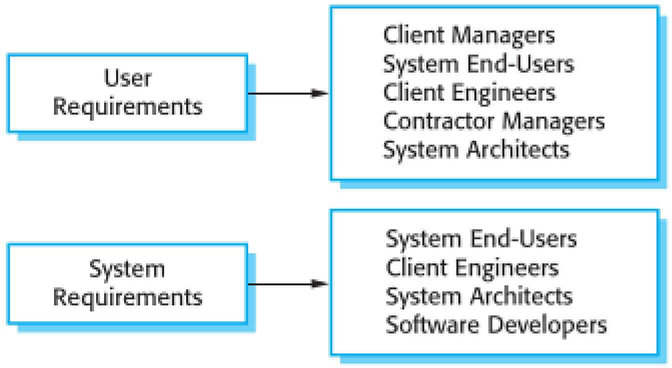
Cavalier Institute - https://cavalierinstitutions.com

Date	Sep-19-2024	Session No	2

Topic : Requirements Engineering

Requirements Engineering

Software system requirements are classified as functional or non-functional requirements.



Readers of different types of requirements specification

Functional Requirements

- **Definition**: Functional requirements define what a system should do. They describe the specific behaviors, tasks, and functionalities the system must perform to satisfy the needs of users and stakeholders.
- **Examples**:

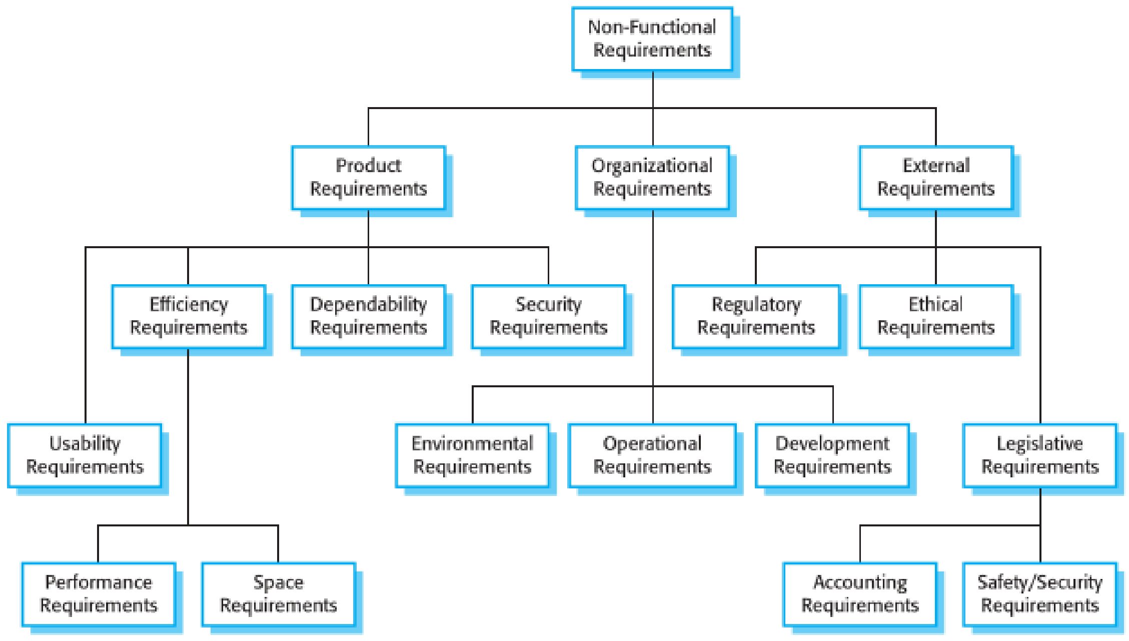
  - **Login Functionality**: The system must allow users to log in with a username and password.
  - **Search Feature**: The system should allow users to search for products using keywords.
  - **Order Processing**: The system must process payments and send confirmation emails for completed orders.

Non-Functional Requirements

- **Definition**: Non-functional requirements describe how the system performs its functions. They define system attributes such as performance, security, usability, and reliability.
- **Examples**:
  - **Performance**: The system should load the homepage in under 2 seconds for all users.
  - **Security**: The system must use encryption to protect sensitive user data.
  - **Scalability**: The system should be able to handle up to 10,000 concurrent users without performance degradation.

Functional requirements describe **what** the system should do, while non-functional requirements describe **how well** the system should do it.

Non-Functional Requirements



Requirements Engineering Process

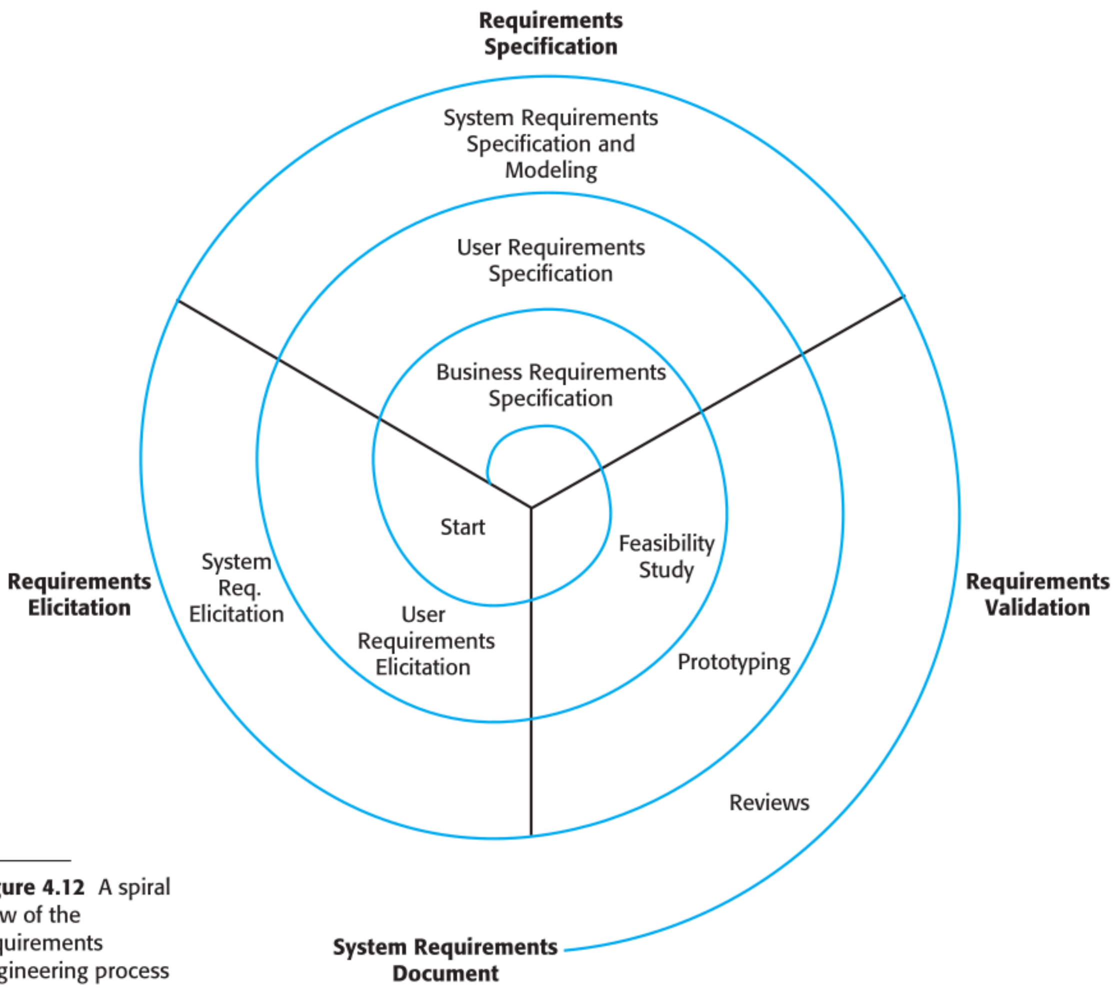
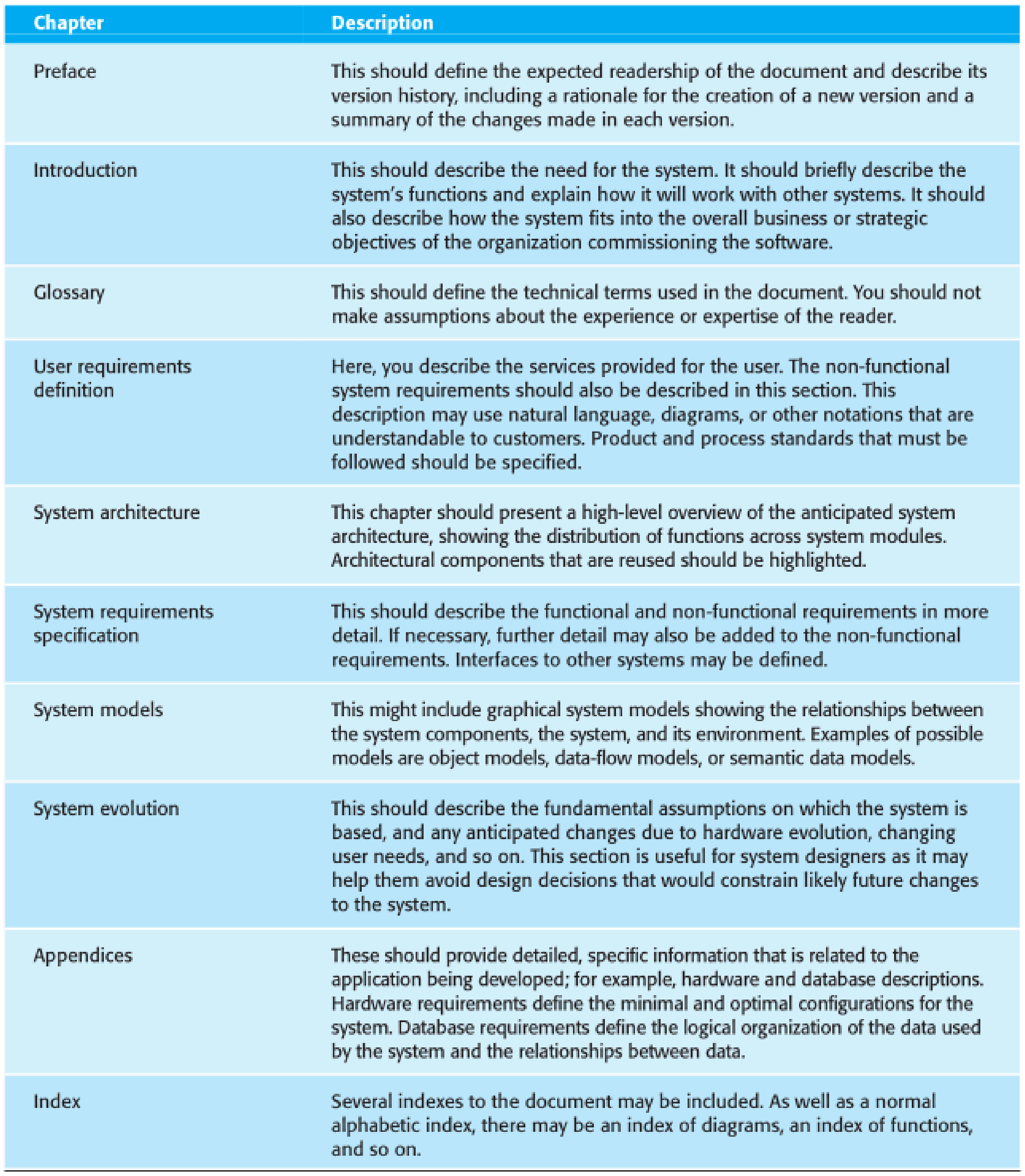

Requirements Specification
System Requirements Specification and Modeling
User Requirements Specification
Business Requirements Specification
Start
Feasibility Study
Prototyping
Reviews
Requirements Validation
Requirements Elicitation
System Req. Elicitation
User Requirements Elicitation
System Requirements Document

Figure 4.12 A spiral view of the requirements engineering process

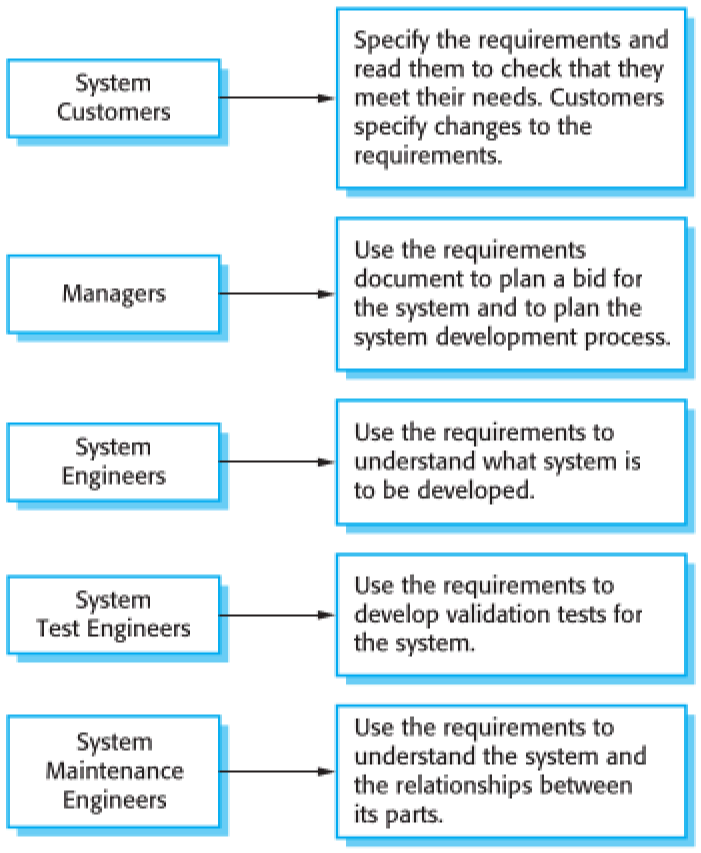
Requirements Engineering (RE) is a critical process in software engineering that involves defining, documenting, and maintaining the needs and requirements for a software system. It is one of the foundational activities in the software development lifecycle and ensures that the final product meets the user's expectations, business goals, and technical constraints. Requirements engineering typically encompasses several stages, including elicitation, specification, validation, and management. Below is a detailed breakdown of these stages:

Requirements Document Structure



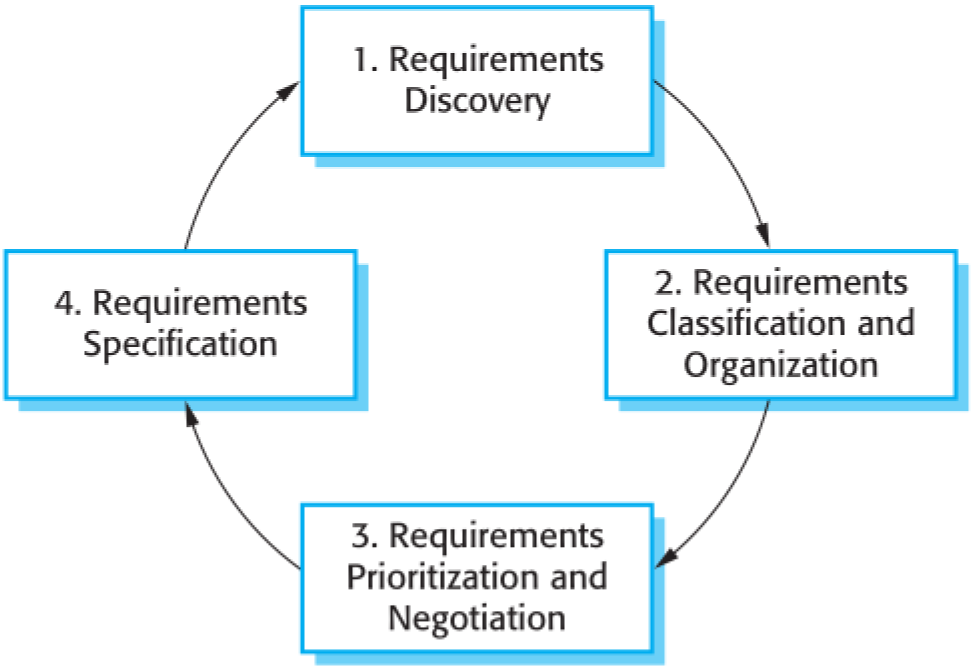
Chapter	Description
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	Here, you describe the services provided for the user. The non-functional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.
System architecture	This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.
System requirements specification	This should describe the functional and non-functional requirements in more detail. If necessary, further detail may also be added to the non-functional requirements. Interfaces to other systems may be defined.
System models	This might include graphical system models showing the relationships between the system components, the system, and its environment. Examples of possible models are object models, data-flow models, or semantic data models.
System evolution	This should describe the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on. This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system.
Appendices	These should provide detailed, specific information that is related to the application being developed; for example, hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organization of the data used by the system and the relationships between data.
Index	Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, and so on.

Users of Requirements Document

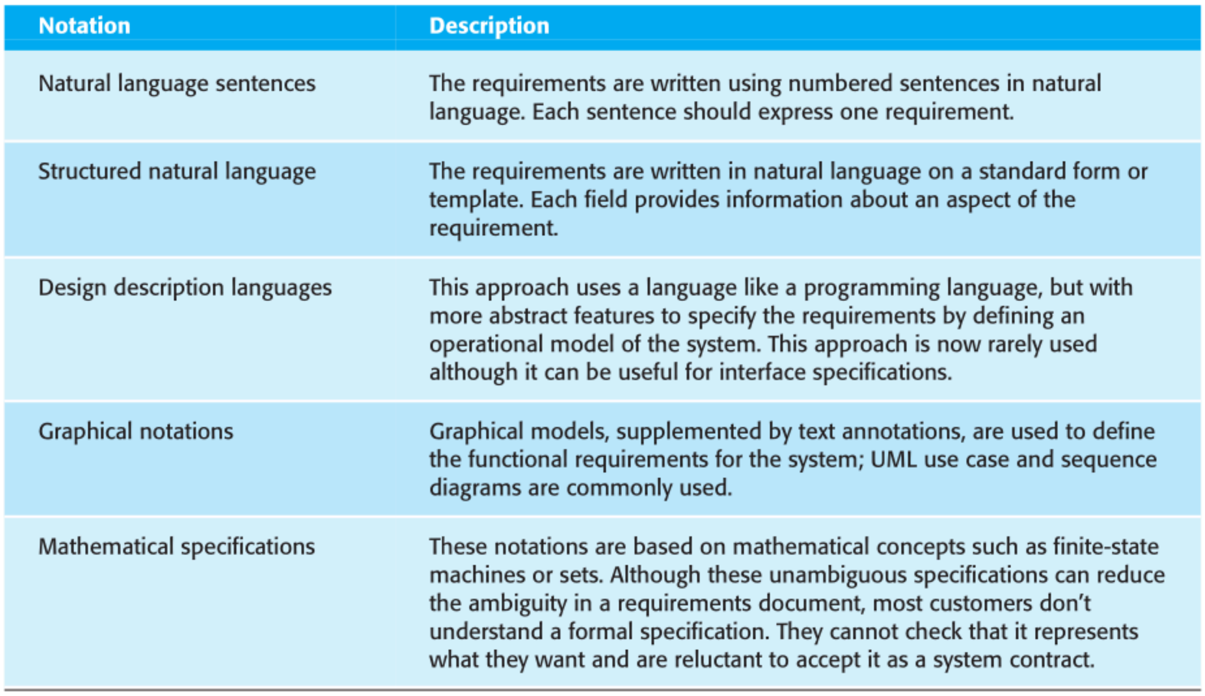

System Customers
Specify the requirements and read them to check that they meet their needs. Customers specify changes to the requirements.
Managers
Use the requirements document to plan a bid for the system and to plan the system development process.
System Engineers
Use the requirements to understand what system is to be developed.
System Test Engineers
Use the requirements to develop validation tests for the system.
System Maintenance Engineers
Use the requirements to understand the system and the relationships between its parts.

1. Requirements Elicitation

- **Purpose**: Gathering information from stakeholders to understand their needs and expectations from the software.
- **Methods**:
    - **Interviews**: Directly questioning stakeholders to understand their needs.
    - **Surveys and Questionnaires**: Collecting information from a broader group of stakeholders in a structured manner.
    - **Workshops and Brainstorming**: Group activities to encourage discussion and clarify requirements.
    - **Observation**: Watching how users interact with current systems to discover implicit requirements.
    - **Prototyping**: Developing a working model to gather feedback and refine requirements.
- **Challenges**:
    - Communicating with non-technical stakeholders.
    - Identifying conflicting requirements from different stakeholders.
    - Discovering implicit (unstated) requirements.

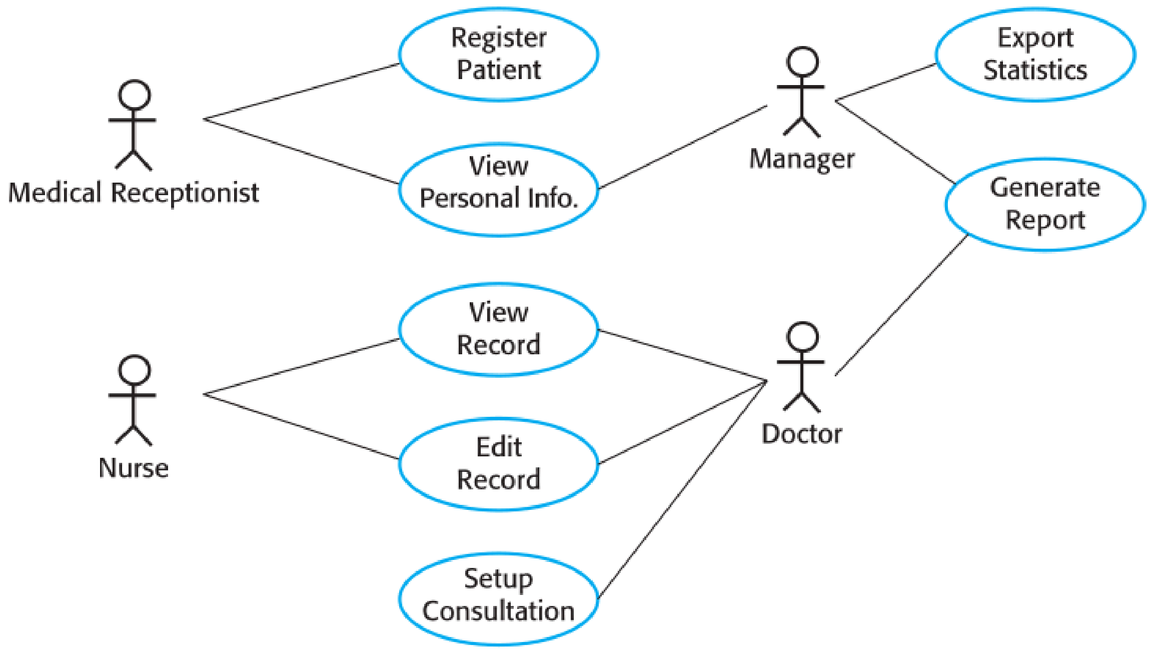

1. Requirements Discovery
2. Requirements Classification and Organization
3. Requirements Prioritization and Negotiation
4. Requirements Specification

2. Requirements Specification



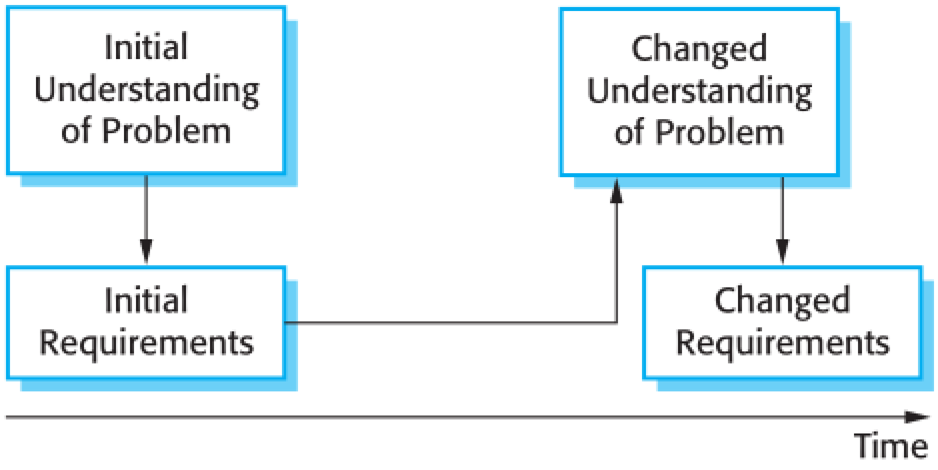
Notation	Description
Natural language sentences	The requirements are written using numbered sentences in natural language. Each sentence should express one requirement.
Structured natural language	The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement.
Design description languages	This approach uses a language like a programming language, but with more abstract features to specify the requirements by defining an operational model of the system. This approach is now rarely used although it can be useful for interface specifications.
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML use case and sequence diagrams are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don't understand a formal specification. They cannot check that it represents what they want and are reluctant to accept it as a system contract.

- **Purpose**: Converting gathered requirements into a formal document that can be used throughout the software development lifecycle.
- **Key Documents**:
  - **Software Requirements Specification (SRS)**: A detailed document outlining functional and non-functional requirements.
  - **Use Case Diagrams**: Visual representations of interactions between users and the system.
  - **User Stories**: Simple descriptions of a feature from the user's perspective (common in Agile environments).
  - **Process Models**: Diagrams like flowcharts or activity diagrams to depict the steps in a business process.
- **Types of Requirements**:
  - **Functional Requirements**: What the system should do (e.g., login, data processing).
  - **Non-Functional Requirements**: How the system should behave (e.g., performance, security, scalability).



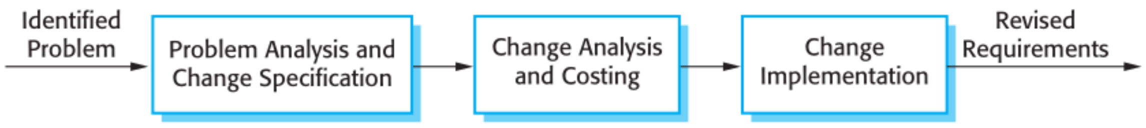
3. Requirements Validation and Verification

- **Purpose**: Ensuring that the requirements accurately reflect the needs of stakeholders and are feasible to implement.
- **Activities**:
  - **Reviews**: Conducting stakeholder and team reviews to ensure completeness and correctness.
  - **Prototyping**: Building mock-ups or simple implementations to validate key requirements.
  - **Test Cases**: Creating test cases based on requirements to verify that they can be validated later.
- **Challenges**:
  - Incomplete or ambiguous requirements leading to misunderstandings.
  - Detecting inconsistencies and conflicts in the early stages.


Initial Understanding of Problem
Changed Understanding of Problem
Initial Requirements
Changed Requirements
Time

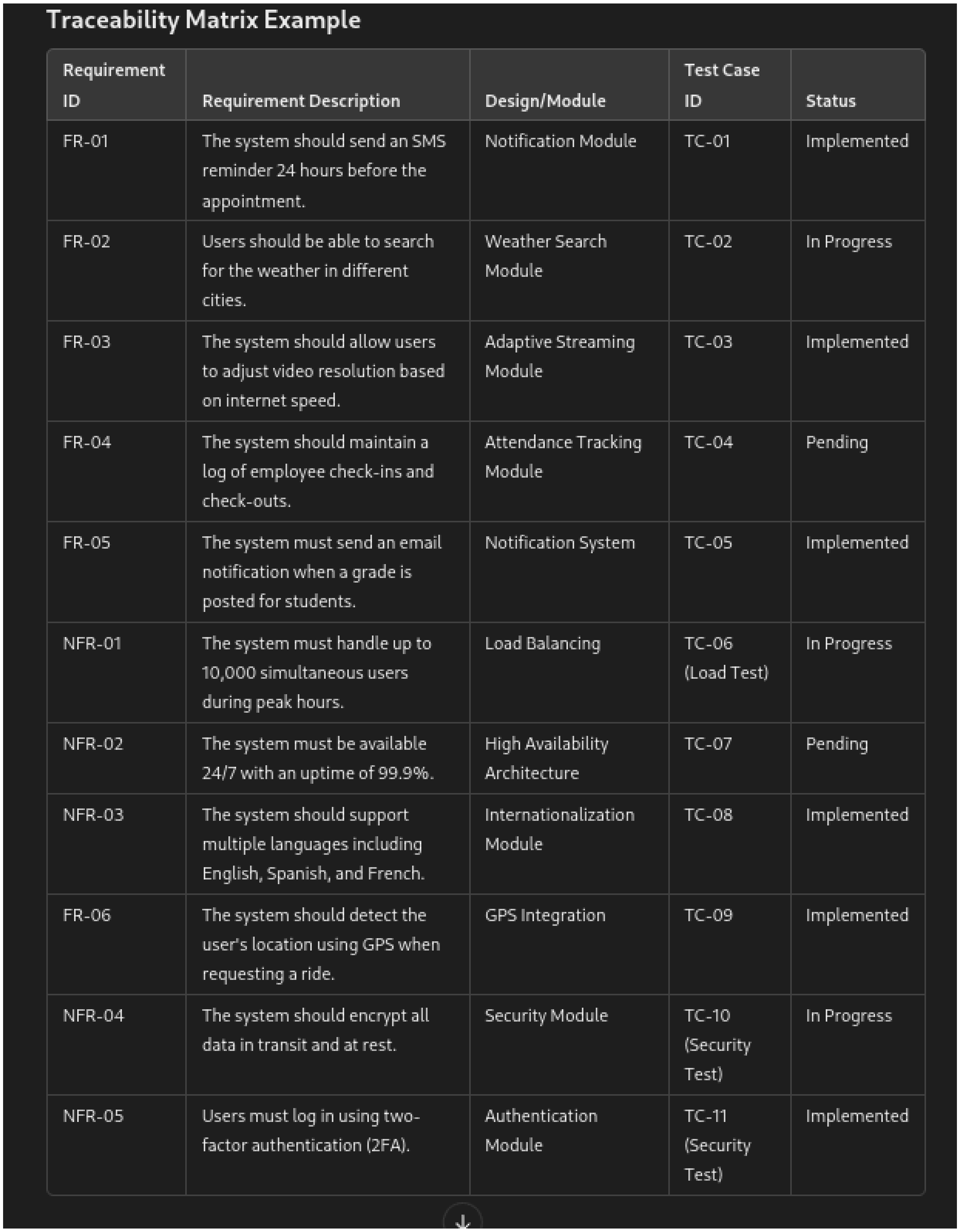
4. Requirements Management

- **Purpose**: Handling the evolution of requirements throughout the software development lifecycle.
- **Activities**:
    - **Version Control**: Tracking changes made to the requirements.
    - **Traceability**: Ensuring that each requirement can be traced back to its origin and is linked to corresponding design, code, and test cases.
    - **Change Management**: Defining processes for handling changes in requirements due to evolving business needs or market conditions.
- **Challenges**:
    - Handling changing requirements in Agile and dynamic environments.
    - Keeping documentation updated as the software evolves.
    - Maintaining alignment between stakeholder expectations and actual system capabilities.


Identified Problem
Problem Analysis and Change Specification
Change Analysis and Costing
Change Implementation
Revised Requirements

Traceability Matrix

A traceability matrix helps in mapping the relationship between different types of requirements (such as user requirements, system requirements, design, and test cases) to ensure that each requirement is adequately covered and validated throughout the software development process. Below is a basic example of a traceability matrix:



Traceability Matrix Example

Requirement ID	Requirement Description	Design/Module	Test Case ID	Status
FR-01	The system should send an SMS reminder 24 hours before the appointment.	Notification Module	TC-01	Implemented
FR-02	Users should be able to search for the weather in different cities.	Weather Search Module	TC-02	In Progress
FR-03	The system should allow users to adjust video resolution based on internet speed.	Adaptive Streaming Module	TC-03	Implemented
FR-04	The system should maintain a log of employee check-ins and check-outs.	Attendance Tracking Module	TC-04	Pending
FR-05	The system must send an email notification when a grade is posted for students.	Notification System	TC-05	Implemented
NFR-01	The system must handle up to 10,000 simultaneous users during peak hours.	Load Balancing	TC-06 (Load Test)	In Progress
NFR-02	The system must be available 24/7 with an uptime of 99.9%.	High Availability Architecture	TC-07	Pending
NFR-03	The system should support multiple languages including English, Spanish, and French.	Internationalization Module	TC-08	Implemented
FR-06	The system should detect the user's location using GPS when requesting a ride.	GPS Integration	TC-09	Implemented
NFR-04	The system should encrypt all data in transit and at rest.	Security Module	TC-10 (Security Test)	In Progress
NFR-05	Users must log in using two-factor authentication (2FA).	Authentication Module	TC-11 (Security Test)	Implemented

Explanation of the Columns:

- **Requirement ID:** A unique identifier for each functional (FR) or non-functional (NFR) requirement.
- **Requirement Description:** A brief summary of the requirement.
- **Design/Module:** The specific design component or module that implements the corresponding requirement.
- **Test Case ID:** The ID of the test case that validates the requirement.
- **Status**: The current status of the requirement implementation (e.g., "Implemented", "In Progress", "Pending").

This traceability matrix helps ensure that every requirement has been mapped to a design component and is being validated by a corresponding test case. It ensures traceability from requirements through to testing, which is crucial for verifying completeness and quality assurance.

5. Types of Requirements

- **Business Requirements**: High-level requirements that capture the goals of the organization for the system.
- **User Requirements**: Specifications from the user's perspective that define how the system should support user tasks.
- **System Requirements**: Detailed descriptions of the functionalities that the system must provide, including hardware, software, and network requirements.

6. Importance of Requirements Engineering

- **Minimizing Project Risks**: Well-defined requirements reduce the chances of scope creep, budget overruns, and project delays.
- **Improved Communication**: A thorough requirements document helps bridge the gap between stakeholders (business and technical teams).
- **Higher Quality Software**: Clear requirements lead to better design, implementation, and testing, which results in fewer defects and higher customer satisfaction.
- **Cost Control**: Requirements management helps in controlling cost by identifying changes early in the process and ensuring that they are justified.

7. Tools Used in Requirements Engineering

- **JIRA/Confluence**: Used for tracking requirements, creating user stories, and managing Agile projects.
- **IBM Rational DOORS**: A tool for capturing, tracking, analyzing, and managing changes to requirements.
- **Enterprise Architect**: Used for creating and managing requirements, especially with UML and modeling approaches.

- **Microsoft Excel/Word**: Simpler tools for documenting requirements in small projects.
- **ReqView**: A tool for managing structured requirements and traceability.

8. Challenges in Requirements Engineering

- **Ambiguity and Incompleteness**: Requirements may be vague or incomplete, leading to misinterpretation.
- **Changing Requirements**: Stakeholders' needs may evolve, causing the requirements to change, which can lead to scope creep.
- **Stakeholder Conflicts**: Different stakeholders may have conflicting priorities or needs, which need to be reconciled.
- **Technical Constraints**: Certain requirements may be impossible or too costly to implement due to technical limitations.

9. Best Practices in Requirements Engineering

- **Involve Stakeholders Early and Often**: Continuous collaboration with stakeholders ensures that requirements are well-understood and agreed upon.
- **Use Models and Visualizations**: Diagrams, flowcharts, and mock-ups help clarify and communicate requirements more effectively.
- **Focus on Clear, Testable Requirements**: Write requirements that can be validated through testing to ensure the end product meets expectations.
- **Iterate and Refine**: Review and refine requirements throughout the development process, especially in Agile environments.
- **Prioritize Requirements**: Not all requirements have the same importance. Use techniques like MoSCoW (Must have, Should have, Could have, Won't have) to prioritize them.

Requirements engineering is essential for creating software that satisfies the needs of users and stakeholders. By carefully eliciting, specifying, validating, and managing requirements, software engineers can reduce project risks, improve communication between teams, and deliver high-quality software products. Proper RE processes ensure that the software aligns with business objectives and provides real value to users.

END