System Analysis and Design

1. What is the purpose of systems analysis? Why is it important?
    1. The purpose of systems analysis is to study and understand a system in detail. It helps identify problems in the current system, understand how the system works, and find out what users really need. Systems analysis clearly defines what the new or improved system should do before moving to the design and development stage.
    2. Systems analysis is important because it:
        - Helps **find problems early** in the system
        - Saves **time and money** by avoiding mistakes
        - Makes sure the system **meets user needs**
        - Improves **communication between users and developers**
        - Helps create a **clear plan** for system design.

2. List the six core processes for software systems development.
    - The six core processes for software systems development are:
        1. **Planning**: Identify the problem, goals, scope, schedule, and resources of the system.
        2. **Systems Analysis**: Study the current system and gather user requirements.

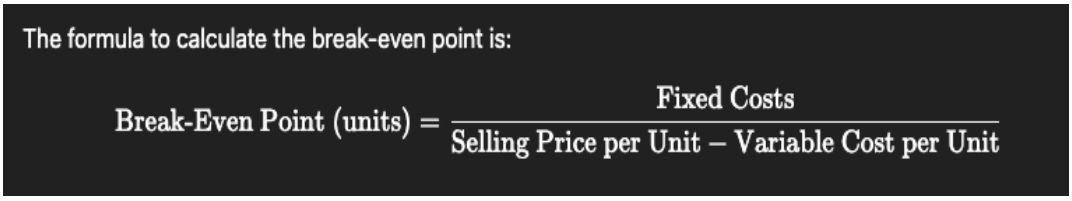
3. **Systems Design :** Design how the system will work, including data, processes, and user interface.
4. **Development (Implementation):** Write the code and build the system based on the design.
5. **Testing:** Check the system for errors and make sure it works correctly.
6. **Deployment and Maintenance**: Install the system, support users, fix bugs, and improve the system over time.

3. Which phase in the SDLC is the most important? Justify your answer.

1. The **most important phase in the SDLC** is **Systems Analysis (Requirement Analysis)**.
2. Justification
    a. **Defines what the system should do:** If requirements are not clear, the system may be built incorrectly.
    b. **Prevents mistakes:** Finding problems early saves time and money.
    c. **Guides design and development:** Designers and developers rely on clear requirements.
    d. **Ensures user satisfaction:** The system will meet the actual needs of users.

4. What is the break-even point for a project? How is it calculated?

1. Break-Even Point for a Project: The **break-even point (BEP)** is the point at which a project's **total costs equal its total revenues**, meaning the project **neither makes a profit nor a loss**. It helps businesses know **how much they need to sell or produce to cover costs**.
2. How It Is Calculated
    - The formula to calculate the break-even point is:



    - Where:
        - **Fixed Costs** = Costs that do not change (e.g., salaries, rent)
        - **Variable Cost per Unit** = Cost to produce one unit (e.g., materials, labor)
        - **Selling Price per Unit** = Price at which the product/service is sold

5. Describe how projects are selected in organizations.

Organizations select projects based on **their goals, resources, and potential benefits**. The selection process usually involves the following steps:

1. **Identifying Needs or Problems**: Organizations look for problems to solve or opportunities to improve business processes.
2. **Evaluating Feasibility:** Check if the project is technically, economically, and operationally feasible.
3. **Estimating Costs and Benefits:** Analyze the financial impact, including costs, profits, and break-even points.
4. **Prioritizing Projects**: Projects are ranked based on **importance, urgency, risk, and strategic alignment** with organizational goals.
5. **Selecting the Project:** Decision-makers choose projects that provide the most value while fitting within resources and time limits.

6. What are Computer-Aided Software Engineering (CASE) tools? Explain their advantages in system analysis and design.

1. **CASE**(Computer-Aided Software Engineering) tools are software applications that help system analysts and developers design, develop, and maintain software systems more efficiently. They

provide support for tasks like diagramming, documentation, code generation, testing, and project management.

2. Advantages of CASE Tools in System Analysis and Design:
    - **Improves Productivity**: Automates repetitive tasks, such as drawing diagrams or generating code.
    - **Reduce Mistakes**: CASE tools help make diagrams, designs, and code more accurate, which reduces errors in the system.
    - **Better Documentation**: keep all system documents, diagrams, and reports organized, easy to update, and easy to understand.
    - **Facilitates Collaboration**: Multiple people can work together on the same system easily using the same CASE tools.
    - **Speeds Up Development:** From analysis to design to coding, CASE tools make the process faster and smoother.
    - **Improves System Quality**: By standardizing designs and following rules, CASE tools help build systems that work correctly and meet user needs.

7. What is meant by **Agile development** and **Iterative development**?
    1. **Agile Development**: is a way of building software in **small steps** called *iterations*.

- The team **develops a part of the system**, shows it to users, gets feedback, and then improves it.
- This process **repeats** until the full system is ready.
- Agile focuses on **flexibility, teamwork, and responding to changes** quickly.

2. **Iterative Development**: is similar to Agile but focuses more on **repeating the development cycle** to improve the system.
    - The system is **built in versions**, tested, and then improved in the next version.
    - Each version is **better than the previous one** until the final system is complete.

Agile and iterative development are ways to build software in **small parts**. Agile focuses on **flexibility and user feedback**, while iterative focuses on **repeating development to improve the system**.

8. List the different elicitation techniques used in requirements gathering.

Elicitation Techniques in Requirements Gathering:

Requirements elicitation is the process of **collecting information from users and stakeholders** about what a system should do. Some common techniques are:

1. **Interviews** – Talking directly to users or stakeholders to ask questions.
2. **Questionnaires / Surveys** – Using written questions to collect information from many people.
3. **Observation** – Watching how users work to understand current processes and problems.
4. **Document Analysis** – Reviewing existing documents, forms, and reports to understand requirements.
5. **Workshops / Joint Application Development (JAD)** – Group meetings where users and developers discuss and define requirements.
6. **Prototyping** – Building a simple model of the system to show users and get feedback.
7. **Brainstorming** – Generating ideas in a group to find possible system features and solutions.

Requirements elicitation techniques help **understand user needs and system problems**. Common techniques include **interviews, questionnaires, observation, document analysis, workshops, prototyping, and brainstorming**.

9. "Interviews should always be conducted as structured interviews." Do you agree? Justify your answer.
    1. Do you agree: No, I do **not completely agree**.

2. **Justification:**
    - **Structured interviews** use prepared questions and are good for **collecting consistent information** from many users.
    - But sometimes, **unstructured interviews** are better because they allow users to **share ideas freely** and give more **detailed explanations**.
    - **Semi-structured interviews** are also common, combining both methods for flexibility and consistency.
    - The choice depends on the **type of information needed** and the situation.

Structured interviews are useful for consistency, but unstructured or semi-structured interviews are sometimes better to get **detailed and flexible information** from users.

10. What are the different types of **feasibility** analysis, and when are they used?
    1. Feasibility analysis is done to **determine whether a project is possible, practical, and worth doing** before spending time and money on it. It helps organizations **reduce risks** and make sure the system will succeed. There are **three main types of feasibility analysis**:
        a. Technical Feasibility

b. Economic Feasibility (Cost-Benefit Analysis)

c. Operational Feasibility

2. What they used for

1) Technical Feasibility

- This checks whether the technology, hardware, software, and technical skills needed for the system are available.
- It also looks at whether the system can be developed with current tools and techniques.
- Technical feasibility is important because if the organization cannot provide the required technology, the project may fail even if it is well planned.
- Example: Before building a bus booking system, the organization checks if they have servers, programming tools, and skilled developers to make the system.

2) Economic Feasibility (Cost-Benefit Analysis)

- This evaluates whether the project is financially worthwhile.
- It compares the total costs of developing and maintaining the system with the expected benefits.
- A project is economically feasible if the benefits are greater than or equal to the costs.

- **Example**: If a school wants a new attendance system, they calculate the cost of software, hardware, and staff training and compare it with the benefit of saving time and reducing errors in attendance records.

3) Operational Feasibility

- This checks if the system will work properly in the organization and if users can operate it effectively.
- It ensures the system solves the intended problems and fits into the current business processes.
- Operational feasibility also considers user acceptance, because a system that users do not like or understand will fail.
- **Example**: Even if a new online shopping system is technically and economically feasible, it must be easy for customers and staff to use, or it will not be successful.

11. What is a process model? What is a data flow diagram (DFD)? How are they related?

❖ **<u>What is a Process Model?</u>**

A **process model** is a diagram or representation that shows how a system works. It focuses on the processes or **activities** in a

system and how they **transform input into output**. Process models help analysts and developers **understand, analyze, and improve a system** before building it.

Example: In a school management system, a process model may show how **student information is collected, stored, and used to generate reports**.

❖ **<u>What is a Data Flow Diagram (DFD)?</u>**

A Data Flow Diagram (DFD) is a type of process model that visually shows how data moves through a system. It illustrates:

- **Processes:** Tasks that transform data (e.g., "Calculate Student Grades")
- **Data Stores:** Where data is stored (e.g., "Student Database")
- **Data Flows:** How data moves between processes, stores, and external entities
- **External Entities:** People or systems that interact with the system (e.g., "Teacher" or "Parent")

Example: A DFD for a bus booking system shows how user booking requests go from the website to the database and back with confirmation.

❖ ***<u>How They Are Related</u>***

- A **process model** is a **general concept** that shows system activities.
- A **DFD is a type of process model** that **focuses on data movement**.
- Both help **understand the system clearly, improve design, and communicate with users**.

12. Give an example of a closed-ended question, an open-ended question, and a probing question. When would each be used?

1. Closed-Ended Question
    - **Definition:** A question that can be answered with a **"yes/no" or a short, specific answer**.
    - **Example:** "Do you use the current attendance system every day?"
    - **When Used:** To get quick, clear, and specific information.
2. Open-Ended Question
    - **Definition:** A question that **requires a detailed answer** and allows the person to explain.
    - **Example:** "What problems do you face when using the current attendance system?"
    - **When Used:** To **understand opinions, ideas, or experiences** in more detail.

3. Probing Question
    - **Definition:** A question used to **get more information or clarify an answer**.
    - **Example:** "Can you explain why it takes so long to enter attendance?"
    - **When Used:** To **dig deeper** when the initial answer is **not enough or unclear**.

Conclusion (Exam-Friendly) (*no need*):

- **Closed-ended questions** → get short, specific answers
- **Open-ended questions** → get detailed information
- **Probing questions** → get more explanation or clarify answers

13. List the characteristics of a good user interface.

A **good user interface (UI)** makes it easy and pleasant for users to interact with a system. The main characteristics are:

1. **Simple and Clear:** Easy to understand and use without confusion.
2. **Consistent:** Similar actions and layouts behave the same way throughout the system.
3. **Responsive:** Works quickly and reacts well to user actions.
4. **User-Friendly / Intuitive:** Users can figure out how to use the system without much training.

5. **Informative Feedback:** The system gives clear messages about actions, errors, or progress.

6. **Flexible:** Allows users to choose options and adapt the system to their needs.

7. **Visually Appealing:** Clean and organized layout with readable text and proper colors.

8. **Error Prevention and Recovery:** Helps users avoid mistakes and recover easily if an error happens.

14. Write a short note on structure charts. *(Answer not sure)*

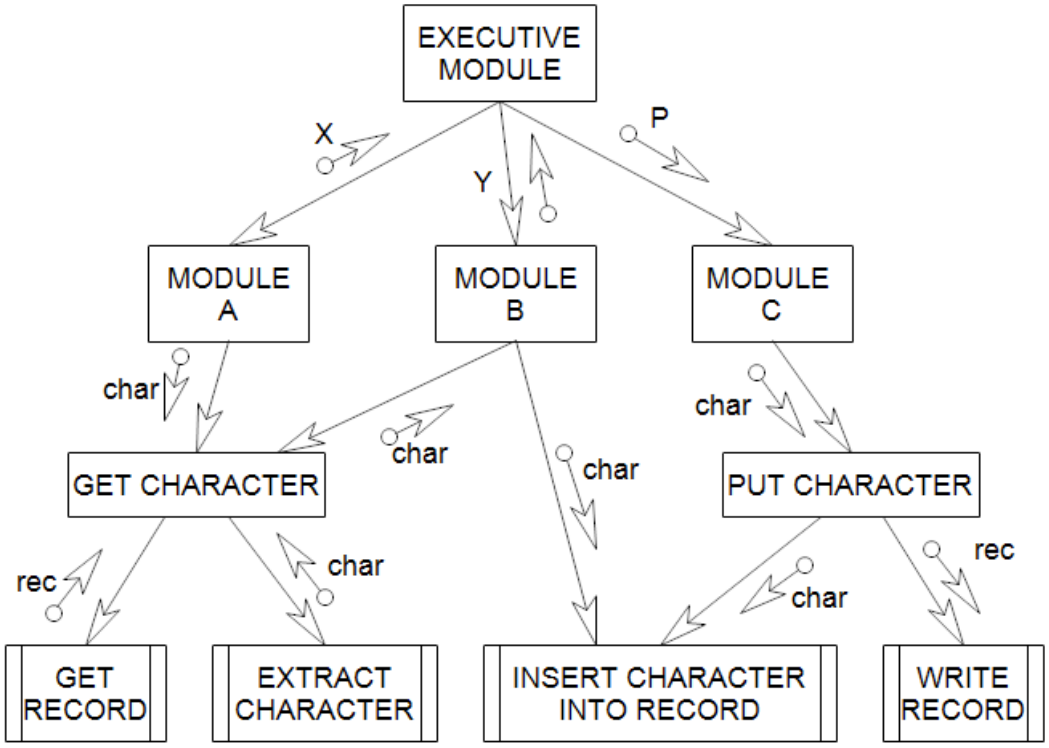
A structure chart is a hierarchical diagram in software design that visually maps a system's modules, showing functions, relationships, and data flow from a top-down perspective, using boxes for modules and arrows for data/control, breaking large systems into manageable, testable sub-problems like a tree structure. They use specific symbols for data flow, control (selection/iteration), and module interfaces, distinct from flowcharts by focusing on modular structure rather than sequence, aiding in planning code and visualizing complexity.

Key Components and Symbols

Structure charts use specific notation to show the flow of control and data between modules:

- **Modules**: Represented by rectangular boxes, these are the basic units of code (functions or procedures).
- **Module Invocation**: Indicated by vertical lines or arrows showing which module "calls" or controls another.
- **Data Couples:** Small arrows with an empty circle at the end, representing the movement of data between modules.
- **Control Couples (Flags)**: Arrows with a filled circle at the end, indicating control signals (e.g., an "End of File" message) sent between modules.
- **Condition (Selection)**: A diamond shape at the base of a module indicating that it will call only one of several sub-modules based on a specific condition.
- **Iteration (Loops)**: A curved arrow around a control line, showing that one or more modules are executed repeatedly.

Structure Chart



---

15. When would you use ***electronic reports*** rather than ***paper reports***, and vice versa?

- **Electronic Reports:** are reports viewed on a computer, tablet, or phone.
  - ➢ Used when:
    - Information needs to be **updated often**
    - Reports must be **shared quickly** with many people
    - Users need to **search, filter, or analyze data**
    - The organization wants to **save paper and cost**

- Reports are used for **daily operations**

Example: Online sales reports or attendance reports viewed on a dashboard.

- **Paper Reports :** Paper reports are printed reports.
  - Used when:
    - Reports are needed for **meetings, signatures, or legal records**
    - Users prefer **reading printed documents**
    - Computers or internet access are **not always available**
    - Reports are needed for **official documentation**

Example: Printed financial statements or signed approval forms.

16. Explain the transition from requirements to design in system development.

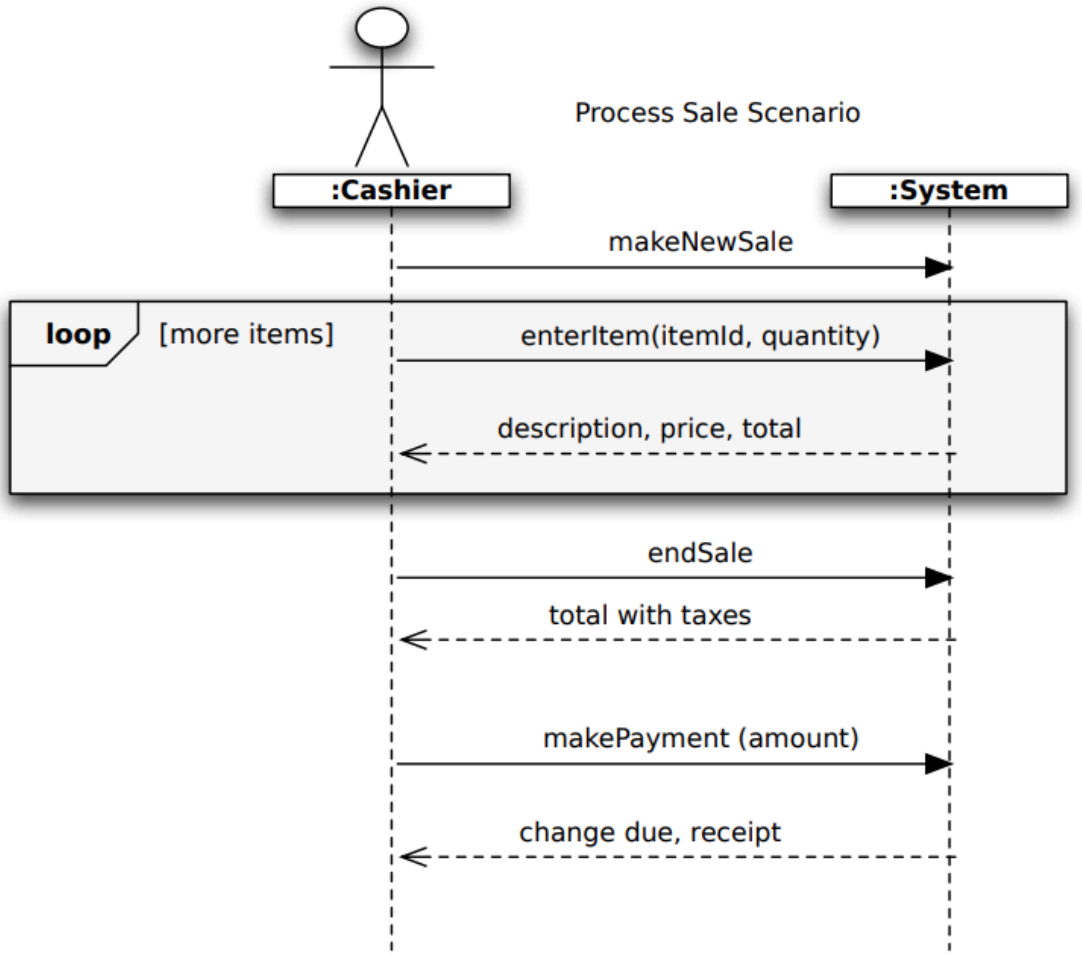
17. What is the purpose of a sequence diagram? (*Answer not sure*)

Purpose of a Sequence Diagram is used to show how different parts of a system communicate with each other over time. It shows the order of messages sent between users, objects, or system components to complete a task. It's important because:

- Show the **step-by-step flow** of actions
- Help understand **how objects interact**
- Make system behavior **easy to understand**

- Help developers **plan and write correct code**
- Reduce misunderstanding during design

Simple Sequence Diagram (Process Sale Scenario):



18. Define software testing. (*Answer not sure*)

❖ **Definition of Software Testing**

Software testing in system analysis and design is the process of verifying and validating a software application to

ensure it functions correctly, securely, and efficiently according to the specified requirements. It is a critical, ongoing activity within the Software Development Lifecycle (SDLC) that aims to identify defects, improve product quality, and ensure the final system meets user needs and business objective

❖ **The 4 Fundamental Levels of Testing**

Systems are tested hierarchically, moving from small components to the complete integrated system.

1. **Unit Testing**: The first level where developers test individual units (functions, modules, or classes) in isolation to ensure they function correctly at the smallest level.
2. **Integration Testing:** Focuses on verifying the interaction and data flow between integrated modules to identify communication or compatibility issues.
3. **System Testing:** Tests the entire, fully integrated application as a whole to evaluate it against the functional and non-functional requirements (e.g., performance, security).
4. **Acceptance Testing:** The final phase where stakeholders or end-users verify if the system meets their business needs and expectations before it goes live (often categorized as Alpha or Beta testing).

❖ **Core Testing Methodologies**

- **Black-Box Testing:** Testing purely based on functional requirements without knowledge of internal code. Testers interact with the system as an end-user.
- **White-Box Testing:** Testing with full access to the internal logic and code structure, often performed by developers to ensure code paths are correct.
- **Gray-Box Testing:** A combination of both, where the tester has partial knowledge of internal structures while testing functional behavior.

❖ **Key Quality Characteristics Tested**

Beyond functionality, designers and analysts use testing to ensure the system's long-term viability:

- **Performance**: Assessing speed, responsiveness, and stability under different workloads.
- **Security**: Identifying vulnerabilities to protect sensitive data from unauthorized access.
- **Usability**: Evaluating how user-friendly and intuitive the interface is for the target audience.
- **Regression Testing**: Re-testing after updates or bug fixes to ensure new changes haven't negatively impacted existing functionality.

❖ **Role in Systems Analysis & Design**

In **SAD**, testing is not just about finding "bugs"; it is the mechanism for Verification (confirming the product is built correctly according to design specs) and Validation (confirming the right product is being built to meet user needs). A successful test plan is established early in the design phase to define the scope, resources, and criteria for system success.

19. Differentiate pilot conversion, phased conversion, and simultaneous conversion.*(Answer not sure)*

Answer:

1. *Pilot Conversion (Location-Based):*
   The new system is fully implemented in one part of the organization (a "pilot site") before being rolled out to the rest.

   - **Process**: One branch or department acts as a test site. Problems are found and fixed before the system is used everywhere.
   - **Risk:** Low risk, because problems affect only one area.
   - **Suitability:** Best for complex systems that cannot be safely implemented in one step

2. *Phased Conversion (Module or Site-Based)*

   In phased conversion, the new system is introduced step by step.

   - **Process**
     - **Module-based:** One function at a time (e.g., billing first, then inventory).

    - **Location-based:** One site at a time until all sites are converted.
- **Risk:** Low to medium risk, but it takes more time and effort.
- **Suitability:** Best for large or complex systems that cannot be changed all at once.

3. *Simultaneous Conversion (Time-Based / Direct Conversion)*

   In simultaneous conversion, the old system is stopped and the new system starts at the same time for everyone.
   - **Process**: All users switch to the new system on a fixed date.
   - **Risk**: High risk, because any problem affects the whole organization.
   - **Suitability**: Used for small or simple systems, or when the old system must be replaced immediately.

20. Explain unit testing, integration testing, system testing, and acceptance testing.

1. **Unit testing** is a type of testing where we check one small part of the system , method , or module to make sure it work correctly by itself. It is usually done by the developer before combining that part with other part of the system
2. **Integration Testing** is a type of testing where we check how different parts of the system work together after unit testing is done. It focuses on whether the connection between modules is correct, such as whether data is passed correctly between them or whether one component can successfully call another component
3. **System Testing** is a type of testing where we test the entire system as a complete product to make sure it meets the system requirements. In

this testing, we check the system like a real user would use it and confirm that all features work together properly in a full working environment.

4. **Acceptance Testing** is a type of testing where the user or client tests the system to confirm it matches their needs and the original requirements. If the user is satisfied during acceptance testing, they approve the system, and it can be released or put into real uses.

21. What types of documentation are produced during system implementation?

During system implementation, several types of documentation are produced to help developers build the system, help users operate it, and help the organization maintain it later.

1. **Technical Documentation** is produced to explain how the system is built and how it works internally. It usually includes the system architecture, database design , program code notes, configuration details and how different modules connect to each other. This document is mainly used by developers and future maintenance teams.
2. **User Documentation** is produced to guide end users on how to use the system correctly. It usually includes user manuals , step-by-step instructions, screenshots, FAQs, and basic troubleshooting so users can complete tasks without needing a developer.

3. **Operation and Admin Documentation** is produced to help IT staff run and manage the system after it is installed. It includes installation steps, server setup, backup and recovery procedures, security settings, system monitoring and instructions for handling common system problems.
4. **Testing Documentation** is produced to record how the system was tested and what the results were. It usually includes test plans , test cases , test reports , bug lists and confirmation that fixes were completed, which proves the system was checked before going live.

22. Explain the bug life cycle and its importance in system implementation.

The bug life cycle is the step-by-step process that a bug follows from the time it is first found until it is finally fixed and closed. IT helps the team track bugs clearly so nothing is missed and everyone knows the current status of each problem.

Importance of the bug life cycle in system implementation :

1. Make sure every bug is recorded and not forgotten.
2. Helps the team track the current status of each bug clearly (new, assigned , fixed , rest , closed)
3. It improves communication between tester, developer and manager.

4. It helps prioritize work by separating high priority bugs from minor ones.
5. It reduce duplicate bugs because the team can check if the issue was already reported
6. It lowers the risk of releasing a system with serious errors, so the system becomes more stable and reliable
7. It creates a history issue , which helps for maintenance and future improvements.

23. List the main characteristics of object-oriented systems.

1. **Object and Classes** : the system is built using objects (real things in the system) and classes (the blueprint that defines what an object has and what it can do)
2. **Encapsulation** : Data and the functions that use that data kept together inside an object and other parts of the system access it through controlled methods.
3. **Abstraction** : The system shows only the important details to the user and hides the complex internal details to keep things simpler.
4. **Inheritance** : A new class can reuse and extend an existing class, which reduces repeated code and saves time.
5. **Polymorphism** : The same action or method name can behave differently depending on the object , which makes the system more flexible.

24. What is the purpose of a use case diagram?

The purpose of a use case diagram is to show what the system should do from the user's point of view. It explains the main services or functions the system provides and who will use them . It is used to :

1. identify the **actors** (users or other systems) that interact with the system
2. list the **use cases** (tasks/functions) the system must support
3. show the **relationship** between actors and those tasks
4. help stakeholders and developers **understand the system requirements clearly** before designing or coding.

25. What is the purpose of a class diagram? Explain with an example.

The purpose of a class diagram is to show the structure of an object-oriented system by describing the classes in the system, what data they store (attributes) , what actions they can do (methods) and how classes are related to each other. It is used during analysis and design to help developers understand what to build and how the system parts connect.

Example : In an online shopping system, we might have classes like **Customer**, **Order**, and **Product**. A **Customer** can place many **Orders**, and each **Order** can contain many **Products**. In the class

diagram, the *Customer* class may have attributes like customerId and name, and methods like login() or placeOrder(). The *Order* class may have attributes like orderId and orderDate, and methods like addProduct() and calculateTotal(). The relationships in the diagram show that a customer is linked to orders, and orders are linked to products, which helps the team clearly see the system structure before coding.

26. Explain the behavioral state machine diagram with an example scenario.

A behavioral state machine diagram is used to show how one object changes its state over time. It focuses on the different states the object can be in (like Idle, Processing , Completed) and the events/conditions that cause it to move from one state to another (called transitions). It is useful when the object's behavior depends a lot on its current state.

Example Scenario : ATM Card / Bank Transaction: Imagine the ATM system for one session. The ATM session can move through these states :

1. The session starts in the **Idle** state (waiting for a card).
2. When the user **inserts a card**, it moves to **Card Inserted**.
3. The ATM asks for PIN, so it moves to **PIN Entry**.
4. If the user enters the **correct PIN**, it moves to **Authenticated**. If the PIN is wrong 3 times, it moves to **Card Blocked** and ends.

5. After authentication, the user chooses a transaction, so it moves to **Transaction Selected**.
6. When the user confirms, it moves to **Processing**.
7. If the transaction is successful, it moves to **Completed**, then to **Eject Card**, and finally returns to **Idle**.
8. If there is not enough balance or a system error, it moves to **Failed**, then still **Eject Card**, and returns to **Idle**.

27. Compare and contrast black-box testing and white-box testing.

1. **BlackBox Testing**
    1. **Focus**: on functionality (the outside view)
    2. **Knowledge requirements**: testers don't need to know code. In this case system testing is also a type of black box testing since it's only testing to see if all the functions of the software works correctly while acceptance testing doesn't need to test all the functions. It's just needed to test if it meets the requirements of the end user from a group of beta users or stakeholders.
    3. **Approach**: like in the knowledge, black box testing is usually done from a user perspective.
2. **White Box Testing**
    1. **Focus** : on the internal structure (inside view , code or component of hardware)
    2. **Knowledge requirements** : testers of white box need to know code in order to understand the internal structure and

to ensure that the code is working as expected. Unit Testing is of white box testing as it is done while the developer is coding to verify if their logic is correct. There's also integration testing which is a type of testing that tests the integration or the interaction between components or modules code to ensure that they communicate and work together properly.

3. **Approach** : this testing is usually done from the developer side or code perspective.