3... Software Requirements

Learning Objectives...

- To understand the concept of Requirement engineering with its types.
- To learn the Requirement engineering tasks.
- To study about Requirement gathering and Feasibility study.
- to learn about Fact finding techniques.

3.1 INTRODUCTION REQUIREMENT ENGINEERING

- Requirement Engineering is the process of identifying, documenting, and managing the needs and requirements of stakeholders for a software system. It ensures that the final product aligns with user expectations and business goals.

3.2 TYPES OF REQUIREMENTS

3.2.1 Functional- Non-functional Requirements

1. Functional requirements:

- These define what a system is supposed to do. They describe the specific behaviors, tasks, or functions the system must support.

Examples:

- Users should be able to log in with email and password.
- The system shall send a confirmation email after user registration.
- Admins can view, edit, and delete user accounts.
- The shopping cart shall update the total price automatically when items are added or removed.

Characteristics:

1. Specific to user interactions or system tasks.
2. Often written as "The system shall..."
3. Directly testable.

2. Non-functional Requirements:

- These define how a system performs a function rather than the function itself. The describe the system's quality attributes.

Examples:

- The system shall respond to user queries within 2 seconds.
- The system must support 10,000 concurrent users.
- Data must be encrypted in transit and at rest.
- The system should have 99.99% uptime.

Categories of NFRs:

- Performance (speed, response time)
- Scalability
- Security
- Usability
- Reliability
- Maintainability
- Portability

3.2.2 Domain Requirements

- Domain requirements are a special category of requirements that come from the application domain of the system rather than from the system's functions or quality attributes. They reflect industry-specific rules, standards, constraints, or terminology that the system must comply with.

Examples:

(i) In a Healthcare System:

- Patient data must comply with HIPAA regulations.
- Only certified medical professionals can access electronic prescriptions.

(ii) In a Banking System:

- All financial transactions must follow Know Your Customer (KYC) regulations.
- The system must generate daily transaction reports for auditing.

(iii) In an E-commerce System:

- Product prices must include Value Added Tax (VAT) based on the user's location.
- Orders cannot be shipped internationally without validating customs rules.

3.2.3 Software Requirements

(a) User Requirements:

- High-level description of what users expect from the system.
- Written in natural language with visual aids for better understanding.

(b) System Requirements

- Detailed, technical description of system behavior.
- Often used as a contract between the client and the developer.

3.3 REQUIREMENT ENGINEERING TASKS

(i) Inception:

"What is the problem and who are the stakeholders?"

- Initial understanding of the project and its context.
- Identify stakeholders, business goals, and system scope.
- Define the project vision at a high level.

Activities:

- Meet with clients and stakeholders.
- Identify project constraints and risks.
- Define key system objectives.

(ii) Elicitation:

"What do the stakeholders really need?"

- Gathering user needs, expectations, and requirements.
- Often involves dealing with conflicting or vague requirements.

Techniques:

- Interviews
- Surveys and questionnaires
- Workshops
- Brainstorming
- Observation
- Document analysis

(iii) Elaboration:

"What are the detailed and structured requirements?"

- Refine, expand, and model the gathered requirements.
- Translate high-level goals into detailed functional and non-functional requirements.

Outcomes:

- Use case diagrams
- User stories
- System Models (UML, DFDs, etc.)
- Scenarios and workflows

(iv) Negotiation:

"What can we realistically build?"

- Resolve conflicts between stakeholders.
- Prioritize and agree on the final set of requirements.
- Consider technical constraints, budget, and timeline.

Activities:

- Trade-off analysis
- Feasibility studies
- Requirement prioritization
- Conflict resolution meetings

(v) Specification:

"How do we document the requirements?"

- Document the agreed-upon requirements in a formal and usable way.

Deliverables:

- Software Requirements Specification (SRS)
- User stories
- Functional and non-functional requirement lists

(vi) Validation:

"Are the requirements correct and complete?"

- Ensure that the documented requirements reflect the real needs.
- Detect errors, inconsistencies, omissions, or ambiguities.

Techniques:

- Requirement reviews
- Prototyping
- Simulations
- Stakeholder walkthroughs

3.4 REQUIREMENT GATHERING

(i) Collaborative Requirement Gathering:

- Collaborative Requirement Gathering is an interactive approach to collecting software requirements by actively involving stakeholders, users, analysts, designers, and sometimes developers in shared sessions. The goal is to ensure that the requirements reflect real business needs through mutual understanding and alignment.

(ii) Quality Function Deployment (QFD):

- Quality Function Deployment (QFD) is a structured method used to translate customer needs (the "voice of the customer") into technical requirements for a product or system. It ensures that the final design or system delivers maximum customer satisfaction.

(iii) Usage Scenarios:

- Usage Scenarios describe how users will interact with a system to achieve specific goals. They are narrative stories that illustrate real-world use of the software, helping developers understand user behavior, system context, and functional requirements.

(iv) Elicitation Work Products:

- Elicitation work products are the tangible outputs or deliverables created during the requirements elicitation phase of requirements engineering. These artifacts help document, organize, and communicate the gathered information from stakeholders.

3.5 FEASIBILITY STUDY

(i) Technical Feasibility:

- Technical feasibility is an assessment of whether a proposed system or solution can be successfully developed and implemented using the current or available technology, tools, expertise, and infrastructure.
- Technical feasibility determines whether the technology needed to build the system exists, is accessible, and can support the requirements within constraints such as time, budget, and skills.
- It answers the question:
  - "Can we actually build this with the tools and knowledge we have?"
  - "Can the system be developed with existing technology and resources?"

When is Technical Feasibility Done?

- During the feasibility study phase of project initiation.
- Before committing resources to development.
- Often alongside economic, legal, and operational feasibility studies.

(ii) Operational Feasibility:

- Operational feasibility assesses whether a proposed system will function effectively within the existing organizational structure, culture, and processes. It evaluates whether the solution will be accepted, adopted, and successfully used by the people and departments involved.
- Operational feasibility is the study of how well a proposed system solves business problems, meets user needs, and fits into the organization's day-to-day operations.
- It answers the question:
  - "Will the system work in practice, and will people actually use it?"

Operational Feasibility Addresses:

- **Human Factors:** Will employees adapt to using the system?
- **Process Compatibility:** Does the system integrate well with existing processes?
- **Policy Compliance:** Does it follow company policies and standards?
- **Customer Impact:** Will it improve or hinder customer experience?

(iii) Economic Feasibility:

- Economic feasibility also known as cost-benefit analysis evaluates whether a project is financially viable by comparing its expected costs with its anticipated benefits. It helps decision-makers determine if the investment in a new system or solution is worthwhile.
- Economic feasibility assesses whether the expected financial benefits of a project outweigh its costs, ensuring that the project is a sound economic decision.
- It answers the question:
  - "Is the project worth the money and resources we're planning to spend?"

3.6 FACT FINDING TECHNIQUES

- Fact-finding techniques are systematic methods used by business analysts, system analysts, or requirement engineers to gather information about a system, its processes, and stakeholder needs. These techniques are critical during the requirements elicitation phase to ensure the final system is well-understood and accurately reflects user expectations.

3.6.1 Interviews

- An interview is a direct, face-to-face or virtual conversation between the system analyst and stakeholders to gather information about the current system, business needs, and user expectations.
- Interviews are one of the most effective fact-finding techniques, especially for understanding complex or sensitive requirements.

(a) Structured Interview:

- In a structured interview, the analyst prepares a fixed set of predefined questions and follows them strictly during the session.

Characteristics:

- All interviewees are asked the same questions in the same order.
- Focused on quantitative or clearly defined information. Responses are easier to compare and analyze.

Advantages:

1. Easy to replicate and analyze.
2. Helps stay on topic.
3. Reduces interviewer bias.

Disadvantages:

1. Less flexibility.
2. May miss important but unanticipated information.

Example:

- Asking HR staff a fixed list of questions about leave policy and procedures.

(b) Unstructured Interview:

- In an unstructured interview, the analyst has a general topic or objective but allows the conversation to flow naturally, asking follow-up questions based on responses.

Characteristics:

- Open-ended questions.
- Exploratory and flexible.
- Ideal for discovering unknown or hidden requirements.

Advantages:

1. Provides deeper insights.
2. Encourages open communication.
3. Good for complex or new problem domains.

Disadvantages:

1. Harder to compare responses.
2. Requires more time and skill to conduct.
3. Risk of going off-topic.

Example:

- Talking informally with a manager to understand workflow issues and uncover pain points.

3.6.2 Questionnaires

- Questionnaires are a popular fact-finding technique where a set of written questions is prepared and distributed to a group of stakeholders or users to collect information about their needs, preferences, and opinions.

Purpose:

- To gather data from a large number of people efficiently.
- To obtain consistent information that can be easily compared and analyzed.

Types of Questions:

Table 3.1

Question Type	Description	Example
Closed-ended	Fixed set of answers (Yes/No, multiple choice, rating scales).	"Do you use the current system daily? (Yes/No)".
Open-ended	Respondents provide free-text answers.	"What features would you like to see in the new system?"

Advantages:

1. Can reach a large audience quickly and inexpensively.
2. Easy to quantify and analyze responses, especially closed-ended questions.
3. Allows respondents to answer at their convenience.
4. Helps gather data from geographically dispersed stakeholders.

Disadvantages:

1. Limited depth; may not capture complex or nuanced information.
2. Risk of misinterpretation if questions are unclear.
3. Low response rates can reduce reliability.
4. No opportunity for immediate clarification or follow-up.

3.6.3 Record View

- Record View refers to the way data is presented or accessed as individual records within a system, database, or application interface. Each record represents a single unit of information – for example, a customer, an order, or an employee entry.

Purpose:

- To collect detailed information by examining individual records of data within existing systems or documents.
- Helps analysts understand the structure, content, and attributes of each data entity.
- Useful for verifying how data is stored, what fields are used and identifying data quality issues.

Advantages:

1. **Detailed Insight:** Provides a complete picture of each record's data, helping identify missing or inconsistent information.
2. **Data Validation:** Helps detect errors, anomalies, or redundant data by reviewing full records.
3. **Clear Understanding:** Offers clarity about what information is actually captured and how it relates to business processes.
4. **Easy Documentation:** Can be used to create accurate data dictionaries or system specifications.

Disadvantages:

1. **Time-Consuming:** Reviewing records one-by-one can be slow, especially with large datasets.
2. **Limited Scope:** Focuses on individual records, so it may miss broader process-level or system-wide issues.
3. **Data Overload:** Large records with many fields can overwhelm analysts and obscure important details.
4. **May Not Reveal User Needs:** Only shows existing data; does not capture unspoken user requirements or workflows.

3.6.4 Observation

- Observation is a fact-finding technique where the analyst watches users perform their tasks in their natural work environment to gather first-hand information about processes, workflows, and system usage.

Purpose:

- To understand the real-world execution of tasks.
- To identify how users interact with existing systems.
- To uncover implicit knowledge or undocumented procedures.
- To spot problems, inefficiencies, or workarounds users employ.

Advantages:

1. Provides accurate, real-time insight into actual user behavior.
2. Captures non-verbal cues, environmental factors, and informal processes.
3. Useful for discovering hidden problems users may not mention.
4. Does not rely on user memory or verbal explanation.

Disadvantages:

1. Time-consuming and resource-intensive.
2. Users may change their behavior if they know they are being obser (Hawthorne Effect).
3. Ethical and privacy concerns if observation is intrusive.
4. Difficult to observe complex cognitive tasks that are mostly mental.

3.7 SOFTWARE REQUIREMENT SPECIFICATION (SRS)

- A Software Requirements Specification (SRS) is a document which is used as communication medium between the customer and the supplier. When the softwa requirement specification is completed and is accepted by all parties, the end of requirements engineering phase has been reached. This is not to say, that after acceptance phase, any of the requirements cannot be changed, but the changes m be tightly controlled. The software requirement specification should be edited by bo the customer and the supplier, as initially neither has both the knowledge of what required (the supplier) and what is feasible (the customer).

Definition:

- SRS is a set of precisely stated properties or constraints that a software system mu satisfy.
- SRS is a software requirements document which establishes boundaries on th solution space of the problem of developing a useful software system.
- A software requirements document allows a design to be validated - if the constrain and properties specified in the document are satisfied by the software design, the that design is an acceptable solution to the problem. The task should not b underestimated, example the requirements document for a ballistic missile defen system (1977) contained over 8000 distinct requirements and support paragraphs an was 2500 pages in length.
- According to Heninger, six requirements which a software requirements docume should satisfy:

(i) It should specify only external system behavior.

(ii) It should specify constraints on the implementation.

(iii) It should be easy to change.

(iv) It should serve as a reference tool for system maintainers.

(v) It should record forethought about the life cycle of the system.

(vi) It should characterize acceptable responses to undesired events.

3.7.1 Characteristics of a Software Requirements Specification

- A good SRS should be:
  1. **Unambiguous:**
     - Every requirement has only one interpretation.
     - Each characteristic of the final product is described using a single unique term.
     - A glossary should be used when a term used in a particular context could have multiple meanings.
  2. **Complete:** A complete SRS must possess the following qualities:
     - Inclusion of all significant requirements.
     - Definition of the responses of the software to all realizable classes of input.
  3. **Verifiable:**
     - Every requirement must be verifiable.
     - There must exist some finite cost-effective process with which a person or machine can check that the software meets the requirement.
  4. **Consistent:**
     - No set of individual requirements described in the SRS can be in conflict.
     - **Types of likely conflicts:**
       - Two or more requirements describe the same real world object in different terms.
       - The specified characteristics of real world objects might conflict.
       - There may be a logical or temporal conflict between two specified actions.
  5. **Modifiable:**
     - The structure and style of the SRS are such that any necessary changes to the requirements can be made easily, completely and consistently.
     - **Requirements:** A coherent and easy-to-use organization (including a table of contents, index and cross-referencing). Not be redundant as this can lead to errors.
  6. **Traceable:**
     - The origin of each requirement must be clear.
     - The SRS should facilitate the referencing of each requirement in future development or enhancement documentation.
     - These are of two types:
       - **Backward traceability:** Each requirement must explicitly reference its source in previous documents.
       - **Forward traceability:** Each requirement must have a unique name or reference number.

.7.2 Structure of SRS Document

ntroduction:

The following subsections of the Software Requirements Specifications (SRS) document should provide an overview of the entire SRS. The thing to keep in mind as you write this document is that you are telling what the system must do-so that designers can ultimately build it.

(i) **Purpose:** Identify the purpose of this SRS and its intended audience. In this subsection, describe the purpose of the particular SRS and specify the intended audience for the SRS.

(ii) **Scope:** In this subsection, identify the software product(s) to be produced by name. Explain what the software product(s) will, and, if necessary, will not do. Describe the application of the software being specified, including relevant benefits, objectives, and goals. Be consistent with similar statements in higher-level specifications if they exist. This should be an executive-level summary. Do not enumerate the whole requirements list here.

(iii) **Definitions, Acronyms, and Abbreviations:** Provide the definitions of all terms, acronyms, and abbreviations required to properly interpret the SRS. This information may be provided by reference to one or more appendices in the SRS or by reference to documents. This information may be provided by reference to an Appendix.

(iv) **References:** In this subsection,

(a) Provide a complete list of all documents referenced elsewhere in the SRS.

(b) Identify each document by title, report number (if applicable), date, and publishing organization.

(c) Specify the sources from which the references can be obtained. This information can be provided by reference to an appendix or to another document. If your application uses specific protocols then reference them here so designers know where to find them.

(v) **Overview:** In this subsection, describe what the rest of the SRS contains. Explain how the SRS is organized. Don't rehash the table of contents here. Point people to the parts of the document they are most concerned with.

he Overall Description:

Describe the general factors that affect the product and its requirements. This section does not state specific requirements. Instead, it provides a background for those requirements, which are defined in section 3.3, and makes them easier to understand. In a sense, this section tells the requirements in plain English. Section 3.3 will contain a specification written for the developers.

(i) **Product Perspective:** Put the product into perspective with other related products. If the product is independent and totally self-contained, it should be so stated here. If the SRS defines a product that is a component of a larger system, as frequently occurs, then this subsection relates the requirements of the larger system to functionality of the software and identifies interfaces between that system and the software. If you are building a real system, compare its similarity and differences to other systems in the marketplace. If you are doing a research-oriented project, what related research compares to the system you are planning to build.

A block diagram showing the major components of the larger system, interconnections, and external interfaces can be helpful. This is not a design or architecture picture. It is more to provide context, especially if your system will interact with external actors.

(ii) **Product Functions:** Provide a summary of the major functions that the software will perform. Sometimes the function summary that is necessary for this part can be taken directly from the section of the higher-level specification (if one exists) that allocates particular functions to the software product.

For clarity: The functions should be organized in a way that makes the list of functions understandable to the customer or to anyone else reading the document for the first time.

Textual or graphic methods can be used to show the different functions and their relationships. Such a diagram is not intended to show a design of a product but simply shows the logical relationships among variables. This describes the functionality of the system in the language of the customer.

(iii) **User Characteristics:** Describe those general characteristics of the intended users of the product including educational level, experience, and technical expertise. Do not state specific requirements but rather provide the reasons what certain specific requirements are:

- What is it about your potential user base that will impact the design? Their experience and comfort with technology will drive UI design. Other characteristics might actually influence internal design of the system.

(iv) **Constraints:** Provide a general description of any other items that will limit the developer's options. These can include:

(a) Regulatory policies

(b) Hardware limitations (For example, signal timing requirements)

(c) Interface to other applications

(d) Parallel operation

(e) Audit functions
(f) Control functions
(g) Higher-order language requirements
(h) Signal handshake protocols (For example, XON-XOFF, ACK-NACK).
(i) Reliability requirements
(j) Criticality of the application
(k) Safety and security considerations.

This section captures non-functional requirements in the customer's language. A more formal presentation of these will occur in Section 3.3.

(v) **Assumptions and Dependencies:** List each of the factors that affect the requirements stated in the SRS. These factors are not design constraints on the software but are, rather, any changes to them that can affect the requirements in the SRS. For example, an assumption might be that a specific operating system would be available on the hardware designated for the software product. If, in fact, the operating system were not available, the SRS would then have to change accordingly. This section is a catch all for everything else that might influence the design of the system and that did not fit in any of the categories above.

(vi) **Apportioning of Requirements:** Identify requirements that may be delayed until future versions of the system. After you look at the project plan and hours available, you may realize that you just cannot get everything done. This section divides the requirements into different sections for development and delivery. Remember to check with the customer – they should prioritize the requirements and decide what does and does not get done. This can also be useful if you are using an iterative life cycle model to specify which requirements will map to which interaction.

3.7.3 Specific Requirements

- This section contains all the software requirements at a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements. Throughout this section, every stated requirement should be externally perceivable by users, operators, or other external systems. These requirements should include at minimum description of every input (stimulus) into the system, every output (response) from the system and all functions performed by the system in response to an input or in support of an output.

List every piece of information that is required so the designers can build the right UI and data tables.

(i) **External Interfaces:** This contains a detailed description of all inputs into and outputs from the software system. It complements the interface descriptions in section 3.2 but does not repeat information there.

(ii) **Functions:** Functional requirements define the fundamental actions that must take place in the software in accepting and processing the inputs and in processing and generating the outputs. These are generally listed as 'shall' statements starting with "The system shall..."

(iii) **Performance Requirements:** This subsection specifies both the static and the dynamic numerical requirements placed on the software or on human interaction with the software, as a whole.

(iv) **Logical Database Requirements:** This section specifies the logical requirements for any information that is to be placed into a database.

(v) **Design Constraints:** Specify design constraints that can be imposed by other standards, hardware limitations, etc.

(vi) **Software System Attributes:** There are a number of attributes of software that can serve as requirements. It is important that required attributes are specified so that their achievement can be objectively verified. These are also known as non-functional requirements or quality attributes.

3.7.4 Change Management Process

- Identify the change management process to be used to identify, log, evaluate, and update the SRS to reflect changes in project scope and requirements. How are you going to control changes to the requirements. Can the customer just call up and ask for something new? Does your team have to reach consensus? How do changes to requirements get submitted to the team? Formally in writing, email or phone call?

3.7.5 Document Approvals

- Identify the approvers of the SRS document. Approver name, signature, and date should be used.

3.7.6 Supporting Information

- The supporting information makes the SRS easier to use. It includes:
  (i) Table of Contents
  (ii) Index in. Appendices
- The Appendices are not always considered part of the actual requirements specification and are not always necessary. They may include:
  (a) Sample I/O formats, descriptions of cost analysis studies, results of user surveys.
  (b) Supporting or background information that can help the readers of the SRS.
  (c) A description of the problems to be solved by the software.

(d) Special packaging instructions for the code and the media to meet security, export, initial loading, or other requirements. When Appendices are included, the SRS should explicitly state whether or not the Appendices are to be considered part of the requirements.

Check Your Understanding

1. What is the main goal of Requirement Engineering?
   (a) Write code
   (b) Define system requirements clearly
   (c) Conduct testing
   (d) Manage hardware
2. Functional requirements describe:
   (a) System architecture
   (b) Budget of the project
   (c) How the system should perform certain tasks
   (d) System size
3. Non-functional requirements include all of the following except:
   (a) Performance (b) Usability
   (c) Reliability (d) Login process
4. Domain requirements are derived from:
   (a) Budget documents
   (b) Specific application domain or environment
   (c) User interface
   (d) Testing plan
5. User requirements are usually:
   (a) Written in technical language
   (b) Designed for software engineers
   (c) Written in natural language for end-users
   (d) Stored in source code
6. System requirements are also known as:
   (a) Executive needs (b) Technical specifications
   (c) User manuals (d) Design patterns
7. Which task involves understanding the problem domain and stakeholders?
   (a) Inception (b) Validation
   (c) Testing (d) Deployment

8. The process of extracting requirements from stakeholders is known as:
   (a) Elaboration (b) Elicitation
   (c) Negotiation (d) Specification
9. In which RE task are conflicts in requirements resolved?
   (a) Validation (b) Specification
   (c) Elaboration (d) Negotiation
10. The final detailed document of the requirements is created during:
   (a) Elicitation (b) Elaboration
   (c) Specification (d) Validation
11. Collaborative Requirement Gathering mainly involves:
   (a) One-on-one interviews
   (b) Multiple stakeholders working together
   (c) Studying manuals
   (d) System testing
12. QFD (Quality Function Deployment) is used to:
   (a) Transform customer needs into technical requirements
   (b) Track bugs
   (c) Design network
   (c) Manage finances
13. Usage Scenarios help in:
   (a) Tracking errors
   (b) Designing databases
   (c) Understanding how users interact with the system
   (d) Writing source code
14. Technical Feasibility checks:
   (a) Budget
   (b) User satisfaction
   (c) Market share
   (d) Technology resources and capabilities

15. Operational Feasibility focuses on:
    (a) Is the system usable by the organization?
    (b) Cost and expenses
    (c) Data storage
    (d) Project scheduling
16. Economic Feasibility is mainly about:
    (a) Business structure (b) Functional diagrams
    (c) Cost-benefit analysis (d) Database performance
17. A structured interview is:
    (a) Informal and spontaneous
    (b) Without any questions
    (c) A group discussion
    (d) Based on a set of predefined questions
18. An unstructured interview allows:
    (a) Only closed questions (b) Strict format
    (c) Free-flowing conversation (d) Only written responses
19. Questionnaires are good when:
    (a) You need face-to-face communication (b) Large group input is required
    (c) One person is involved (d) Observation is difficult
20. Record view technique involves:
    (a) Checking employee attendance
    (b) Code reviews
    (c) Hosting meetings
    (d) Analyzing existing documents and files
21. Observation is useful when:
    (a) Users can't express needs clearly (b) The project is already deployed
    (c) Only financial data is required (d) Interviews fail

Answers

1. (b)	2. (c)	3. (d)	4. (b)	5. (c)	6. (b)	7. (a)
8. (b)	9. (d)	10. (c)	11. (b)	12. (a)	13. (c)	14. (d
15. (a)	16. (c)	17. (d)	18. (c)	19. (b)	20. (d)	21. (

Practice Questions

Q.I Answer the following questions in short.

1. What is requirement engineering?
2. Give 3 examples each of functional and non-functional requirements.
3. What are domain requirements?
4. What are system requirements?

Q.II Answer the following questions in detail:

1. Why is requirement engineering important in software development?
2. Differentiate between functional and non-functional requirements with examples.
3. How do domain requirements affect software design?
4. Compare user and system requirements.
5. Describe the following tasks with examples:
   (a) Inception (b) Elicitation
   (c) Elaboration (d) Negotiation
   (e) Specification (f) Validation
6. What is requirement validation? Why is it necessary?
7. How does collaboration improve requirement quality?
8. What is a feasibility study? Why is it performed before development?
9. How do feasibility studies help in project decision-making?
10. Differentiate between structured and unstructured interviews.
11. What are the advantages and limitations of interviews in requirement gathering? How are questionnaires used in fact-finding?
12. When are questionnaires more effective than interviews?
13. What is the purpose of reviewing existing records in requirement analysis?
14. What is observation technique in requirement gathering?
15. Discuss the pros and cons of observation.
16. Explain the requirement engineering process with its activities.
17. Describe technical, operational, and economic feasibility with examples.

18. What are the different requirement gathering techniques? Explain any two.
19. Short notes on:
    (a) Inception and Elaboration
    (b) Usage Scenarios
    (c) System vs User Requirements
20. Explain fact-finding techniques with advantages and disadvantages.

***