Requirements Engineering Processes

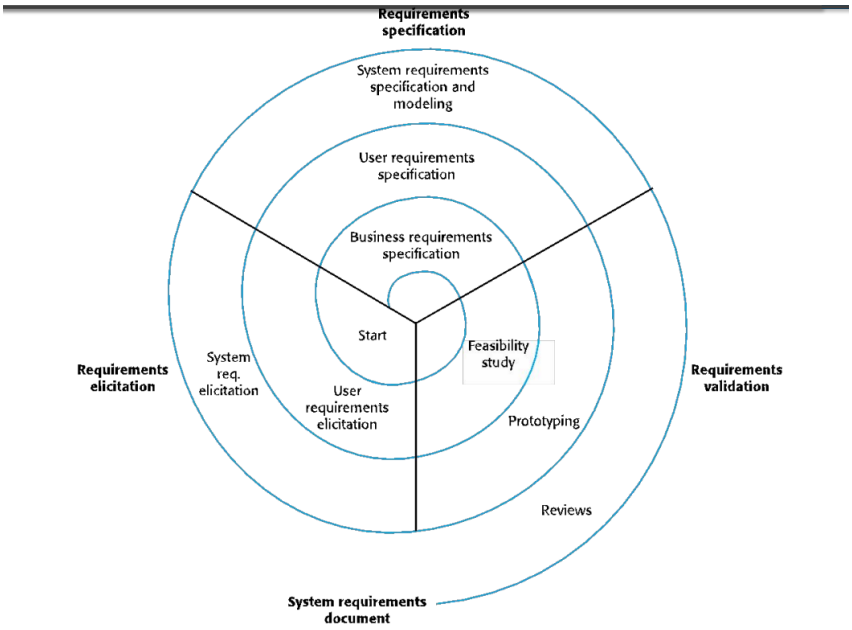
Notes By Mannan Ul Haq

Processes for Requirements Engineering (RE) differ based on the application domain, stakeholders, and the organization developing the requirements.

Common Activities: Despite variations, there are several generic activities present in all RE processes:

1. **Requirements Elicitation**: Gathering requirements from stakeholders.
2. **Requirements Analysis**: Examining and understanding gathered requirements.
3. **Requirements Validation**: Ensuring that gathered requirements meet stakeholders' needs and are feasible.
4. **Requirements Management**: Organizing, documenting, and tracking requirements throughout the project lifecycle.

Iterative Nature: RE is typically iterative, with these processes being interwoven and revisited throughout the project.



Requirements Elicitation

Definition:

Also known as requirements discovery, it is the initial phase of gathering information about the application domain and system requirements.

Participants:

- Involves technical staff collaborating with customers and stakeholders.

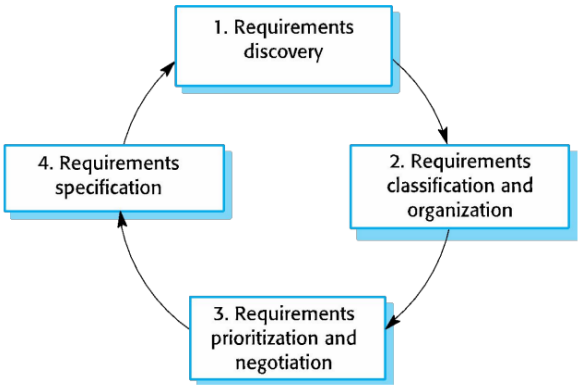
- Stakeholders may include end-users, managers, engineers, domain experts, and other relevant parties.

Objectives:

- Understand the application domain, required system services, and operational constraints.
- Identify system stakeholders and their roles in the project.

Key Activities:

1. **Requirements Discovery:**
   - Interact with stakeholders to understand their needs and expectations.
   - Identify domain requirements along with user and system requirements.
2. **Requirements Classification and Organization:**
   - Group related requirements and organize them into coherent clusters for better understanding.
3. **Prioritization and Negotiation:**
   - Prioritize requirements based on importance and resolve any conflicts that arise among stakeholders' needs.
4. **Requirements Specification:**
   - Document the gathered requirements and prepare them for the next phase of the project.



Challenges:

- Stakeholders' uncertainty about their needs.
- Expression of requirements in varied terms by stakeholders.
- Possibility of conflicting requirements from different stakeholders.
- Influence of organizational and political factors on system requirements.
- Evolution of requirements during the analysis process due to emerging stakeholders or changes in the business environment.

Interviewing:

- **Purpose**: Formal or informal interviews with stakeholders are integral to most Requirements Engineering (RE) processes.
- **Types of Interviews**:
  - **Closed Interviews**: Based on predetermined questions.
  - **Open Interviews**: Explore various issues with stakeholders, allowing for more flexibility.
- **Effective Interviewing**:
  - Stay open-minded and let people share their thoughts.
  - Ask questions to get conversations going.

Interviews in Practice:

- Typically involve a mix of closed and open-ended approaches.
- Helps understand what people do and how they'll use the system.
- Interviewers should stay open and guide discussions.

Problems with Interviews:

- Language barriers may arise when application specialists use technical terminology unfamiliar to requirements engineers.
- Interviews may not effectively capture domain requirements due to difficulties in understanding specific domain terminology or concepts.

Ethnography:

- Involves social scientists observing and analyzing how people work over time.
- Provides insights into social and organizational factors influencing work practices.
- Reveals rich and complex aspects of work not captured by simplistic system models.

Scope of Ethnography:

- Shows how people actually work, not just how we think they do.
- Highlights how people work together and adapt.
- Effective for understanding existing processes but may not identify new system features.

Requirements Specification

- **Definition**: Writing down what users and the system need in a document.
- **User Requirements**: Easy for non-technical people to understand.
- **System Requirements**: More detailed and technical.

Ways to Write Specifications:

- **Natural Language**: Using simple sentences.

- **Structured Natural Language**: Using a standard form with different sections.
- **Graphical Notations**: Using diagrams like UML.
- **Mathematical Specifications**: Using math concepts, but not easy for everyone to understand.

Guidelines for Writing Requirements:

- Use a consistent format and language.
- Highlight key points.
- Explain why each requirement is necessary.
- Avoid technical jargon.

Problems with Natural Language:

- Lack of clarity: It's hard to be precise without making the document confusing.
- Confusion between functional and non-functional requirements.
- Mixing several requirements together in one statement.

Structured Specifications

- **Definition**: Structured specifications involve writing requirements in a standardized format, restricting the freedom of the writer. This method aims to bring consistency and clarity to requirement documents.
- **Example**: Consider a requirement for an insulin pump, a medical device used by diabetic patients. A structured specification might look like this:

  Requirement ID: INS-01

  Title: Insulin Delivery

  Description: The insulin pump shall deliver the prescribed insulin dosage to the patient at the specified time intervals.

  Acceptance Criteria: The insulin pump must accurately deliver the prescribed dosage within ±5% of the target amount, and must not exceed the specified delivery time window by more than 1 minute.
- **Suitability**: Structured specifications work well in technical domains, such as embedded control systems, where precision and consistency are crucial for safety and functionality.
- **Limitations**: However, for business systems or projects with evolving requirements, structured specifications may be too rigid, as they can stifle creativity and flexibility in capturing complex user needs and system behaviors.

Tabular Specification

- **Definition**: Tabular specifications are used alongside natural language to provide a structured format for defining requirements, especially when multiple alternative actions need to be described.

- **Example**: In the context of an insulin pump system, a tabular specification might outline how to calculate the insulin dosage based on different scenarios of blood sugar level changes:

Condition	Action
Sugar level falling (r2 < r1)	CompDose = 0
Sugar level stable (r2 = r1)	CompDose = 0
Sugar level increasing and rate of increase decreasing ((r2 - r1) < (r1 - r0))	CompDose = 0
Sugar level increasing and rate of increase stable or increasing ((r2 - r1) ≥ (r1 - r0))	CompDose = round((r2 - r1)/4)

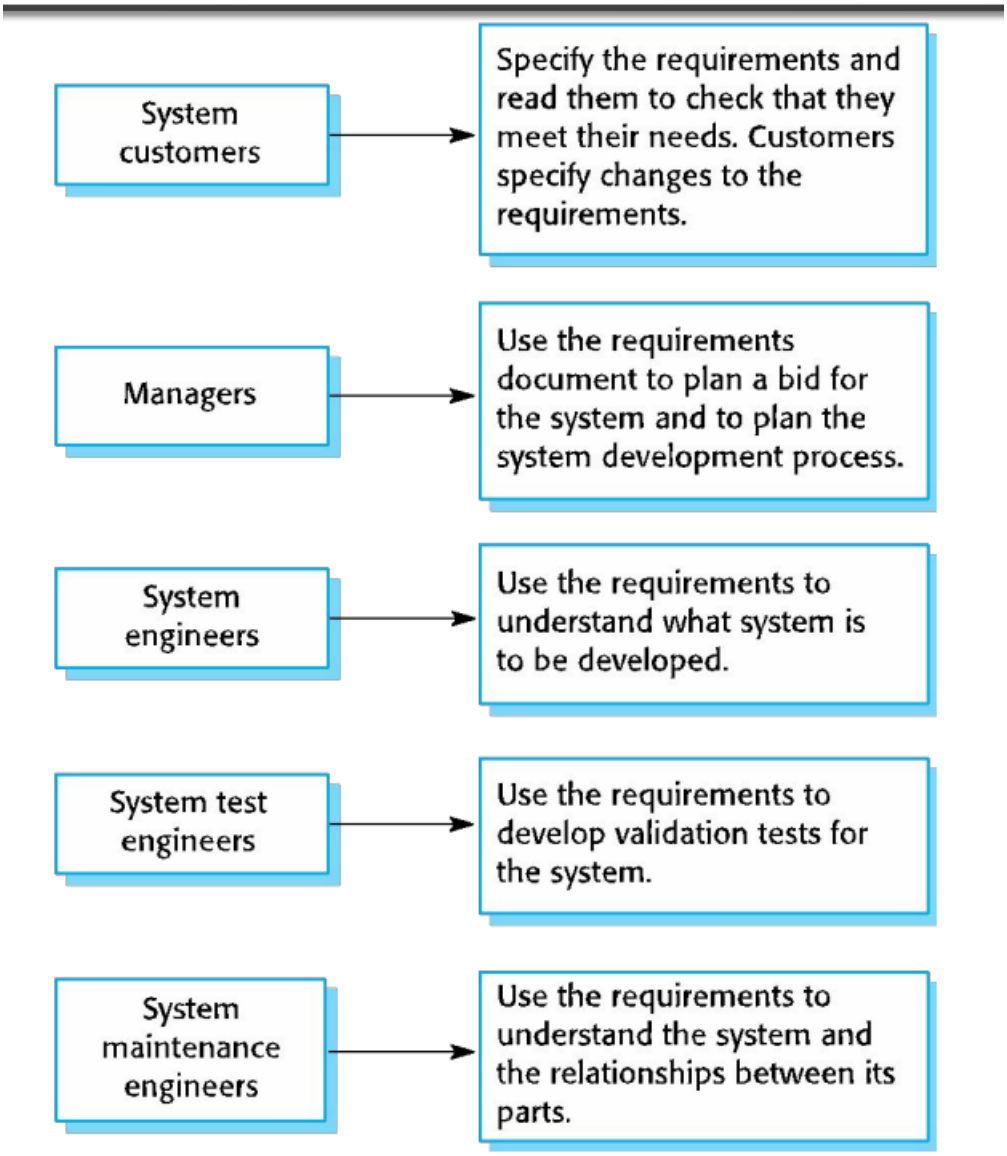
- **Suitability**: Tabular specifications are particularly useful when there are multiple conditions or scenarios to consider, providing a clear and concise way to outline each action.

Use Cases:

- **Definition**: Use cases are scenarios included in the Unified Modeling Language (UML) that identify the actors involved and describe their interactions with the system.
- **Purpose**: They help capture all possible interactions with the system and provide a high-level graphical model supplemented by more detailed tabular descriptions.
- **Detailing**: Use cases can be further detailed using UML sequence diagrams, which show the sequence of event processing in the system.

The Software Requirements Document

- **Definition**: The software requirements document serves as the official statement outlining what is required of the system developers during the development process.
- **Contents**:
  - It should encompass both a clear definition of user requirements and a detailed specification of the system requirements.
  - It's important to note that this document is not a design document. Rather, it focuses on outlining WHAT the system should do, rather than HOW it should do it.
- **Purpose**:
  - Provides a comprehensive understanding of the system's functionalities and constraints to all stakeholders involved in the development process.
  - Serves as a reference point for system developers, guiding them throughout the development lifecycle.
- **Users of a requirements document:**



Structure of a Requirements Document

1. **Preface**
   - Defines the expected readership of the document.
   - Describes the version history, including reasons for new versions and summaries of changes.
2. **Introduction**
   - Describes the system's necessity.
   - Briefly outlines system functions and interactions with other systems.
   - Explains how the system aligns with the organization's business or strategic objectives.
3. **Glossary**

- Defines technical terms used in the document.
- Ensures clarity and understanding, regardless of reader expertise.

4. **User Requirements Definition**
    - Describes services provided for users.
    - Specifies nonfunctional system requirements.
    - May use natural language, diagrams, or other understandable notations.
    - Specifies product and process standards.
5. **System Architecture**
    - Presents a high-level overview of the anticipated system architecture.
    - Highlights reused architectural components.
6. **System Requirements Specification**
    - Describes functional and nonfunctional requirements in detail.
    - May include further detail on nonfunctional requirements.
    - Defines interfaces to other systems.
7. **System Models**
    - Includes graphical system models depicting relationships between system components and the environment.
    - Examples include object models, data-flow models, or semantic data models.
8. **System Evolution**
    - Describes fundamental assumptions of the system.
    - Addresses anticipated changes due to hardware evolution or changing user needs.
    - Aids system designers in avoiding design decisions that could restrict future changes.
9. **Appendices**
    - Provides detailed, specific information related to the application being developed.
    - Includes hardware and database descriptions.
10. **Index**
    - May include several indexes, such as alphabetic, diagrams, and functions indexes, for easy navigation and reference.

Requirements Validation

Requirements validation ensures that the defined requirements accurately reflect what the customer wants and that they can be effectively implemented.

Importance of Requirements Validation:

- Errors in requirements can be costly to rectify post-delivery.
- Fixing a requirements error after delivery may cost up to 100 times more than fixing an implementation error.

Criteria for Requirements Checking:

1. **Validity:** Do the system functions support the customer's needs optimally?
2. **Consistency:** Are there any conflicts among requirements?
3. **Completeness:** Are all customer-required functions included?
4. **Realism:** Can the requirements be implemented within available resources?
5. **Verifiability:** Can the requirements be effectively verified?

Techniques for Requirements Validation:

1. **Requirements Reviews:**
   - Systematic manual analysis of requirements.
   - Involves both client and contractor staff.
   - Can be formal or informal.
2. **Prototyping:**
   - Uses an executable model to validate requirements.
   - Allows for early testing and feedback.
3. **Test-Case Generation:**
   - Develops tests to validate requirements.
   - Checks for testability and effectiveness.

Review Checks:

1. **Verifiability:**

   Is the requirement realistically testable?
2. **Comprehensibility:**

   Is the requirement properly understood?
3. **Traceability:**

   Is the requirement's origin clearly stated?
4. **Adaptability:**

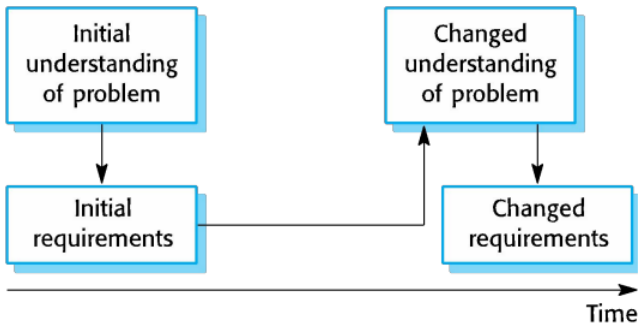
   Can the requirement be changed without significantly affecting other requirements?

Requirements Change

Requirements for a system often change due to shifts in business and technical environments, as well as evolving user needs.

Factors Contributing to Requirements Change:

- **Environmental Changes:** Introduction of new hardware, interface requirements with other systems, evolving business priorities, and new regulations.
- **Diverse Stakeholder Needs:** System customers and end-users may have conflicting requirements, necessitating compromises.
- **User Community Diversity:** Large systems typically serve diverse user communities with conflicting priorities, leading to the need for balancing support.



Requirements Management

Requirements management is like keeping track of a moving target during the process of building a system. It involves making sure that any changes or new ideas about what the system should do are properly handled.

Important Parts of Requirements Management:

- **Finding New Ideas:** Sometimes new ideas come up while building the system or after it's finished.
- **Keeping Track:** It's like keeping a list of all the things the system needs to do and making sure they're organized and connected.
- **Making Changes Official:** There's a formal way to suggest and decide on changes to the system's requirements.

Planning for Requirements Management:

- **Figuring Out Details:** Deciding how closely to watch each requirement.
- **Making Key Choices:**
  - **Identifying Requirements:** Giving each requirement its own name so we can keep track of it.
  - **Handling Changes:** Setting up rules for how changes are proposed and decided on.
  - **Keeping Track:** Deciding how to link different requirements and parts of the system.
  - **Using Tools:** Using software or other tools to help manage all this information.

Dealing with Changes:

Making Decisions:

- **Checking Ideas:** Looking at new ideas to see if they make sense.
- **Seeing the Impact:** Figuring out how changes will affect the system.
- **Putting Changes into Action:** Making the actual changes to the plans and designs.

Why Requirements Management Matters:

- **Staying in Control:** Making sure the project stays on track without getting too complicated.
- **Talking Clearly:** Making sure everyone involved understands what's happening with the system.
- **Avoiding Problems:** Preventing problems that might come up if changes aren't handled well.