OOSE Module 2 - QB Answers

[4 - marks]

1. State the full form and main purpose of a Software Requirement Specification (SRS) document

- **Full Form of SRS**
  SRS stands for **Software Requirements Specification**. It is a formal document that clearly describes what a software system should do and how it should perform.
- **Purpose of SRS**
  The main purpose of an SRS document is to provide a **clear and complete description of software requirements**. It acts as a common reference point for developers, testers, project managers, and clients.
- **Guidance for Development**
  SRS helps the development team understand exactly what needs to be built. This reduces confusion and ensures that development work follows agreed requirements.
- **Basis for Testing and Validation**
  Testers use the SRS to create test cases and verify whether the software meets expectations. It ensures that all requirements are validated properly before release.
- **Contractual Role**
  The SRS also acts as a **contract between the developer and the customer**. It defines scope and prevents misunderstandings later in the project lifecycle.

2. Differentiate between a Functional Requirement and a Non-functional Requirement with one example of each

- **Meaning of Functional Requirements**
  Functional requirements describe **what the system should do**. They define features, services, and actions performed by the software.
- **Example of Functional Requirement**
  In an online banking system, users should be able to **log in using a username and password**. This requirement directly describes a system function visible to users.
- **Meaning of Non-functional Requirements**
  Non-functional requirements describe **how well the system should perform**. They focus on quality attributes like speed, security, reliability, and usability.

- **Example of Non-functional Requirement**
  The system should **respond to user actions in less than 2 seconds**. This does not add a feature but improves user experience.
- **Key Difference**
  Functional requirements drive core features, while non-functional requirements influence system performance and architecture. Both are necessary for successful software development.

3. List the Four Phases of Requirement Engineering

- **Feasibility Study**
  This phase checks whether the project is technically, economically, operationally, legally, and schedule-wise possible. It helps decide if the project should proceed.
- **Requirement Elicitation**
  In this phase, requirements are collected from stakeholders using interviews, surveys, observation, and prototyping. The goal is to understand user needs clearly.
- **Requirement Specification**
  Here, collected requirements are documented in a structured form like the SRS. Both functional and non-functional requirements are clearly written.
- **Requirement Verification and Validation**
  This phase ensures requirements are complete, consistent, achievable, and match customer needs. Errors found here prevent costly changes later.
- **Importance of Phases**
  These phases together ensure accurate requirement definition and reduce project risks. They form the foundation of successful software development.

4. Explain why it is crucial to follow the IEEE standard format for SRS

- **Standardization of Documentation**
  IEEE format provides a **standard structure** for writing SRS documents. This ensures consistency and clarity across different projects.
- **Improved Understanding**
  A standard format helps developers, testers, and stakeholders easily understand requirements. It removes ambiguity by using controlled language.
- **Better Communication**
  Following IEEE standards improves communication between technical and non-technical stakeholders. Everyone refers to the same structured document.
- **Ease of Validation and Testing**
  IEEE SRS makes it easier to trace requirements to design, code, and test cases. This improves verification and validation activities.
- **Regulatory and Quality Compliance**
  Standard formats help meet quality and compliance requirements. They also support version control and proper documentation management.

5. Name any four essential sections that must be present in a comprehensive SRS document

- **Introduction Section**
  This section explains the purpose, scope, and overview of the document. It gives readers a high-level understanding of the software system.
- **Functional Requirements Section**
  It describes all system functions in detail, including inputs, outputs, and expected behavior. Requirements are often prioritized in this section.
- **Non-functional Requirements Section**
  This section defines quality attributes such as performance, security, scalability, and reliability. These requirements ensure system effectiveness.
- **Interface Requirements Section**
  It explains how the system interacts with users or other systems. Examples include user interfaces, APIs, and communication protocols.
- **Importance of Sections**
  Together, these sections ensure completeness and clarity in the SRS document.

6. Explain the primary goal of the Requirement Elicitation phase

- **Understanding Stakeholder Needs**
  The main goal of requirement elicitation is to understand what stakeholders expect from the system. It focuses on identifying real problems to be solved.
- **Gathering Accurate Information**
  Information is collected using interviews, surveys, observation, focus groups, and prototyping. These techniques help capture diverse viewpoints.
- **Clarifying Expectations**
  Elicitation helps clarify unclear or hidden requirements. Stakeholders often do not fully express needs without proper discussion.
- **Reducing Future Errors**
  Correct elicitation prevents misunderstandings that can cause rework later. Early clarity saves time and development cost.
- **Foundation for SRS**
  The collected requirements form the base for requirement specification and SRS preparation.

7. Give four examples of Non-functional Requirements related to performance and security

- **System Response Time**
  The system should respond to user actions within **2 seconds**. This ensures good performance and user satisfaction.
- **Concurrent User Handling**
  The system should support **50,000 concurrent users** during peak hours. This ensures scalability and performance stability.
- **Data Encryption**
  All sensitive data and transactions must be **encrypted**. This protects user data from unauthorized access.
- **Secure Authentication**
  The system should prevent unauthorized access using secure login mechanisms. Security requirements protect system integrity.
- **Importance of NFRs**
  These requirements improve reliability, safety, and user trust in the system.

8. Briefly explain the Requirement Validation phase of Requirement Engineering

- **Purpose of Validation**
  Requirement validation ensures that requirements truly reflect stakeholder needs. It checks whether the right requirements are defined.
- **Checking Completeness and Consistency**
  Validation ensures no requirement is missing or conflicting. Complete and consistent requirements reduce future defects.
- **Stakeholder Involvement**
  Stakeholders review requirements through meetings and walkthroughs. Their feedback confirms correctness.
- **Avoiding Costly Rework**
  Early validation prevents errors from propagating to design and coding stages. This saves time and resources.
- **Continuous Process**
  Validation is iterative and continues throughout the software lifecycle. It ensures quality and customer satisfaction.

[5 – Marks]

1. Explain the IEEE standard format for SRS document. Focus on the main sections and their contents

- **Introduction**
  This section explains the purpose of the SRS document and why it is needed. It also defines the scope of the system and gives an overall overview of the product.
- **Overall Description / General Description**
  This part describes the product perspective, user characteristics, and main features of the system. It helps stakeholders understand how the system fits into a larger environment.
- **Functional Requirements**
  This section clearly explains what the system should do. All system functions, inputs, outputs, and expected behaviors are described in detail.
- **Non-functional Requirements**
  It defines quality attributes like performance, security, reliability, and scalability. These requirements explain how well the system should perform its functions.
- **Interface Requirements**
  This section describes how the system interacts with users or other systems. It includes software interfaces, communication methods, and data formats.
- **Design Constraints and Appendices**
  Design constraints list limitations such as hardware, software, or standards to be followed. Appendices provide references, definitions, and additional information for clarity.

2. For an Online Bookstore System, formulate three Functional Requirements and three Non-functional Requirements

- **Functional Requirement 1**
  The system should allow users to register and log in using valid credentials. This helps identify users and manage their purchase history.
- **Functional Requirement 2**
  Users should be able to search for books by title, author, or category. This feature helps users find books easily.
- **Functional Requirement 3**
  The system should allow users to place orders and make online payments. Order confirmation should be generated after successful payment.

- **Non-functional Requirement 1 (Performance)**
  The system should load search results within 2 seconds. Fast response time improves user experience.
- **Non-functional Requirement 2 (Security)**
  All payment transactions should be encrypted. This ensures protection of sensitive user data.
- **Non-functional Requirement 3 (Availability)**
  The system should be available 24/7 with minimal downtime. High availability increases customer trust.

3. Explain the relationship between the Four Phases of Requirement Engineering and how they lead to the final SRS document

- **Feasibility Study Phase**
  This phase checks whether the project is technically, economically, legally, and operationally possible. It decides whether the project should proceed or not.
- **Requirement Elicitation Phase**
  In this phase, requirements are gathered from stakeholders using interviews, surveys, and observation. It helps understand real user needs.
- **Requirement Specification Phase**
  Collected requirements are documented clearly in a structured format. This phase directly produces the SRS document.
- **Requirement Verification and Validation Phase**
  This phase checks that requirements are complete, consistent, and correct. Errors are corrected before development starts.
- **Relationship Between Phases**
  Each phase builds on the previous one and refines the requirements. Together, they ensure a clear and accurate final SRS document.

4. Describe how the Structure and contents of an SRS document contribute to clarity and communication among stakeholders

- **Clear Organization of Information**
  The structured format divides requirements into logical sections. This helps readers quickly locate required information.
- **Common Understanding for All Stakeholders**
  Developers, testers, managers, and clients all refer to the same document. This avoids misunderstandings and miscommunication.
- **Use of Simple and Consistent Language**
  Requirements are written using simple language and a consistent format. This reduces ambiguity and confusion.
- **Improved Traceability**
  Each requirement can be linked to design, development, and testing. This ensures all requirements are properly implemented.
- **Better Decision Making**
  A clear SRS helps stakeholders make informed decisions regarding scope, cost, and timeline.

5. Discuss the impact of poorly defined requirements on the success and cost of a software project

- **Increased Development Cost**
  Poor requirements lead to frequent changes during development. This increases time and cost significantly.
- **Project Delays**
  Unclear requirements cause confusion among developers. This results in delays in implementation and delivery.
- **Low Quality Software**
  If requirements are incomplete or ambiguous, the final product may not meet user needs. This affects customer satisfaction.
- **Rework and Maintenance Issues**
  Errors in requirements propagate to later stages. Fixing them later requires extensive rework.
- **Risk of Project Failure**
  Poor requirements can lead to scope creep and mismatched expectations. This can cause complete project failure.

6. For a university examination system, classify the following into Functional or Non-functional requirements

- **i) System shall check for malpractice – Functional Requirement**
  This requirement describes a specific action performed by the system. It directly defines system behavior.
- **ii) System shall generate a merit list – Functional Requirement**
  Generating a merit list is a core feature of the system. It produces a concrete output based on input data.
- **iii) System shall be available 99.9% of the time – Non-functional Requirement**
  This requirement describes system availability. It focuses on quality and performance, not functionality.
- **Importance of Classification**
  Correct classification helps in proper design and testing. It ensures both system behavior and quality are addressed.

7. Briefly explain the difference between Product Requirements and Process Requirements in the context of SRS

- **Product Requirements**
  Product requirements describe what the final software product should do. They include functional and non-functional requirements.
- **Focus of Product Requirements**
  These requirements define system features, performance, security, and usability. They directly affect end users.
- **Process Requirements**
  Process requirements describe how the software should be developed. They focus on development methods, standards, and tools.
- **Focus of Process Requirements**
  These requirements guide the development team rather than users. They ensure quality and compliance during development.
- **Key Difference**
  Product requirements define the software, while process requirements define the development approach.

8. Describe the steps and tools used during the Requirement Analysis phase to detect inconsistencies and ambiguities

- **Requirement Review and Walkthroughs**
  Requirements are reviewed by stakeholders and team members. This helps detect missing or conflicting requirements.
- **Verification and Validation Techniques**
  Requirements are checked for completeness, consistency, and feasibility. Errors are corrected early.
- **Use of Prototyping**
  A working model is created to validate requirements. Stakeholders provide feedback to clarify ambiguities.
- **Use of Tools and Techniques**
  Tools like questionnaires, interviews, use cases, user stories, and mind mapping are used. These tools help organize and analyze requirements.
- **Documentation and Traceability**
  Requirements are documented clearly and linked to other artifacts. This ensures transparency and clarity.

[10 – marks]

1. Analyze the importance of Non-functional Requirements. Explain how they differ from Functional Requirements and categorize the various types of Non-functionalRequirements with examples

Importance of Non-functional Requirements

Non-functional Requirements (NFRs) are critical because they define **how** a system operates rather than just **what** it does. While specific features (Functional Requirements) are important, NFRs play a vital role in shaping the overall user experience and ensuring the system's long-term success. They ensure that the software is efficient, reliable, and user-friendly. Without NFRs, a system might work correctly but be too slow, insecure, or difficult to use, leading to failure. They heavily influence the system architecture and performance optimization.

Difference between Functional and Non-functional Requirements

The main differences between these two types of requirements are:

- **Definition:** Functional Requirements describe specific behaviors, features, or operations the system must perform (what it does). In contrast, Non-functional Requirements describe the quality attributes or conditions under which the system operates (how well it performs).
- **Visibility:** Functional requirements are directly visible to users as explicit features (e.g., a login button). Non-functional requirements are not always visible features but are qualities like speed or security that users experience.
- **Measurement:** Functional requirements are easy to measure by checking if a result acts as expected. Non-functional requirements are harder to measure and are often validated against specific benchmarks or metrics.

Types of Non-functional Requirements with Examples

NFRs can be categorized into several key quality attributes. Below are the types mentioned in the document with specific examples:

1. **Performance:** This defines how fast or efficiently the system responds under specific conditions.
    - *Example:* The system should respond to user actions in less than 2 seconds.
    - *Example:* An app should load a menu in under 1 second.
2. **Scalability:** This ensures the system can handle growth in user traffic or data volume.
    - *Example:* The system should handle 100 million users with minimal downtime.
    - *Example:* Supporting up to 50,000 concurrent orders during peak hours.
3. **Security:** This protects the system against unauthorized access and ensures data safety.
    - *Example:* All transactions must be encrypted and comply with security standards.
4. **Usability:** This focuses on how easy the system is to learn and use.
    - *Example:* The app should have an intuitive interface that is easy for first-time users.
5. **Reliability & Maintainability:** These ensure the system is dependable and easy to update or fix after deployment.

2. Prepare a detailed outline for an SRS document for a Hotel Room Booking System following the IEEE standard. Briefly describe the content that would go into the major sections.

Outline for srs based on IEEE -

1. Introduction

This section provides an overview of the entire document and the software product.

- **1.1 Purpose**: Define the objective of this document, which is to outline the requirements for the Hotel Room Booking System. It explains that this document is intended for developers, project managers, and stakeholders to understand what needs to be built.
- **1.2 Document Conventions**: Describe the formatting or standards used in the document (e.g., how priorities are ranked).
- **1.3 Scope**: Describe the software's goals and objectives.
    - *Context:* The system will allow users to search for rooms, make reservations, and process payments, while allowing hotel staff to manage room inventory.
    - *Benefits:* It will reduce manual errors and improve booking efficiency.
- **1.4 References**: List any documents or sources from which information was gathered.

2. Overall Description

This section describes the general factors that affect the product and its requirements.

- **2.1 Product Perspective**: Describe the context of the system. Is it a standalone website, a mobile app, or a sub-system of a larger travel network?
- **2.2 Product Functions**: A summary of the major functions the software will perform.
- **2.3 User Classes and Characteristics**: Identify the different types of users who will use the system.
- **2.4 Operating Environment**: Describe the hardware and software environment in which the system must operate (e.g., Android, iOS, Chrome browser).
- **2.5 Design and Implementation Constraints**: Specify limitations such as regulatory compliance, budget, or required technologies (e.g., must use a specific database).
- **2.6 Assumptions and Dependencies**: List factors that are assumed to be true, such as the availability of a third-party payment gateway.

3. System Features (Functional Requirements)

This is the core section where specific behaviors and operations are defined. Requirements should be ranked by importance.

- **3.1 Search and Reservation:**
    - *Description:* The system must allow users to input dates and location to find available rooms.
    - *Input/Output:* Input dates/guest count; Output list of available rooms and prices.
- **3.2 User Authentication:**
    - *Description:* Users must be able to create accounts and log in securely.

4. External Interface Requirements

This section details how the software communicates with outside elements.

- 4.1 User Interfaces: Describe the logical characteristics of the interface (e.g., screen layouts, buttons, standard error messages).
- 4.2 Hardware Interfaces: Define interactions with physical devices if any (e.g., connecting to a key card writer at the front desk).
- 4.3 Software Interfaces: Specify connections to other software systems, such as banking APIs for payments or email servers for sending confirmations.
- 4.4 Communications Interfaces: Describe network protocols required, such as HTTP/HTTPS or specific data streams.

5. Other Nonfunctional Requirements

This section defines the quality attributes of the system.

- 5.1 Performance Requirements: Specify static and dynamic requirements, such as the number of concurrent users supported or maximum response time for search results (e.g., < 2 seconds).
- 5.2 Security Requirements: Define needs for password encryption, data integrity, and protection of customer financial data.

6. Other Requirements

- Appendix A: Glossary: Definitions of specific terms and acronyms used in the document.
- Appendix B: Analysis Models: Diagrams (like Use Case or Data Flow diagrams) that support the requirements.

3. Explain the Four Phases of Requirement Engineering in detail. Evaluate the importance of the Validation phase in reducing rework in later stages of development

The Four Phases of Requirement Engineering

Although the document lists five steps in the process, the first four sequential phases that focus on defining and creating the requirements are:

1. Feasibility Study This is the preliminary phase where the project's viability is analyzed to decide if it is worth pursuing. It focuses on five key areas:

- **Economic Feasibility:** Analyzes the cost-benefit ratio to see if the project is financially beneficial.
- **Technical Feasibility:** Assesses if the current hardware, software, and technical skills are sufficient to develop the project.
- **Operational Feasibility:** Determines if the product will be easy to operate and maintain and if the solution is acceptable to the organization.
- **Legal Feasibility:** Ensures compliance with laws, regulations, and intellectual property rights.
- **Schedule Feasibility:** Evaluates if the project can be completed within a realistic timeline.

2. Requirements Elicitation This is the process of gathering needs and expectations from stakeholders. It is critical for understanding the problem the software is intended to solve. Techniques used include:

- **Interviews & Surveys:** One-on-one conversations or questionnaires distributed to stakeholders.
- **Focus Groups & Observation:** Discussing needs in groups or observing users in their work environment.
- **Prototyping:** Creating a working model to gather feedback.

3. Requirements Specification The goal of this phase is to create a clear, comprehensive document (SRS) that describes the system requirements. It involves specifying:

- **Functional Requirements:** What the system should do (e.g., input validation, data processing).
- **Non-Functional Requirements:** How well the system should perform (e.g., performance, security, reliability).
- **Constraints & Acceptance Criteria:** Limitations and conditions for release.

4. Requirements Verification and Validation This phase ensures the requirements are correct and meet stakeholder needs.

- **Verification:** Checks if the requirements are complete, consistent, and error-free (ensuring the product is built correctly).
- **Validation:** Checks if the built software matches the customer's actual requirements (ensuring the right product is built)

Importance of the Validation Phase in Reducing Rework

The Validation phase is crucial for minimizing cost and effort in later development stages. Its primary importance lies in **preventing error propagation**.

- **Prevention of Rework:** If requirements are not validated, errors in the definitions propagate to successive stages (like design and coding). Catching these errors early prevents the need for extensive modification and rework later, which saves significant time and budget.
- **Ensuring Customer Satisfaction:** Validation ensures the software is traceable to the customer's actual requirements, not just the developer's interpretation. By testing requirements against stakeholder expectations (e.g., via prototypes), teams can resolve conflicts and misunderstandings before development begins.
- **Consistency and Completeness:** It ensures requirements are practical, achievable, and do not conflict with one another, providing a solid foundation for the development team.

4. Discuss the characteristics of a good SRS document. How can one ensure that the requirements are verifiable, unambiguous, and complete?

Characteristics of a Good SRS Document

A well-constructed SRS document acts as a contract between the developer and the customer. To be effective, it should possess the following key characteristics:

- **Unambiguous:** The document should provide a clear description of requirements to reduce misunderstandings. It must use controlled language to remove ambiguity, ensuring that every requirement has only one interpretation.
- **Complete:** The requirements should be "complete in every sense," meaning no necessary information, features, or constraints are missing.
- **Consistent:** There should be no conflicts or contradictions between requirements. For example, one requirement should not ask for a feature that another requirement prohibits.
- **Verifiable (Testable):** A good SRS facilitates testing by ensuring that all features can be validated. Requirements must be written in a way that allows testers to create specific test cases.
- **Traceable:** It should link requirements to design, code, and tests. This helps in compliance and version control.
- **Feasible (Practically Achievable):** The requirements defined must be practically achievable given the available resources and technology.
- **Understandable:** It should be written in a way that is understandable by both the development team and the stakeholders.

How to Ensure Requirements are Verifiable, Unambiguous, and Complete

To ensure these specific qualities, the Requirement Engineering process suggests the following approaches:

1. Ensuring Requirements are Verifiable

- **Use Quantifiable Metrics:** Avoid vague terms like "fast" or "reliable." Instead, use measurable values. For example, specify that the "system should respond to user actions in less than 2 seconds". Non-functional requirements should be validated against benchmarks or metrics.
- **Create Test Cases:** One of the methods to ensure verifiability is to verify if test cases can be made for the requirement. If you cannot write a test for it, the requirement is likely not verifiable.

2. Ensuring Requirements are Unambiguous

- **Use Simple, Natural Language:** Requirements should be written in natural language using simple terms.
- **Avoid Jargon:** Strictly avoid technical jargon that might confuse stakeholders who are not technical experts.
- **Use Visual Aids:** Incorporate diagrams, models, and visual aids to communicate requirements effectively, as text alone can often be misinterpreted.
- **Consistent Format:** Use a consistent format throughout the document to maintain clarity.

3. Ensuring Requirements are Complete

- **Thorough Elicitation:** Use diverse techniques like interviews, surveys, and focus groups to gather needs from all stakeholders, not just a few.
- **Stakeholder Reviews:** Once specified, requirements must be reviewed and validated by stakeholders to ensure they accurately reflect their needs.
- **Check for Edge Cases:** During analysis, ask questions about possible "edge cases" to ensure the design considers scenarios outside the normal flow.
- **Validation Tasks:** Perform validation tasks to ensure the software to be built is traceable to customer requirements, preventing errors from propagating to later stages

5. Explain the importance of Stakeholder Identification in the requirement engineering process. Describe the challenges in eliciting requirements from diverse stakeholders.

Importance of Stakeholder Identification

While the text does not explicitly label a section "Stakeholder Identification," the importance of this step is embedded in the goals of Requirements Elicitation and the usage of the SRS document. Identifying the right stakeholders is the foundation for the entire engineering process for several reasons:

- Understanding the Core Problem: The primary goal of requirements elicitation is to understand the problem the software is intended to solve and the specific needs of the stakeholders who will use the system.
- Ensuring Success: The process is critical to the success of the software development project because it ensures that the software being developed meets the actual needs and expectations of those involved.
- Comprehensive Coverage: It helps ensure that the software system meets the needs of *all* stakeholders, not just a few. This includes diverse roles such as Product Owners (who define requirements), Developers (who need to understand what to build), Testers (who create validation cases), and Clients (who approve features).
- Facilitating Communication: Identifying stakeholders allows for better communication and collaboration between the development team and the people relying on the product.

Challenges in Eliciting Requirements from Diverse Stakeholders

Eliciting requirements is rarely straightforward. The text outlines several disadvantages and challenges when dealing with diverse stakeholder groups:

- Conflicting Needs: It can be difficult to elicit requirements from stakeholders who often have different needs and priorities. Conflicts may arise between different stakeholders, which can be difficult to resolve.
- Lack of Agreement: It is challenging to ensure that all stakeholders understand and agree on the final requirements.
- Incomplete Identification: It can be difficult to ensure that *all* stakeholders' needs and expectations are taken into account, potentially leading to missed requirements.
- Changing Requirements: Requirements may change over time as stakeholders evolve their understanding, which can result in delays and additional costs.
- Complexity: The process can be time-consuming and expensive, especially if the requirements are complex or if the gathering process is not well-managed.

6] Describe the role and contents of the Overall Description and Specific Requirements sections in an SRS. Why is separating these two sections beneficial?

1. Overall Description

- **Role**: This section provides a high-level summary or overview of the product. Its goal is to describe the general factors that affect the product and its requirements, focusing on the context rather than specific technical details.
- **Contents**: According to the IEEE standard template and the provided descriptions, this section typically includes:
    - Product Perspective: The relationship of the product to other systems.
    - Product Functions: A summary of the major functions the software will perform.
    - User Classes and Characteristics: A description of the different types of users (user community) who will use the system.
    - Operating Environment: The environment in which the software will run.
    - Design and Implementation Constraints: Limitations such as hardware, regulatory policies, or specific standards that must be followed.
    - Assumptions and Dependencies: Factors assumed to be true that could impact the project if changed.

2. Specific Requirements (System Features)

- **Role:** This section contains the detailed description of all aspects of the software to be built. It specifies the expected behavior of the system, detailing exactly what outputs should be produced from given inputs. This is the primary section used by developers to understand what to build and by testers to create validation plans.
- **Contents:** This section is much more granular and includes:
    - Functional Requirements: Detailed descriptions of inputs, their sources, units of measure, valid ranges, and the resulting outputs.
    - External Interface Requirements: Details on how the software communicates with users, hardware, and other software.
    - Performance Requirements: Specific constraints on speed, memory usage, and error rates (static and dynamic requirements).
    - Non-functional Attributes: Qualities like security, portability, and reliability.

Separating these sections improves the clarity and utility of the document for different stakeholders:

1. **Context vs. Detail**: The Overall Description allows stakeholders (like clients and project managers) to understand the "big picture," objectives, and user needs without getting lost in technical minutiae. In contrast, the Specific Requirements provide the precise, unambiguous instructions needed by the technical team (developers and testers) to construct and validate the system.

2. **Managing Constraints**: By separating general constraints (like Design Constraints) in the overview, the design team can identify limitations (e.g., hardware limits or standards) before diving into the specific logic of the features.

3. **Traceability and Testing:** Specific requirements are written in a ranked, detailed order to facilitate measuring outcomes and generating test plans, which is distinct from the general summary provided in the overview.

7. Consider a system that requires high security. Outline the Functional and Nonfunctional requirements needed. Explain how the non-functional requirements will affect the architectural design.

1. Functional Requirements (What the system must do)

Functional requirements describe the specific behaviors, features, and operations the system must provide. For a high-security system, these explicitly requested features might include:

- User Authentication: The system must verify the identity of users, allowing them to log in with a username and password.
- Access Control: The system must restrict access to specific data or features based on user roles (implied by "unauthorized access" concerns).
- Transaction Processing: If financial, users should be able to perform transactions safely.
- Input Validation: The system must validate inputs to prevent malicious data entry.

2. Non-Functional Requirements (How the system must perform)

Non-functional requirements define the attributes or qualities of the system, such as security and reliability. For a high-security system, these constraints include:

- Data Encryption: All transactions must be encrypted and comply with industry security standards to ensure data safety.
- Unauthorized Access Prevention: The system must be robust against unauthorized access attempts.
- Data Integrity: The system must ensure that data remains accurate and consistent, preventing corruption or tampering.
- Reliability: The system must perform its security functions dependably under defined conditions.

3. Impact on Architectural Design

While functional requirements drive the core design and specific features of the system, non-functional requirements dictate how the system is built and structured.

- Influence on Architecture: NFRs explicitly influence the system architecture and performance optimization. For example, a requirement for high security forces the architecture to include layers for encryption, secure communication protocols, and rigorous authentication services.
- Design Constraints: Security requirements act as "Design Constraints". These are limitations or standards that the design team *must* follow, such as using specific cryptographic algorithms or hardware limitations.
- Defining Constraints: Unlike functional requirements which define actions, NFRs define the conditions under which those actions occur. If the security NFR is not met, the architectural design is considered a failure, even if the features work functionality

8. Critically evaluate the challenges faced during the Requirement Engineering process. Suggest methods and best practices to overcome these challenges.

Challenges in Requirement Engineering

The Requirement Engineering process is critical but prone to several significant difficulties that can impact the project's success:

- Stakeholder Conflicts and Diversity: It is often difficult to elicit requirements from stakeholders who have different needs and priorities. There may be direct conflicts between stakeholders that are hard to resolve, making it challenging to ensure everyone understands and agrees on the final requirements.
- Completeness and Coverage: It can be difficult to ensure that *all* stakeholders' needs and expectations are taken into account. Missing a key stakeholder's perspective can lead to incomplete requirements.
- Volatility (Changing Requirements): Requirements often change over time, which is a major risk. These changes can lead to delays, increased costs in the development process, and scope creep.
- Clarity and Consistency: It is challenging to ensure that requirements are clear, consistent, and complete. Without rigorous verification, ambiguity can lead to misunderstandings.
- Resource Intensity: The process can be time-consuming and costly, particularly if the gathering process is not well-managed or if the requirements are complex.

Methods and Best Practices to Overcome Challenges

To mitigate these challenges, the document suggests several methods, tools, and best practices:

1. diverse Elicitation Techniques To handle stakeholder diversity and ensure completeness, use a variety of techniques rather than just one:

- Interviews and Surveys: Use one-on-one conversations and questionnaires to gather input from a larger audience.
- Focus Groups: Bring stakeholders together to discuss needs and potentially resolve conflicts through group interaction.
- Prototyping: Create working models to gather concrete feedback and validate requirements early, reducing the risk of misunderstanding.

2. Robust Requirements Management To handle the volatility of changing requirements, implement a formal management process:

- Track and Control Changes: Monitor changes to identify their source and assess their impact before approval.
- Version Control and Traceability: Keep track of different document versions and link requirements to design and testing elements. This helps prevent scope creep and ensures alignment with project goals.

3. Standardization and Verification To ensure clarity and consistency:

- Use Standard Formats: Follow a consistent format (like the SRS template) and use natural language with simple terms to avoid technical jargon.
- Visual Aids: Use diagrams, models, and visual aids to communicate requirements more effectively than text alone.
- Verification Tasks: Perform reviews, buddy checks, and create test cases to detect conflicts and errors early.

4. Adaptability

- Flexible Approach: As a best practice, Requirement Engineering should be flexible and adaptable, aligning constantly with overall project goals to handle complexity and changes effectively.