

### (Write-Up) THEORY

- 1) Objectives
- 2) Basic Definitions
- 3) UML Notations
- 4) Design Methodology for the given Problem statement
- 5) Conclusion

#### TW1:

1. Requirements identification is the first step of any software development project. Until the requirements of a client have been clearly identified, and verified, no other task (design, coding, testing) could begin.

After completing this experiment you will be able to:

- Identify ambiguities, inconsistencies and incompleteness from a requirements specification.
- Identify and state functional requirements.
- Identify and state non-functional requirements.

2. **Unambiguity:** There should not be any ambiguity what a system to be developed should do.

**Consistency:** There should be no inconsistency between the two end users.

**Completeness:** A particular requirement for a system should specify what the system should do and also what it should not.

**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

3.

4. write it on your own about how did you come up with these data

5. In conclusion, identifying the requirements from problem statements is a crucial step in any project development process. It helps to ensure that everyone involved in the project has a clear understanding of what is needed to solve the problem at hand. The process involves analysing the problem statement, breaking it down into smaller, more manageable parts, and identifying the key requirements that need to be met.

## TW2:

1. Use case diagrams belong to the category of behavioural diagram of UML diagrams. Use case diagrams aim to present a graphical overview of the functionality provided by the system. It consists of a set of actions (referred to as use cases) that the concerned system can perform, one or more actors, and dependencies among them

After completing this experiment you will be able to:

- How to identify different actors and use cases from a given problem statement
- How to associate use cases with different types of relationships
- How to draw a use-case diagram

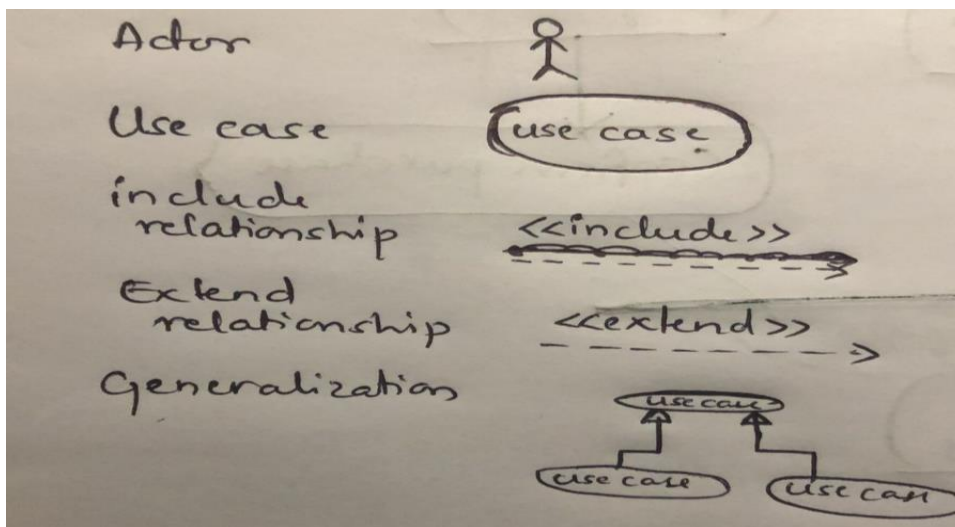
2. **Actor:** An actor can be defined as an object or set of objects, external to the system, which interacts with the system to get some meaningful work done. Actors could be human, devices, or even other systems.

**Use Case:** A use case is simply a functionality provided by a system.

**Include Relationship:** an include relationship is a relationship between two use cases where one use case includes the functionality of another use case. The included use case is not complete on its own and is dependent on the including use case for its functionality.

**Extend Relationship:** an extend relationship is a relationship between two use cases where one use case adds functionality to another use case. The extended use case is complete on its own and provides a basic set of functionalities. The extending use case adds optional or alternative behaviour to the extended use case.

**Use Case Generalization:** use case generalization is a relationship between two use cases where one use case is a specialized version of another use case. The specialized use case inherits the behaviour of the general use case and adds or modifies its behaviour to meet specific requirements.



3.

4. write it on your own about how did you come up with these data and diagram

5. In conclusion, UML use case diagrams are a powerful tool for modelling the behaviour of a system from the point of view of external users or actors. They provide a graphical representation of the system's functionality and help to ensure that the system meets the needs of its users. When creating a use case diagram, it is important to identify the actors involved, the use cases they need to perform, and any relationships between them.

### TW3:

1. Entity-Relationship model is used to represent a logical design of a database to be created. In ER model, real world objects (or concepts) are abstracted as entities, and different possible associations among them are modelled as relationships. An **entity set** is a collection of all similar entities.

Similarly, a **"Relationship" set** is a set of similar relationships.

After completing this experiment you will be able to:

- Identify entity sets, their attributes, and various relationships
- Represent the data model through ER diagram.



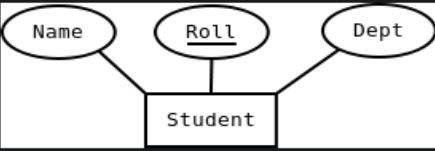
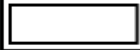




2. An **entity set** is a collection of all similar entities.

Similarly, a **"Relationship" set** is a set of similar relationships.

**Attributes** are the characteristics describing any entity belonging to an entity set.

**Weak Entity:** An entity set is said to be weak if it is dependent upon another entity set.

3.

Term	Notation	Remarks
Entity set		Name of the set is written inside the rectangle
Attribute		Name of the attribute is written inside the ellipse
Entity with attributes		Roll is the primary key; denoted with an underline
Weak entity set		
Relationship set		Name of the relationship is written inside the diamond
Related entity sets		
Relationship cardinality		A person can own zero or more cars but no two persons can own the same car
Relationship with weak entity set		

4. write it on your own about how did you come up with these data and diagram

5. In conclusion, E-R modelling is a powerful tool for identifying the relationships between entities in a system and for designing a database schema. When creating an E-R model from problem statements, it is important to identify the entities involved, their attributes, and the relationships between them. This helps to ensure that the model accurately reflects the system's behaviour and can be used as a blueprint for designing and implementing the database schema.

#### TW4:

1. After completing this experiment, you will be able to:

- Design a relational database using phpmySQL from xampp server
- Connect this database via php to an html form and add data from html form to mysql
- Learn to run xampp server

2.

3.

4. write it on your own about how did you come up with these data and diagram

5. to conclude, we have designed a database, understood the requirements and created tables with appropriate attributes and data types. Then we have connected this table to an HTML form via php to demonstrate frontend-backend connection for database and program.

#### TW5:

1. A **statechart diagram** is a pictorial representation of such a system, with all it's states, and different events that lead transition from one state to another.

After completing this experiment you will be able to:

- Identify the distinct states a system has
- Identify the events causing transitions from one state to another
- Represent the above information pictorially using simple states
- Identify activities representing basic units of work, and represent their flow

2. **State:** A state is any "distinct" stage that an object (system) passes through in its lifetime. All such states can be broadly categorized into following three types:

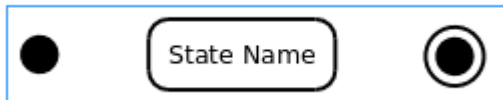
- Initial: The state in which an object remains when created.
- Final: The state from which an object do not move to any other state [optional].
- Intermediate: Any state, which is neither initial, nor final.

**Transition:** Transition is movement from one state to another state in response to an external stimulus (or any internal event).

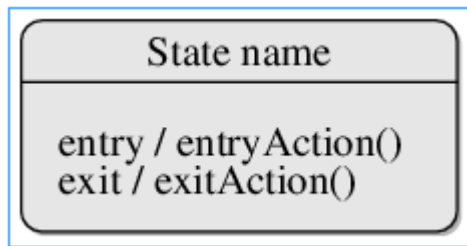
**Action:** As mentioned are the actions that represents behaviour of the system.

**Activity Diagrams:** Activity diagrams fall under the category of behavioural diagrams in Unified Modelling Language. It is a high-level diagram used to visually represent the flow of control in a system. It has similarities with traditional flow charts.

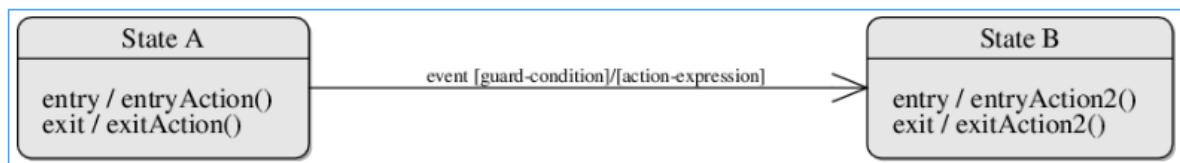
3.



initial, intermediate and final states



intermediate state in state diagram



transition

Activity diagram:

Component	Graphical Notation
Activity	
Flow	
Decision	
Merge	
Fork	
Join	
Note	

4. write it on your own about how did you come up with these data and diagram

5. Statechart and activity modelling are powerful tools for visualizing complex systems and processes. In statechart diagrams, the various states of the system are represented, along with the transitions between them triggered by events and actions. In activity diagrams, the different activities of a process are depicted, as well as the flow of the process and its conditions. These models help to provide a better understanding of the system or process and to identify potential issues. Developing these models requires critical thinking and problem-solving skills, as well as proficiency in visual modelling tools.

#### **TW6:**

1. After completing this experiment, you will be able to:

- Graphically represent a class, and associations among different classes
- Identify the logical sequence of activities undergoing in a system, and represent them pictorially

2. **Classes** are the structural units in object-oriented system design approach, so it is essential to know all the relationships that exist between the classes, in a system.

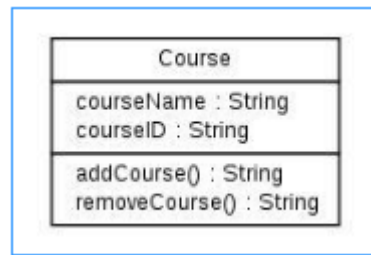
**Class diagram:** It is a graphical representation for describing a system in context of its static construction.

1. **Class:** A set of objects containing similar data members and member functions is described by a class. In UML syntax, class is identified by solid outline rectangle with three compartments which contain
2. **Class name:** A class is uniquely identified in a system by its name. A textual string is taken as class name. It lies in the first compartment in class rectangle.
3. **Attributes:** Property shared by all instances of a class. It lies in the second compartment in class rectangle.
4. **Operations:** An execution of an action can be performed for any object of a class. It lies in the last compartment in class rectangle.

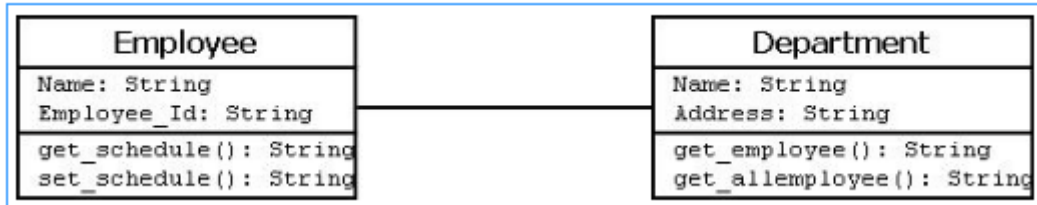
**Sequence diagram:** It represents the behavioural aspects of a system. Sequence diagram shows the interactions between the objects by means of passing messages from one object to another with respect to time in a system. Sequence diagram contains the objects of a system and their life-line bar and the messages passing between them.

1. **Object:** Objects appear at the top portion of sequence diagram. Object is shown in a rectangle box. Name of object precedes a colon ':' and the class name, from which the object is instantiated.
2. **Life-line bar:** A down-ward vertical line from object-box is shown as the life-line of the object. A rectangle bar on life-line indicates that it is active at that point of time.

3.



class



relationships

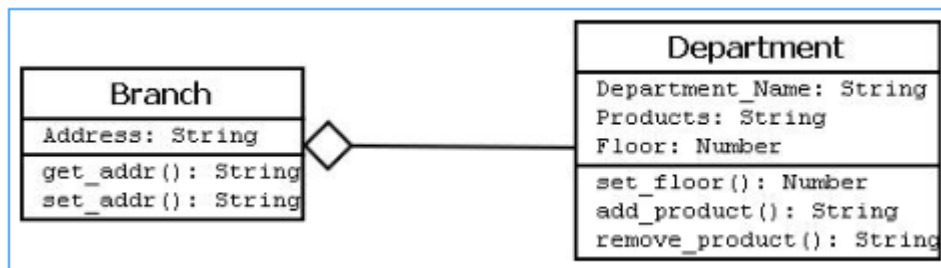


Figure 04.

aggregation



Figure 05.

composition

Single instance	1
Zero or one instance	0..1
Zero or more instance	0..*
One or more instance	1..*
Particular range(two to six)	2..6

multiplicity

Sequence diagrams:

Message Type	Notation
Synchronous message	
Asynchronous message	
Response message	

4. write it on your own about how did you come up with these data and diagram

5. Modeling UML class diagrams and sequence diagrams are critical steps in designing and developing software systems. UML class diagrams depict the classes in a system, their attributes, and their relationships with one another. Sequence diagrams illustrate how objects in a system interact with one another over time, highlighting the flow of messages and the order in which they are sent. Developing these models requires knowledge of object-oriented programming, proficiency in UML modeling tools, and attention to detail.

#### TW7:

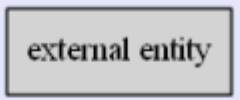


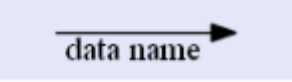
1. After completing this experiment you will be able to:

- Identify external entities and functionalities of any system.
- Identify the flow of data across the system.
- Represent the flow with Data Flow Diagrams.

2. **Data Flow Diagram:** it provides the functional overview of a system. The graphical representation easily overcomes any gap between 'user and system analyst' and 'analyst and system designer' in understanding a system.

Explanation of Symbols used in DFD:-

1. **Process:** Processes are represented by circle. The name of the process is written into the circle.
2. **External entity:** External entities are only appear in context diagram. External entities are represented by a rectangle and the name of the external entity is written into the shape.
3. **Data store:** Data stores are represented by a left-right open rectangle. Name of the data store is written in between two horizontal lines of the open rectangle.
4. **Data flow:** Data flows are shown as a directed edge between two components of a Data Flow Diagram. Data can flow from external entity to process, data store to process, in between two processes and vice-versa.

Term	Notation
External entity	
Process	
Data store	
Data flow	

3.



4. . write it on your own about how did you come up with these data and diagram

5. Modelling Data Flow Diagrams (DFD) is an important part of systems analysis and design. DFDs are visual representations that depict how data flows through a system, identifying the inputs, processes, and outputs involved in the system. These diagrams help to identify potential bottlenecks, inefficiencies, and areas for improvement. Developing DFDs requires an understanding of the system being modelled, as well as proficiency in modelling tools. The resulting models help to improve communication, identify areas for improvement, and ensure the successful implementation of systems.

#### **TW8:**

1. After completing this experiment, you will be able to:

- Learn about different techniques of testing a software.
- Design unit test cases to verify the functionality and locate bugs, if any.

2. **Testing software** is an important part of the development life cycle of a software. The purpose of testing is to verify and validate a software and to find the defects present in a software. The purpose of finding those problems is to get them fixed.

1. **Verification** is the checking or we can say the testing of software for consistency and conformance by evaluating the results against pre-specified requirements.
2. **Validation** looks at the systems correctness, i.e. the process of checking that what has been specified is what the user actually wanted.
3. **Defect** is a variance between the expected and actual result. The defect's ultimate source may be traced to a fault introduced in the specification, design, or development (coding) phases.

3.

4. write it on your own about how did you come up with these test suites and whats the procedure

5. Designing test suites is an essential part of software development, ensuring that software systems meet user requirements and are free of bugs and defects. Test suites are collections of test cases that are designed to thoroughly test different aspects of the system. They may include unit tests, integration tests, and acceptance tests. Developing effective test suites requires a comprehensive understanding of the system being tested, as well as knowledge of testing tools and methodologies.