or mean use of the available memory and the amount of memory available. The definition of available hardware is especially important when the extension of the hardware, late in the development life cycle is impossible or expensive.

- **Verification Requirements**

Verification requirements take into account how customer acceptance will be conducted at the completion of the project. Here a reference should be made to the verification plan document. Verification requirements specify how the functional and the performance requirements are to be measured and verified. The measurements taken may include simulation, emulation and live tests with real or simulated inputs. The requirements should also state whether the measurement tests are to be staged or completed on conclusion of the project, and whether a representative from the client's company should be present.

- **Acceptance Testing Requirements**

Acceptance test requirements detail the types of tests which are to be performed prior to customer acceptance. These tests should be formalised in an acceptance test document.

- **Documentation Requirements**

Documentation requirements specify what documentation is to be supplied to the client, either through or at the end of the project. The documentation supplied to the client may include project specific documentation as well as user guides and any other relevant documentation.

- **Quality Requirements**

Quality requirements will specify any international as well as local standards which should be adhered to. The quality requirements should be addressed in the quality assurance plan, which is a core part of the quality assurance document. Typical quality requirements include following ISO9000-3 procedures. The National Aeronautics and Space Administration's software requirement specification - SFW-DID-08 goes to

the extent of having subsections detailing relevant quality criteria and how they will be met. These sections are Quality Factors:

- Correctness
- Reliability
- Efficiency
- Integrity
- Usability
- Maintainability
- Testability
- Flexibility
- Portability
- Reusability
- Interoperability
- Additional Factors

Some of these factors can be addressed directly by requirements, for example, reliability can be stated as an average period of operation before failure. However most of the factors detailed above are subjective and may only be realised during operation or post-delivery maintenance. For example, the system may be vigorously tested, but it is not always possible to test all permutations of possible inputs and operating conditions. For this reason, errors may be found in the delivered system. With correctness the subjectifies of how correct the system is, is still open to interpretation and needs to be put into context with the overall system and its intended usage. An example of this can be taken from the recently publicised 15th point rounding error found in Pentium (processors. In the whole most users of the processor will not be interested in values of that order, so as far as they are concerned, the processor meets their correctness quality criteria, however a laboratory assistant performing minute calculations for an experiment this level or error may mean that the processor does not have the required quality of correctness.

- **Safety Requirements**

Safety requirements cover not only human safety, but also equipment and data safety. Human safety considerations include protecting the operator from moving parts, electrical circuitry and other physical dangers. There

may be special operating procedures, which if ignored my lead to a hazardous or dangerous condition occurring. Equipment safety includes safeguarding the software system from unauthorised access either electronically or physically. An example of a safety requirement may be that a monitor used in the system will conform to certain screen emission standards or that the system will be installed in a Faraday Cage with a combination door lock.

- **Reliability Requirements**

Reliability requirements are those which the software must meet in order to perform a specific function under certain stated conditions, for a given period of time. The level of reliability requirement can be dependent on the type of system, i.e. the more critical or life threatening the system, the higher the level of reliability required. Reliability can be measured in a number of ways including number of bugs per x lines of code, mean time to failure and as a percentage of the time the system will be operational before crashing or an error occurring. Davis states however that the mean time to failure and percent reliability should not be an issue as if the software is fully tested, the error will either show itself during the initial period of use, if the system is asked to perform a function it was not designed to do or the hardware/software configuration of the software host has been changed. Davis suggests the following hierarchy when considering the detail of reliability in a software requirement specification.

- Destroy all humankind
- Destroy large numbers of human beings
- Kill a few people
- Injure people
- Cause major financial loss
- Cause major embarrassment
- Cause minor financial loss
- Cause mild inconvenience
- Naming conventions
- Component headers

- In-line document style
- Control constructs
- Use of global/common variables

- **Maintainability Requirements**

Maintainability requirements look at the long-term lift of the proposed system. Requirements should take into consideration any expected changes in the software system; any changes of the computer hardware configuration and special consideration should be given to software operating at sites where software support is not available. Davis suggests defining or setting a minimum standard for requirements which will aid maintainability i.e.

- **Characteristics of a Good Software Requirements Specification**

A software requirements specification should be clear, concise, consistent and unambiguous. It must correctly specify all of the software requirements, but no more. However, the software requirement specification should not describe any of the design or verification aspects, except where constrained by any of the stakeholder's requirements.

- **Complete**

For a software requirements specification to be complete, it must have the following properties:

Description of all major requirements relating to functionality, performance, design constraints and external interfaces.

Definition of the response of the software system to all reasonable situations.

Conformity to any software standards, detailing any sections which are not appropriate

Have full labelling and references of all tables and references, definitions of all terms and units of measure.

Be fully defined, if there are sections in the software requirements specification still to be defined, the software requirements specification is not complete.

- **Consistent**

A software requirement specification is consistent if none of the requirements conflict. There are a number of different

types of confliction:

Multiple descriptors - This is where two or more words are used to reference the same item, i.e. where the term cue and prompt are used interchangeably.

Opposing physical requirements - This is where the description of real-world objects clash, e.g. one requirement states that the warning indicator is orange, and another states that the indicator is red.

Opposing functional requirements - This is where functional characteristics conflict, e.g. perform function X after both A and B has occurred, or perform function X after A or B has occurred.

- **Traceable**

A software requirement specification is traceable if both the origins and the references of the requirements are available. Traceability of the origin or a requirement can help understand who asked for the requirement and also what modifications have been made to the requirement to bring the requirement to its current state. Traceability of references are used to aid the modification of future documents by stating where a requirement has been referenced. By having foreword traceability, consistency can be more easily contained.

- **Unambiguous**

As the Oxford English dictionary states the word unambiguous means "not having two or more possible meanings". This means that each requirement can have one and only one interpretation. If it is unavoidable to use an ambiguous term in the requirements specification, then there should be clarification text describing the context of the term. One way of removing ambiguity is to use a formal requirements specification language. The advantage to using a formal language is the relative ease of detecting errors by using lexical syntactic analysers to detect ambiguity. The disadvantage of using a formal requirements specification language is the learning time and loss of understanding of the system by the client.

- **Verifiable**

A software requirement specification is verifiable if all of the requirements contained within the specification are verifiable. A requirement is verifiable if there exists a finite cost-effective method by which a person or machine can check that the software product meets the requirement. Non-verifiable requirements include "The system should have a good user interface" or "the

software must work well under most conditions" because the performance words of good, well and most are subjective and open to interpretation. If a method cannot be devised to determine whether the software meets a requirement, then the requirement should be removed or revised.

- **Scenario:**

INTRODUCTION:

OVERALL DESCRIPTION

SPECIFICATION REQUIREMENTS

- **Appendices:**

- **Index:**

- **Conclusion:**

(10)	(20)	(10)	(10)	TOTAL (50)

Practical No. 04

Aim:- Develop USE case diagrams for various scenario.

Theory :-

- **What is a USE case diagram?**

In the Unified Modelling Language (UML), a use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use a set of specialized symbols and connectors. An effective use case diagram can help your team discuss and represent:

  - Scenarios in which your system or application interacts with people, organizations, or external systems.
  - Goals that your system or application helps those entities (known as actors) achieve.
  - The scope of your system.

- **When to apply use case diagrams**

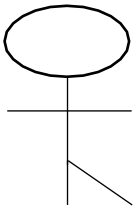
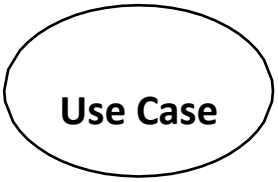
Use case diagrams specify the events of a system and their flows. But use case diagram never describes how they are implemented. Use case diagram can be imagined as a black box where only the input, output, and the function of the black box is known.

These diagrams are used at a very high level of design. This high- level design is refined again and again to get a complete and practical picture of the system. A well-structured use case also describes the pre-condition, post condition, and exceptions. These extra elements are used to make test cases when performing the testing.

Use case diagrams can be used for –

- Requirement analysis and high-level design.
- Model the context of a system.
- Reverse engineering
- Forward engineering

Notations:-

Notation Description	**Visual Representation**
Actor **Someone interacts with use case (system function).** **Similar to the concept of user, but a user can play different roles** **Actor has a responsibility toward the system (inputs), and Actor has expectations from the system (outputs).**	
Use Case **System function (process - automated or manual)** **i.e. Do something** **Each Actor must be linked to a use case, while some use cases may not be linked to actors.**	

Communication Link **Actors may be connected to use cases by associations, indicating that the actor and the use case communicate with one another using messages.**	————
Boundary of system **The system boundary is potentially the entire system as defined in the requirements document.** **For example, for an ERP system for an organization, each of the modules such as personnel, payroll, accounting, etc.**	

Relationships in Use Case Diagrams

There are five types of relationships in a use case diagram. They are

- Association between an actor and a use case
- Generalization of an actor
- Extend relationship between two use cases
- Include relationship between two use cases
- Generalization of a use case

- **Scenario:**

- **Questions:**

1. **What is the Importance of Use Case Diagrams?**

1. **Draw a USE case diagram for given scenario?**

- **Conclusion:**

(10)	(20)	(10)	(10)	TOTAL

PRACTICAL NO: 05

Aim: Develop E-R (Entity Relationship) diagram.

Theory:

- **What is Entity?**

An entity is any object in the system that we want to model and store information about. Entities are usually recognizable concepts, either concrete or abstract, such as person, places, things, or events which have relevance to the database.

Some specific examples of entities are Employee, Student, Lecturer.

- **What is E-R diagram?**

An entity relationship model, also called an entity - relationship (ER) diagram, is a graphical representation of entities and their relationships to each other, typically used in computing in regard to the organization of data within databases or information systems.

For example: In the following ER diagram we have - two entities Student and College and these two entities have many to one relationship as many student's study in a single college**.**

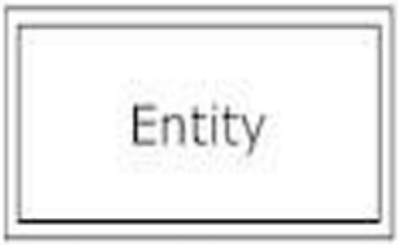
- **ER Diagram Uses:**

When documenting a system or process, looking at the system in multiple ways increases the understanding of that system.

ERD diagrams are commonly used in conjunction with a data flow diagram to display the contents of a data store.

- **Common Entity Relationship Diagram Symbols:**
  - **Entities,** which are represented by rectangles. An entity is an object or concept about which you want to store information.

  - **weak entity** is an entity that must defined by a foreign key relationship with another entity as it cannot be uniquely identified by its own attributes alone.



  - **Relationship**, which are represented by diamond shapes, show how two entities share information in the database.

- **Attributes,** which are represented by ovals. A key attribute is the unique, distinguishing characteristic of the entity.

- **multivalued attribute** can have more than one value. For example, an employee entity can have multiple skill values.

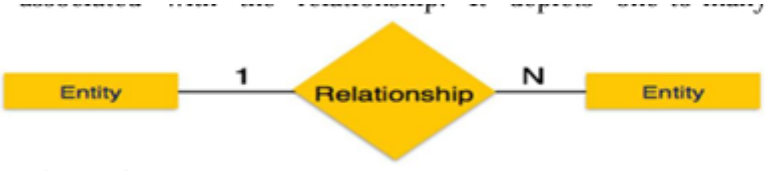
- **Connecting lines,** solid lines that connect attributes to show the relationships of entities in the diagram.

- **Relationship**

Relationships are represented by diamond-shaped box. Name of the relationship is written inside the diamond-box. All the entities (rectangles) participating in a relationship, are connected to it by a line.

- **<u>One-to-one</u>** − When only one instance of an entity is associated with the relationship, it is marked as '1:1'. The following image reflects that only one instance of each entity should be associated with the relationship. It depicts one-to-one relationship.

- **<u>One-to-many</u>** − When more than one instance of an entity is associated with a relationship, it is marked as '1: N'. The following image reflects that only one instance of entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts one-to-many relationship.



- **Many-to-one** − When more than one instance of entity is associated with the relationship, it is marked as 'N:1'. The following image reflects that more than one instance of an entity on the left and only one instance of an entity on the right can be associated with the relationship

- **Many-to-many** − The following image reflects that more than one instance of an entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts many-to-many relationship.

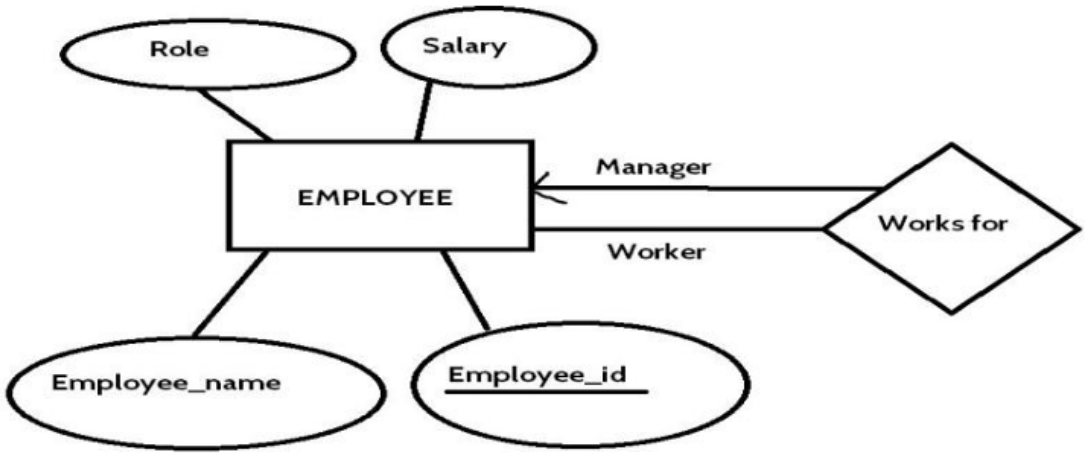
• How to Draw ER Diagrams?

Below points show how to go about creating an ER diagram.

1. Identify all the entities in the system. An entity should appear only once in a particular diagram. Create rectangles for all entities and name them properly.

2. Identify relationships between entities. Connect them using a line and add a diamond in the middle describing the relationship.

3. Add attributes for entities. Give meaningful attribute names so they can be understood easily



- **Advantages of ER Diagram:**

Conceptually it is very simple: ER model is very simple because 1 6 if we know relationship between entities and attributes, then we can easily draw an ER diagram.

Better visual representation: ER model is a diagrammatic representation of any logical structure of database. By seeing ER diagram, we can easily understand relationship among entities and relationship.

Effective communication tool: It is an effective communication tool for database designer.

Highly integrated with relational model: ER model can be easily converted into relational model by simply converting ER model into tables.

Easy conversion to any data model: ER model can be easily converted into another data model like hierarchical data model, network data model and so on.

- **Disadvantages of ER Diagram:**

Limited constraints and specification.

Loss of information content: Some information be lost or hidden in ER model.

Limited relationship representation: ER model represents limited relationship as compared to another data models like relational model etc.

No representation of data manipulation: It is difficult to show data manipulation in ER model.

Popular for high level design: ER model is very popular for designing high level design

- **Scenario:**

- **Questions: 1. Draw the ER diagram for given scenario ?**

- **Conclusion:**

(10)	(20)	(10)	(10)	TOTAL

PRACTICAL NO : 06

Aim : Develop DFD (Data Flow Diagram) .

Theory:

- What is Data Flow Diagram?

Data flow diagrams are used to graphically represent the flow of data in a business information system. DFD describes the processes that are involved in a system to transfer data from the input to the file storage and reports generation.

Data flow diagrams can be divided into logical and physical. The logical data flow diagram describes flow of data through a system to perform certain functionality of a business. The physical data flow diagram describes the implementation of the logical data flow.

- DFD Symbols:

There are four basic symbols that are used to represent a data-flow diagram.

- Process

A process receives input data and produces output with a different content or form. Processes can be as simple as collecting input data and saving in the database, or it can be

complex as producing a report containing monthly sales of all retail stores in the northwest region.

Every process has a name that identifies the function it performs. The name consists of a verb, followed by a singular noun.

Example:

- Apply Payment
- Calculate Commission
- Verify Order

- Data Flow

A data-flow is a path for data to move from one part of the information system to another. A data-flow may represent a single data element such the Customer ID or it can represent a set of data element (or a data structure).

Example:

- Customer info (Last Name, FirstName, SS#, Tel #, etc.)
- Order info (Ordered, Item#, Order Date, Customer, etc.).

- Data Store

A data store or data repository is used in a data-flow diagram to represent a situation when the system must retain data

because one or more processes need to use the stored data in a later time.

- External Entity

It Is also known as actors, sources or sinks, and terminators, external entities produce and consume data that flows between the entity and the system being diagrammed. These data flows are the inputs and outputs of the DFD.

- How to draw a data flow diagram?

Lucid chart makes it easy to create a customized data flow diagram starting with a simple template. Choose the symbols you need from our library—processes, data stores, data flow, and external entities— and drag-and-drop them into place. Since Lucid chart is an online tool, it facilitates collaboration and bypasses the hassles of desktop DFD software.

- Levels in Data Flow Diagrams (DFD)

In Software engineering DFD (data flow diagram) can be drawn to represent the system of different levels of abstraction. Higher level DFDs are partitioned into low levels-hacking more information and functional elements. Levels in

DFD are numbered 0, 1, 2 or beyond. Here, we will see mainly 3 levels in data flow diagram, which are: 0- level DFD, 1-level DFD, and 2-level DFD.

0- level DFD:

It is also known as context diagram. It's designed to be an abstraction view, showing the system as a single process with its relationship to external entities. It represents the entire system as single bubble with input and output data indicated by incoming/outgoing arrows.

1- level DFD:

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

2- level DFD:

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

- Advantages of data flow diagram:
  - A simple graphical technique which is easy to understand.
  - It helps in defining the boundaries of the system.
  - It is useful for communicating current system knowledge to the users.
  - It is used as the part of system documentation file.
  - It explains the logic behind the data flow within the system.

- Disadvantages of data flow diagram:
  - Data flow diagram undergoes lot of alteration before going to users, so makes the process little slow.
  - Physical consideration is left out. It makes the programmers little confusing towards the system.

Scenario: 1.

- **Questions:**

  1) Draw DFD diagram for given scenario ?

  Level 0 :

  Level 1 :

Level 2 :

- **Conclusion:**

10	20	10	10	Total

PRACTICAL NO: 07

Aim: Develop a Sequence diagram for various problem statement.

Theory:

- What is a Sequence diagram?

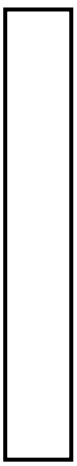
A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are sometimes called event diagrams or event scenarios. Sequence diagrams are preferred by both developers and readers alike for their simplicity.

A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

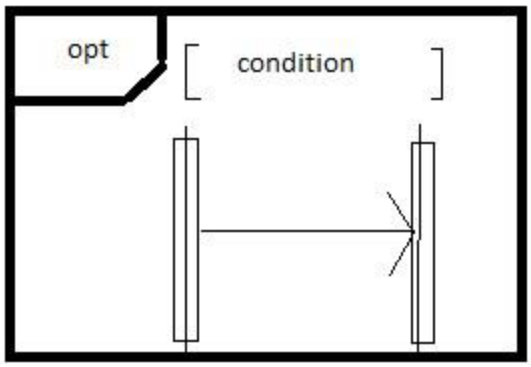
Sequence Diagram (Notations):

Object/Actor :

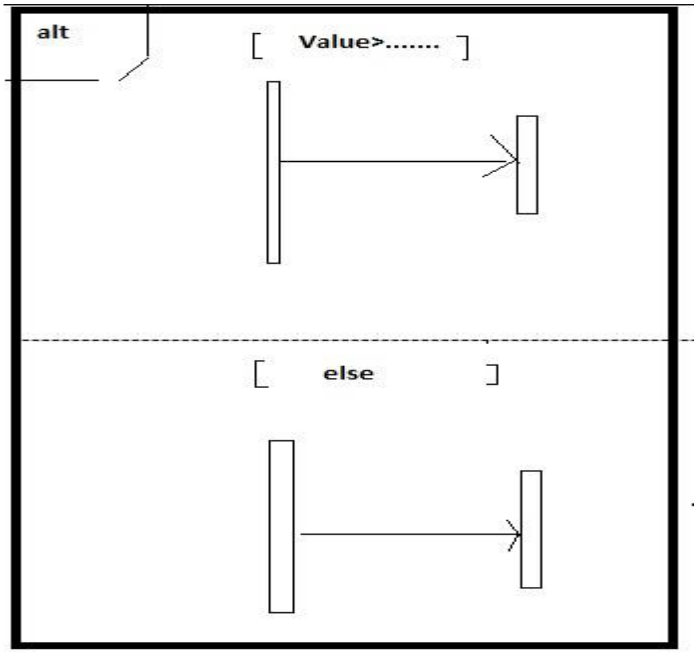
Active on lifeline:



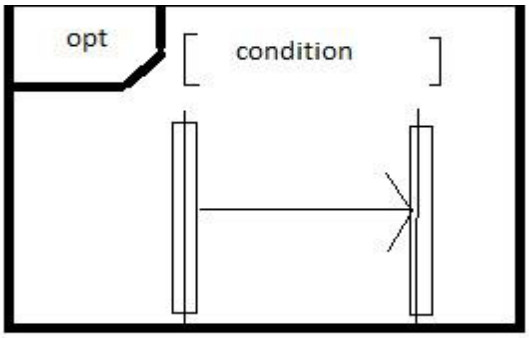
If :



If else :



Loop:



1. Draw Sequence diagram for given scenario.

   Scenario:

2. State the Notations of Sequence Diagram .

3. State the importance of Sequence Diagram.

- **Conclusion:**

(10)	(20)	(10)	(10)	TOTAL

Practical No . 8

AIM : Develop project scheduling for system using GANTT chart and PERT chart.

THEORY:

GANTT CHART

- Time-line chart, also called a Gantt chart,A time-line chart can be developed for the sentire project.
- Alternatively, separate charts can be developed for each project function or for each individual working on the project.
- Figure illustrates the format of a time-line chart. It depicts a part of a software project schedule
- All project tasks (for concept scoping) are listed in the lefthand column.
- T he horizontal bars indicate the duration of each task.
- When multiple bars occur at the same time on the calendar, task concurrency is implied.
- The diamonds indicate milestones.

PERT CHART

Definition: A flowchart-like tool used to represent and analyze the tasks involved in a project, especially when dealing with uncertain time estimates.

- **Purpose:** Focuses on identifying the critical path—the sequence of tasks that directly affects the project's completion time.
- **Features:**
    - Uses nodes to represent events (start or completion of tasks) and arrows to depict tasks and their sequence.
    - Provides three time estimates for each task: optimistic, pessimistic, and most likely.
    - Helps in identifying project risks and efficient time management.

ECT (Earliest Completion Time):

- The earliest time by which a task or event in a project can be completed, assuming all its preceding tasks are finished on time.
- It is calculated by moving **forward through the network diagram**, adding the duration of each task as you progress.

LCT (Latest Completion Time):

- The latest time by which a task or event must be completed without causing any delay to the project timeline.
- It is determined by moving **backward through the network diagram**, starting from the project's final completion time.

Critical Path:

- Represents the **longest sequence of dependent tasks** in a project.
- Tasks on the critical path have zero slack, meaning any delay in these tasks will directly delay the overall project.
- Identifying the critical path helps project managers focus on the most important tasks that require strict monitoring.

Q.1) Draw a GANTT CHART for considered data.

Work tasks	Planned start	Actual start	Planned complete	Actual complete	Assigned person	Effort allocated	Notes
I.1.1 Identify needs and benefits							Scoping will require more effort/time
Meet with customers	wk1, d1	wk1, d1	wk1, d2	wk1, d2	BLS	2 p-d	
Identify needs and project constraints	wk1, d2	wk1, d2	wk1, d2	wk1, d2	JPP	1 p-d	
Establish product statement	wk1, d3	wk1, d3	wk1, d3	wk1, d3	BLS/JPP	1 p-d	
Milestone: Product statement defined	wk1, d3	wk1, d3	wk1, d3	wk1, d3			
I.1.2 Define desired output/control/input (OCI)							
Scope keyboard functions	wk1, d4	wk1, d4	wk2, d2		BLS	1.5 p-d	
Scope voice input functions	wk1, d3	wk1, d3	wk2, d2		JPP	2 p-d	
Scope modes of interaction	wk2, d1		wk2, d3		MLL	1 p-d	
Scope document diagnostics	wk2, d1		wk2, d2		BLS	1.5 p-d	
Scope other WP functions	wk1, d4	wk1, d4	wk2, d3		JPP	2 p-d	
Document OCI	wk2, d1		wk2, d3		MLL	3 p-d	
FTR: Review OCI with customer	wk2, d3		wk2, d3		all	3 p-d	
Revise OCI as required	wk2, d4		wk2, d4		all	3 p-d	
Milestone: OCI defined	wk2, d5		wk2, d5				
I.1.3 Define the function/behavior							

ANSWER:-

Q.2) Draw a PERT CHART for considered data.

TASK ID	Task Description	Preceed ID	Succ. ID	Duration
A	Specification	1	2	3
B	High Level Design	2	3	2
C	Detailed Design	3	4	2
D	Code/Test Main	4	5	7
E	Code/Test DB	4	6	6
F	Code/Test UI	4	7	3
G	Write test plan	4	8	2
	Dummy Task	5	8	
	Dummy Task	6	8	
	Dummy Task	7	8	
H	Integrate/System Test	8	9	5
I	Write User Manual	8	10	2
J	Typeset User Manual	10	9	1

ANSWER:-

1) Construct a project network.

2) Calculate ECT

3) Calculate LCT .

4) Find the CPM.

CONCLUSION:-

(10)	(20)	(10)	(10)	Total (50)

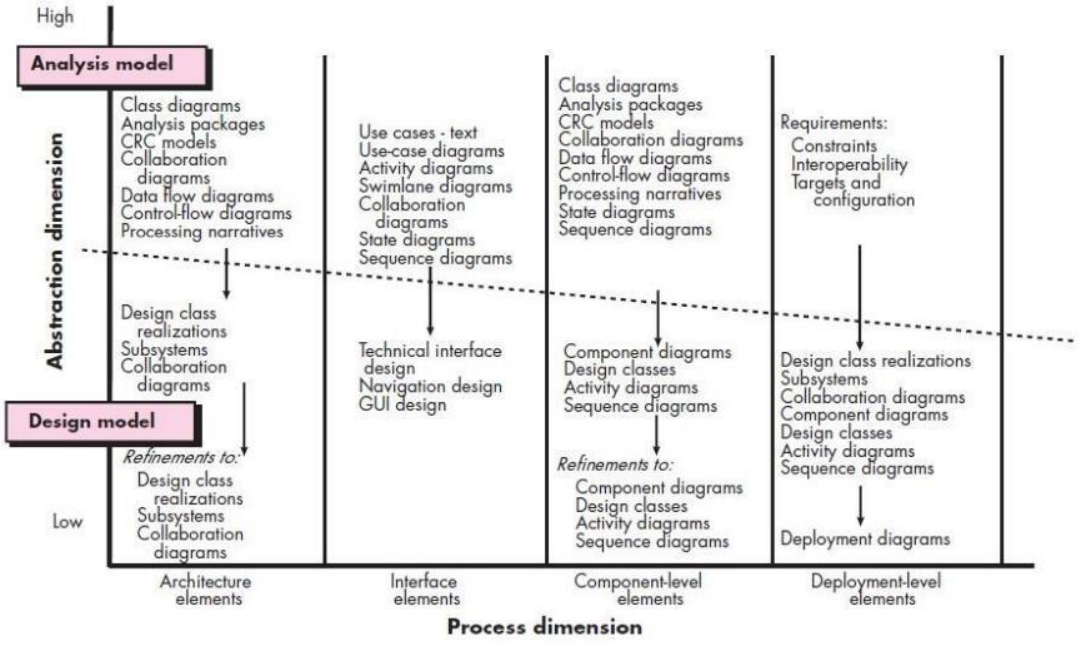
EXPERIMENT NO:09

Aim: Apply software modeling and data design concepts in software development.

Theory:

Design Model

- The design model can be viewed in two different dimensions
- The process dimension indicates the evolution of the design model as design tasks are executed as part of the software process.
- The abstraction dimension represents the level of detail as each element of the analysis model is transformed into a design equivalent and then refined iteratively. As shown fig
- The dashed line indicates the boundary between the analysis and design models.
- In some cases, a clear distinction between the analysis and design models is possible. In other cases, the analysis model slowly blends into the design and a clear distinction is less obvious.
- The elements of the design model use many of the same UML diagrams that were used in the analysis model.
- The difference is that these diagrams are refined and elaborated as part of design; more implementation-specific detail is provided



Types of design elements:-

1. Data Design Elements
2. Architectural Design Elements
3. Interface Design Elements
4. Component-level diagram elements
5. Deployment-level design elements

Data Design:-

- Like other software engineering activities, data design creates a model of data and/or information that is represented at a high level of abstraction
- This data model refined into progressively more implementationspecific representations that can be processed by the computerbased system.
- In many software applications, the architecture of the data will have a profound influence on the architecture of the software that must process it.
- The structure of data has always been an important part of software design.
- At the program component level, the design of data structures and the associated algorithms required to manipulate them is essential to the creation of high-quality applications.

At the application level, the translation of a data model-into a database is pivotal
- to achieving the business objectives of a system.
- At the business level, the collection of information stored in disparate databases and reorganized into a “data warehouse” enables data mining or knowledge discovery that can have an impact on the success of the business itself.

Scenario:

-

Questions:

- **1. What are Data design elements?**

2. Draw a model for the given scenario using Data design Model

3. **Why was Data design Model used instead of the other models?**

4. **State another detailed scenario where you can use Data Design Model.**

Conclusion:

(10)	(20)	(10)	(10)	Total(50)

EXPERIMENT NO: 10

Aim: Estimate time, size and cost for software product.

Theory:

Observations on Estimating:

• Estimation of **resources, cost, and schedule** for a software engineering effort requires experience in it

• It's an access **to good historical information, requires for the development of software.**

• Although **estimating is as much art as it is science**, this important action need not be conducted

• **Useful techniques** for time and effort estimation do exist.

• Process and project metrics can provide historical perspective and powerful input for the generation of quantitative estimates.

• **Past experience** can aid immeasurably as estimates are developed and reviewed.

• Because estimation lays **a foundation for all other project planning actions, and project planning provides the road map for successful software engineering**,

• Estimation of resources, cost, and schedule for a software engineering effort requires experience, access to good historical information (metrics), and the courage to commit to quantitative predictions when qualitative information is all that exists.

• Problem decomposition, an important approach to estimating, becomes more difficult.

• The availability of historical information has a strong influence on estimation risk. By looking back, you can emulate things that worked and improve areas where problems arose.

• **When comprehensive software metrics are available for past projects, estimates can be made with greater assurance,**

schedules can be established to avoid past difficulties, and overall risk is reduced.

• Estimation risk is measured by the degree of uncertainty in the quantitative estimates established for resources, cost, and schedule.

• **If project scope is poorly understood** or project requirements are subject to change

• As a planner, you and the customer should recognize **that variability in software requirements means instability in cost and schedule**.

• However, you should not become obsessive about estimation. Modern software engineering approaches take an iterative view of development.

Software Project Estimation:

• Software cost and effort estimation will never be an exact science. Too many variables—**human, technical, environmental**, political—can affect the ultimate cost of software and effort applied to develop it.

• To achieve reliable cost and effort estimates, a number of options arise:

1. We can achieve 100 percent accurate estimates **after the project is Complete.**
2. Base **estimates on similar projects** that have already been completed.
3. Use relatively **simple decomposition techniques** to generate project
cost and effort estimates.
4. Use one or more empirical models for software cost and effort estimation.

• **Longer you wait, the more you know**, and the more you know, the less likely you are to make serious errors in your estimates.

• If the current project is quite similar to past efforts and other project influences (e.g., the customer, business conditions, the software engineering environment, deadlines) are roughly equivalent.

• Ideally, the techniques noted for each option should be applied in tandem; each used as a **cross-check** for the other.

• **Decomposition techniques take a divide-and-conquer** approach to software project estimation.

• By decomposing a project into major functions and related software engineering activities, cost and effort estimation can be performed in a stepwise fashion.

• If no historical data exist, costing rests on a very based foundation.

Decomposition Techniques:

• Software project estimation is a form of problem solving,

• Estimation uses one or both forms of partitioning.

• But before an estimate can be made, you must understand the scope of the software to be built and generate an estimate of its "size."

• **Software Sizing** The accuracy of a software project estimate is predicated on a number of things:

The degree to which you have properly estimated the **size of the product** to be built;

The ability to translate the **size estimate into human effort, calendar time, and costs**

The degree to which the project plan reflects the **abilities of the software team**;

The stability of product requirements and the environment that supports the **software engineering effort**.

• In the context of project planning, **size refers to a quantifiable outcome of the software project**.

• If a **direct approach** is taken, size can be measured in lines of code **(LOC).**

• If **an indirect approach** is chosen, size is represented **as function points** (FP).

Various approaches to the sizing problem:

"**Fuzzy logic" sizing**. To apply this approach, the **planner must identify the type of application**, establish its magnitude on a qualitative scale, and then refine the magnitude within the original range.

Function point sizing. The **planner develops estimates of the information domain** characteristics discussed.

Standard component sizing. Software is composed of a number of different "standard components" that are generic to a particular application area. For example, the standard components for an information system are subsystems, modules, screens, reports, interactive programs, batch programs, files, LOC, and object-level instructions.

Change sizing. use of existing software that must be modified in some way as part of a project. The planner estimates the number and type e.g., reuse, adding code, changing code, deleting code of modifications that must be accomplished.

Example: LOC-Based Estimation

Function	Estimated LOC
User interface and control facilities (UICF)	2,300
Two-dimensional geometric analysis (2DGA)	5,300
Three-dimensional geometric analysis (3DGA)	6,800
Database management (DBM)	3,350
Computer graphics display facilities (CGDF)	4,950
Peripheral control function (PCF)	2,100
Design analysis modules (DAM)	8,400
Estimated lines of code	*33,200*

Scenario:

- ## Questions:

1. **Estimate Time for given scenario.**

2. **Estimate Cost.**

3. **Estimate Size.**

- **Conclusion:**

(10)	(20)	(10)	(10)	TOTAL

Experiment Number : 11

Aim : Estimate risk and perform RMMM (Risk Monitoring Mitigation and Management.

Theory:

Risk Mitigation Monitoring and Management

• The goal of the risk mitigation, monitoring and management plan is to identify as many potential risks as possible.

• An effective strategy must consider three issues: risk avoidance or mitigation, risk monitoring, and risk management and contingency planning.

RISK MITIGATION:

• If a software team adopts a proactive approach to risk, avoidance is always the best strategy.

• This is achieved by developing a plan for risk mitigation.

RISK MONITORING:

• The project manager monitors factors that may provide an indication of whether the risk is becoming more or less.

RISK MANAGEMENT:

• Risk management and contingency planning assumes that mitigation efforts have failed and that the risk has become a reality.

Scenario:

- **Risk Mitigation**

- **Risk Monitoring:**

- **Risk Management:**

- **Conclusion:**

(10)	(20)	(10)	(10)	TOTAL

Experiment Number : 12

Aim : Estimate scope and feasibility of various systems in terms of technical aspects**.**

Theory:

- **Scope:**
  1. Software scope describes the functions and features that are to be delivered to end users.
  2. The first activity in software project planning is the determination of software scope.
  3. Function and performance allocated to software during system engineering should be assessed to establish a project scope.
  4. A statement of software scope must be bounded. Software scope describes the data and control to be processed, function, performance, constraints, interfaces, and reliability.
  5. Functions described in the statement of scope are evaluated and, in some cases, refined to provide more detail prior to the beginning of estimation.
  6. As both cost and schedule estimates are functionally oriented, some degree of decomposition is often useful.
  7. Performance considerations encompass processing and response time requirements. Constraints identify limits placed on the software by external hardware, available memory, or other existing systems.
  8. Initiate the communication that is essential to establish the scope of the project.
  9. But a question and answer meeting format are not an approach that has been overwhelmingly successful for understanding the scope
  10. The Question **&Answers session** s**hould** be used for the first encounter only and then be replaced by a meeting format that combines elements of problem solving, negotiation, and specification.
  11. Customers and software engineers often have an unconscious "us and them" mindset.
  12. A number of independent investigators have developed a team- oriented approach to requirements gathering that can be applied to help establish the scope of a project.
  13. **Facilized application specification techniques (FAST)**, this approach encourages the **creation of a joint team of customers and developers who work together to identify the problem, propose elements of the solution**, negotiate different approaches, and specify a preliminary set of requirements.

- **Feasibility:**
  1. Once scope has been identified it is reasonable to ask: “Can we build software to meet this scope? Is the project feasible?”
  2. Software engineers rush past these questions only to become mired in a project that is doomed from the onset.
  3. After a few hours or sometimes a few weeks of investigation, you are sure you can do it again.
  4. Projects on the margins of your experience are not so easy.
  5. A team may have to spend several months discovering what the central, difficult-to-implement requirements of a new application actually are.
  6. Do some of these requirements pose risks that would make the project infeasible? Can these risks be overcome?
  7. The feasibility team ought to carry initial architecture and design of the high-risk requirements to the point at which it can answer these questions.
  8. In some cases, when the team gets negative answers, a reduction in requirements may be negotiated.
  9. Once scope is understood, the software team and others must work to determine if it can be done within the dimensions just noted.
  10. This is a crucial, although often overlooked, part of the estimation process.

Scenario:

- **Scope Estimation:**

- **Feasibility Estimation:**

- **Conclusion:**

(10)	(20)	(10)	(10)	TOTAL

EXPERIMENT NO.:-13

Aim: Estimate risk and impact factor for a various system.

Theory:

- System/Scenario Name: Software Project Risk Management

Risks	Category	Probability	Impact	RMMM
Size estimate may be significantly low	PS	60%	2	
Larger number of users than planned	PS	30%	3	
Less reuse than planned	PS	70%	2	
End-users resist system	BU	40%	3	
Delivery deadline will be tightened	BU	50%	2	
Funding will be lost	CU	40%	1	
Customer will change requirements	PS	80%	2	
Technology will not meet expectations	TE	30%	1	
Lack of training on tools	DE	80%	3	
Staff inexperienced	ST	30%	2	
Staff turnover will be high	ST	60%	2	
Σ				
Σ				
Σ				

Impact values:
1—catastrophic
2—critical
3—marginal
4—negligible

Related Points:-

1. Category

The category represents the origin or nature of the risk within a project. It helps in understanding which part of the project the risk is associated with. The table below explains the abbreviations used:

Code Description

PS Product Size

BU Business Impact

Code	Description
CU	Customer-Related
TE	Technology Risk
DE	Development Environment
ST	Staff/Personnel

2. Probability

Probability refers to the likelihood that a specific risk might occur during the course of a project. It is usually expressed as a percentage.

How it's identified:

- Derived from previous project experiences
- Determined through expert evaluations
- Estimated during team brainstorming or risk assessment meetings

For example, if clients often request changes mid-project, the risk of requirement changes might be estimated at 80%.

3. Impact

Impact indicates the seriousness of the consequences if a risk becomes a reality. A numeric scale is typically used:

Value	Meaning
1	Severe (Catastrophic)
2	High (Critical)
3	Medium (Marginal)

Value	Meaning
4	Low (Negligible)

4.RMMM Calculation

Once category, probability, and impact are defined for a risk, the RMMM (Risk Mitigation, Monitoring, and Management) value is calculated using this formula:

RMMM = Probability (as decimal) × Impact

- Scenario:

- Ouestions:

1.Write the Risk Category.

2.Write Probability.

3.Write Impact factor.

4.Calculate RMMM.

- Conclusion:

(10)	(20)	(10)	(10)	Total(50)

EXPERIMENT NO: 14

Aim: Identify Software Quality Assurance techniques performed during different phases of a project for various system.

Theory:

BACKGROUND ISSUES

- Definition- Software quality assurance is a "planned and systematic pattern of actions that are required to ensure high quality in software"
- Quality control and assurance are essential activities for any business that produces products
- Prior to the twentieth century, quality control was the sole responsibility of the craftsperson who built a product. As time passed and mass production, quality control became an activity performed by people other than the ones who built the product.
- The first formal quality assurance and control function was introduced at BellLabs in 1916 and spread rapidly throughout world.
- During the 1940s, more formal approaches to quality control were suggested. These relied on measurement and continuous process improvement as key elements of quality management.
- Today, every company has mechanisms to ensure quality in its products.
- The history of quality assurance in software development parallels the history of quality in hardware manufacturing.
- During the early days of computing quality was the sole responsibility of the programmer .
- Standards for quality assurance for software were introduced in military contract software development during 1970s and have spread rapidly into software development in world.

ELEMENTS OF SOFTWARE QUALITY ASSURANCE

Standards.

- The IEEE, ISO, and other standards organizations have produced a broad array of software engineering standards and related documents .

- Standards may be adopted voluntarily by a software engineering organization or imposed by the customer or other stakeholders.

- The job of SQA is to ensure that standards that have been adopted are followed and that all work products conform to them.

Reviews and audits.

- Technical reviews are a quality control activity performed by software engineers. Their intent is to uncover errors.

- Audits are a type of review performed by SQA personnel with the intent of ensuring that quality guidelines are being followed for software engineering work .

Testing.

- Software testing, is A quality control function that has primary goal to find errors.

- The job of SQA is to ensure that testing is properly planned and efficiently conducted so that it has the highest likelihood of achieving its primary goal.

Error/defect collection and analysis.

- The only way to improve is to measure how you're doing.

- SQA collects and analyzes error and defect data to better understand how errors are introduced and what software engineering activities are best suited to eliminating them.

Change management.

- Change is one of the most disruptive aspects of any software project.

- If it is not properly managed, change can lead to confusion, and confusion almost always leads to poor quality.

Education.

- Every software organization wants to improve engineering practices. its software

- A key contributor to improvement is education of software engineers, their managers, and other stakeholders.

- The SQA organization takes the lead in software process improvement and is a key proponent and sponsor of educational programs.

Vendor management.

• Three categories of software are acquired from external software vendors—⎫ shrink-wrapped packages, ⎫ a tailored shell that provides a basic skeletal structure that is custom tailored to the needs of a purchaser, and ⎫ contracted software that is custom designed and constructed from specifications provided by the customer organization.

• The job of the SQA organization is to ensure that high-quality software results by suggesting specific quality practices that the vendor should follow when possible, and incorporating quality mandates as part of any contract with an external vendor.

Security management.

• With the increase in cyber crime and new government regulations regarding privacy,

• every software organization should institute policies that protect data at all levels, establish firewall protection for Web Apps, and ensure that software has not been tampered with internally.

• SQA ensures that appropriate process and technology are used to achieve software security.

Safety.

• SQA may be responsible for assessing the impact of software failure and for initiating those steps required to reduce risk.

Risk management.

• Although the analysis and mitigation of risk is the concern of software engineers,

• The SQA organization ensures that risk management activities are properly conducted and that risk-related contingency plans have been established.

Scenario:

- **Conclusion:**

(10)	(20)	(10)	(10)	Total (50)

EXPERIMENT NO: 15

Aim: Perform Mini project, considering any one from various concepts for software engineering **.**

Theory:

- **Software Engineering:**

Software Engineering is the systematic application of engineering principles to the design, development, testing, deployment, and maintenance of software. It involves using structured methods and best practices to build reliable, efficient, and scalable software systems that meet user requirements.

- **Software Development Life Cycle (SDLC)**

  - Requirement Analysis
  - System Design
  - Implementation (Coding)
  - Testing
  - Deployment
  - Maintenance

- **Project Management Concepts**

  - Gantt Chart – to show project timeline
  - Risk Management
  - Team roles and responsibilities

- **Software Quality Assurance (SQA)**

  - Ensures the software meets quality standards
  - Involves code reviews, test cases, validation, and verification

Scenario :

Conclusion:

(10)	(20)	(10)	(10)	Total (50)