Question 2: Derive Boundary Classes, Controller classes, Entity Classes

Boundary classes interact with the user, displaying options and collecting input. For example, PaymentGatewayBoundary handles payment method selection and PaymentResultBoundary shows the payment result.

Controller classes manage the payment flow; the PaymentController processes the selected payment method, interacting with boundary classes.

Entity classes represent the payment methods themselves, like CardPayment, WalletPayment, CashPayment, and NetBankingPayment, each implementing a PaymentMethod interface to process payments.

The system's structure ensures clear separation of concerns, with boundary classes managing user interaction, controller classes orchestrating logic, and entity classes handling core operations.

Question 3: Place these classes on a three tier Architecture

A **3-tier architecture** is a way to organize an application into three separate layers, each with a specific role. This structure helps make the system more organized, scalable, and easier to manage.

|  |
| --- |
| Application layer |
| PaymentGatewayBoundary  PaymentResultBoundary |

|  |
| --- |
| Business Logic Layer |
| PaymentController |

|  |
| --- |
| Data Layer |
| CardPayment WalletPayment CashPayment NetBankingPayment |

Question 4:

Question 7: What is MVC architecture? Explain MVC rules to derive classes from use case diagram and guidelines to place classes in 3-tier architecture

MVC stands for Model-View-Controller, a software architectural pattern commonly used for developing user interfaces that separates an application into three interconnected components. This separation assists in managing complex applications by enabling modularization and separate development.

- Model: Represents the data and the business logic of the application. It directly manages the data, logic, and rules of the application. The Model component is responsible for retrieving data from storage, processing it, and sending it to the View.

- View: Represents the presentation layer, which displays data to the user. It reflects the state of the Model and presents it in a specific format, allowing users to interact with the application.

- Controller: Acts as an intermediary between the Model and the View. It receives user input from the View, processes it (often involving changes to the Model), and returns the results to the View for display

Rules to derive the classes from use case diagram-

1. Combination of One actor and a use case results in one boundary class.

2. Combination of Two actors and a use case results in two boundary class.

3. Combination of Three actors and a use case results in three boundary class.

4. Use case will result in a controller class.

5. Each actor will result in one entity class.

Consider the example of Online shopping application with the following use case:

Model Classes-

Customer, Payment, Net Banking, Card, Cash

View Classes-

Login View, PaymentOptionView, NetBankingView, BankSelectionView, CredentialsView, PaymentAmountView, PaymentConfirmationView, LogoutView

Controller Classes-

Login Controller, PaymentOptionController, NetBankingController, BankSelectionController, CredentialsController, PaymentAmountController, PaymentConfirmationController, LogoutController

Guidelines to place classes in 3-tier architecture-

Presentation Layer-

This layer is nothing but a user interface.

View is inside this layer.

These classes interact directly with the user.

Presentation layer is responsible for displaying information and receiving the input from the

user.

Application Layer-

This layer is nothing but business logic.

Model and controller are inside this layer.

Controller handles the user input, process the request and co-ordinates interaction between the model and view.

Data Layer-

Classes which are responsible for data access and storage are in this layer.

It contains the classes which interacts with the database, files and other data sources.

Question 8: Explain BA contributions in project (Waterfall Model – all Stages)

**Requirement Gathering (Initial Phase)**

* Identify stakeholders and gather input from business users, clients, and other key participants
* Conduct interviews, workshops, surveys, and document analysis to gather detailed business requirements
* Categorize and prioritize requirements into functional (what the system must do) and non-functional (system performance, security, etc.) requirements
* Document the requirements in a clear, comprehensive **Software Requirements Specification (SRS)**
* Validate the requirements with stakeholders to ensure accuracy and completeness

**Design Phase**

* Review the business requirements document (SRS) to ensure all business needs are captured and understood.
* Work closely with system architects, designers, and developers to ensure that the design aligns with business objectives.
* Provide feedback on design proposals to ensure they address the user and business requirements.
* Create or review business process models, wireframes, and flow diagrams to visualize how the system will operate.
* Ensure that the design specification includes detailed technical solutions that meet business needs, such as performance and security requirements.

**Implementation (Development Phase)**

* Support the development team by clarifying requirements and resolving ambiguities in the documentation.
* Participate in progress reviews and ensure that the development is aligned with business expectations.
* Ensure that any changes or new requirements are properly documented and communicated to the development team.
* Help manage and prioritize requirements, making sure that the most critical business functions are implemented first.
* Communicate with stakeholders to keep them updated on development progress and ensure their needs are being met.

**Integration and Testing (Verification Phase)**

* Develop detailed **test cases** based on the business requirements and ensure they align with the functional and non-functional requirements.
* Collaborate with the testing team to verify that the system is being tested in line with business goals.
* Participate in **User Acceptance Testing (UAT)** by coordinating with end-users to validate the system against business needs.
* Review test results and work with the development team to resolve any issues or defects identified during testing.
* Ensure that all business requirements are met before the system is approved for deployment.

**Deployment (Release Phase)**

* Support the deployment team in preparing deployment plans and training materials.
* Assist in **user training**, ensuring that end-users understand how to use the system effectively.
* Ensure that system documentation (e.g., user manuals) is clear, accurate, and aligned with the final solution.
* Communicate with stakeholders and manage expectations around the deployment process and timeline.
* Monitor the deployment to ensure the system functions as expected and resolve any immediate issues that arise.

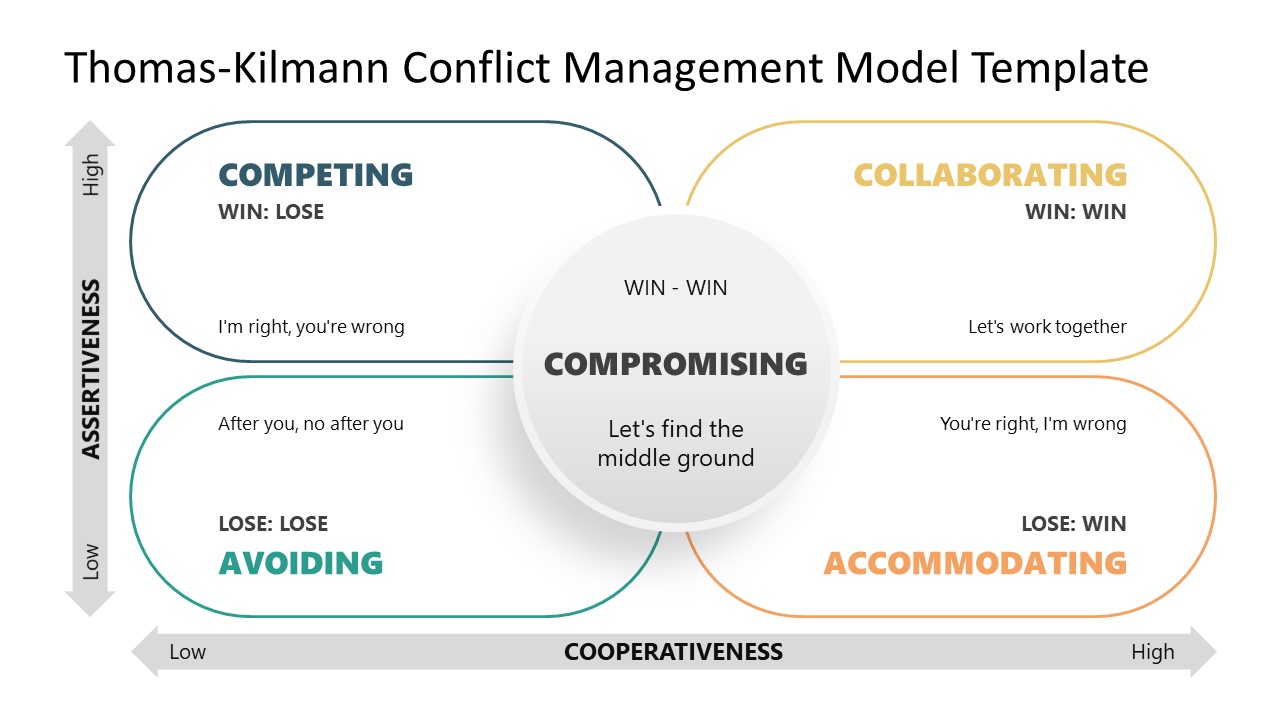
**Maintenance (Post-Deployment)**

* Gather feedback from users on system performance and identify any issues or improvement opportunities.
* Document user-reported issues and new business requirements for future updates or patches.
* Prioritize enhancements and bug fixes based on business needs and user feedback.
* Work with the development team to implement any required changes and updates.
* Provide ongoing support and ensure the system continues to meet the evolving needs of the business.

Question 9: What is conflict management? Explain using Thomas – Kilmann technique

Conflict management is the systematic and efficient handling of conflicts, aiming to minimize disruptions, enhance communication, and foster a collaborative environment, especially in business settings where conflicts arise from differing goals, values, personalities, and work styles among team members or stakeholders.

The Thomas-Kilmann Conflict Mode Instrument (TKI) is a widely recognized framework for managing conflicts, developed by Kenneth Thomas and Ralph Kilmann.



Competing (Win-Lose): High assertiveness, low cooperativeness. You push for your own needs, often at others' expense.

Accommodating (Lose-Win): Low assertiveness, high cooperativeness. You prioritize others' needs over your own.

Avoiding (Lose-Lose): Low assertiveness, low cooperativeness. You sidestep the conflict, hoping it resolves itself.

Collaborating (Win-Win): High assertiveness, high cooperativeness. You work with others to find a mutually beneficial solution.

Compromising (Partial Win/Partial Lose): Moderate assertiveness and cooperativeness. Both parties make concessions to reach a middle ground.

Question 10: List down the reasons for project failure

**Lack of Clear Objectives and Goals** - Undefined project goals can lead to differing interpretations among stakeholders, causing inefficiency and misalignment within teams.

**Poor Planning and Scheduling** - Inadequate planning causes unrealistic timelines and missing milestones. Lack of contingency planning leads to delays and cost overruns.

**Inadequate Resource Management** - Not having enough people, tools, or money can slow down progress. Giving team members too much work or not providing enough resources can cause burnout and lower their productivity.

**Unclear Roles and Responsibilities** - Unclear roles cause task overlap and missed responsibilities. This confusion results in errors and accountability problems.

**Poor Communication** - Ineffective communication and lack of timely updates lead to misunderstandings, missed deadlines, and unaddressed issues that escalate over time.

**Scope creep** - Scope creep causes project delays and increased costs when changes are introduced without proper evaluation, leading to deviations from original objectives.

**Lack of Stakeholder Engagement** - Stakeholders' lack of involvement and unmanaged expectations can lead to dissatisfaction and rejection of project deliverables

**Risk Management Failures** - Failing to proactively identify and mitigate risks leaves projects vulnerable to unforeseen challenges, leading to disruptions and potential failure

**Unrealistic Expectations** - Setting unrealistic expectations can lead to disappointment and perceived failure, even with progress, as overpromising without considering constraints sets the project up for failure.

**Lack of Change Management** - Insufficient planning for managing change can lead to project failure, as neglecting the human side of change causes resistance, confusion, and inefficiencies.

**Budget and Financial Mismanagement -** Poor budget planning and lack of financial oversight can lead to cost overruns, potentially halting the project or resulting in incomplete deliverables.

Question 11: List the Challenges faced in projects for BA

Lack of Domain Knowledge

Lack of training.

Obtaining sign-off on the requirement.

Change management.

Co-ordination between developers and testers.

Conducting meeting.

Making sure status report is effective.

Driving clients for UAT completion.

Making sure that the project is going on right track and delivered as per the timelines without any issues.

Gathering clear and unambiguous requirements can be challenging.

Unable to understand what stakeholder is trying to convey.

Scope creep- change in requirement or scope of the project during the project lifecycle can lead to scope creep.

Managing the stakeholder with conflicting interest can be a difficult task for BA.

BA may face difficulties in understanding the requirements if the domain is not familiar to him.

Question 12: Write about Document Naming Standards

Document Naming Standards ensure consistency and easy retrieval by using clear titles, version numbers, dates, and categories. Avoid special characters, use simple formats, and maintain uniformity for better file organization.

Ex. Suppose we have a project with the ID "PROJ123," and we're working with a Requirements Specification Document.

Project ID: PROJ123

Document Type: REQ

Version: 1.0

Date: 2024-05-26

The document identifier could be: PROJ123-REQ-1.0-2024-05-26.

Question 13: What are the Do’s and Don’ts of a Business analyst

|  |  |
| --- | --- |
| **Do's** | **Don'ts** |
| 1. Document Clear and Concise Requirements | 1. Don't Make Assumptions without Validation |
| 2. Communicate Effectively with Stakeholders | 2. Don't Ignore Stakeholder Feedback |
| 3. Prioritize Requirements Based on Business Value | 3. Don't Allow Scope Creep without Proper Control |
| 4. Understand Business Processes Thoroughly | 4. Don't Overcomplicate Solutions |
| 5. Ensure Requirements are Feasible and Realistic | 5. Don't Skip Validation or Testing |
| 6. Collaborate with Cross-Functional Teams | 6. Don't Work in Isolation |
| 7. Maintain Detailed and Organized Documentation | 7. Don't Forget to Track Changes |
| 8. Adapt and Be Flexible to Changing Requirements | 8. Don't Rush Through Requirement Gathering |

Question 14: Write the difference between packages and sub-systems

A package refers to a grouping of related classes, interfaces, or components that serve a specific function within a software application.

For example, in a project to develop a customer relationship management (CRM) system, a package might include classes related to user authentication, such as login and registration functionalities. This modular approach allows business analysts to focus on specific areas of the project and manage dependencies efficiently.

A sub-system is a larger structural entity that encompasses multiple packages and fulfills a broader functional role within the overall system.

For example, in the same CRM system, a sub-system could be the sales management module, which includes packages for order processing, customer tracking, and reporting. The sub-system integrates various functionalities and resources, allowing business analysts to oversee the interdependencies and interactions among different packages, ensuring that they work cohesively to meet business goals.

Question 15: What is camel-casing and explain where it will be used

The name "camel case" comes from the idea that the capital letters look like the humps on a camel's back. Camel-casing is a naming convention in programming where a multi-word name is written without spaces, and each word, except the first, starts with an uppercase letter. This makes it easier to read and understand the purpose of the variable, method, or function. Camel casing is commonly used for naming variables, functions, and methods in many programming languages, It is particularly helpful when naming elements in code, as it avoids spaces (which are not allowed in names) while maintaining clarity.

For example: A project folder for customer feedback might be named customerFeedbackReport (instead of customer\_feedback\_report or customer feedback report).

Question 16: Illustrate Development server and what are the accesses does business analyst has?

A Development Server is a server used by developers to create, test, and deploy applications in a controlled environment before moving them to a production server. It is primarily used during the development phase of a project to write, test, debug, and modify the application’s code.

Typical Access Levels for a BA:

Read-Only Access: BAs typically have read-only access to view the progress of the application, review functionality, and check if it meets business requirements.

Access to Documentation/Reports: BAs may also have access to logs, reports, or test case results from the development server to track the completion and functionality of different features.

Limited Testing Access: In some cases, BAs may be allowed to conduct light testing on specific features (UAT), but they don't typically modify the server environment or code

Question 17: What is Data Mapping

The database contains multiple tables in it.

There may come a scenario, where we need to map the data from one table to another.

Data mapping is necessary in cases where we want quick manner.

Data mapping is nothing but a process to establish connection between multiple data sources.

The purpose of data mapping is to ensure that the data is accurately transferred or converted into different format.

Data integration- While combining the data from different sources, it ensures that the data is properly matched.

Data Migration- While migrating the data from legacy system(source) to the new system(destination), the data elements are mapped accurately into the new system.

Required techniques are applied to covert the data into the format that is required by the new system.

Data Transformation- Data transformation means converting the data from one format to other.

In data mapping, data transformation plays very important role which ensures that the data of legacy system (source) is mapped correctly to the data in new system (destination).

Question 18: 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.

An API (Application Programming Interface) is a set of rules, protocols, and tools that allows different software applications to communicate with each other. It defines the methods and data formats that systems can use to request and exchange information. APIs enable systems to interact seamlessly without needing to understand the underlying code of the other system.

For the above scenario,

Establish API communication- set up API communication between your application and other application to exchange data.

Do Data formatting- while sending the data from one application to other, convert the date format from dd-mm-yyyy to mm-dd-yyyy.

While receiving the data from other application, parse the data and extract the date, month and year end re-arrange them accordingly.

Perform Data Validation and ensure that the converted date remains in a valid format