



MARWADI UNIVERSITY

FACULTY OF TECHNOLOGY

[ INFORMATION &amp; COMMUNICATION TECHNOLOGY ]

SEM: 4

MU FINAL EXAM

**May: 2023****Subject: - (Software Engineering ) (01CT0615)****Date:-12/05/2023****Total Marks:-100****Time: - 3 HOUR**Instructions:

1. Attempt all questions.
2. Make suitable assumptions wherever necessary.
3. Figures to the right indicate full marks.
4. English version is authentic.

Question: 1/48.9.

- (a) [10]
- 1 Which one is not requirements prioritization techniques?
    - (a) Ranking
    - (b) **One Dollar Method**
    - (c) Numerical Assignment
    - (d) Bubble Sort Technique
  - 2 Full form of RAD model
    - (a) **Rapid Application Development**
    - (b) Rapid Application Document
    - (c) Relative Application Development
    - (d) Relative Application Developer
  - 3 Which pattern is using plug-in modules?
    - (a) Client-Server Pattern
    - (b) Event-Driven Pattern
    - (c) Microservice Pattern
    - (d) **Microkernel Pattern**
  - 4 \_\_\_\_\_ diagram represent how software is installed on the hardware component
    - (a) Component Diagram
    - (b) Activity Diagram
    - (c) **Deployment Diagram**
    - (d) Use case Diagram
  - 5 \_\_\_\_\_ box testing, also called behavioral testing
    - (a) White-box
    - (b) Gray-box
    - (c) **Black-box**
    - (d) Green-box
  - 6 The process in which specifications are described, recorded and maintained throughout the design process is called \_\_\_\_\_
    - (a) Test planning
    - (b) **Requirement Engineering**
    - (c) Feasibility Study
    - (d) System Testing

- 7 In MoScow Technique W states.....  
 (a) **Would**  
 (b) What  
 (c) Why  
 (d) Where
- 8 The Verification and Validation Model is also known as which of the following?  
 (a) Waterfall Model  
 (b) **V-Model**  
 (c) Spiral Model  
 (d) Prototype Model
- 9 Building websites with JavaScript and e-commerce websites in general is the example of \_\_\_\_pattern  
 (a) **Event-Driven Pattern**  
 (b) Client-Server Pattern  
 (c) Layered Pattern  
 (d) Level Pattern
- 10 Which one is the Agile Process Models?  
 (a) External Programming  
 (b) Advanced software development  
 (c) Scrub  
 (d) **None of Above**

(b)


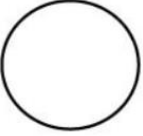
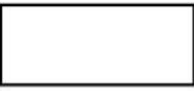
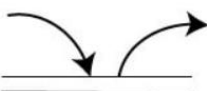
[10]

1. **What do you mean by agility?**

Agility is ability to move quickly and easily. It is a property consisting of quickness, lightness, & ease of movement The ability to create and respond to change in order to profit in a turbulent global business environment

2. **Define DFD. what are the different symbols are used in DFD diagram**

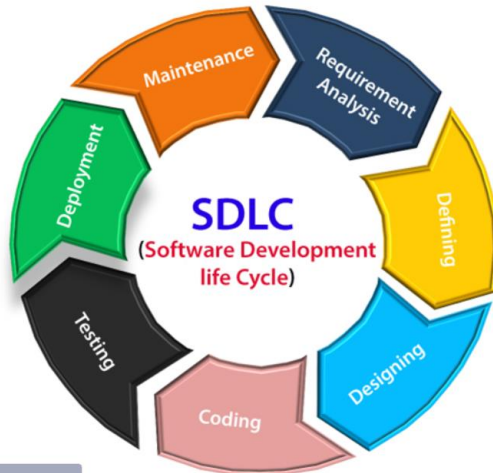
A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both.

| Symbol  | Name                                    | Function   |
|---|---|--|
|  | <b>Data flow</b>                        | Used to Connect Processes to each other, to sources or Sinks; the arrow head indicates direction of data flow. |
|  | <b>Process</b>                          | Performs Some transformation of Input data to yield output data.   |
|  | <b>Source of Sink (External Entity)</b> | A Source of System inputs or Sink of System outputs.   |
|  | <b>Data Store</b>                       | A repository of data; the arrow heads indicate net inputs and net outputs to store.                            |

3. **What do you mean by SDLC?**

SDLC stands for Software Development Life Cycle. It is a structured process used in software engineering to design, develop, and maintain software systems.

The SDLC typically consists of several phases that are performed in a sequential or iterative manner, depending on the project's requirements and complexity. The phases of the SDLC typically include:



4. **Enlist the characteristic of SRS.**

- Clear
- Correct
- Consistent
- Coherent
- Comprehensible
- Modifiable
- Verifiable
- Prioritized
- Unambiguous
- Traceable
- Credible source
- 

5. **What is Software Architecture**

The architecture of a system describes its major components, their relationships (structures), and how they interact with each other.

Software architecture and design includes several contributory factors such as Business strategy, quality attributes, human dynamics, design, and IT environment.

6. **Define Requirements Prioritization in brief.**

Software development or any other project facing multiple requirements, budgetary constraints, and tight deadlines often necessitate the need to prioritize stakeholders' requirements. At some point, it's usually necessary to make decisions on which set of requirements need to be implemented first and which ones can be delayed till a later release

7. **Differentiate testcase and SRS**

Test case and SRS (Software Requirement Specification) are two different documents used in software development, and they serve different purposes.

A test case is a document that describes the steps to be executed to verify if a particular functionality or requirement of the software is working as expected. Test cases are created based on the requirements and are used to ensure that the software meets the functional and non-functional requirements defined in the SRS.

On the other hand, SRS is a document that captures the requirements of the software

system to be developed. It describes what the software should do and what it should not do. The SRS document includes functional and non-functional requirements, design constraints, assumptions, and acceptance criteria

8. **Define entity, attributes and relationship.**

**Entity:** An entity is an object or concept in the real world that can be distinguished from other objects or concepts. In a database, an entity is represented as a table that contains data related to that particular object or concept. For example, a "customer" entity in a database for an e-commerce site may include data such as the customer's name, address, phone number, and email address.

**Attribute:** An attribute is a characteristic or property of an entity. In a database, an attribute is represented as a column in a table that contains data related to that characteristic or property. For example, in the "customer" entity mentioned above, the "name" attribute may contain data such as "John Smith" or "Jane Doe".

**Relationship:** A relationship is a connection between two or more entities. In a database, a relationship is represented by a link between two or more tables that indicates how the entities are related to each other. For example, in a database for an e-commerce site, the "order" entity may be related to the "customer" entity through a relationship that indicates which customers have placed which orders.

9. **What are the different type of Feasibility?**

- **Technical Feasibility :** Technical feasibility assesses the latest technology required to meet customer requirements in time and on budget.
- **Operational Feasibility -** Operational feasibility assesses the context in which the software needed performs a variety of levels to solve business problems and consumer needs.
- **Economic Feasibility –** Economic feasibility determines if the software required to produce financial benefits for an organization.
- 

10. **Write disadvantage of waterfall model.**

**Lack of flexibility:** The Waterfall model does not allow for changes to be made easily once the development process has started. If changes are needed, it may require restarting the entire process, which can be time-consuming and expensive.

**High risk:** Because the Waterfall model does not allow for changes to be made easily, it can lead to high project risk. If requirements are not properly understood or defined at the outset, it can lead to significant problems later in the development process.

**Limited customer involvement:** The Waterfall model assumes that requirements are fixed at the beginning of the project, which can limit customer involvement and feedback throughout the development process.

**Long development cycle:** The Waterfall model can lead to a long development cycle due to the sequential nature of the process. This can result in delays in delivering the software product to customers.

**Lack of testing:** The Waterfall model does not prioritize testing until the later stages of the development process, which can lead to quality issues and bugs that are not discovered until late in the development process

**Question: 2/ 48.2.**

(a(i)) Differentiate verification and validation (minimum 4 points)

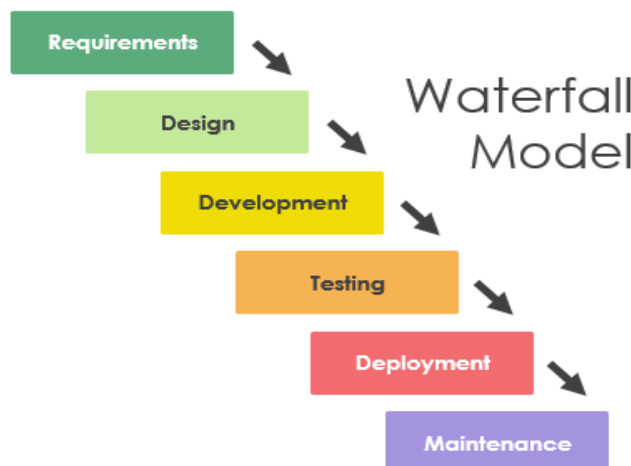
[4]

| Verification  | Validation  |
|---|---|
| Are we implementing the system right?   | Are we implementing the right system?   |
| Evaluating products of a development phase  | Evaluating products at the closing of the development process                   |
| The objective is making sure the product is as per the requirements and design specifications   | The objective is making sure that the product meets user's requirements         |
| Activities included: reviews, meetings, and inspections   | Activities included: black box testing, white box testing, and grey box testing |
| Verifies that outputs are according to inputs or not  | Validates that the users accept the software or not                             |
| Items evaluated: plans, requirement specifications, design specifications, code, and test cases | Items evaluated: actual product or software under test                          |
| Manual checking of the documents and files  | Checking the developed products using the documents and files                   |

(a(ii)) **Explain waterfall model in detail.**

[4]

The waterfall model is a breakdown of project activities into linear sequential phases, where each phase depends on the deliverables of the previous one and corresponds to a specialization of tasks. The approach is typical for certain areas of engineering design.



- Requirements analysis and specification phase: The aim of this phase is to understand the exact requirements of the customer and to document them properly.
- Design Phase: This phase aims to transform the requirements gathered in the SRS into a suitable form which permits further coding in a programming language. It defines the overall software architecture together with high level and detailed design. All this work is documented as a Software Design Document (SDD).
- Implementation and unit testing: During this phase, design is implemented. If the SDD is complete, the implementation or coding phase proceeds smoothly, because all the information needed by software developers is contained in the SDD.
- Integration and System Testing: This phase is highly crucial as the quality of the end product is determined by the effectiveness of the testing carried out.
- Operation and maintenance phase: Maintenance is the task performed by every user once the software has been delivered to the customer, installed, and operational.

(b) **Explain the characteristic of SRS in detail.**

[8]

- **Correctness:** User review is used to provide the exactness of requirements stated in the SRS. SRS is said to be ideal if it covers all the requirements that are truly expected from the system
- **Completeness:** The SRS is absolute if, and only if, it includes the following essentials:
  - All essential requirements, whether relating to functionality, performance, design, constraints, attributes, or external interfaces.
  - Definition of their responses of the software to all attainable classes of input data in all available classes of situation.
- **Consistency:** The SRS is reliable if, and only if, no subset of individual requirements mentioned in its conflict
- **Unambiguousness:** SRS is unambiguous when every fixed requirement has only one understanding. This proposes that each element is solely interpreted.
- **Ranking for importance and stability:** The SRS is ranked for importance and stability if each requirement in it has an identifier to indicate either the significance or stability of that particular requirement
- **Modifiability:** SRS should be made as modifiable as likely and should be capable of speedily obtain changes to the system to some extent. Alterations should be perfectly indexed and cross-referenced.
- **Verifiability:** SRS is perfect when the specified requirements can be confirmed with a cost-effective system to ensure whether the final software meets those requirements. The requirements are verified with the help of reassess.
- **Traceability:** The SRS is outlined if the source of each of the requirements is clear and if it facilitates the referencing of each condition in future development or enhancement documentation.
- **Design Independence:** There should be a choice to choose from multiple design alternatives for the ultimate system. More specifically, the SRS should not include any implementation details.
- **Testability:** An SRS should be written in such a way that it is easy to generate test cases and test plans from the report.
- **Understandable by the customer:** An end user may be an expert in his/her precise domain but might not be coached in computer science. Hence, the purpose of formal information and representations should be avoided too as much extent as possible. The language should be kept plain and clear.
- **The right level of abstraction:** If the SRS is written for the requirements stage, the details should be explained openly whereas, for a feasibility study, less analysis can be used. Hence, the level of abstraction changes according to the objective of the SRS.
- **Concise:** The SRS report should be brief and at the same time, unambiguous, consistent, and complete. Verbose and unrelated descriptions decrease readability and also increase error possibilities.
- **Structured:** It should be well organized. A well organized document is simple to understand and change

**OR**(b) **What do you mean by Requirements Prioritization? Explain different techniques of Requirement Prioritization**

[8]

Software development or any other project facing multiple requirements, budgetary constraints, and tight deadlines often necessitate the need to prioritize stakeholders' requirements. At some point, it's usually necessary to make decisions on which set of requirements need to be implemented first and which ones can be delayed till a later release

This list of requirements prioritization techniques provides an overview of common techniques that can be used in prioritizing requirements.

- Ranking
- Numerical Assignment (Grouping)
- MoScow Technique
  - MUST (Mandatory)
  - SHOULD (Of high priority)
  - COULD (Preferred but not necessary)
  - WOULD (Can be postponed and suggested for future execution)
- Bubble Sort Technique
- Hundred Dollar Method

**NOTE: you suppose to explain each above technique in brief**

**Question: 3/ 4.3.**

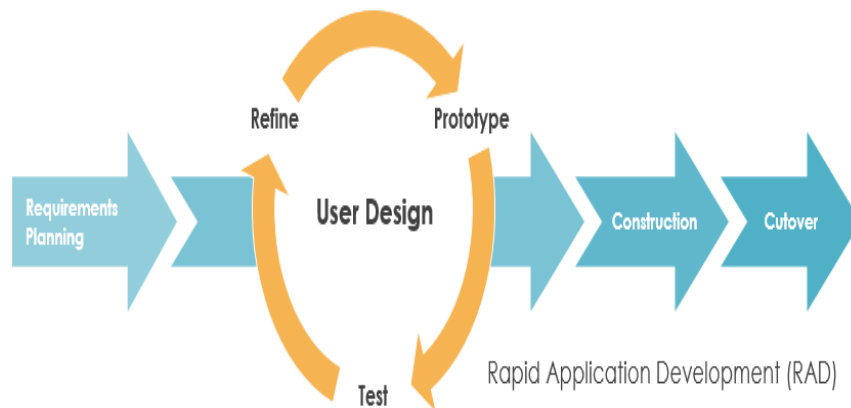
(a) **Explain RED model in detail.**

[8]

RAD is a linear sequential software development process model that emphasizes a concise development cycle using an element based construction approach. If the requirements are well understood and described, and the project scope is a constraint, the RAD process enables a development team to create a fully functional system within a concise time period.

RAD (Rapid Application Development) is a concept that products can be developed faster and of higher quality through:

- Gathering requirements using workshops or focus groups
- Prototyping and early, reiterative user testing of designs
- The re-use of software components
- A rigidly paced schedule that refers design improvements to the next product version
- Less formality in reviews and other team communication



**Advantage of RAD Model**

- This model is flexible for change.
- In this model, changes are adoptable.
- Each phase in RAD brings highest priority functionality to the customer.
- It reduced development time.
- It increases the reusability of features

**Disadvantage of RAD Model**

- It required highly skilled designers.
- All application is not compatible with RAD.
- For smaller projects, we cannot use the RAD model.

- On the high technical risk, it's not suitable.
- Required user involvement.

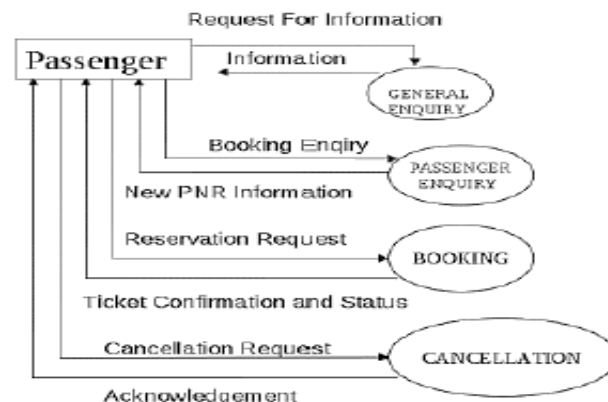
(b) **Differentiate User Requirements and System Requirements**

[4]

| User Requirements Vs. System Requirements  |   |
|--|---|
| 7  |   |
| User Requirements  | System Requirements   |
| <ul style="list-style-type: none"> <li>▪ Written for customers .</li> <li>▪ Statements in natural language.</li> <li>▪ Describe the services that system provides and its operational constraints.</li> <li>▪ May include diagrams or tables.</li> <li>▪ Should describe functional and non-functional requirements.</li> <li>▪ Should be understandable by system users who don't have detailed technical knowledge.</li> </ul> | <ul style="list-style-type: none"> <li>▪ Statements that set out detailed descriptions of the system's functions, services and operational constraints.</li> <li>▪ Defines what should be implemented so may be part of a contract between client and contractor.</li> <li>▪ Intended to be a basis for designing the system.</li> <li>▪ Can be illustrated using system models.</li> </ul> |
| <b>We provide a definition for a user requirement.</b>   | <b>We provide a <u>specification</u> for a system requirement.</b>  |

(c) **Draw and Explain Data flow diagram for Airline reservation system.**  
**text diagram / Level 0 DFD**

[4]



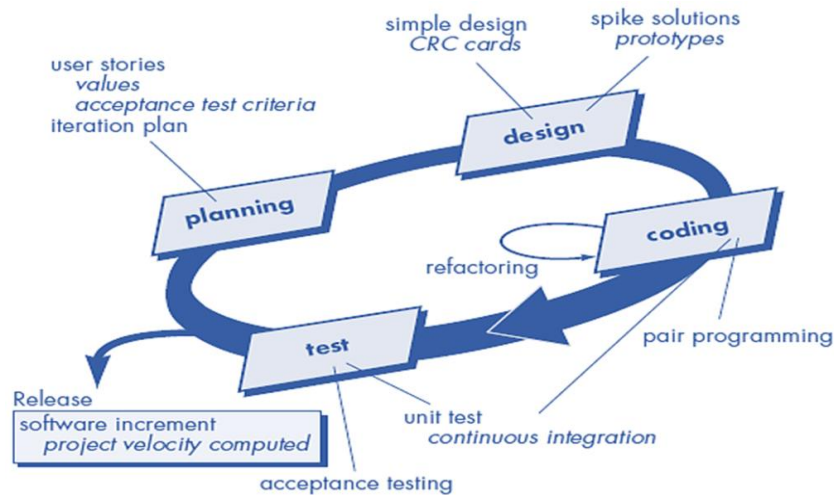
OR

(a) **Explain XP agile model in detail.**

[8]

- The most widely used approach to agile software development
- A variant of XP called Industrial XP (IXP) has been proposed to target process for large organizations
- It uses object oriented approach as its preferred development model
- Communication: To achieve effective communication, it emphasized close & informal (verbal) collaboration between customers and developers
- Simplicity: It restricts developers to design for immediate needs not for future needs
- Feedback: It is derived from three sources the implemented software, the customer and other software team members, it uses Unit testing as primary testing
- Courage: It demands courage (discipline), there is often significant pressure to design for future requirements, XP team must have the discipline (courage) to design for today
- Respect: XP team respect among members





- User Stories
  - Customers assigns value (priority)
  - Developers assigns cost (number of development weeks)
- Project velocity
  - Computed at the end of first release
  - Number of stories implemented in first release
  - Estimates for future release
  - Guard against over-commitment
- Keep-it-Simple (Design of extra functionality is discouraged)
- Preparation of CRC(Class-responsibility-collaboration) card is work
- project
  - CRC cards identify and organize object oriented classes
  - Spike Solutions
  - Operational prototype intended to clear confusion
- Refactoring
- Modify internals of code, No observable change
- Integrate code with other team members, this “continuous integration” helps to avoid compatibility & interfacing problems, “smoke testing” environment to uncover errors early
- Unit test by developers & fix small problems
- Acceptance tests - Specified by customer

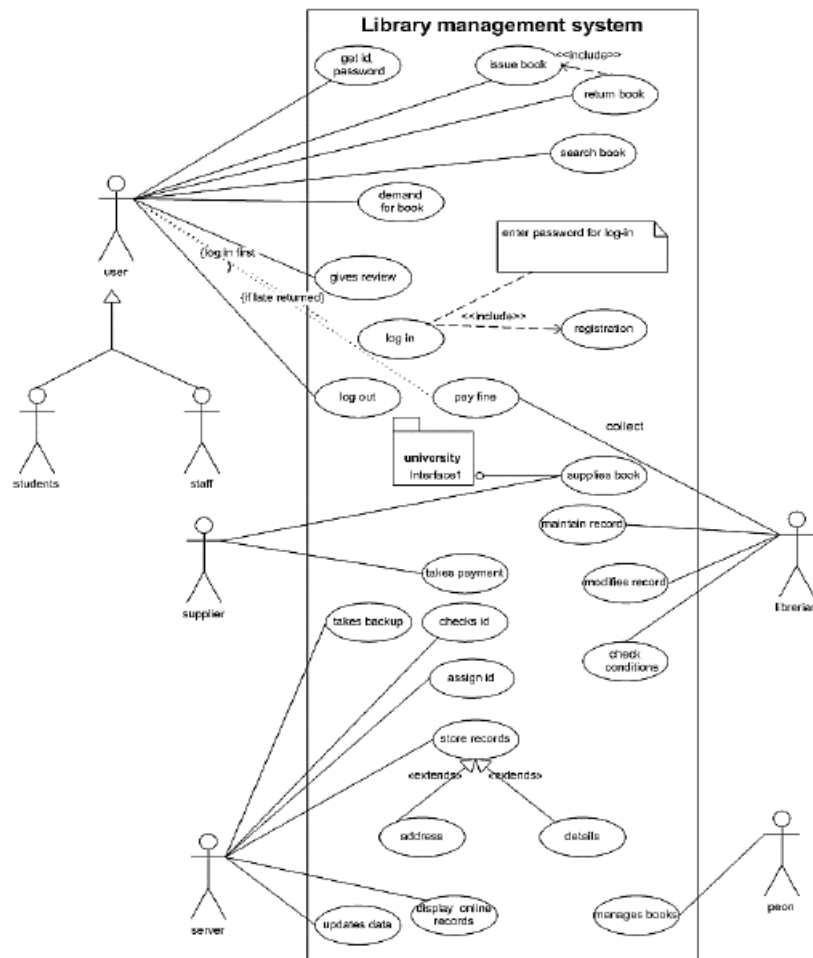
(b) **Differentiate Functional and Non-functional Requirements**

[4]

| FUNCTIONAL vs NONFUNCTIONAL REQUIREMENTS |  |  |
|--|--|--|
|  | Functional Requirements  | Nonfunctional Requirements   |
| Objective                                | What the product does  | How the product works  |
| Focus                                    | Focus on user requirements   | Focus on user expectations   |
| Documentation                            | in use case  | as a quality attribute   |
| End Result                               | Product features   | Product properties   |
| Essentiality                             | Mandatory  | Not mandatory, but desirable   |
| Origin type                              | Defined by user  | Defined by developers or tech experts  |
| Testing                                  | Component, API, UI testing, etc. Tested before nonfunctional testing     | Performance, usability, security testing, etc. Tested after functional testing |
| Types                                    | External interface, authentication, authorization levels, business rules | Usability, reliability, scalability, performance                               |

(c) **Draw and Explain use case diagram for library management system**

[4]



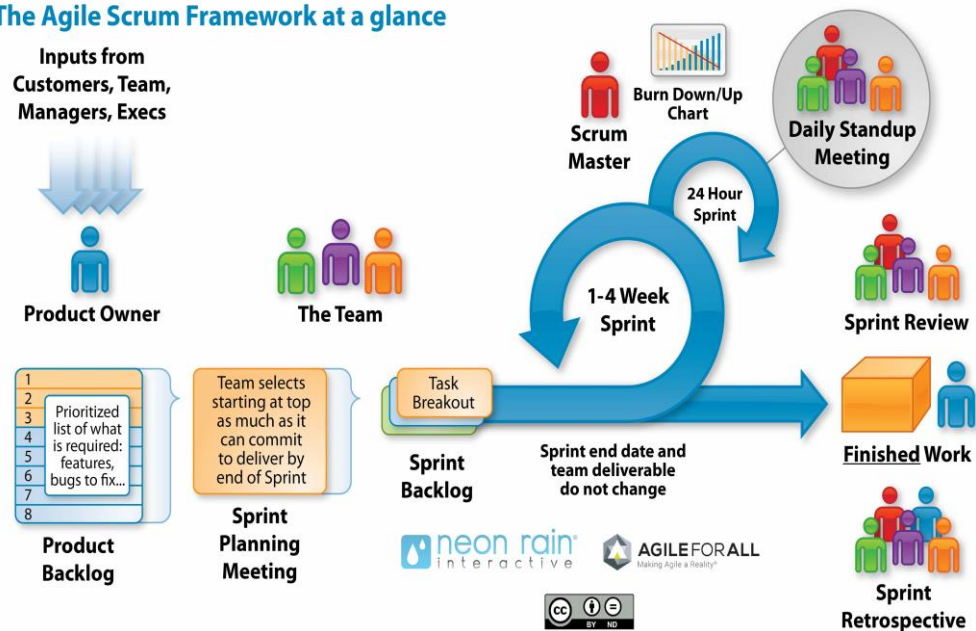
**Question: 4/ ۛۛ.ۛ.**

(a) **Explain scrum agile model in detail.**

[8]

1. Scrum is an agile process model which is used for developing the complex software systems & it is a lightweight process framework.
2. Lightweight means the overhead of the process is kept as small as possible in order to maximize the productivity.

### The Agile Scrum Framework at a glance



#### 3. Backlog

- It is a **prioritized list of project requirements** or features that must be provided to the customer.
- The **items can be included** in the backlog at **any time**.
- The **product manager analyses** this **list** and **updates** the **priorities** as per the requirements.

#### 4. Sprint

- These are the **work units** that are needed to **achieve** the requirements mentioned in the backlogs.
- Sprints have **fixed duration** or time box (of **2 to 4 weeks, 30 days**).
- **Change** are **not introduced** during the **sprint**.
- Thus **sprints** allow the team **members** to **work** in **stable** and **short-term environment**.

#### 5. Scrum Meetings

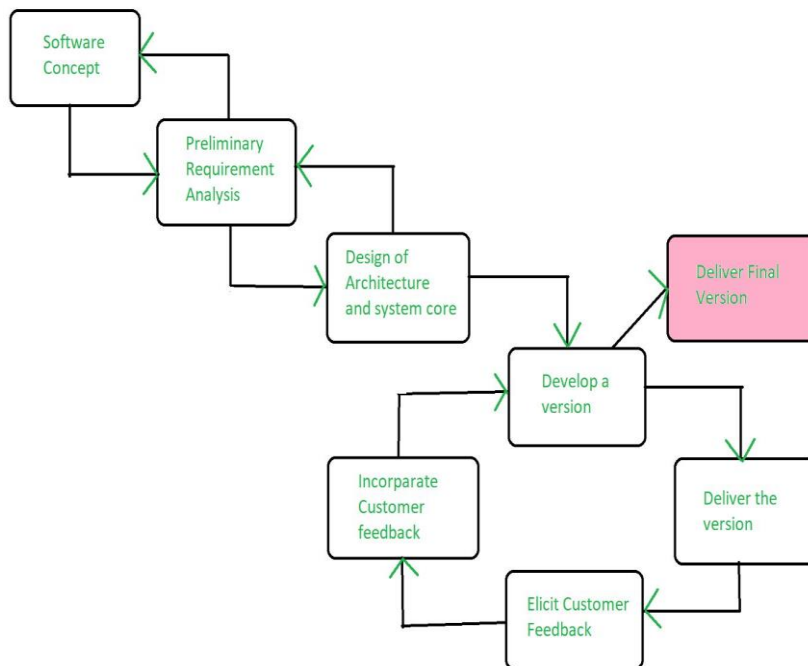
- There are **15 minutes daily meetings** to **report** the **completed** activities, **obstacles** and **plan** for **next** activities.
- Following are three questions that are mainly discussed during the meetings.
  1. What are the **tasks done** since **last meeting** ?
  2. What are the **issues** that team is **facing** ?
  3. What are the **next activities** that are **planned**?
  4. The **scrum master** leads the meeting and **analyses** the **response** of each team member.
  5. Scrum meeting **helps** the **team** to **uncover potential problems** as early as possible
  6. It leads to “**knowledge socialization**” & promotes “**self-organizing team structure**”

#### 6. Demo

- Deliver **software increment** to customer
- Implemented functionalities are **demonstrated** to the customer

(b) Explain Evolutionary Process Models in detail

[8]



- Evolutionary model is a combination of Iterative and Incremental model of software development life cycle.
- Delivering your system in a big bang release, delivering it in incremental process over time is the action done in this model.
- Some initial requirements and architecture envisioning need to be done.
- It is better for software products that have their feature sets redefined during development because of user feedback and other factors.
- The Evolutionary development model divides the development cycle into smaller, incremental waterfall models in which users are able to get access to the product at the end of each cycle.
- **Feedback** is provided **by the users** on the product for the planning stage of the next cycle and the development **team responds**, often by changing the product, plan or process. Therefore, **the software product evolves withtime**. All the models have the **disadvantage** that the duration of time from start of the project to the **delivery time of a solution is very high**. Evolutionary model solves this problem in a different approach.
- Feedback is provided by the users on the product for the planning stage of the next cycle and the development team responds, often by changing the product, plan or process. Therefore, the software product evolves withtime.
- All the models have the disadvantage that the duration of time from start of the project to the delivery time of a solution is very high. Evolutionary model solves this problem in a different approach.

**OR**

(a) What are the different types of messages in sequence diagram explain in detail

[8]

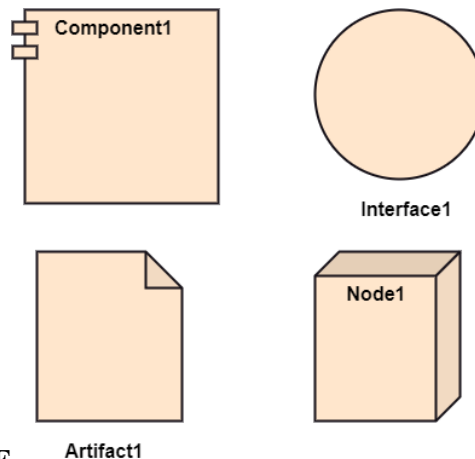
- **Call Message:** It defines a particular communication between the lifelines of an interaction, which represents that the target lifeline has invoked an operation.
- **Return Message:** It defines a particular communication between the lifelines of interaction that represent the flow of information from the receiver of the corresponding caller message.
- **Self Message:** It describes a communication, particularly between the lifelines of an interaction that represents a message of the same lifeline, has been invoked.
- **Recursive Message:** A self message sent for recursive purpose is called a recursive message. In other words, it can be said that the recursive message is a special case of the self message as it represents the recursive calls.
- **Create Message:** It describes a communication, particularly between the lifelines of an

- interaction describing that the target (lifeline) has been expressed.
- Destroy Message: It describes a communication, particularly between the lifelines of an interaction that depicts a request to destroy the lifecycle of the target.
- Duration Message: It describes a communication particularly between the lifelines of an interaction, which portrays the time passage of the message while modeling a system.

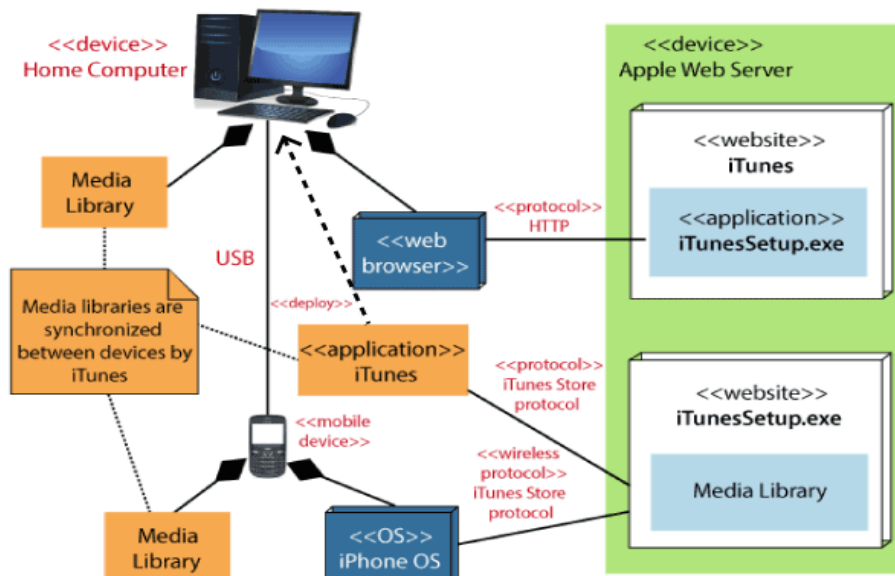
(b) Explain Deployment diagram with example.

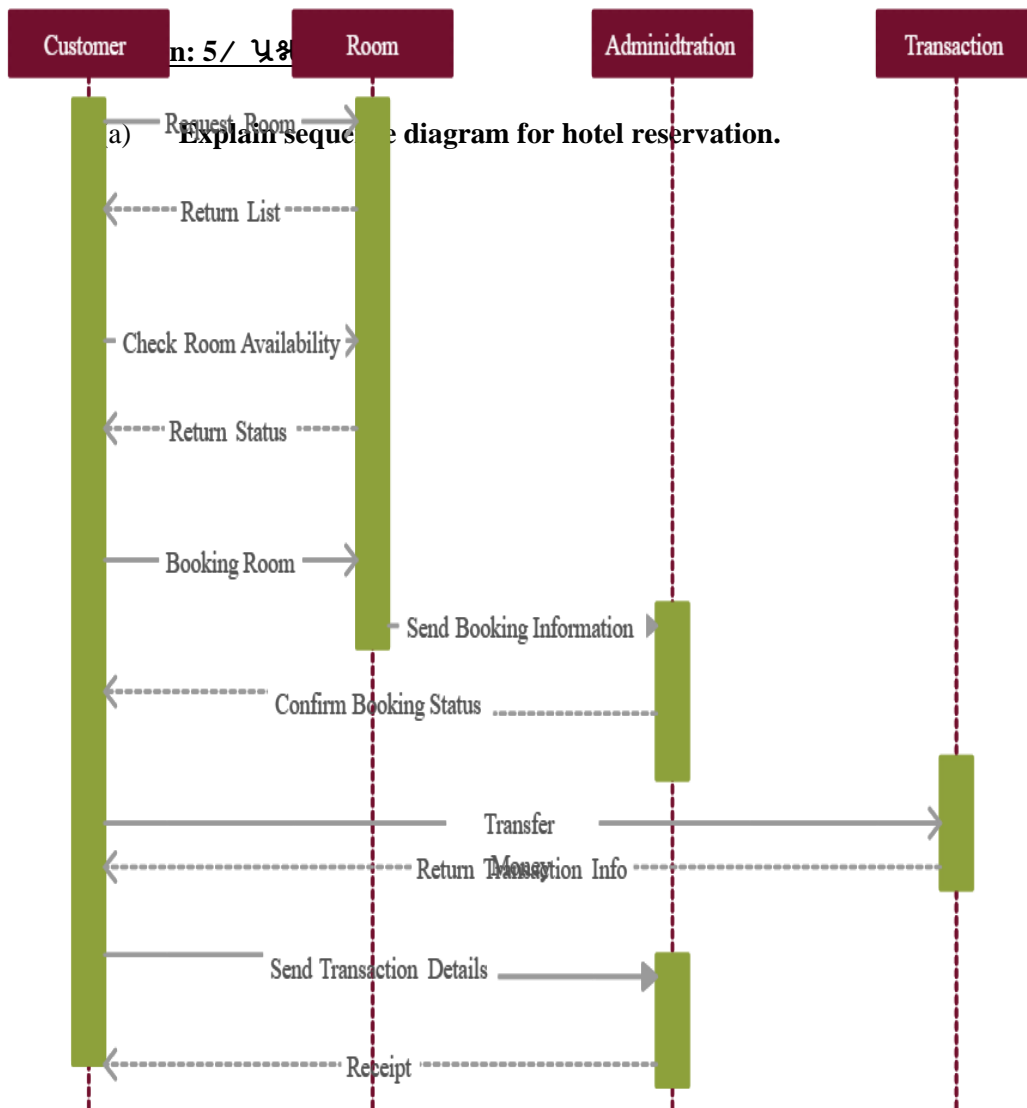
[8]

- The main purpose of the deployment diagram is to represent how software is installed on the hardware component. It depicts in what manner a software interacts with hardware to perform its execution.
- The deployment diagram does not focus on the logical components of the system, but it put its attention on the hardware topology.
- Following are the purposes of deployment diagram enlisted below:
  - To envision the hardware topology of the system.
  - To represent the hardware components on which the software components are installed.
  - To describe the processing of nodes at the runtime
- The deployment diagram consist of the following notations:
  - A component
  - An artifact
  - An interface
  - A node



### EXAMPLE





[8]

- (b) Explain Software Architecture in detail.

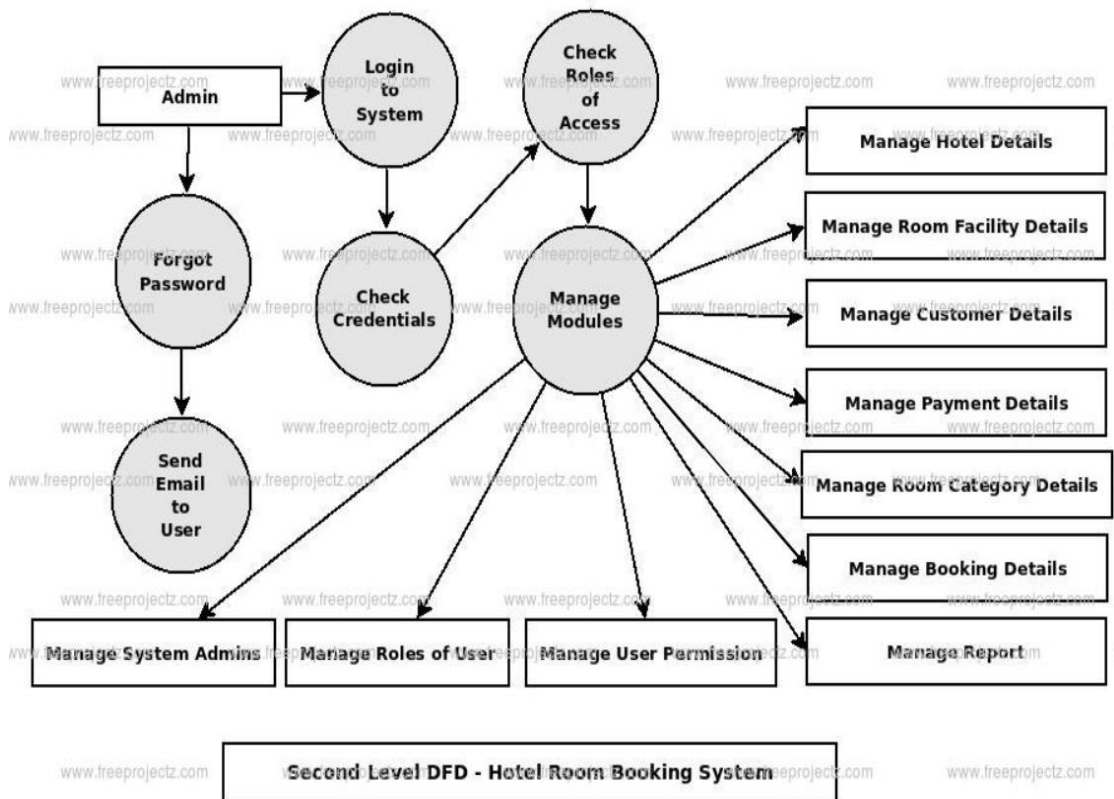
[8]

Software architecture refers to the high-level design and organization of a software system. It defines the components of the system, how they interact with each other, and the overall structure of the system. In essence, it is the blueprint or roadmap for how the software will be built.

**OR**

- (a) Explain DFD diagram 2 for hotel reservation

[8]



(b) Explain Layered Pattern in detail

[8]

As the name suggests, components(code) in this pattern are separated into layers of subtasks and they are arranged one above another.

Each layer has unique tasks to do and all the layers are independent of one another. Since each layer is independent, one can modify the code inside a layer without affecting others.

It is the most commonly used pattern for designing the majority of software. This layer is also known as 'N-tier architecture'. Basically, this pattern has 4 layers.

- Presentation layer (The user interface layer where we see and enter data into an application.)
- Business layer (this layer is responsible for executing business logic as per the request.)
- Application layer (this layer acts as a medium for communication between the 'presentation layer' and 'data layer'.

Data layer (this layer has a database for managing data.)

Ideal for:

E-commerce web applications development like Amazon.

Note : Need to draw the diagram

### Question: 6/ ୩୫.୫.

(a) Explain ASD agile model in detail

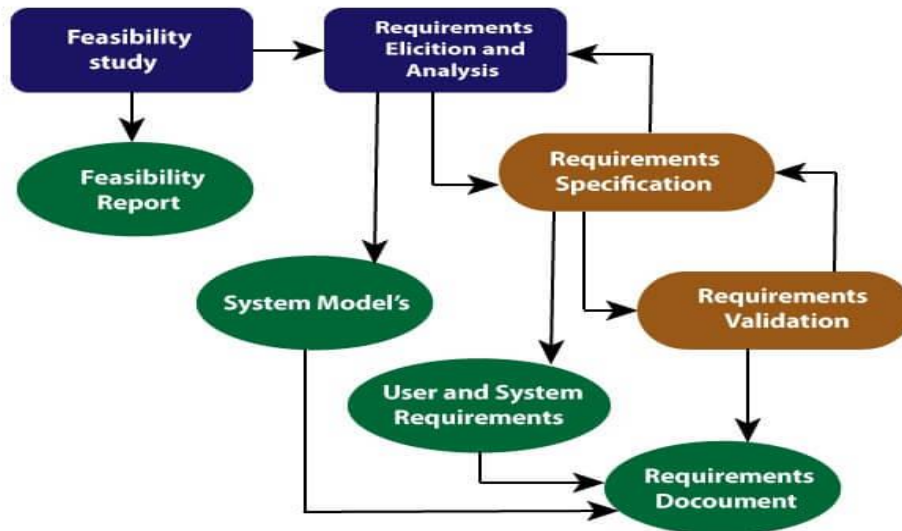
[8]

- This is a technique for building complex software systems using iterative approach.
- ASD focus on working in collaboration and team self-organization.
- ASD — distinguishing features
  - Component-based focus
  - Uses “time-boxing”
  - Explicit consideration of risks
  - Emphasizes collaboration for requirements gathering
- Emphasizes “learning” throughout process



(b) Explain Requirement Engineering Process. [4]

- It is a four-step process, which includes -
- Feasibility Study
- Requirement Elicitation and Analysis
- Software Requirement Specification
- Software Requirement Validation



**Requirement Engineering Process**

(c) Define following terms: [4]

- i) Association ii) Generalization iii) Composition iv) Multiplicity  
In the context of object-oriented programming and UML (Unified Modeling Language), the following terms have the following definitions:

i) Association: Association refers to a relationship between two or more objects or classes. It indicates that the objects or classes are related to each other in some way, such as through a shared attribute or method. An association is typically represented in a UML class diagram using a line connecting the classes, with an arrow indicating the direction of the relationship.

ii) Generalization: Generalization refers to a relationship between a more general class (called the superclass or parent class) and one or more more specific classes (called the subclass or child class). It indicates that the subclass inherits the attributes and behaviors of the superclass, and can also add its own attributes and behaviors. Generalization is represented in a UML class diagram using an arrow pointing from the subclass to the superclass.

iii) Composition: Composition is a type of association that indicates a strong relationship between two or more objects or classes, where one object or class is part of another object or class. It indicates that the composed object or class cannot exist without the container object or class. Composition is represented in a UML class diagram using a diamond symbol on the container object or class end of the line connecting the two classes.

iv) Multiplicity: Multiplicity is a UML notation used to indicate the number of instances of a class that can be associated with another class. It is represented as a range of numbers or a specific number, such as "0..1" to indicate that the associated class can have zero or one instances, or "1..\*" to indicate that the



associated class must have at least one instance. Multiplicity is typically shown near the end of the association line in a UML class diagram.

OR

- (a) Explain Requirement Elicitation Techniques in detail. [8]

- Interviews
- Surveys
- Questionnaires
- Task analysis
- Domain Analysis
- Brainstorming
- Prototyping
- Observation

- (b) What are the Attributes of a good SRS [4]

- Concise: The SRS report should be brief and at the same time, unambiguous, consistent, and complete. Verbose and unrelated descriptions decrease readability and also increase error possibilities.
- Structured: It should be well organized. A well organized document is simple to understand and change. In practice, the SRS document undergoes several alterations to cope up with the user requirements. Often, user requirements grow over a span of time. Therefore, to make the modifications to the SRS document simple, it is vital to make the report well organized
- **Black-box view:** It should only state what the system should do and avoid doing from stating how to do these. This means that the SRS document should identify the external behavior of the system and not discuss the implementation issues.
- Black-box view: It should only state what the system should do and avoid doing from stating how to do these. This means that the SRS document should identify the external behavior of the system and not discuss the implementation issues.

- (c) Draw and explain different notations of class diagram with example. [4]

The class diagram depicts a static view of an application. It represents the types of objects residing in the system and the relationships between them. A class consists of its objects, and also it may inherit from other classes. A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code.

It shows the attributes, classes, functions, and relationships to give an overview of the software system. It constitutes class names, attributes, and functions in a separate compartment that helps in software development. Since it is a collection of classes, interfaces, associations, collaborations, and constraints, it is termed as a structural diagram.

It analyses and designs a static view of an application.

It describes the major responsibilities of a system.

It is a base for component and deployment diagrams.

It incorporates forward and reverse engineering.

It can represent the object model for complex systems.

It reduces the maintenance time by providing an overview of how an application is structured before coding.

It provides a general schematic of an application for better understanding.

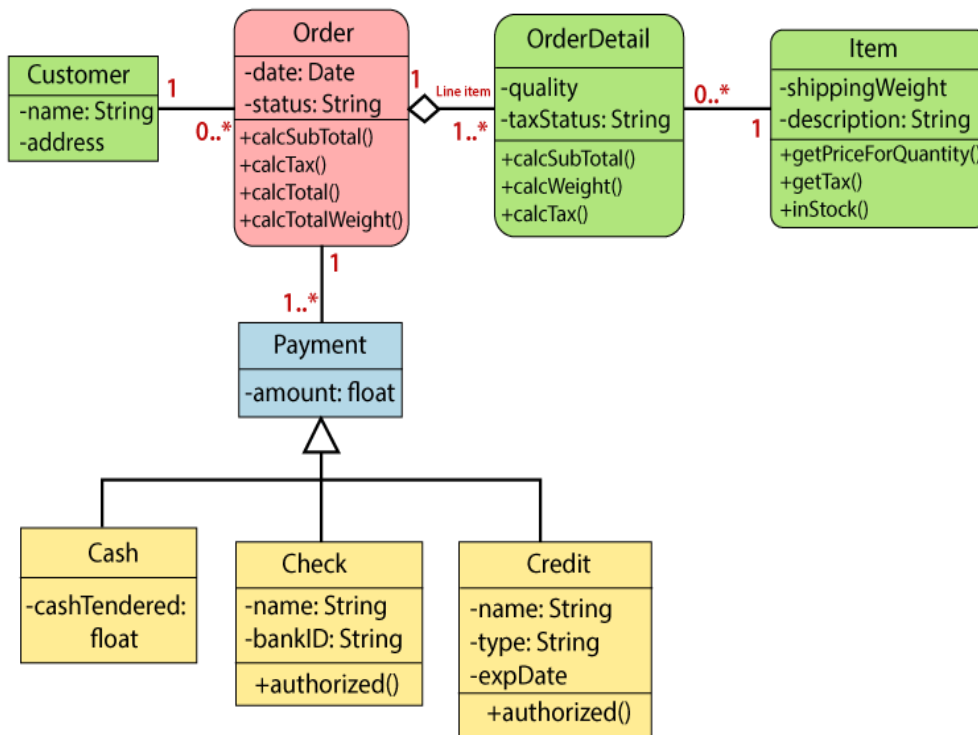
It represents a detailed chart by highlighting the desired code, which is to be programmed.

It is helpful for the stakeholders and the developers.

Upper Section: The upper section encompasses the name of the class. A class is a representation of similar objects that shares the same relationships, attributes, operations, and semantics

Middle Section: The middle section constitutes the attributes, which describe the quality of the class

Lower Section: The lower section contain methods or operations. The methods are represented in the form of a list, where each method is written in a single line. It demonstrates how a class interacts with data.

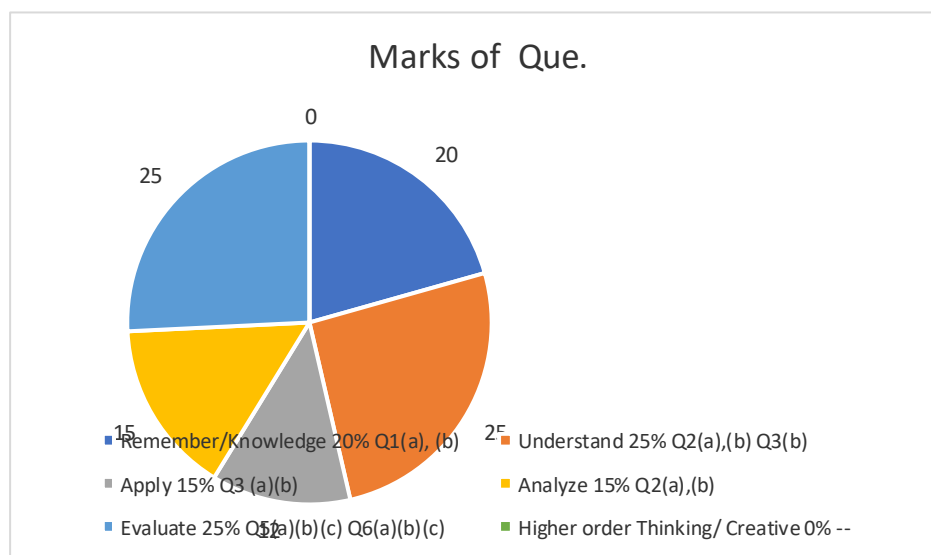


**\*\*Best of Luck (શુભેચ્છા)\*\***

## – Bloom's Taxonomy Report –

**Sub: Software Engineering****Sem. 6<sup>th</sup>****Branch: ICT (FOT)****Que. Paper weightage as per Bloom's Taxonomy**

| LEVEL                           | % of weightage | Question No.            | Marks of Que. |
|---------------------------------|----------------|-------------------------|---------------|
| Remember/Knowledge              | 20%            | Q1(a), (b)              | 20            |
| Understand                      | 25%            | Q2(a),(b) Q3(b)         | 25            |
| Apply                           | 15%            | Q3 (a)(b)               | 12            |
| Analyze                         | 15%            | Q2(a),(b)               | 15            |
| Evaluate                        | 25%            | Q5(a)(b)(c) Q6(a)(b)(c) | 25            |
| Higher order Thinking/ Creative | 0%             | --                      | --            |

**Chart/Graph of Bloom's Taxonomy**

## Course Outcome Wise Questions

|              |                 |         |                             |
|--------------|-----------------|---------|-----------------------------|
| Subject Code | <b>01CT0615</b> | Subject | <b>SOFTWARE ENGINEERING</b> |
|--------------|-----------------|---------|-----------------------------|

| CO No.     | Course Outcome  |
|------------|---|
| <b>CO1</b> | Understand various software engineering principles and their application                  |
|            | <b>1(A), 1(B), 5(C)</b>   |
| <b>CO2</b> | Demonstrate use of various Agile methodologies for software development                   |
|            | <b>1(A), 1(B), 3(C), 4(A), 5(C-Or), 6(A-Or)</b>   |
| <b>CO3</b> | Apply various modeling techniques for designing system requirement                        |
|            | <b>1(B), 2(B), 3(A), 3(A-Or), 4(A-Or), 4(B), 4(B-Or), 5(B-Or), 6(C)</b>                   |
| <b>CO4</b> | Identify different types of risk and evaluate its impact on software system               |
|            | <b>1(B), 3(B), 3(C-Or), 5(A), 6(A)</b>  |
| <b>CO4</b> | Distinguish different testing strategies and Create test cases                            |
|            | <b>1(A), 2(A), 2(B-Or), 3(B-Or), 5(A-Or), 5(B), 6(B), 6(C-Or)</b>                         |
| <b>CO6</b> | Able to understand and apply the basic project management practices in real life projects |
|            | <b>1(B), 6(B-Or)</b>  |

| Blooms Taxonomy                         | Question List   |
|---|---|
| <b>Remember / Knowledge</b>             | 1(A), 1(B), 4(A)  |
| <b>Understand</b>                       | 1(A), 1(B), 2(B), 3(B), 4(B), 5(A), 5(C), 5(C-Or), 6(A), 6(A-Or), 6(B), 6(C-Or) |
| <b>Apply</b>                            | 1(A), 1(B), 2(B-Or), 3(A), 3(B-Or), 4(B-Or), 5(A-Or), 5(B), 5(B-Or), 6(C)       |
| <b>Analyze</b>                          | 1(B), 2(A), 3(C), 3(C-Or), 4(A-Or), 6(B-Or)                                     |
| <b>Evaluate</b>                         | 1(B), 3(A-Or)   |
| <b>Higher order Thinking / Creative</b> |   |