#  Project 3 Part 1

**Q1. Draw a Use Case Diagram**

**A. **

****

****

**Q2. Derive Boundary Classes, Controller classes, Entity Classes.**

**A. Boundary Classes (View/UI Layer):** This represent the interface between the system and external actors, such as users, external systems, or APIs. These classes handle user inputs and outputs but do not contain business logic.

**Characteristics:**

* Interact directly with users or external systems.
* Responsible for data collection and display.
* Do not store business logic or system data.
* May include Graphical User Interface (GUI) components, API endpoints, or command-line interfaces.

**2. Controller Classes (Control Layer):** This serve as intermediaries between boundary and entity classes. They process user inputs, apply business rules, and interact with entity classes to retrieve or modify data.

**Characteristics:**

* Handle system logic and coordinate between boundary and entity classes.
* Validate and process user inputs before passing them to entity classes.
* Ensure data consistency by enforcing business rules.
* May include decision-making mechanisms based on application logic.

**3. Entity Classes (Model/Business Layer):** This represent the core business objects in the system. They encapsulate data and define the fundamental structure and behavior of real-world objects modeled within the application.

**Characteristics:**

* Store and manage application data.
* Contain attributes and behaviors relevant to the domain model.
* May include validation rules and internal consistency checks.
* Typically mapped to database tables in a persistence layer.

**Q3. Place these classes on a three tier Architecture**

**A. **

**Q4. Explain Domain Model for Customer making payment through Net Banking**

**A.**

Client

Bank

Savings Account

Checking Account

Bank Assets

**Q5. Draw a sequence diagram for payment done by Customer Net Banking**

**A. **

****

**Q6. Explain Conceptual Model for this Case**

**A.**

**Q7. What is MVC architecture? Explain MVC rules to derive classes from use case diagram and guidelines to place classes in 3-tier architecture.**

**A. MVC (Model-View-Controller) architecture** is a software design pattern that separates an application into three interconnected components: **Model, View, and Controller**. This separation helps organize code, making applications easier to develop, maintain, and scale. The **Model** represents the application's data and business logic, handling database interactions and rules. The **View** is responsible for displaying the data to users and managing the UI (User Interface). The **Controller** acts as an intermediary between the Model and View, processing user inputs, updating data, and determining which view should be displayed. This structured approach allows developers to work on different components independently, improving efficiency and reusability.

**Advantages of MVC Architecture**

1. **Separation of Concerns**: Each component (Model, View, Controller) has a specific role, making code easier to manage.
2. **Reusability**: Components can be reused in different parts of the application, reducing redundancy.
3. **Scalability**: Supports the development of large-scale applications by allowing teams to work on separate modules simultaneously.
4. **Better Maintenance**: Debugging and updating the code is easier due to the clear separation of logic.
5. **Parallel Development**: Frontend and backend developers can work simultaneously, speeding up development time.
6. **Improved Code Organization**: Helps in keeping the application structured, leading to better readability and maintainability.

 Class Class

Business Layer or Logical Layer

Data Layer or Data Access Layer

Application or Presentation Layer

From or the project

Class

Business Layer or Logical Layer

**Q8. Explain BA contributions in project (Waterfall Model – all Stages)**

**A.** The Waterfall Model is a sequential software development methodology where each phase must be completed before moving to the next. BA plays a crucial role in every phase of this model.

1. Requirement Gathering & Analysis

* Conducts stakeholder interviews and workshops to gather requirements.
* Documents Business Requirement Document (BRD) and Functional Requirement Specification (FRS).
* Identifies business rules, constraints, and use cases.
* Validates requirements through stakeholder reviews.

2. System Design

* Assists in preparing system design documents by providing detailed business logic.
* Collaborates with the technical team to ensure requirements are accurately translated into design.
* Reviews wireframes, mockups, and process flow diagrams.
* Ensures data models align with business needs.

3. Implementation (Development)

* Clarifies requirements for developers and resolves ambiguities.
* Reviews development progress to ensure alignment with business requirements.
* Updates stakeholders on the development phase.
* Identifies potential risks and gaps in requirement implementation.

4. Testing (Verification)

* Creates and validates test cases, scenarios, and acceptance criteria.
* Conducts User Acceptance Testing (UAT) to ensure the product meets business needs.
* Works with QA teams to verify that requirements are correctly implemented.
* Documents and communicates defects, working with developers to resolve them.

5. Deployment (Implementation)

* Supports the deployment process by ensuring business readiness.
* Conducts training sessions and prepares user manuals for stakeholders.
* Assists in the transition from development to production.
* Ensures smooth go-live and provides post-implementation support.

6. Maintenance (Support & Feedback)

* Monitors system performance and gathers feedback from users.
* Identifies enhancements or changes needed for future releases.
* Ensures compliance with regulatory and business changes.
* Acts as a bridge between business users and IT support teams for issue resolution.

**Q9. What is conflict management? Explain using Thomas – Kilmann technique**

**A. Conflict management:** It is the process of resolving disagreements effectively while maintaining positive relationships and achieving organizational goals. It involves strategies to handle conflicts in a way that minimizes negative impacts and promotes collaboration.

**Thomas-Kilmann Conflict Model (TKI)** is a framework for understanding different approaches to managing conflicts. It is based on two key factors:

1. Assertiveness: The extent to which an individual tries to satisfy their own needs.
2. Cooperativeness: The extent to which an individual tries to satisfy others' needs.

Based on these factors, the model identifies five conflict resolution styles:

1. Competing (High Assertiveness, Low Cooperativeness)
	* A win-lose approach where one party pursues their goal at the expense of the other.
	* Suitable for quick decisions, emergencies, or when standing firm is necessary.
	* Example: A manager enforces a strict deadline despite resistance.
2. Collaborating (High Assertiveness, High Cooperativeness)
	* A win-win approach where both parties work together to find a mutually beneficial solution.
	* Requires open communication and problem-solving.
	* Example: Two teams work together to merge their ideas into a new strategy.
3. Compromising (Moderate Assertiveness, Moderate Cooperativeness)
	* A middle-ground approach where both parties make concessions.
	* Useful when a fair, quick resolution is needed.
	* Example: A project team agrees on a revised timeline to accommodate both business and technical needs.
4. Avoiding (Low Assertiveness, Low Cooperativeness)
	* The conflict is ignored or postponed.
	* Used when the issue is trivial, or when waiting is beneficial.
	* Example: An employee avoids engaging in a minor argument to maintain workplace harmony.
5. Accommodating (Low Assertiveness, High Cooperativeness)
	* One party prioritizes the other’s needs over their own.
	* Suitable when maintaining relationships is more important than winning.
	* Example: A team member agrees to another's proposal to promote teamwork.

This model helps individuals and organizations choose the right conflict resolution strategy based on the situation and desired outcome.

**Q10. List down the reasons for project failure**

**A.** Below are the possible reasons for project failure:

1. Poor Requirement Gathering

* Incomplete, ambiguous, or changing requirements can lead to a project that does not meet business needs.
* Lack of stakeholder involvement in requirement gathering results in incorrect assumptions.
* Example: A software project fails because user needs were not properly documented, leading to multiple revisions.

2. Lack of Proper Planning

* Inadequate project planning results in unrealistic timelines and budget miscalculations.
* Missing key milestones or overlooking dependencies leads to project delays.
* Example: A construction project stalls due to poor planning of resource allocation.

3. Weak Communication

* Poor communication among teams, stakeholders, and management causes misunderstandings.
* Lack of documentation and unclear instructions lead to incorrect implementations.
* Example: A project team assumes different interpretations of a requirement, resulting in a product that does not match expectations.

4. Scope Creep (Uncontrolled Scope Expansion)

* Additional features and requirements keep getting added without proper impact analysis.
* Leads to budget overruns and missed deadlines.
* Example: A website development project initially planned for five pages expands to 20 pages without timeline adjustments.

5. Inadequate Risk Management

* Not identifying potential risks early can cause major disruptions later in the project.
* Failing to create contingency plans for risks such as technology failures, regulatory changes, or stakeholder conflicts.
* Example: A software upgrade fails due to ignoring the risk of compatibility issues with existing systems.

6. Insufficient Budget Allocation

* Underestimating costs leads to financial constraints mid-project.
* Sudden cost-cutting can impact quality, resources, and timelines.
* Example: A mobile app project is abandoned due to a lack of funds for proper testing and deployment.

7. Lack of Stakeholder Involvement

* Stakeholders, including end-users, are not actively engaged, leading to misalignment of expectations.
* Delays in feedback and approvals result in project hold-ups.
* Example: A new HR software fails to gain adoption because HR managers were not involved in the design phase.

8. Poor Leadership and Project Management

* Ineffective decision-making and lack of clear vision can confuse and demotivate teams.
* Poor delegation, conflict resolution, and leadership skills lead to inefficiencies.
* Example: A project manager frequently changes priorities, causing confusion and delays.

9. Technology Challenges and Integration Issues

* Using outdated, incompatible, or untested technology leads to performance failures.
* Poor integration planning causes operational disruptions.
* Example: A new CRM system fails because it is incompatible with existing databases.

10. Inexperienced Project Team

* Team members without the required skills lead to slow progress and poor-quality output.
* Lack of proper training and experience affects project execution.
* Example: A team unfamiliar with Agile methodology struggles to adapt, causing inefficiencies.

11. Unrealistic Timelines and Deadlines

* Setting aggressive deadlines without considering realistic work effort.
* Rushed project execution results in poor quality and missed goals.
* Example: A software launch deadline is set too soon, leading to insufficient testing and numerous post-release issues.

12. Lack of Testing and Quality Control

* Insufficient testing results in bugs, security vulnerabilities, and system failures.
* Ignoring quality assurance processes leads to poor end-user experience.
* Example: A financial software launches without thorough testing, leading to incorrect transactions and customer complaints.

13. Vendor or Third-Party Failures

* Dependence on unreliable external vendors or suppliers causes unexpected delays.
* Poor contract management results in service disruptions.
* Example: A logistics software project fails due to a third-party API shutdown.

14. Resistance to Change

* Employees or stakeholders resist adopting new systems or processes, leading to low adoption rates.
* Poor change management strategies lead to confusion and dissatisfaction.
* Example: A company implements a new ERP system, but employees refuse to use it due to lack of training and awareness.

15. Legal or Compliance Issues

* Not following industry regulations, data security laws, or contractual obligations.
* Leads to legal penalties, lawsuits, or project shutdowns.
* Example: A healthcare application fails due to non-compliance with patient data privacy laws.

**Q11. List the Challenges faced in projects for BA**

**A.** Possible challenges and their solutions for a BA

1. Unclear or Evolving Requirements

Challenge: Stakeholders often provide vague, incomplete, or frequently changing requirements, leading to scope creep and rework.

Solution: Establish clear, detailed requirement documentation and use techniques like User Stories or Use Case Diagrams. Implement a Change Management Process to handle evolving requirements and ensure any changes are properly assessed before implementation.

2. Stakeholder Conflicts

Challenge: Conflicting interests and priorities between stakeholders can delay decisions and create tensions in the project.

Solution: Facilitate regular meetings and open communication channels. Use conflict resolution techniques like Thomas-Kilmann's Conflict Resolution Model and document key decisions to align all parties.

3. Poor Communication

Challenge: Miscommunication between teams and stakeholders can lead to misunderstandings and incorrect solutions.

Solution: Set up regular check-ins, use clear documentation, and adopt standardized communication methods (e.g., RACI matrices) to ensure clarity and avoid confusion.

4. Lack of Business or Domain Knowledge

Challenge: BAs may struggle with understanding business processes in new industries, leading to misaligned solutions.

Solution: Engage in research, use resources such as Subject Matter Experts (SMEs), and collaborate with business stakeholders to gather domain-specific knowledge and insights.

5. Inadequate Time for Requirement Gathering

Challenge: Tight timelines may result in rushed or incomplete requirement gathering.

Solution: Prioritize critical requirements, use time-boxing to limit the time spent on each task, and focus on high-value items first to ensure the most important requirements are gathered early.

6. Technology Constraints

Challenge: Existing systems and technology may limit the scope of feasible solutions, causing integration issues.

Solution: Collaborate with the technical team early to assess technology limitations and propose solutions that align with current systems. Consider system upgrades or workarounds where possible.

7. Managing Expectations

Challenge: Stakeholders may have unrealistic expectations regarding timelines, costs, and deliverables.

Solution: Set clear, realistic expectations upfront, and use regular progress updates to keep stakeholders informed. Use Agile methods to deliver incremental updates and manage expectations along the way.

8. Resistance to Change

Challenge: Employees or stakeholders may resist new processes or systems, which affects project adoption.

Solution: Implement effective Change Management strategies, including communication campaigns, training, and user involvement throughout the project lifecycle to ease the transition.

9. Poor Documentation Practices

Challenge: Inconsistent or incomplete documentation can result in confusion and missed requirements.

Solution: Use standardized documentation templates such as Business Requirements Documents (BRD) and Software Requirements Specifications (SRS). Regularly update documentation to reflect any changes in the project.

10. Difficulty in Validating Requirements

Challenge: It can be challenging to validate requirements with stakeholders due to differing perspectives.

Solution: Use prototyping, wireframes, or mock-ups to validate requirements early and ensure that they meet business needs. Conduct User Acceptance Testing (UAT) to confirm alignment.

11. Challenges in Agile Projects

Challenge: Agile projects often involve frequent changes in requirements, which can be overwhelming for the BA.

Solution: Regularly refine the product backlog, conduct frequent sprint planning sessions, and maintain clear communication with the Product Owner to keep the project on track.

12. Lack of Support from Leadership

Challenge: Insufficient backing from senior management can result in a lack of resources or project delays.

Solution: Regularly engage leadership through status reports and business case justification. Ensure they understand the strategic importance of the project and its outcomes.

13. Insufficient Access to End Users

Challenge: BAs may only have limited access to the actual users of the system, leading to a gap in user feedback.

Solution: Arrange direct user interviews, conduct surveys, and use usability testing to ensure that end-users' needs are captured accurately.

14. Difficulty in Measuring Success

Challenge: Defining and measuring success criteria for a project can be complex, especially when metrics are not clearly defined.

Solution: Collaborate with stakeholders to define Key Performance Indicators (KPIs) and Success Metrics at the beginning of the project, and ensure these metrics are tracked throughout the lifecycle.

15. Multiple Priorities and Workload

Challenge: BAs often juggle multiple projects or tasks, which can lead to workload management issues.

Solution: Prioritize tasks using techniques like the Eisenhower Matrix or Kanban boards, and leverage project management tools such as Trello or Asana to track and manage work effectively.

**Q12. Write about Document Naming Standards**

**A.** Document naming standards are essential for ensuring consistency, clarity, and easy access to project-related files. Proper naming conventions make it easier to organize, search for, and retrieve documents, especially in large or complex projects with multiple stakeholders and contributors. Below are the key components and best practices for creating effective document naming standards:

Key Components of a Naming Standard:

A typical document name should include the following components:

* **Document Type/Category:** This specifies the type of document, such as "BRD" for Business Requirement Document, "SRS" for Software Requirements Specification, or "UAT" for User Acceptance Testing.
* **Project/Module Name or Identifier:** A unique identifier for the project or module to which the document belongs (e.g., "HRSystem", "AppUpgrade").
* **Version Number:** A versioning system is crucial for tracking document revisions (e.g., "v1", "v2", "v1.2").
* **Date:** The date the document was created or last modified, often in the format of YYYYMMDD (e.g., "20250218").
* **Document Status (Optional):** Indicates the status of the document, such as "Draft", "Final", or "Reviewed".

Best Practices for Naming Documents:

A. Consistent Format

* Use a standard format for all document names to avoid confusion. A common format could be:
[DocumentType][ProjectName][Version][Date][Status]

B. Use Descriptive Terms

* Include descriptive names for clarity. Avoid using abbreviations that may confuse team members. For example, use "BusinessRequirement" instead of "BR" unless it is a commonly understood abbreviation.

C. Avoid Special Characters

* Refrain from using special characters like !, #, %, &, or spaces in file names. These can cause issues with file systems and file-sharing platforms. Instead, use underscores (\_) or hyphens (-) to separate words.

D. Use Leading Zeros for Versioning and Dates

* When using versioning or dates, add leading zeros to maintain uniformity, especially when dealing with numbers. For example, "v01" instead of "v1", or "20250218" instead of "2025218".

E. Keep It Short but Descriptive

* The file name should be short enough to avoid truncation while still being descriptive enough to identify its content. Avoid overly long file names.

F. Version Numbering: Keep track of document revisions with version numbers. Use a consistent numbering scheme, such as:

* v1 for the first draft.
* v1.1 for minor updates or changes.
* v2 for significant revisions.

G. File Naming for Updates: When updating a document, ensure that the version number and the modification date reflect the latest changes.

**Q13. What are the Do’s and Don’ts of a Business analyst**

**A. Do’s of a Business Analyst:**

1. **Understand Business Needs Thoroughly**
* Always gather comprehensive business requirements through stakeholder meetings, surveys, and research.
* Ensure a clear understanding of business objectives before moving forward with analysis.
1. **Communicate Effectively**
* Use clear and concise communication when interacting with stakeholders, developers, and testers.
* Translate complex business requirements into understandable documents for technical teams.
1. **Engage Stakeholders Regularly**
* Keep stakeholders involved throughout the project to avoid misunderstandings and ensure alignment.
* Schedule regular check-ins, workshops, and feedback sessions.
1. **Document Everything Clearly**
* Maintain proper documentation of business requirements, workflows, use cases, and functional specifications.
* Ensure all documents are version-controlled and accessible to relevant stakeholders.
1. **Use Visual Tools**
* Create flowcharts, wireframes, process diagrams, and mockups to visually represent requirements.
* Tools like Microsoft Visio, Lucidchart, and Balsamiq can help in creating clear visual models.
1. **Focus on Problem-Solving**
* Identify and analyze business problems before suggesting solutions.
* Ensure that the proposed solution aligns with business goals and adds value.
1. **Adapt to Changing Requirements**
* Stay flexible to accommodate requirement changes while maintaining proper change management.
* Ensure impact analysis is done before implementing changes.
1. **Collaborate with Cross-Functional Teams**
* Work closely with project managers, developers, testers, and end-users to ensure project success.
* Foster a culture of teamwork and knowledge-sharing.
1. **Prioritize Requirements**
* Classify requirements as **Must-have, Should-have, Could-have, and Won’t-have** (MoSCoW Method).
* Ensure critical business needs are addressed first.
1. **Ensure User-Centric Design**
* Always consider the end-user experience while defining solutions.
* Validate usability, accessibility, and efficiency in business processes.

**Don’ts of a Business Analyst**

1. **Don’t Assume Requirements**
* Never assume what the business needs without proper validation.
* Always confirm requirements with stakeholders before documenting them.
1. **Don’t Use Jargon with Business Stakeholders**
* Avoid using technical or complex terms when discussing requirements with non-technical stakeholders.
* Use business-friendly language and explain technical terms when necessary.
1. **Don’t Ignore Stakeholder Conflicts**
* Address conflicts and differing opinions among stakeholders proactively.
* Use facilitation techniques to resolve misunderstandings and find a common ground.
1. **Don’t Overload Documents with Unnecessary Information**
* Keep requirement documents structured, precise, and easy to read.
* Avoid redundant information that may cause confusion.
1. **Don’t Neglect Testing and Validation**
* Ensure that business requirements are properly tested before final implementation.
* Participate in **User Acceptance Testing (UAT)** and validate the solution meets business needs.
1. **Don’t Ignore Business Constraints**
* Consider budget, timeline, and resource constraints when defining requirements.
* Avoid proposing solutions that are unrealistic or unfeasible.
1. **Don’t Delay Decision-Making**
* Take proactive steps to resolve ambiguities and finalize requirements on time.
* Seek input from stakeholders but avoid unnecessary delays.
1. **Don’t Focus Only on Technical Aspects**
* A BA should consider business impacts, process improvements, and user experience along with technology.
* Think holistically and ensure solutions align with business strategy.
1. **Don’t Work in Isolation**
* A BA should collaborate with all relevant teams instead of working alone.
* Ensure continuous engagement with stakeholders throughout the project lifecycle.
1. **Don’t Overpromise Unrealistic Outcomes**
* Be realistic about project feasibility, system capabilities, and expected results.
* Always validate assumptions and expectations before making commitments.

**Q14. Write the difference between packages and sub-systems**

**A.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Feature** | **Package** |

| **Sub-System** |
| --- |

|  |
| --- |
|  |

 |
| Definition | A logical grouping of related classes, interfaces, or components within a system. | A self-contained module that performs a specific function within the system and can interact with other sub-systems. |
| Purpose | Organizes and manages code by grouping related elements for better modularity. | Represents a larger functional unit that can operate independently within a system. |
| Scope | Primarily used for structuring software projects and code organization. | Represents a higher-level functional division in a system that may consist of multiple packages. |
| Independence | A package is not necessarily self-contained; it may depend on other packages. | A sub-system is typically more self-contained and can function independently. |
| Visibility Control | Provides encapsulation by controlling access to elements within the package. | Controls system-wide functionality and interaction between different sub-systems. |
| Interaction | Packages do not directly interact but organize elements that interact. | Sub-systems interact with each other through well-defined interfaces. |
| UML Representation | Represented as a folder-like symbol in UML diagrams.  | Represented as a rectangular box with internal components and interfaces. |

**Q15. What is camel-casing and explain where it will be used**

**A.** Camel-casing is a naming convention used in programming and software development where multiple words are combined into a single string without spaces, and each word (except the first in some cases) begins with a capital letter. It is called "camel case" because the capital letters within the word resemble the humps of a camel.

**Types of Camel-Casing**

1. **Lower Camel Case (camelCase)**
* The first word starts with a **lowercase** letter, and subsequent words start with **uppercase** letters.
* Example: employeeName, totalAmount, getUserDetails().
* **Usage:** Used for **variables, function names, and method names** in many programming languages.
1. **Upper Camel Case (PascalCase)**
* Every word starts with an **uppercase** letter, including the first word.
* Example: EmployeeName, TotalAmount, GetUserDetails().
* **Usage:** Used for **class names, object names, and namespaces** in various programming languages.

**Advantages of Using Camel-Casing**

1. **Improves Readability** – Makes variable and function names easier to read without underscores.
2. **Standardizes Naming Conventions** – Ensures consistency across the codebase.
3. **Widely Accepted in Programming** – Used in languages like Java, JavaScript, C#, and Python.
4. **Prevents Conflicts** – Helps differentiate variables, classes, and functions based on their case style.

Camel-casing is **widely used in programming**, especially in **variables, methods, class names, JSON data, APIs, UI development, and frameworks**. It ensures a **consistent, readable, and maintainable** code structure across different technologies.

**Q16. Illustrate Development server and what are the accesses does business analyst has?**

A. **Development Server** is an environment used by developers to build, test, and debug applications before they are deployed to production. It provides a controlled setting for coding, testing new features, fixing bugs, and integrating components before moving to further stages like **Testing (QA), Staging, and Production**.

**Characteristics of a Development Server:**

* Used for **coding and initial testing**.
* **Not accessible** to end-users or customers.
* Contains **test data**, not real customer data.
* Frequent updates, changes, and debugging occur here.
* Supports **version control systems** (Git, SVN) to track changes.

**Business Analyst (BA)** does not typically write code but plays a crucial role in **validating requirements, assisting developers, and ensuring alignment with business goals**. The BA may have limited access based on project needs.

Access Rights for a BA in Development Server:

|  |  |  |  |
| --- | --- | --- | --- |
|

|  |
| --- |
| **Access Type** |

|  |
| --- |
|  |

 | **Description** |
|

|  |
| --- |
| Read-Only Access |

|  |
| --- |
|  |

 |

|  |
| --- |
| The BA can view application changes, test new features, and verify implementation. |

|  |
| --- |
|  |

 |
|

|  |
| --- |
| Test Data Access |

|  |
| --- |
|  |

 |

|  |
| --- |
| The BA can access test data to validate workflows and functionality. |

|  |
| --- |
|  |

 |
|

|  |
| --- |
| Requirement Validation |

|  |
| --- |
|  |

 |

|  |
| --- |
| The BA ensures that developed features match business requirements. |

|  |
| --- |
|  |

 |
|

|  |
| --- |
| Bug Reporting & Tracking |

|  |
| --- |
|  |

 |

|  |
| --- |
| Using tools like JIRA, Trello, or Asana, BAs track and report issues. |

|  |
| --- |
|  |

 |
|

|  |
| --- |
| User Interface (UI) Review |

|  |
| --- |
|  |

 |

|  |
| --- |
| The BA checks if UI matches design documents and usability guidelines. |

|  |
| --- |
|  |

 |
|

|  |
| --- |
| API/Integration Testing (Limited) |

|  |
| --- |
|  |

 | If APIs are involved, the BA may verify API responses using tools like Postman. |

BA utilizes the development server for Requirement Validation, Testing and Feedback, Collaboration with Developers, Change Request Management, Demo and Walkthroughs.

**Q17. What is Data Mapping**

**A. Data Mapping** is the process of **matching and transforming data fields** from one system, format, or database to another. It ensures data consistency, accuracy, and usability during data migration, integration, or transformation.

Data mapping is an essential part of many data management processes. If not properly mapped, data may become corrupted as it moves to its destination. Quality in data mapping is key in getting the most out of your data in data migrations, integrations, transformations and in populating a data warehouse.

**Q18. 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**

**A.** An **API (Application Programming Interface)** is a set of rules that allows different software applications to communicate with each other. It enables data exchange between systems, services, or applications by sending and receiving structured requests and responses.

**Scenario: Handling Date Format Differences in API Integration**

* **Application** accepts dates in dd-mm-yyyy format.
* **The external US-based application** provides dates in mm-dd-yyyy format.
* **API Integration Challenge:** Convert the incoming date format before processing it.

Solution: Process and Convert Date Format

* Extract the order\_date field.
* Convert "03-15-2024" (MM-DD-YYYY) to "15-03-2024" (DD-MM-YYYY).
* This transformation can be done using:
	+ Backend logic (e.g., Python, Java, or JavaScript).
	+ Middleware/ETL tools (e.g., Talend, MuleSoft).