**Q1: Draw use case diagram**



**Q2: What is boundary class, controller class and entity classes?**

Boundary classes (Presentation layer)

Boundary classes handle interaction between the system and external entities (user, other systems or devices). They represent user interfaces (UI), API and communication points. Used to handle interaction between the system and external actor.

Characteristics:

* Represent system entry points.
* Handle user input and output.
* Facilitate interaction with controller classes.

Examples:

* Login page (handles user authentication input)
* Web page (display data to the user)
* Customer portal (Interface for customer)

Controller classes (logic layer)

Controller classes manage the flow of data between the model and the view. They handle user input and update the model and view accordingly. These classes act as intermediaries, processing business logic and controlling interaction between boundary and entity classes. Acts as intermediary between boundary and entity classes.

Characteristics:

* Manage workflow and decision making.
* Implement business rule and validation.
* Invoke entity operation and update boundary classes.
* Do not store persistent data but coordinate data flow.

Examples:

* Order controller (manages order placement workflow)
* Payment processor (controls payment transactions)
* Booking controller(manages seat reservations in a system)

Entity classes (Data layer)

Entity classes represent core business objects and store essential data. They are typically mapped to database table. Basically represent core data and business logic of the application.

Characteristics:

* Contain attributes and methods related to business logic.
* Exist independently of boundary and controller classes.

Examples:

* Users (represent a system user with attributes like name, email etc.)
* Product (Stores product details like price, stock and description)
* Order (Holds details of customers order).

**Q3: Place these classes on three tier architecture.**

|  |
| --- |
| **User Layer/Presentation layer** |
| Payment method selection boundary |
| Card payment boundary |
| Cash payment boundary |
| Net banking payment boundary |
| Wallet payment boundary |
| **Business logic layer** |
| Payment controller |
| Card payment controller |
| Cash payment controller |
| Net banking payment controller |
| Wallet payment controller |
| **Data tier** |
| Customer- Entity class |
| Payment- Entity class |
| Card – Entity class |
| Wallet- Entity class |
| Bank account – Entity class |

**In this three-tier architecture:**

* The presentation layer handles user interaction and input/output.
* The business logic layer manages the flow of data and implements the core logic of payment processing.
* The data storage layer (or logical data layer in this case) encapsulates the data and business logic related to payment methods.

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

|  |
| --- |
| **Customer** |
| **Customer ID** | **Customer name** | **Contact No** | **Address** | **Account number** |
|   |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |
| --- |
| **Bank** |
| **Bank name** | **Location** | **Branch code** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |
| --- |
| **Payment** |
| **Payment ID** | **Amount** | **Payment date** | **Status** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |
| --- |
| **Account** |
| **Account ID** | **Account type** | **Balance** | **Acct holder name** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |
| --- |
| **Net banking service** |
| **Authentication** | **Funds transfer** | **Transaction** | **Account management** |
|  |  |  |  |
|  |  |  |  |

|  |
| --- |
| **Authentication** |
| **User name** | **Password**  | **OTP** |  |
|  |  |  |  |
|  |  |  |  |

|  |
| --- |
| **Transaction** |
| **Transaction ID** | **Recipient details** | **Amount** | **timestamp** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Q5: Draw sequence diagram for payment done by customer net banking.**



**Q6. Explain conceptual modelling for this case.**

A conceptual model represents the high level abstract representation of a system or problem domain. In this case study of customer making a payment through various methods (including net banking), a conceptual model would provide a clear understanding of the key concepts, relationships and interactions involved in the payment process.

**Key Concepts:**

* Customer: Represents an individual who intends to make a payment.
* Payment Methods: Include card, wallet, cash, and net banking, representing different ways customers can make payments.
* Payment Transaction: Represents the transfer of funds from the customer to the recipient through a chosen payment method.
* Authorization: The process of verifying the customer's identity and confirming their intention to initiate a payment.
* Processing: The actual transfer of funds from the customer's account to the recipient's account.
* Confirmation: Providing feedback to the customer regarding the successful completion of the payment transaction.

**Relationships:**

* Customer-Payment Method: A customer can choose from multiple payment methods to make a payment.
* Payment Method-Payment Transaction: Each payment method is associated with one or more payment transactions.
* Payment Transaction-Confirmation: Each payment transaction results in a confirmation message being sent to the customer.

**Interactions:**

* Initiation: The customer initiates the payment process by selecting a payment method and providing necessary details.
* Validation and Authorization: The system validates the provided information and authorizes the payment if everything is in order.
* Processing: The system processes the payment transaction, transferring funds from the customer's account to the recipient's account.
* Confirmation: Upon successful completion, the system provides a confirmation message to the customer.

**Abstractions:**

* Customer Interface: Represents the interaction between the customer and the payment system, including selecting a payment method and providing payment details.
* Payment Processing Logic: Represents the business logic involved in validating, authorizing and processing payment transactions.
* Feedback Mechanism: Represents the system's ability to provide feedback to the customer regarding the status of the payment transaction.

**Constraints and assumptions:**

* Security: Assumption that the system implements robust security measures to protect sensitive payment information.
* Availability of Funds: Constraint that the customer must have sufficient funds or credit limit to complete the payment transaction.
* Accuracy and Reliability: Expectation that the system processes payments accurately and reliably, minimizing the risk of errors or failures.

**Q7: MVC Architecture and MVC Rules to derive the classes for use case diagram and guidelines to places the 3-tier architecture.**

MVC, which stands for Model-View-Controller is a software architectural pattern commonly used in developing user interfaces, particularly in web applications. It separates an application into three interconnected components each responsible for specific aspects of the application's functionality.

**Model**: The Model represents the application's data and business logic. It is responsible for managing the data, processing requests and responding to queries from the View and Controller components. It does not depend on the View or Controller. It operates independently ensuring that changes to the user interface or application logic do not directly affect the data management.

**View:** The View represents the presentation layer of the application. It is responsible for rendering the user interface and displaying data to the user. It receives data from the Model and presents it to the user in a format that is understandable and visually appealing. Views are typically passive and do not contain business logic. They only display information and respond to user interactions forwarding user input to the Controller.

**Controller**: The Controller acts as an intermediary between the Model and View components. It handles user input, processes requests and updates the Model accordingly. It receives user input from the View, interpret it and invoke appropriate actions on the Model to perform the necessary operations and also it determines which view to display based on the application's state and data provided by the Model.

**MVC architecture rules:**

**Identify Actors**: Actors in the use case diagram represent external entities interacting with the system. Each actor may correspond to a separate class or a set of classes in the system.

**Identify Use Cases**: Use cases represent specific functionalities or tasks that the system must perform to satisfy user requirements. Each use case may involve one or more classes that collaborate to fulfil the use case.

**Apply CRC (Class-Responsibility-Collaborator) Cards**: Use CRC cards to brainstorm and define classes, their responsibilities and collaborations. Each class should encapsulate related responsibilities and collaborate with other classes as necessary.

**Identify Associations**: Associations between actors and use cases, as well as between different use cases, may suggest relationships between classes. These associations help define the structure of the classes and their interactions.

**Consider Inheritance and Composition**: Identify commonalities and relationships between classes to determine whether inheritance or composition should be used to model those relationships.

**Guidelines to place classes in 3 tier architecture:**

**Presentation layer:**

* Place boundary classes (UI components) such as forms, screens and widgets in the presentation layer.
* These classes interact directly with the user and handle input/output operations.
* Examples include classes responsible for handling user input, displaying data, and managing user interactions.

**Business logic layer (Application layer):**

* Place controller classes responsible for controlling the flow of data and application logic in the business logic layer.
* These classes mediate between the presentation layer and the data access layer, coordinating the processing of user requests and responses.
* Examples include classes implementing use case logic, orchestrating interactions between components and enforcing business rules.

**Data access layer (Persistence layer):**

* Place entity classes representing data entities, data access objects (DAOs) and other components responsible for data storage and retrieval in the data access layer.
* These classes interact with the database or external data sources to perform CRUD (Create, Read, Update, and Delete) operations on persistent data.
* Examples include classes representing database tables, entities mapped to object-relational models and data access components encapsulating database operations.

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

|  |  |  |
| --- | --- | --- |
| Stages | Activities | Artefacts and resources |
| Pre project | Enterprise Analysis – SWOT Analysis, GAP Analysis, Market Research, Feasibility Study, Root Cause Analysis, Decision Analysis, Strategy Analysis, Enterprise Architectural Frameworks, Project Scope and Business case writing, Risk analysis | Business CaseSOW (Statement of Work)PO (Purchase Order)Sr. BA, Business ArchitectsPre sales Consultants |
| Planning, Estimations & Assessment | 1.Understand Assumptions and Constraints along with Business Rules and Business Goals2. Plan Packages for Big Projects3. Understands the project plan from PM4. BA conducts stakeholders Analysis5. Plan BA approach strategy (Req. gathering techniques, communication, Req. management, Documents to follow, Tools to use, Change Request Handling methodology for this Project | PM Sr. BA |
| Requirements Gathering | 1. Stakeholders identify and document2. Client gives BRD or BA prepares BRD by interacting with Client – Brainstorming , Document Analysis, Reverse engineering, Interviews, workshops, FocusGroups, Observation, Questionnaires.3. Prototyping can be used by BA to make the Client to give more specific requirements4. Sort the gathered Requirements (avoiding duplicate Reqs , grouping into similar functionality or intomodules)5. Prioritize requirements – MoSCoW6. Validate Requirements - FURPS | BRD (Business Requirements Document)BA PM |
| Requirements Analysis | 1. Draws UML Diagrams ( Use case and Activity Diagrams)2. Prepares Functional Requirements from Business Requirements.3. All Architects comes up with Technical Requirements (SSD)4. SRS will have Functional Requirements and Technical Requirements5. Takes Signoff on SRS from Client. SRS is the first legal binding Doc between the Business and the technical Team6. BA prepared RTM from SRS before Design phase Starts. (BA is the owner of RTM).7. BA traces how requirements are dealt in each phase of development life cycle from Design till UAT | Functional Requirements SpecificationSSD(Supplementary SupportDocument) SRS (Software RequirementsSpecification)RTM (Requirements TraceabilityMatrix)BA PM Solution-Architect DB – Architect NW – Architect |
| Design | 1. From Use case Diagram , Test Manager or BA will prepare Test Cases2. Communicates with Client on the design and Solution documents (updates Status to Client and make them understand how the solution would look like to prepare them to drive UAT)3. BA will initiate the preparation of End user manuals4. updates RTM5. From Use case Diagram Solution-Architect recommends Architecture of the IT solution6. DB Architect uses Persistence Classes (Entity Classes) and comes up with ER Diagrams or DB Schema.7. GUI Designer will look into Transient Classes (Boundary Classes) and designs all possible Screens for the IT Solution | Solution DocumentDesign Document – HDD – ADDBAPMSolution-ArchitectDB – ArchitectNW – ArchitectGUI - DesignerTest Manger |
| Coding | 1.BA organizes JAD Sessions2. BA clarifies queries of Technical Team during Coding3. Developers refer Diagrams and Transient (Controller Classes) of BA and code their unit4. Update End user manuals5. Update RTM6. Conducts regular Status meetings with technical team and the Client and tuning Client for participation in UAT | LDD – CDDApplicationDevelopment TeamBAPM |
| Testing | 1. BA- Prepares Test Cases from Use Cases or assists test Manager to do so2. BA performs high level testing3. BA prepares Client for UAT4. Test Data is requested by BA from Client5. Updates End User Manuals6. Updates RTM7. Take signoff from Client on Client Project Acceptance form | Test Concerning DocumentsApplication with less errorsTesting TeamBAPMClient |
| Deployment and Implementation | 1.Forwards RTM to Client or the PM which should be attached to the Project Closure Document2. Coordinates to complete and share End User Manuals3. Plans and Organizes Training Sessions for End Users4. Prepares Lessons learned from this project (to take precautions for coming projects) |  |

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

**Conflict management:** Conflict management refers to the process of identifying, addressing, and resolving conflicts or disagreements that arise within individuals, teams, organizations, or communities. Conflicts can occur due to differences in opinions, values, goals, interests, or personalities. Effective conflict management involves understanding the underlying causes of conflicts, facilitating constructive communication and negotiation, and finding mutually acceptable solutions to resolve disagreements.

**Thomas- kilmann technique**:

The Thomas-Kilmann Conflict Mode Instrument (TKI) is a popular conflict resolution tool developed by Kenneth Thomas and Ralph Kilmann. It is based on their research on conflict management and identifies five primary conflict-handling styles or modes. These modes represent different ways individuals approach and respond to conflicts. The TKI is widely used in various organizational contexts to improve communication, collaboration, and decision-making. Let's explore each conflict mode:

**Competing (Assertive, Uncooperative):**

* Individuals who adopt a competing style are assertive in pursuing their own goals and interests often at the expense of others' concerns.
* They tend to use their authority, power, or influence to win arguments or achieve their objectives even if it leads to conflict or negative outcomes for others.
* This style is appropriate in situations where quick decisions are needed, or when the individual's position is vital and must be defended.

**Collaborating (Assertive, Cooperative):**

* Individuals who collaborate seek to find mutually beneficial solutions to conflicts by addressing the concerns and interests of all parties involved.
* They value open communication, active listening and creative problem-solving to achieve win-win outcomes.
* Collaboration is effective in complex situations where multiple perspectives are valuable and long-term relationships need to be maintained.

**Compromising (Intermediate Assertiveness and Cooperativeness):**

* Compromisers are willing to give up some of their own goals or interests in exchange for concessions from others.
* They seek middle-ground solutions that partially satisfy everyone involved, even if they may not fully address each party's needs or preferences.
* Compromise is suitable when time is limited and a quick resolution is necessary or when maintaining relationships is essential despite differences in goals.

**Avoiding (Unassertive, Uncooperative):**

* Individuals who adopt an avoiding style prefer to evade or postpone conflicts rather than directly addressing them.
* They may withdraw from the situation, ignore the problem or sidestep confrontation to maintain peace or avoid negative emotions.
* Avoidance may be appropriate when the issue is trivial, emotions are running high, or when conflicts cannot be resolved immediately without causing further harm.

**Accommodating (Unassertive, Cooperative**):

* Accommodators prioritize maintaining harmony and preserving relationships over asserting their own interests or viewpoints.
* They are willing to yield to others' demands, accommodate their preferences, or support their goals, even if it means sacrificing their own needs.
* Accommodation is useful when maintaining relationships is paramount, or when the issue at hand is more important to others than to oneself.

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

Project failure can result from various factors, which may occur individually or in combination. Here are some common reasons for project failure:

**Poor Project Planning**: Inadequate planning, inaccurate estimation of resources, timelines, and budgets can lead to project failure. Lack of clear objectives, scope creep, and insufficient risk assessment can also contribute to planning failures.

**Ineffective Communication**: Communication breakdowns between project stakeholders, team members and leadership can lead to misunderstandings, delays and conflicts. Poor communication channels, lack of transparency and ineffective feedback mechanisms hinder progress and decision-making.

**Unclear or Changing Requirements**: Unclear, ambiguous or constantly changing project requirements can result in scope creep, resource overruns and missed deadlines. Failure to gather and prioritize requirements effectively can lead to deliverables that do not meet stakeholders' expectations.

**Inadequate Resource Management**: Misallocation or insufficient allocation of resources, including human resources, budget and equipment can hinder project progress. Lack of skilled personnel, poor team coordination and over-reliance on key individuals can lead to resource constraints and delays.

**Weak Risk Management**: Failure to identify, assess and mitigate project risks can result in unforeseen issues, delays and cost overruns. Ignoring potential risks, failing to develop contingency plans and inadequate risk monitoring can exacerbate project challenges.

**Scope Creep**: Uncontrolled expansion of project scope beyond the initial requirements and objectives can strain resources, extend timelines and increase costs. Lack of scope management processes, indecisive stakeholders and inadequate change control mechanisms contribute to scope creep.

**Poor Stakeholder Engagement**: Inadequate involvement and engagement of project stakeholders, including sponsors, clients, and end-users, can lead to misalignment of expectations and project goals. Failure to address stakeholders' needs and concerns may result in project resistance or rejection.

**Technology Issues:** Technical challenges, such as software bugs, hardware failures, integration complexities and compatibility issues can disrupt project progress and undermine product quality. Inadequate technology expertise and insufficient testing exacerbate these issues.

**Lack of Leadership Support**: Inadequate leadership commitment, direction and support can demotivate project teams, hinder decision-making, and impede progress. Absence of effective project governance, unclear roles and ineffective leadership styles contribute to project failure.

**Poor Quality Management**: Insufficient quality assurance and control processes, inadequate testing and lack of attention to quality standards can result in deliverables that do not meet quality expectations. Poor quality management leads to customer dissatisfaction and rework.

**Financial Constraints**: Budget overruns, insufficient funding, or funding cuts can jeopardize project viability and sustainability. Failure to secure adequate financial resources, inaccurate budget estimation and ineffective cost control measures contribute to financial constraints.

**External Factors**: External factors beyond the project team's control such as changes in market conditions, regulatory requirements, economic downturns, natural disasters or political instability can disrupt project plans and outcomes.

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

Business Analysts (BAs) play a crucial role in project success by bridging the gap between business needs and IT solutions. However, they often encounter various challenges throughout the project lifecycle. Here are some common challenges faced by BAs:

**Unclear or Evolving Requirements**: One of the most significant challenges for BAs is dealing with ambiguous, incomplete or constantly changing requirements. This can result from inadequate stakeholder communication, conflicting priorities or evolving business needs.

**Stakeholder Management**: BAs need to engage with diverse stakeholders, including business users, project sponsors, developers and testers. Managing stakeholder expectations, resolving conflicts and ensuring alignment on project goals can be challenging, especially when stakeholders have competing interests or priorities.

**Scope Creep**: Controlling scope creep is a constant challenge for BAs. As project requirements evolve or stakeholders request additional features, it's essential to manage scope changes effectively to prevent delays, budget overruns and resource constraints.

**Technology Complexity**: BAs must understand the technical aspects of the project and communicate effectively with developers and architects. Dealing with complex systems, new technologies or integration challenges requires continuous learning and collaboration with technical experts.

**Lack of Domain Knowledge:** In projects spanning various industries or domains, BAs may lack domain expertise, making it challenging to understand business processes, terminology, and industry-specific requirements. Investing time in domain research and collaboration with subject matter experts (SMEs) is essential to overcome this challenge.

**Communication Barriers**: Effective communication is critical for BAs to gather requirements, convey information accurately and facilitate collaboration among project stakeholders. Language barriers, cultural differences or poor communication channels can hinder effective communication and lead to misunderstandings.

**Resistance to Change**: Implementing new processes, systems or solutions often faces resistance from users or stakeholders accustomed to existing practices. BAs must address concerns, communicate the benefits of change and ensure adequate training and support to facilitate smooth transitions.

**Resource Constraints**: Limited resources, including time, budget and personnel can constrain BA activities and impact project outcomes. BAs must prioritize tasks, optimize resource utilization and collaborate with project managers to overcome resource constraints effectively.

**Quality Assurance**: Ensuring the quality of requirements documentation, validating solutions against business needs and conducting thorough testing are essential responsibilities of BAs. However, inadequate testing resources, unclear acceptance criteria or lack of testing standards can pose challenges to quality assurance efforts.

**12. Write about document naming standards**

Document naming standards refer to guidelines or conventions established within an organization to create consistent and meaningful names for files, folders or documents. These standards are crucial for effective document management, organization, and retrieval, especially in environments where multiple users collaborate on creating, editing, and accessing files. Here's what they typically entail:

**Consistency**: Naming conventions ensure that files are named consistently across the organization. This consistency makes it easier for users to locate specific documents and reduces confusion.

**Relevance**: File names should accurately reflect the content or purpose of the document. This helps users understand the file's context without needing to open it.

**Clarity**: Names should be clear and easy to understand, avoiding jargon, abbreviations or ambiguous terms. This ensures that users can quickly identify the document they need.

**Brevity:** While names should be descriptive, they should also be concise. Long file names can be cumbersome and may get cut off in certain systems. Aim for a balance between clarity and brevity.

**Avoidance of Special Characters**: Special characters, such as &, %, $, #, etc., should generally be avoided in file names as they can cause compatibility issues across different operating systems or software platforms.

**Use of Dates or Version Numbers**: Including dates or version numbers in file names can help track document revisions and ensure users are accessing the most recent version. For example, "Project\_Report\_v2\_2024-03-12.docx".

**Organization-specific Tags**: Some organizations may include tags or codes specific to their departments, projects, or categories to further organize documents. For example, "HR\_Policy\_Manual.pdf" or "Marketing\_Campaign\_2024.pdf".

**Use of Hyphens or Underscores**: When separating words in a file name, organizations may choose to use hyphens (-) or underscores (\_) to improve readability. For example, "Meeting\_Minutes\_2024-03-12.docx" or "Quarterly-Report-2024-Q1.pdf".

**Version Control**: If multiple versions of a document exist, naming conventions may include a version control system (e.g., v1, v2, v3) to distinguish between them.

**File Format**: It's often useful to include the file format extension at the end of the file name to indicate the file type, such as .docx for Word documents or.pdf for PDF files.

**13. What are the Do’s and Don’ts of a Business Analyst?**

Business analysts play a critical role in organizations by bridging the gap between business objectives and technical solutions. They gather, analyse and document requirements for business processes, systems and projects. Here are some do's and don'ts for business analysts to be effective in their role:

**Do’s:**

**Understand the Business**: Gain a deep understanding of the organization's goals, processes and stakeholders to effectively identify business needs and opportunities.

**Communicate Effectively**: Communicate clearly and effectively with stakeholders, including business users, project managers, developers, and other team members, to ensure alignment and understanding of requirements.

**Ask Questions**: Proactively ask questions to clarify requirements, uncover underlying needs, and ensure comprehensive understanding of business problems and objectives.

**Document Requirements**: Thoroughly document business requirements, use cases, user stories and other artefacts to capture and communicate requirements effectively to stakeholders and development teams.

**Analyse and Prioritize**: Analyse requirements to identify dependencies, conflicts, and potential risks, and prioritize them based on business value and strategic objectives.

**Facilitate Collaboration**: Foster collaboration and consensus among stakeholders by facilitating workshops, meetings and discussions to elicit requirements and drive decision-making.

**Stay Agile**: Embrace agile methodologies and iterative approaches to adapt to changing requirements, deliver value incrementally and respond quickly to feedback and evolving business needs.

**Validate Solutions:** Work closely with stakeholders to validate proposed solutions against business requirements, ensuring that they meet the intended objectives and deliver value to the organization.

**Continuous Learning**: Stay updated on industry trends, best practices and emerging technologies to enhance skills, knowledge, and effectiveness as a business analyst.

**Don’ts:**

**Assume Requirements**: Avoid making assumptions about business requirements without verifying them with stakeholders, as this can lead to misunderstandings and misalignment.

Overlook Stakeholders: Don't overlook key stakeholders or their perspectives, as their input is essential for understanding business needs and ensuring the success of projects

**Ignore Feedback**: Don't ignore feedback from stakeholders or dismiss their concerns, as their insights can uncover valuable information and perspectives that inform better decision-making.

**Be Overly Technical**: Avoid using technical jargon or complex terminology when communicating with non-technical stakeholders, as it can lead to confusion and hinder understanding.

**Focus Solely on Documentation**: While documentation is important, don't prioritize it over meaningful collaboration and understanding with stakeholders. Focus on delivering value and meeting business needs rather than just producing artefacts.

**Rigidly Stick to Processes**: While processes are important, don't rigidly adhere to them at the expense of adaptability and responsiveness to changing business needs. Be flexible and pragmatic in your approach.

**Lose Sight of Business Goals**: Don't lose sight of the overarching business goals and objectives when analysing requirements or proposing solutions. Always align your efforts with the organization's strategic priorities.

**Assume One Size Fits All**: Avoid assuming that a one-size-fits-all approach will work for all projects or stakeholders. Tailor your approach to fit the specific needs, context, and dynamics of each situation.

**Neglect Feedback Loops**: Don't neglect feedback loops or fail to incorporate feedback into the analysis and decision-making process. Continuous feedback ensures that requirements and solutions evolve iteratively to better meet business needs.

Dos and Don’ts as BA

Never say NO to Client

There is NO word called as "BY DEFAULT"

Never imagine anything in terms of GUI

Question the existence of existence / question everything in the world

Ex: what client gives is not always correct

Consult an SME for Clarifications in Requirements every Problem of Client is unique. No two problems of different Client are same. May be the approach, technology, place of use, local laws may be varied to make them (Problems) to be different. Go to Client with a plain mind with no assumptions. Listen carefully and completely until Client is done and then you can ask your Queries. Please do not interrupt the Client, when he/ she are giving you the problem. Maximum Try to extract the leads to Solution from the Client itself. Never try to give Solutions to Client straight away with your previous experience and assumptions. Try to concentrate on the important and truly required Requirements. Don’t be washed away by add on Functionalities or don’t imagine solutions on Screen basis.

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

|  |  |  |
| --- | --- | --- |
| Nature | Package | Sub-systems |
| Scope and Granularity | Packages typically represent a collection of related classes, modules, or components grouped together for organizational purposes. They are often used to manage and encapsulate related functionality or features within a software system. | Subsystems, on the other hand, represent larger, more cohesive units of functionality that encapsulate a significant portion of the system's behaviour. Subsystems may encompass multiple packages, modules, or layers and are designed to achieve specific system-level goals or requirements. |
| Level of Abstraction | Packages are primarily a design-time concept used for organizing and managing the structure of code. They provide a namespace for grouping related elements and help maintain modularity and encapsulation within a software project. | Subsystems are a higher-level architectural concept that defines the overall structure and organization of a system. They abstract away implementation details and focus on defining the system's major components and their interactions. |
| Dependency Management | Packages often exhibit dependencies between one another, with one package relying on the functionality provided by another. Dependency management within packages is typically handled through mechanisms such as import statements, namespaces, or module systems. | Subsystems, on the other hand, are designed to minimize dependencies between different subsystems to promote modularity, maintainability, and scalability. Subsystems encapsulate related functionality and expose well-defined interfaces for communication with other subsystems. |
| Size and Complexity | Packages are typically smaller and more granular in size, containing a limited number of classes or modules related to a specific domain or feature set. They are designed to be manageable and maintainable units of code. | Subsystems are larger and more complex entities that encompass multiple packages and components. They represent significant units of functionality within a system and may include multiple layers, modules, or services. |
| Deployment and Distribution | Packages are often bundled together and deployed as part of a single software artefact, such as a library, module, or application. They can be distributed and reused across different projects or systems. | Subsystems may be deployed and distributed as separate units, allowing for modular development, deployment, and scaling of large systems. Subsystems may also be deployed as part of monolithic application or as independent micro services in a distributed architecture. |

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

Camel casing is a convention used in computer programming, specifically in naming identifiers like variables, functions, and class names. In camel casing, each word in the identifier is capitalized except for the first word, which starts with lowercase. The name is written without spaces, and each subsequent word begins with a capital letter, giving the appearance of humps on a camel's back, hence the name "camel case."

For example:

Camel Case

Number Of Students

Calculate Total Score

Get User Info

Camel casing is commonly used in various programming languages and contexts, including:

**Java**: In Java programming, camel casing is widely used for naming variables, methods, and class names. For example, ‘numberOfStudents’, ‘calculateTotalScore’, ‘getUserInfo’.

**C#**: Similar to Java, C# developers often use camel casing for naming variables, methods, and classes. For example, ‘numberOfStudents’, ‘calculateTotalScore’, ‘getUserInfo’.

**JavaScript**: In JavaScript, camel casing is a common convention for naming variables, functions, and object properties. For example, ‘numberOfStudents’, ‘calculateTotalScore’, ‘getUserInfo’.

**Python**: While camel casing is less common in Python, it is sometimes used for naming variables and functions, especially in situations where consistency with existing codebases or libraries is desired. However, the preferred convention in Python is usually snake case, where words are separated by underscores (‘\_’). For example, ‘number\_of\_students’, ‘calculate\_total\_score’,’ get\_user\_info’.

**JSON and XML**: Camel casing is often used in JSON and XML data formats for naming object keys or XML elements. For example, in JSON: ‘{"numberOfStudents": 10, "total Score": 95}’.

**URLs and Routing**: Camel casing is sometimes used in URL paths or routing patterns in web development frameworks. For example, /’getUserInfo’, /’calculateTotalScore’.

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

A development server is a specialized computer or server used by software developers to build, test, and deploy software applications or websites. It provides an environment where developers can write and debug code, test new features, and collaborate on projects before deploying them to production servers. Here's an illustration of a typical development server setup:

**Hardware Infrastructure**: The development server may be a physical machine located within the organization's premises or a virtual server hosted in the cloud. It is equipped with sufficient processing power, memory, and storage to support the development and testing activities of the development team.

**Operating System and Software**: The development server runs an operating system (e.g., Linux, Windows, macOS) along with the necessary software tools and frameworks required for software development. This includes programming languages (e.g., Java, Python, JavaScript), development environments (e.g., IntelliJ IDEA, Visual Studio Code), version control systems (e.g., Git), databases (e.g., MySQL, PostgreSQL), web servers (e.g., Apache, Nginx), and other development utilities.

**Development Environments**: Developers typically connect to the development server remotely using secure shell (SSH) or remote desktop protocol (RDP) to access their development environments. Each developer may have their own user account and workspace configured with their preferred tools and settings.

**Version Control**: The development server is often integrated with a version control system (e.g., Git, SVN) to manage source code repositories. Developers can check out code, create branches, commit changes, and merge code changes using the version control system.

**Testing Environments**: The development server may host multiple testing environments, including integration testing, user acceptance testing (UAT), and performance testing environments. These environments allow developers to test their code in isolation and validate its functionality before deploying it to production.

Overall, the development server serves as a central hub for software development activities, providing developers with the tools, resources, and environments they need to build, test, and deploy software applications efficiently.

As for the accesses a business analyst has in this environment, it typically depends on the organization's policies, the specific project, and the role of the business analyst within the development process. Generally, a business analyst may have the following accesses on a development server:

**Read-Only Access**: Business analysts may have read-only access to view project documentation, user stories, requirements documents, and other artefacts stored on the development server. This access allows them to stay informed about project progress and understand the functionality being developed.

**Access to Collaboration Tools**: Business analysts may have access to collaboration tools integrated with the development server, such as issue trackers, project management software, and communication platforms. This access enables them to participate in discussions, provide feedback, and track project milestones.

**Limited Access to Testing Environments**: Depending on the project's requirements, business analysts may be granted limited access to testing environments hosted on the development server. This access allows them to review and validate software changes during user acceptance testing (UAT) or other testing phases.

**No Direct Access to Source Code or Development Environments**: In most cases, business analysts do not require direct access to source code repositories or development environments on the development server. Their role primarily focuses on gathering requirements, defining business needs, and validating software functionality rather than writing or testing code.

**17. What is Data Mapping?**

Data mapping is the process of defining the relationships between data elements in two distinct data models or schemas. It involves identifying the source data elements, understanding their structure and format, and determining how they correspond to the target data elements.

**Source Data**: Data mapping starts with an understanding of the source data. This includes identifying the source systems, databases, files, or applications where the data originates. It's essential to understand the structure, format, and semantics of the source data elements.

**Target Data**: Next, the target data is defined. This includes understanding the destination systems, databases, files, or applications where the data will be transformed or loaded. The structure, format, and semantics of the target data elements need to be clearly defined.

**Mapping Rules**: Data mapping involves establishing rules or transformations that define how the source data elements relate to the target data elements. This may include simple mappings, where one source data element corresponds directly to one target data element, or complex mappings involving data transformations, calculations, or aggregations.

**Data Transformation**: In many cases, data mapping also involves data transformation, where the source data is manipulated or converted to meet the requirements of the target data model. This may include data type conversions, data cleansing, data validation, or data enrichment.

**Data Mapping Documentation**: It's important to document the data mapping process comprehensively. This documentation typically includes mapping specifications, data mapping diagrams, mapping rules, and any relevant metadata about the source and target data elements.

**Data Mapping Tools**: There are various tools available to facilitate the data mapping process. These tools provide graphical interfaces for visually defining data mappings, automating data transformation tasks, and generating documentation. Examples of data mapping tools include Informatica Power enter, Talend Data Mapper, and IBM Info Sphere Data Stage.

**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, or Application Programming Interface, is a set of rules, protocols, and tools that allows different software applications to communicate with each other. It defines how software components should interact and enables the exchange of data and functionality between disparate systems.

In the context of software development, an API specifies the methods and data formats that applications can use to request and exchange information. APIs can be used for various purposes, including accessing web services, retrieving data from databases, integrating with third-party applications, and automating processes.

Now, let's consider an application that needs to integrate with another system that uses a different date format. Here's how API integration could be utilized in this scenario:

**Understand the API**: The first step is to understand the API provided by the other application. This includes understanding the endpoints, authentication methods, request/response formats, and any specific requirements or limitations.

**Convert Date Formats**: Since the application receives data from the other system in a different date format (mm-dd-yyyy), it needs to convert this data to its internal format (dd-mm-yyyy) for processing. This conversion can be done programmatically within the application using date manipulation functions or libraries.

**API Integration**: Develop functionality within the application to interact with the other system's API. This involves making HTTP requests to the appropriate endpoints, including any required parameters or headers, and handling the responses returned by the API.

**Data Transformation**: When retrieving data from the other system via the API, the application needs to parse the response and extract relevant information. If the data includes dates, the application should parse them according to the other system's date format (mm-dd-yyyy) and then convert them to the internal date format (dd-mm-yyyy) for further processing.

**Error Handling**: Implement error handling mechanisms to deal with issues such as network errors, API rate limiting, or invalid responses from the other system. This ensures robustness and reliability in the API integration process.

**Testing and Validation**: Thoroughly test the API integration to ensure that data is being exchanged correctly between the two systems and that date conversions are performed accurately. Validate the integration against different scenarios and edge cases to identify and address any potential issues.

**Monitoring and Maintenance**: Once the API integration is in place, monitor its performance and reliability over time. Be prepared to make adjustments or updates as needed, such as handling changes to the other system's API or addressing any issues that arise during operation.