1. **Draw a Use Case Diagram**



1. **Derive Boundary Classes, Controller classes, Entity Classes.**

**Boundary Class:**  It used to handle interactions between the system and external actors.

CustomerInterface

PaymentGatewayInterface

**Controller Class:** Controller classes manage the flow of data between boundary and entity classes.

PaymentController

**Entity Class:** It represents the core data and business logic of the application.

Example: Customer, Payment

1. **Place these classes on a three tier Architecture.**

**Presentation Layer:**

CustomerInterface

PaymentGatewayInterface

**Business Logic:**

PaymentController

WalletController

**Data Layer:**

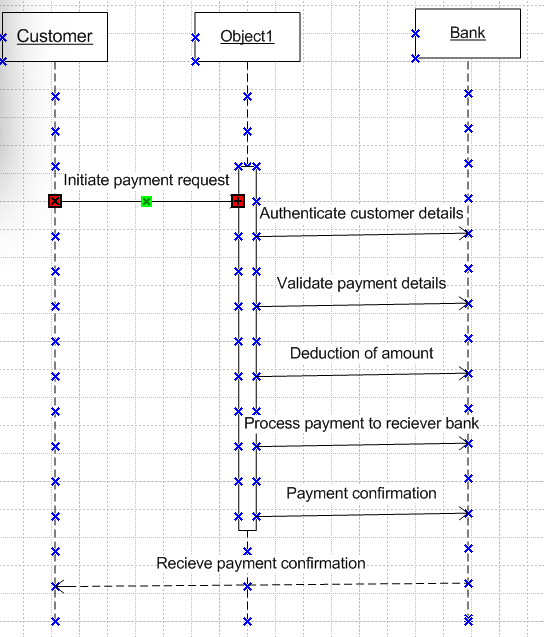
Customer

Payment

1. **Explain Domain Model for Customer making payment through Net Banking.**

A domain model is a conceptual representation that define the structure, relationship, and behaviors of entities within a specific problem domain

1. **Draw a sequence diagram for payment done by Customer Net Banking**

****

1. **Explain Conceptual Model for this Case**

A conceptual model represents the high-level structure of a system, abstracting the main entities, relationships, and their interactions without delving into technical implementation. It helps understand the business domain, clarify requirements, and serve as a foundation for system design.

Key Elements of a conceptual Model:

1. Entities – Customer, Product, Order and Payment
2. Attributes – customerID, Name, email, phoneNumber, order Id, mode of payment
3. Relationships- A customer can make payment with any modes.
4. **What is MVC architecture? Explain MVC rules to derive classes from use case diagram and guidelines to place classes in 3-tier architecture - 8 Marks**

MVC (Model-View-Controller) is a design pattern that separates an application into three components:

Model: Represents the data and business logic.

View: Handles the user interface (UI) and presentation.

Controller: Manages user input, interacts with the model, and updates the view.

MVC Rules to Derive Classes from Use Case Diagram

Model Classes: Derived from entities and business logic in the use case diagram. These classes represent data and the logic behind processing or modifying it (e.g., User, Product, Order).

View Classes: Derived from UI components or interactions in the use case diagram. They handle the display and user interaction (e.g., UserView, ProductView).

Controller Classes: Represent actions or behaviors in the use case diagram. These classes handle the processing of user input and interaction between the model and view (e.g., UserController, ProductController).

Guidelines to Place Classes in 3-Tier Architecture

Presentation Layer (View):

Contains the View classes. It is responsible for displaying the UI and sending user input to the controller.

Example: UserView, ProductView.

Business Logic Layer (Controller & Model):

Controller handles user input, while the Model manages business logic.

Example: UserController, ProductController, UserModel.

Data Access Layer:

The Model classes may interact with the database and handle data retrieval.

Example: UserModel, ProductModel interacting with the database

1. **Explain BA contributions in project (Waterfall Model – all Stages)**

|  |  |  |
| --- | --- | --- |
| **Stage** | **Activities** | **Artifacts & resources** |
| Requirement Gathering | 1. Identify stakeholders and their needs. 2. Conduct elicitation sessions (e.g., interviews, workshops). 3. Analyze and validate requirements. 4. Define scope and constraints. | Business Requirements Document (BRD) Functional Requirements Document (FRD) Use Cases/User Stories Traceability Matrix Stakeholder List elicitation Techniques (e.g., surveys, JAD sessions) |
| System Design | 1. Collaborate with architects and developers to translate requirements into designs. 2. Validate wireframes, mock-ups, and prototypes. 3. Review technical specifications to ensure alignment with business needs. | Wireframes/Prototypes Data Flow Diagrams (DFDs) Entity Relationship Diagrams (ERDs) Updated Requirements Documents Design Specifications |
| Implementation | 1. Address developer queries and provide clarifications. 2. Manage and document requirement changes. 3. Facilitate communication between stakeholders and technical teams. | Clarification Logs Change Requests (CRs) Impact Analysis Reports Development Progress Reports |
| Testing | 1.Collaborate with QA teams to validate test cases against requirements. 2. Plan and conduct User Acceptance Testing (UAT). 3.Track and resolve defects with developers. 4.Validate test results. | Test Case Reviews UAT Plans and Scenarios Defect Logs UAT Sign-Off Document |
| Deployment | 1.Support deployment planning and go-live execution. 2. Train end-users and prepare user documentation. 3. Validate the live system against requirements. 4. Provide post-deployment support. | Deployment Plans User Manuals Training Materials Post-Deployment Validation Reports Feedback Logs |
| Maintenance | 1.Analyze and resolve user-reported issues. 2. Document and evaluate change requests. 3. Identify areas for system optimization. 4. Propose enhancements based on feedback. | Issue Logs Change Requests (CRs) Continuous Improvement Proposals Maintenance Reports |

1. **What is conflict management? Explain using Thomas – Kilmann technique**

Conflict management is the process of identifying, addressing, and resolving disputes or disagreements between individuals, teams, or organizations to minimize negative outcomes and enhance collaboration and understanding. It focuses on finding mutually acceptable solutions while maintaining relationships and achieving organizational goals.

Effective conflict management ensures that conflicts are addressed constructively, leading to better decision-making, improved relationships, and increased productivity.

**Thomas-Kilmann Conflict Management Model**

The **Thomas-Kilmann Conflict Mode Instrument (TKI)** is a widely used framework for understanding and managing conflicts. It identifies five conflict-handling modes based on two dimensions:

1. **Assertiveness**: The degree to which a person tries to satisfy their own needs.
2. **Cooperativeness**: The degree to which a person tries to satisfy the needs of others.

The combination of these dimensions results in the following five conflict-handling styles:

**1. Competing (High Assertiveness, Low Cooperativeness)**

* **Description**:
  + Focuses on satisfying one's own needs, often at the expense of others.
  + Involves a win-lose approach and can be forceful or authoritative.
* **When to Use**:
  + When quick, decisive action is needed (e.g., during emergencies).
  + In situations where an unpopular decision must be enforced.
* **Drawback**:
  + May damage relationships if overused.

**2. Collaborating (High Assertiveness, High Cooperativeness)**

* **Description**:
  + Seeks to find a win-win solution that satisfies all parties.
  + Involves open communication, mutual respect, and a focus on problem-solving.
* **When to Use**:
  + When the issue is highly important to all parties.
  + When building long-term relationships is crucial.
* **Drawback**:
  + Time-consuming and requires significant effort.

**3. Compromising (Moderate Assertiveness, Moderate Cooperativeness)**

* **Description**:
  + Focuses on finding a middle ground where each party gives up something to reach a mutually acceptable solution.
  + Balances assertiveness and cooperativeness.
* **When to Use**:
  + When a quick resolution is needed, and both parties have equal power.
  + When competing goals must be balanced.
* **Drawback**:
  + May result in suboptimal outcomes, as neither party fully achieves their goals.

**4. Avoiding (Low Assertiveness, Low Cooperativeness)**

* **Description**:
  + Involves sidestepping or withdrawing from the conflict without addressing it.
  + Focuses on delaying or ignoring the issue.
* **When to Use**:
  + When the issue is trivial or when emotions are too high to address the conflict constructively.
  + When more information or time is needed.
* **Drawback**:
  + Can lead to unresolved conflicts and missed opportunities.

**5. Accommodating (Low Assertiveness, High Cooperativeness)**

* **Description**:
  + Prioritizes satisfying the needs of others over one's own needs.
  + Involves yielding to maintain relationships or harmony.
* **When to Use**:
  + When preserving relationships is more important than the issue.
  + When one party values the outcome less than the other.
* **Drawback**:
  + Can lead to resentment or exploitation if overused.

1. **List down the reasons for project failure.**

* Wrong Business Requirements: - A thorough analysis of requirements is essential at the initiation phase to avoid future complications. Proper requirements analysis will only lead to delivering correct product.
* Poor planning: - Planning is the first and foremost and the most important step in the journey of the project. It lays the foundation of a project. If planning is improper, the desired outcome will not fall into place.
* Continuous change in requirements: - Especially if the project is dynamic in nature, this becomes very true. Any change request in business needs can lead to the business case becoming outdated before a project has been completed leading to total failure.
* Unrealistic expectations: - It is important that the right people are assigned to perform tasks based on their skills and performance metrics. The concerned person should have awareness and in-depth knowledge of he/she will be doing in the project. Expectations should be realistic.
* Lack of user involvement: - It is essential to engage user during the project lifecycle as their suggestions and feedback play an important role. Failure in user involvement in the project leads to an improper result and the product fails to meet the user expectations.
* Lack of executive support: - Project may also fail if there’s no proper guidance or support from the executives. This is because, without effective leadership, team members will not be able to work together effectively.

1. **List the Challenges faced in projects for BA?**

Challenges that are faced by a business analyst in real time are as follow:

**Lack of Training-** it includes all those areas which are directly or indirectly related to the project by efficient training to the people involved in project we can produce a good and efficient result for the organization.

**Obtaining sign off requirements** - This is one of the major area where a business analyst struggles for while taking sign off on requirement it is necessary for him/her to convince them to the best that all the requirements will be satisfied for taking sign off they have to keep on approaching the clients and be in touch, book an appointment to have the sign off on requirements as well.

**Changing business needs and requirements:** Business analyst have to come up with all the changes that comes in a management while completion of the Project, he/she has to put on the whole process by being involved in the process and managing the changes in the organization.

**Coordination between developers and** testers -In any project if the coordination between developers and tester is not well managed then the project success could not be assumed.in any project it is must to create a coordination between both these teams in an efficient way.

**Conduct meetings**: Conducting meeting is very important and necessary, which is managed by the business analyst as it creates a very great impact in the view of stakeholders. one must note down one thing is to create effective meeting and minutes and agenda should be prepared well in advance.

**Driving clients for UAT completion:** The UAT completion must by the clients, and this will be the main part of the project which must be looked up by business analyst.

**People Management** -People management is the main quality that a business analyst must have, this helps the business analyst to control all the fields in a proper way by which they may create a well and healthy atmosphere with a productive result as well

1. **Write about Document Naming Standards**

Document Naming Standards are guidelines that help ensure consistency, clarity, and easy retrieval of documents within an organization or project. These standards typically involve using a specific naming convention that includes relevant information, such as document type, version, date, and project identifiers. A well-defined naming convention reduces confusion and helps team members quickly identify and organize documents.

Key Elements in Document Naming Standards:

Project/Document Identifier: A short code or acronym representing the project or document type.

Example: HR, PROJ

Document Type: Indicating the nature of the document (e.g., BRD, FRD, SRS).

Example: BRD for Business Requirement Document.

Version Number: To track document revisions.

Example: v1, v2.

Date: A date format (e.g., YYYY-MM-DD) to indicate when the document was created or last updated.

Example: 2025-01-20

Descriptive Title: A brief description of the document's content.

Example: HR\_Policy\_v1\_2025-01-20

1. **What are the Do’s and Don’ts of a Business analyst**

|  |  |
| --- | --- |
| **Do’s** | **Don’ts** |
| 1. Understand the Business Needs | 1. Don’t Assume Requirements |
| 2. Engage with Stakeholders Regularly | 2. Don’t Ignore Stakeholder Input |
| 3. Document Requirements Clearly and Precisely | 3. Don’t Overcomplicate Requirements |
| 4. Prioritize Requirements Effectively | 4. Don’t Allow Scope Creep Without Control |
| 5. Facilitate Communication Between Teams | 5. Don’t Work in Silos |
| 6. Perform Thorough Analysis | 6. Don’t Skip Validation |
| 7. Embrace Change and Adapt | 7. Don’t Resist Change |
| 8. Use Appropriate Tools | 8. Don’t Rely Solely on One Tool |
| 9. Maintain a Strong Focus on Quality | 9. Don’t Ignore Testing |
| 10. Document and Track Changes | 10. Don’t Overlook the Impact of Changes |

1. **Write the difference between packages and sub-systems**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Packages** | **Sub-Systems** |
| **Definition** | A **package** is a grouping of related classes, interfaces, or components within a system, usually based on a certain functionality or feature. | A **sub-system** is a larger, more independent component within a system, which often includes multiple packages or modules to achieve a broader functionality. |
| **Scope** | Packages typically represent a logical grouping of related functionality within a single system or module. | Sub-systems are often larger in scope and may represent a distinct part of a larger system that can operate independently but still interacts with other sub-systems. |
| **Purpose** | The primary purpose of a package is to organize the classes or components to improve maintainability, scalability, and reusability within a specific module. | A sub-system serves a more significant purpose, such as encapsulating specific business functions or processes that can be deployed or run independently. |
| **Independence** | Packages are not usually independent; they exist within a larger system or sub-system and are used to group related classes or components. | Sub-systems are typically independent and often have their own internal structure, data, and even interfaces for interaction with other sub-systems. |
| **Communication** | Packages within a system communicate with each other through well-defined interfaces or APIs. | Sub-systems communicate with each other using defined communication protocols, services, or interfaces. |
| **Level of Abstraction** | Packages are a lower level of abstraction, focusing on grouping related functionality at a class or component level. | Sub-systems represent a higher level of abstraction and encapsulate larger functionalities or processes, often operating as distinct units within the overall system. |
| **Examples** | - PaymentProcessing package containing classes like CreditCardProcessor, PaymentGateway, etc. | - An Accounting Sub-System responsible for general ledger, invoicing, payroll, etc. |

1. **What is camel-casing and explain where it will be used.**

Camel casing (or camelCase) is a naming convention where multiple words are combined without spaces, and each word after the first starts with an uppercase letter, while the first word begins with a lowercase letter. This style resembles the humps of a camel, hence the name.

Example: userName, totalAmount, orderDate.

Where Camel Casing is Used:

Variable Names: Common in programming languages like JavaScript, Java, and C#.

Example: firstName, totalPrice

Function/Method Names: Used for function or method identifiers.

Example: calculateTotal(), getUserData()

API Endpoints: Used in RESTful API naming.

Example: /getUserInfo, /updateOrderStatus

File Names: Sometimes used for file naming where spaces are not allowed.

Example: userProfileData.json

1. **Illustrate Development server and what are the accesses does business analyst has?**

A development server is an environment where developers build, test, and debug applications before deployment. It is separate from the production server, ensuring that testing and development do not impact live systems.

Business Analyst Access to Development Server:

Read-Only Access: BAs can view documentation, logs, or test results.

Test Environment Access: They may test features in a staging or QA environment to ensure they meet business requirements.

API/Service Access: To verify data flows and integrations.

Reports and Data: Access to generated reports for validation purposes.

BAs typically do not have direct access to the code but interact with the development team for updates and testing.

1. **What is Data Mapping**

Data mapping is the process of connecting data elements from one system or database to another. It involves defining how fields in a source system correspond to fields in a target system, ensuring that data is accurately transformed, validated, and transferred. Data mapping is essential in scenarios like data migration, integration, and ETL (Extract, Transform, Load) processes. It ensures that data flows correctly between systems and is formatted to meet the requirements of the destination, maintaining consistency and integrity across different platforms.

**Key Aspects of Data Mapping**

Source Data: The original data, which can be in various formats such as relational databases, flat files (CSV, JSON, XML), or external systems.

Target Data: The destination where the data needs to be mapped, such as another database, application, or data warehouse.

Mapping Rules: The rules or logic that dictate how the source data is transformed into the target data, which may involve data transformation, cleansing, or validation.

Data Types and Formats: Ensuring that the data types (e.g., integer, string, date) in the source system match the expected data types in the target system.

Data Transformation: Any operations that need to be performed to convert or adjust data during the mapping process (e.g., date format conversion, unit conversions, etc.).

**Steps in Data Mapping**

Identify Source and Target Systems: Determine where the data is coming from (source) and where it needs to go (target).

Analyze Data Structures: Review the source and target schemas (databases, tables, files) to understand how data is organized in both systems.

Define Mapping Rules: Establish how each field in the source will be mapped to the corresponding field in the target, considering data transformations, validation, and any required conversions.

Perform Data Transformation: If necessary, perform data transformations such as changing formats, calculating values, or merging fields.

Validate the Mapping: Ensure the data mapping logic is accurate and that the target system can interpret the mapped data correctly.

Testing: Test the data mapping by running sample data through the mapping process to ensure everything works as expected.

1. **What is API. Explain how you would use API integration in the case of your application Date format is dd-mm-yyyy and it is accepting some data from Other Application from US whose Date Format is mm-dd-yyyy**

API stands for application programming interface. An API are mechanism that enables two software components to communicate with each other using a set of definition and protocol. Developers use APIs to make their jobs more efficient by reusing code from before and only changing the part that is relevant to the process they want to improve. A good API makes it easier to create a program

API stands for Application programming interface. Interface can be thought of as a contract of service between two applications. API is a way for two or more computer programs to communicate with each other. API is a software interface that allows two applications to interact with each other.

Let’s understand via one example, you want to book your flight ticket using amazon/paytm of Indigo flight for Mumbai to Goa destination but the amazon/paytm have not excess of Indigo company, in this case via API the application show the require details like available date/seat etc.

API is the intermediate of two applications. using API we can go for the booking. An API used to integrate new applications with existing software system. It serves as the connection between two applications, letting them exchange data. This increase development speed because each functionality doesn't have to be written from scratch