Question 1: Why fixing bugs related to system design and analysis takes more time as compared to bugs in other artifacts of software development life cycle. Briefly explain it with example.

Verified from Book

Fixing design and analysis bugs is often more time-consuming due to their foundational nature. Errors at this stage can **ripple through the entire system**, requiring **extensive rework** across multiple components. This is because they affect the **core structure, logic, and flow** of the software, making them **complex to diagnose and resolve**.

Key Reasons:

- **Foundational Impact:** Design and analysis errors can **invalidate assumptions** and **cause cascading issues** throughout the system.
- **Widespread Effects:** Fixing these bugs often involves **revisiting and redesigning** various **interconnected parts** of the system.
- **Requirement Rework:** Design errors may necessitate **redefining requirements**, leading to **delays** and **increased coordination**.

Example:

Suppose a company is building an e-commerce platform, and during the design phase, the team fails to account for scalability to handle high traffic. When the system is deployed and traffic surges, it starts crashing due to the design flaw. To fix this, the team has to revisit the system's architecture, introduce load balancing, and optimize database interactions—requiring a significant amount of redesign and redeployment across multiple components, making it a time-consuming process.

In comparison, bugs in the later stages (e.g., coding or UI issues) might be isolated to specific features or code blocks and can often be fixed quickly without affecting the overall system structure.

Question 2: How do vision documents help system analysis in the system design process? Explain with an example:

Verified from Book

How Vision Documents Help:

- **Clarify Business Needs:** Vision documents provide a **focused understanding** of the **problem** the system is designed to solve, ensuring alignment between **business goals** and **technical solutions**.
- **Capture Stakeholder Expectations:** They **translate** stakeholder desires into **clear requirements**, **preventing miscommunication** and ensuring the system meets **user needs**.
- **Define Scope and Boundaries:** Vision documents **establish** the **project's scope**, **limiting features** to **avoid unnecessary complexity** and **ensure timely delivery**.
- **Facilitate Team Collaboration:** They serve as a **shared reference point** for **distributed teams**, **promoting understanding** and **alignment** among team members.

For example, in a healthcare information system, a vision document might prioritize patient safety and efficiency, guiding analysts to focus on features like medication management and real-time data access. This ensures that the system meets its intended objectives and avoids misunderstandings.

In smaller system systems, structured approach offers better reusability as compared to OOP approach , do you agree? Explain with an example

NOT Verified from Book

I disagree that structured approaches offer **better reusability** than Object-Oriented Programming (OOP) in smaller systems. In fact, OOP often provides **superior reusability** due to its ability to **model real-world entities** and **promote modular design**.

Explanation:

1. **Structured Approach Limitations:**
    - While structured programming can achieve some reusability through functions, it often faces limitations due to **tight coupling** between data and logic.
    - **Modifications** to data structures can **require significant changes** to functions, **reducing reusability**.
2. **OOP Advantages:**
    - **Encapsulation** in OOP allows **data and behavior** to be packaged together, promoting **reusability** and **maintainability**.
    - **Inheritance** enables **code reuse** by creating **subclasses** that inherit properties and methods from **parent classes**.
    - **Polymorphism** allows objects of different classes to be treated as the same type, providing **flexibility** and **reusability**.

Example:

Consider a **small system** managing **employees**.

Structured Approach:

- **Functions** like calculateSalary() and updateEmployeeData() might manipulate **global data structures**.
- **Changes** to employee types (e.g., part-time) could **require significant modifications** to existing functions.

OOP Approach:

- **Classes** like Employee, FullTimeEmployee, and PartTimeEmployee encapsulate **data** and **behavior**.
- **Inheritance** allows **reusing** common methods while **overriding** specific ones for different employee types.

Conclusion:

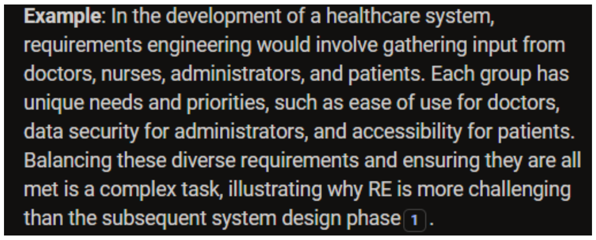
OOP generally offers **better reusability** due to its **modular design** and **object-oriented principles**. Even in smaller systems, these features can **simplify maintenance** and **enable easier additions** of new functionalities.

Why requirements engineering process is more challenging as compared to system design? Briefly explain it with an example

Verified from Book

Requirements engineering is often more arduous than system design due to its inherent complexities.

- **Ambiguity and Complexity:** Requirements can be vague or contradictory, making them difficult to ascertain.
- **Prioritization:** Determining essential requirements amidst competing needs and resource constraints is a complex task.
- **Scope Creep:** Evolving requirements can complicate the engineering process and require constant reevaluation.
- **Stakeholder Alignment:** Managing diverse perspectives and aligning interests can be challenging.


Example: In the development of a healthcare system, requirements engineering would involve gathering input from doctors, nurses, administrators, and patients. Each group has unique needs and priorities, such as ease of use for doctors, data security for administrators, and accessibility for patients. Balancing these diverse requirements and ensuring they are all met is a complex task, illustrating why RE is more challenging than the subsequent system design phase [1].

Why do each requirement must be verifiable or measurable? Briefly explain with an example

Verified from Book

Verifiability and Measurability in Requirements Engineering

Verifiable and measurable requirements are essential for ensuring the successful development and implementation of a system. They provide a **clear and objective foundation** for:

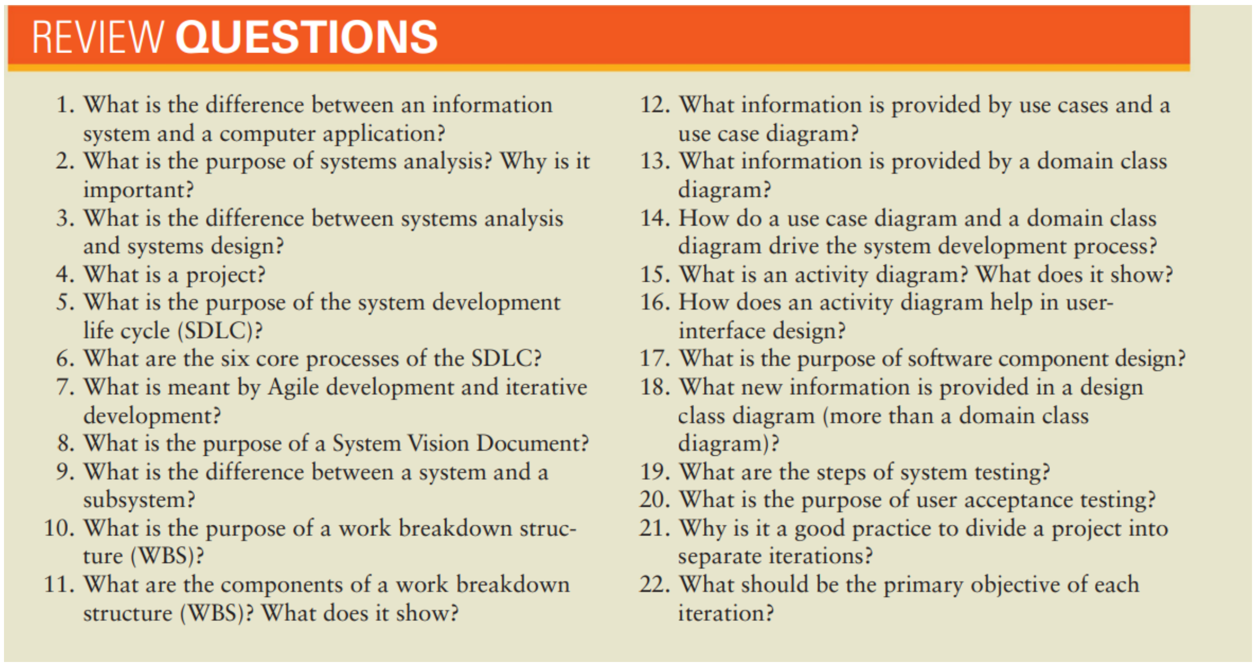
- **Effective Communication:** Verifiable and measurable requirements **facilitate clear communication** among stakeholders, developers, and testers. Everyone involved **understands exactly what is expected** and can **work towards a common goal**.
- **Accurate Assessment:** By **defining requirements in measurable terms**, it becomes **easier to assess progress**, **identify deviations**, and **make informed decisions** throughout the development process.
- **Successful Testing:** Verifiable requirements provide a **solid basis** for **creating effective test cases**. Testers can **verify** if the system **meets the specified criteria**, **ensuring its quality and functionality**.

Example:

Consider a requirement for a mobile app that states, "The app should be user-friendly." While this may seem like a reasonable requirement, it is **vague and not measurable**. A more verifiable and measurable version could be: "The app should have an **average user satisfaction rating of 4.5 out of 5 based on a survey of 100 users**." This revised requirement provides a **clear, quantifiable metric** that can be used to **assess the app's usability**.

Example:

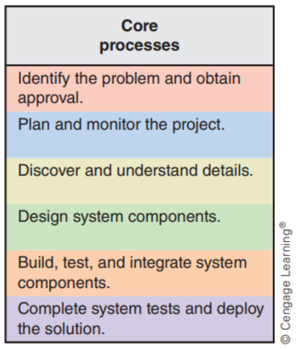
Instead of the vague requirement "The website should load quickly," a more verifiable and measurable requirement would be: "The website should have an **average page load time of 2 seconds or less** on a standard internet connection." This provides a **specific, measurable metric** that can be **tested and verified**.



REVIEW **QUESTIONS**

1. What is the difference between an information system and a computer application?
2. What is the purpose of systems analysis? Why is it important?
3. What is the difference between systems analysis and systems design?
4. What is a project?
5. What is the purpose of the system development life cycle (SDLC)?
6. What are the six core processes of the SDLC?
7. What is meant by Agile development and iterative development?
8. What is the purpose of a System Vision Document?
9. What is the difference between a system and a subsystem?
10. What is the purpose of a work breakdown structure (WBS)?
11. What are the components of a work breakdown structure (WBS)? What does it show?
12. What information is provided by use cases and a use case diagram?
13. What information is provided by a domain class diagram?
14. How do a use case diagram and a domain class diagram drive the system development process?
15. What is an activity diagram? What does it show?
16. How does an activity diagram help in user-interface design?
17. What is the purpose of software component design?
18. What new information is provided in a design class diagram (more than a domain class diagram)?
19. What are the steps of system testing?
20. What is the purpose of user acceptance testing?
21. Why is it a good practice to divide a project into separate iterations?
22. What should be the primary objective of each iteration?

1. **Difference between an information system and a computer application**:
   - An **information system** encompasses all the components (hardware, software, data, people, and processes) that work together to collect, process, store, and provide as output the information needed to complete business tasks
   - A **computer application** is a software program that performs specific tasks for users.
2. **Purpose of systems analysis**:
   - **Purpose**: To understand and specify what the system should do.
   - **Importance**: It ensures that the system meets user requirements and solves the intended problems.
3. **Difference between systems analysis and systems design**:
   - **Systems analysis** focuses on understanding and specifying **what the system should do**.
   - **Systems design** those system development activities that enable a person to describe in detail **how** the resulting information **system will actually be implemented**
4. **What is a project?**:
   - A project is a planned undertaking with a beginning and an end that produces a desired result or product.
5. **Purpose of the system development life cycle (SDLC)**:
   - To provide a structured approach for developing information systems, ensuring that all aspects of the system are properly addressed.
6. **Six core processes of the SDLC**:



Core processes
Identify the problem and obtain approval.
Plan and monitor the project.
Discover and understand details.
Design system components.
Build, test, and integrate system components.
Complete system tests and deploy the solution.

1. Identify the problem or need and obtain approval.
2. Plan and monitor the project.
3. Discover and understand the details.
4. Design the system components.
5. Build, test, and integrate system components.
6. Complete system tests and deploy the solution.
7. **Agile development and iterative development**:
   - **Agile development**: A methodology that emphasizes flexibility and customer satisfaction through iterative development and delivery.
   - **Iterative development**: A process where the system is developed through repeated cycles (iterations) and smaller portions at a time.
8. **Purpose of a System Vision Document**:
   - To provide a clear and concise description of the system's objectives and the problems it aims to solve, ensuring all stakeholders have a shared understanding.
9. **Difference between a system and a subsystem**:
   - A **system** is a set of interrelated components working together to achieve a common goal.
   - A **subsystem** is a smaller component of a larger system, performing specific functions within the overall system.