Software Specification

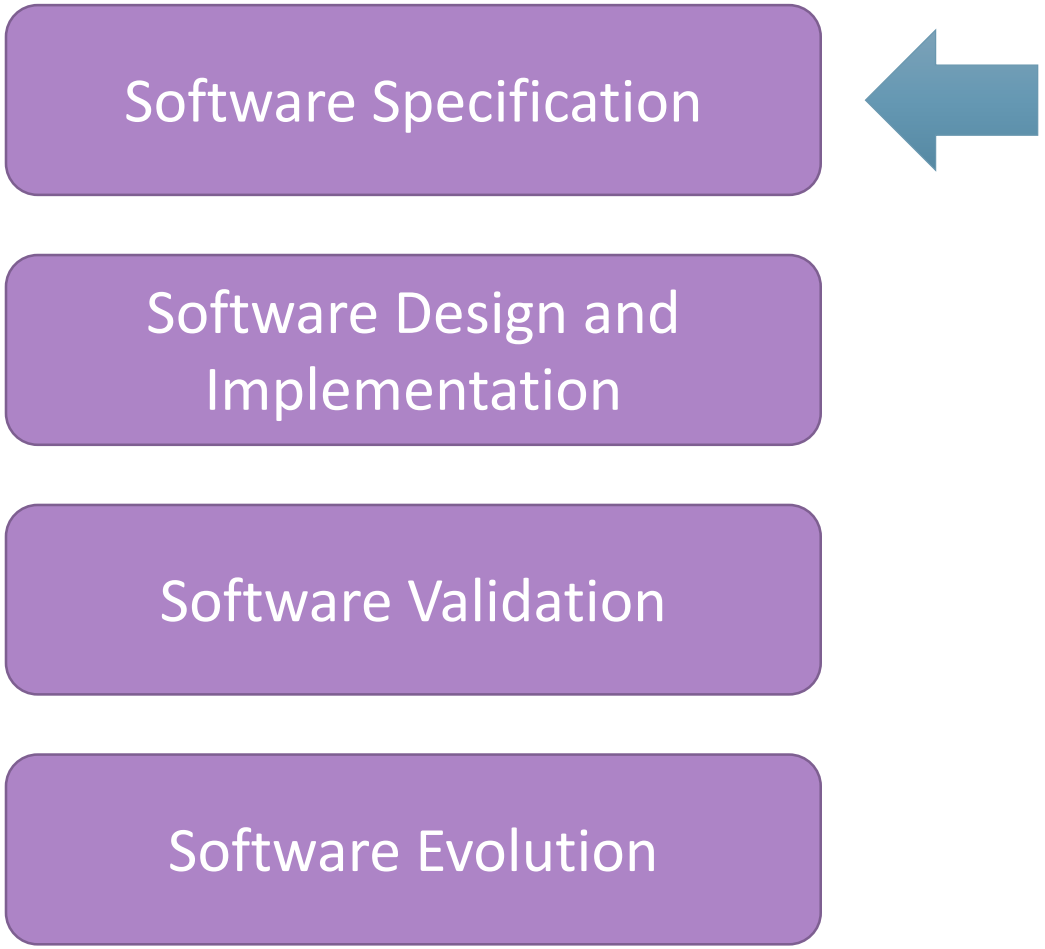
Software Engineering 1

Soon Phei Tin

Objectives

- The objective of this chapter is to introduce software specification and to discuss the processes involved in discovering and documenting these requirements. When you have read the chapter, you will:
- understand the concepts of user and system requirements and why these requirements should be written in different ways;
- Be able to specify functional and nonfunctional software requirements;
- Be able to organize and document software requirements;
- understand the principal software specification activities of elicitation, analysis and validation, and the relationships between these activities;

Fundamental Software Process Activities



Introduction

- **Definition**: Software specification is the process of detailing the functionalities, constraints, and interactions of a software system. It serves as a blueprint for the design and development phases.
- **Purpose**: To ensure that all stakeholders have a clear understanding of what the software system is supposed to do and how it will perform.
- **Output**: Software specification will produce Software Requirements Document which acts as an input to the next activity of the process.

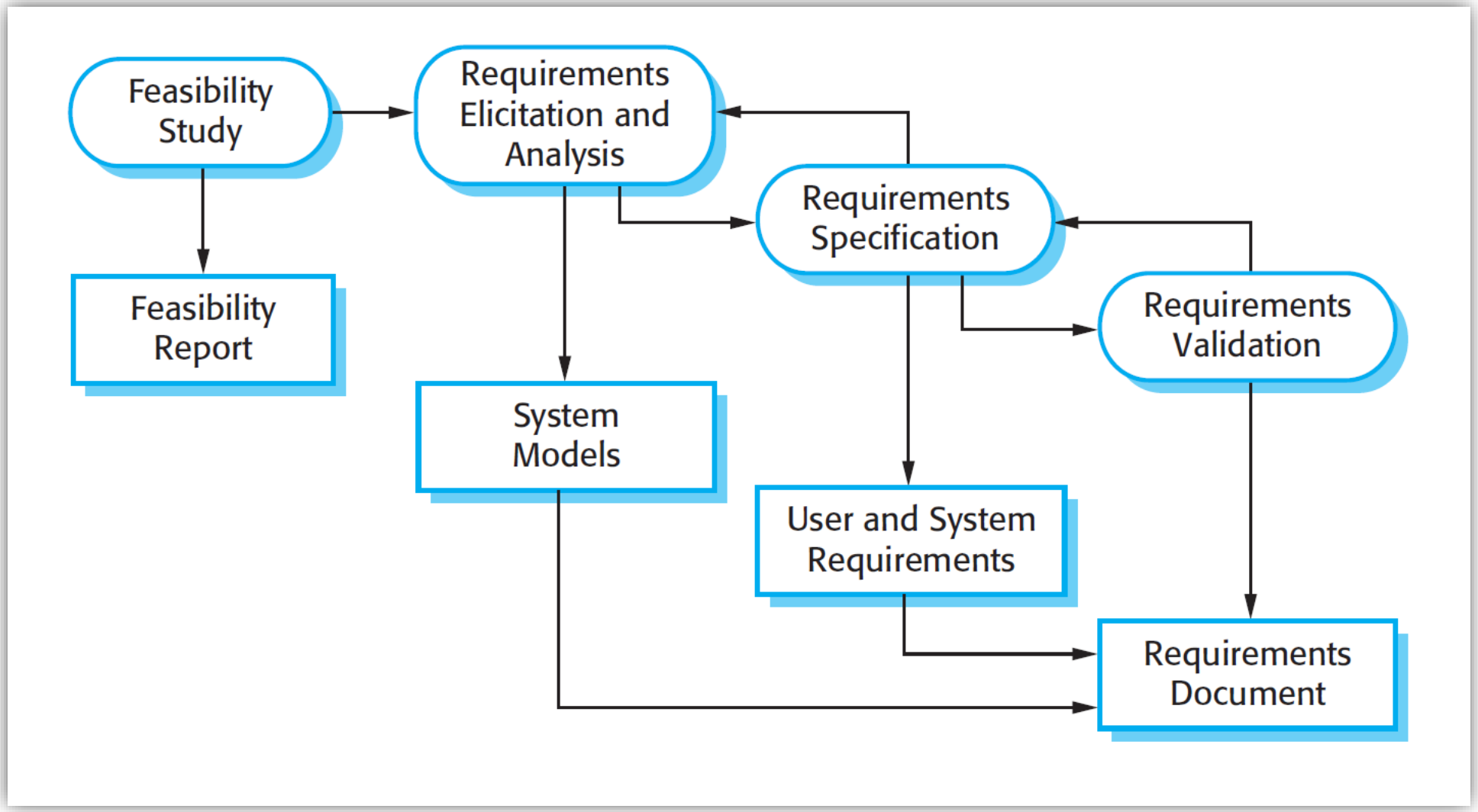
Introduction

- The requirements for a system are the descriptions of what the system should do
  - reflect the needs of customers for a system
- The process of finding out, analyzing, documenting and checking these needs and constraints is called software specification (sometimes requirement engineering)

Importance of Software Specification

- **Clarity and Understanding**: Provides a clear and detailed description of system requirements for all stakeholders.
- **Guidance for Development**: Acts as a roadmap for developers, ensuring that the system is built according to specified requirements.
- **Basis for Testing**: Provides a reference for creating test cases and validating the system against requirements.
- **Risk Reduction**: Identifies potential issues and ambiguities early in the development process, reducing the risk of costly changes later.

Software Specification Processes



Software Specification Processes

- Four high-level activities
  - Feasibility study: To determine whether the proposed system is technically, economically, and operationally feasible.
    - **Outcome**: A feasibility report that provides a recommendation on whether to proceed with the project.
  - Requirements elicitation and analysis: To gather, analyze, and refine the requirements for the software system.
    - **Outcome**: A comprehensive set of requirements that have been analyzed and refined to ensure clarity, completeness, and feasibility.

Software Specification Processes

- Four high-level activities
  - Requirements specification: To document the gathered and analyzed requirements in a clear and structured format.
    - **Outcome**: A detailed Software Requirements Document (SRD) that provides a clear and comprehensive description of the system’s functionalities and constraints.
  - Requirements validation: To ensure that the documented requirements accurately reflect the needs and expectations of stakeholders.
    - **Outcome**: Validated requirements that have been reviewed and approved by stakeholders, ensuring they accurately reflect the desired system.

Software Specification Processes

iterative process in which the activities are interleaved.

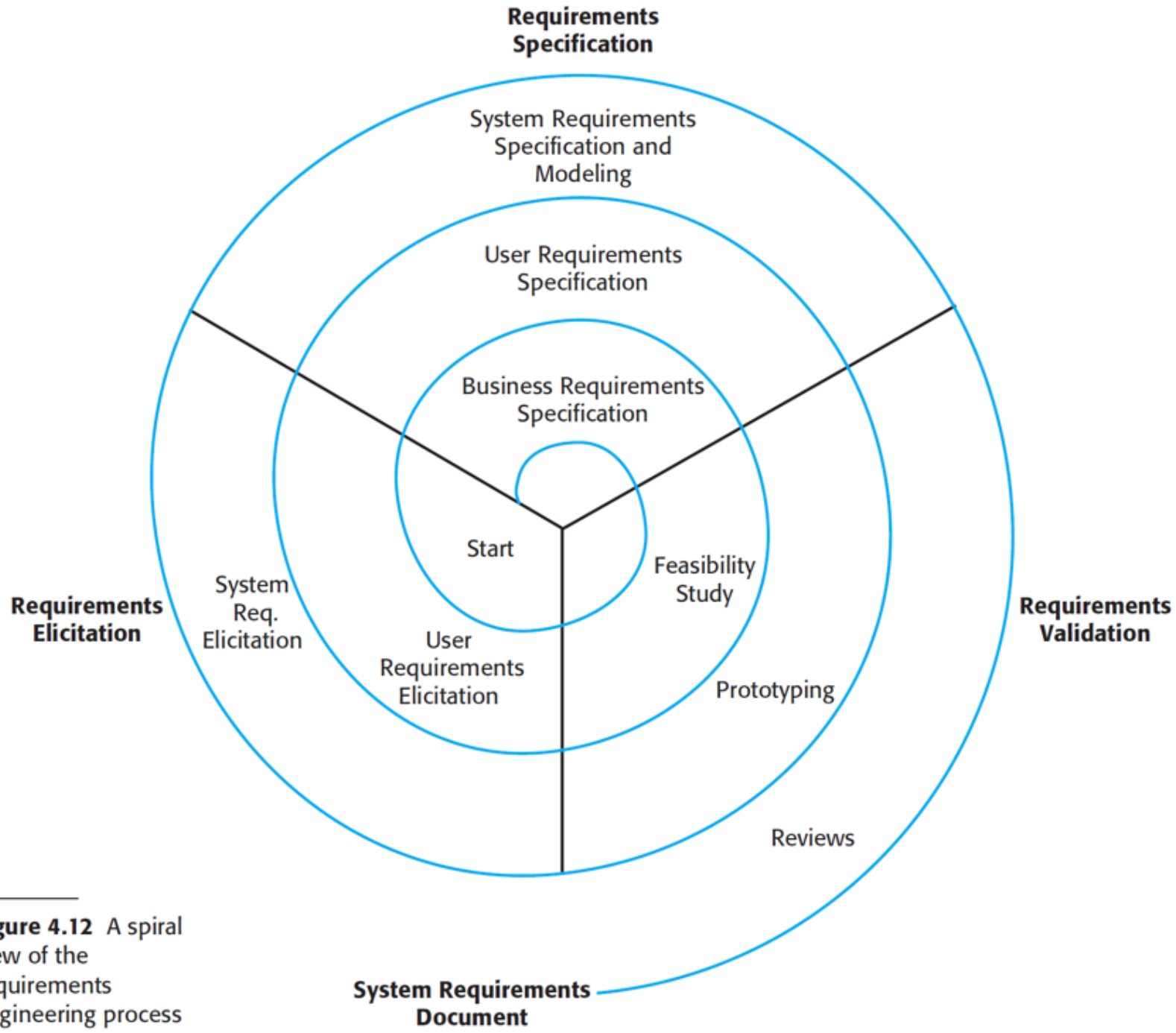


Figure 4.12 A spiral view of the requirements engineering process

Road Map

- *User requirements and system requirements*
- *Functional requirements and non-functional requirements*
- **Feasibility study**
- **Requirements elicitation and analysis**
- **Requirements specification**
- **Requirements validation**

Clarity in Communication

- The requirements can be described in a high-level, abstract statement of a service that a system should provide or a constraint on a system. At the other extreme, it is a detailed, formal definition of a system function
- When should an engineer write the requirements in high-level description and when in detail description?
  - In high-level description when communicating with non-technical personnel such as end-users and business stakeholders
  - In detail description when communicating with technical personnel such as software developers, architects, and technical stakeholders

User requirements and system requirements

- **What are User Requirements?**
- **Definition**: User requirements describe what the users need from the system. They focus on the functionalities and features that will enable users to achieve their goals.
- **Characteristics**:
  - **User-Centric**: Emphasize the needs and perspectives of the end-users.
  - **Functional**: Describe what the system should do in **high-level**.
  - **Non-Technical**: Typically written in a language understandable by users and stakeholders.

Examples of User Requirements

Requirements should state ***what*** the system should do instead of ***how*** it should be done

- **E-commerce Platform**:
  - Users should be able to search for products.
  - Users should be able to add products to a shopping cart.
  - Users should be able to complete a purchase using various payment methods.
- **Social Media Application**:
  - Users should be able to create and edit profiles.
  - Users should be able to post updates and share content.
  - Users should be able to send and receive messages.

Role of Users in Defining User Requirements

- **Primary Contributors**: Users are the primary contributors to user requirements as they best understand their own needs and how the system should support their tasks.
- **Techniques for Gathering User Requirements**:
  - **Interviews**: Conduct one-on-one interviews with users to gather detailed insights.
  - **Surveys and Questionnaires**: Use surveys to gather input from a larger audience.
  - **Workshops and Focus Groups**: Facilitate collaborative discussions to gather diverse perspectives.
  - **Observation**: Observe users in their natural environment to understand their workflows and challenges.
  - **User Stories and Use Cases**: Document requirements in the form of user stories and use cases to capture user interactions with the system.

User requirements and system requirements

- **What are System Requirements?**
- **Definition**: System requirements describe the technical specifications and constraints of the system. They focus on refining user requirements by adding technical details.
- **Characteristics**:
  - **System-Centric**: Emphasize the architecture, design, and technical details of the system.
  - **Functional and Non-Functional**: Include both what the system should do and how it should perform.
  - **Technical**: Written in a language understandable by developers and technical stakeholders.

User Requirement Definition

1. The MHC-PMS shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System Requirements Specification

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost, and the prescribing clinics shall be generated.

1.2 The system shall automatically generate the report for printing after 17.30 on the last working day of the month.

1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed, and the total cost of the prescribed drugs.

1.4 If drugs are available in different dose units (e.g., 10 mg, 20 mg) separate reports shall be created for each dose unit.

1.5 Access to all cost reports shall be restricted to authorized users listed on a management access control list.

Role of Engineers in Defining System Requirements

- **Primary Contributors**: Engineers (including software developers, system architects, and technical experts) are the primary contributors to system requirements as they have the technical expertise to translate user needs into technical specifications.
- **Techniques for Defining System Requirements**:
  - **Requirements Analysis**: Analyze user requirements to derive system requirements.
  - **Technical Workshops**: Conduct workshops with technical stakeholders to define system specifications.
  - **Modeling and Diagrams**: Use modeling techniques (e.g., UML, ER diagrams) to represent system requirements.
  - **Prototyping**: Create prototypes to validate system requirements and design choices.
  - **Traceability**: Ensure traceability between user requirements and system requirements to maintain alignment.

In most cases, Software Requirements Document includes both User Requirements and System Requirements

Question

Which of the following is true?

- User requirements can be further devised into system requirements.
- Engineers collect system requirements before they discovered the related user requirements.
- User requirements are defined by the users, while system requirements are defined by the engineers.
- User requirements are more general, while system requirements are more details and specific.

Functional & Non-functional requirements

- Software requirements are often classified as functional requirements or nonfunctional requirements:
  - Functional requirements: describe the specific behaviors and functions that a system must perform. They detail what the system should do to meet the needs of its users.
  - Non-functional requirements: describe the system’s operational qualities and constraints. They detail how the system should perform rather than what it should do. Non-functional requirements often apply to the system as a whole, rather than individual system features or services.

Functional Requirements

- When expressed as *user requirements*, functional requirements are usually described in an abstract way that can be understood by system users.
- More specific *functional system requirements* describe the system functions, its inputs and outputs, exceptions, etc., in detail.

Examples

- Sample requirement for MHC-PMS system:
  1. A user shall be able to search the appointments lists for all clinics.
  2. The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day.
  3. Each staff member using the system shall be uniquely identified by his or her eight-digit employee number.

Functional Requirements

- The functional requirements specification of a system should be both complete and consistent.
  - Completeness means that all services required by the user should be defined.
  - Consistency means that requirements should not have contradictory definitions.
- In practice, for large, complex systems, it is practically impossible to achieve requirements consistency and completeness.
  - it is easy to make mistakes and omissions when writing specifications for complex systems
  - there are many stakeholders in a large system. Stakeholders have different and often inconsistent needs.

Characteristics of Functional Requirements

- **Behavioral**: Focus on the actions and operations that the system must perform.
- **User-Centric**: Often derived from user needs and use cases.
- **Specific and Measurable**: Clearly defined and can be tested for correctness.

Question

- Which of the following is true?
  - Functional requirements = user requirements.
  - Functional requirements specify the services or operations a system shuold provides to its users.
  - Functional requirements has to be described in general term.

Non-functional Requirements

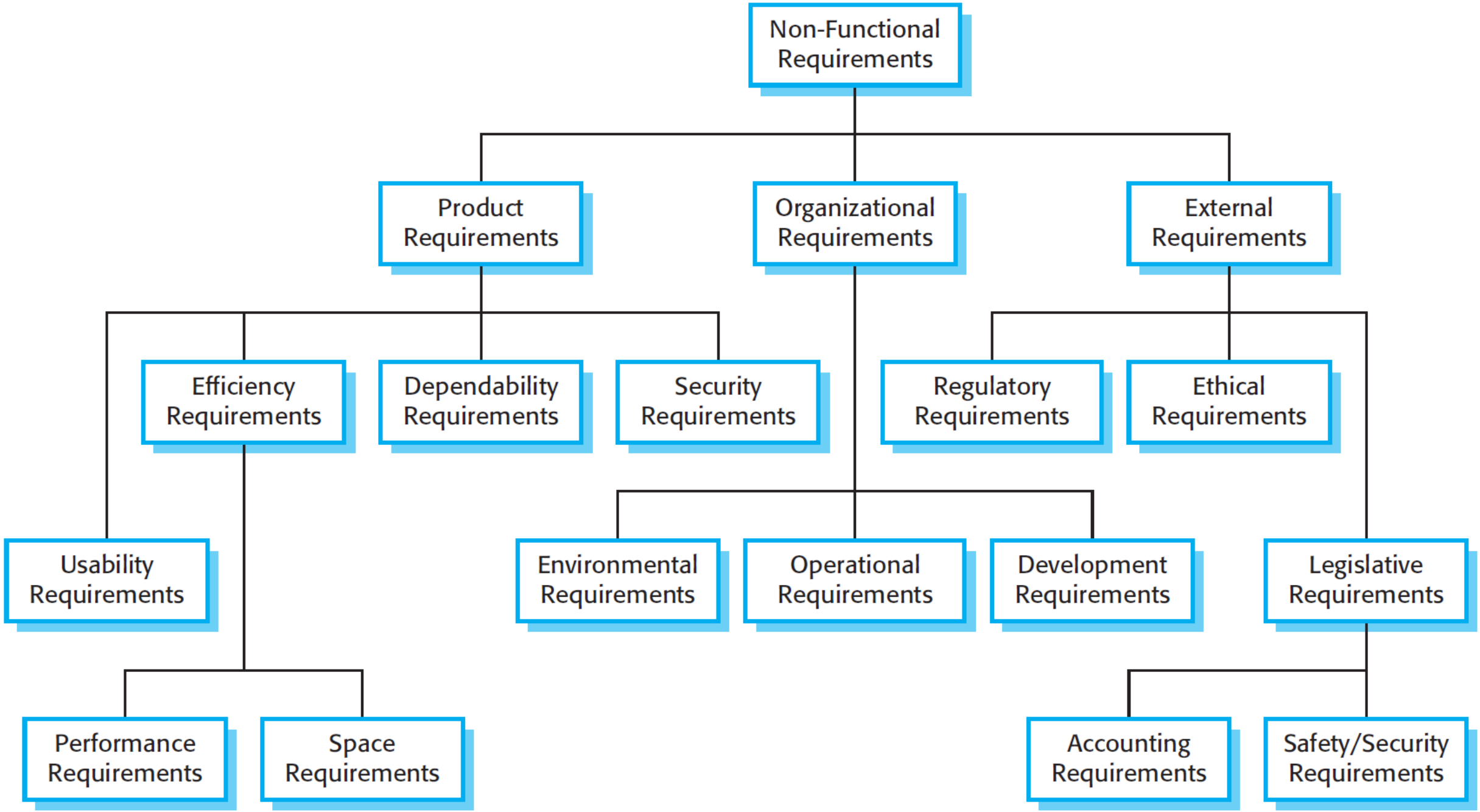
- Non-functional requirements, as the name suggests, are requirements that are not directly concerned with the specific service or operation delivered by the system to its users.
- System properties such as reliability, response time, scalability, security, or availability.
- Usually specify constrain characteristics of the system as a whole

Non-functional Requirements

- Often more critical than individual functional requirements
- failing to meet a non-functional requirement can mean that the whole system is unusable
- The implementation of non-functional requirements may be intricately dispersed throughout the system.
  - They may affect the overall architecture of a system rather than the individual components.
  - A single non-functional requirement may generate a number of related functional requirements.

Characteristics of Non-functional Requirements

- **Quality Attributes**: Focus on performance, security, usability, reliability, scalability, and other attributes.
- **System-Centric**: Often derived from technical and business constraints.
- **Measurable**: Can be quantified and tested for compliance.


Non-Functional Requirements
Product Requirements
Organizational Requirements
External Requirements
Efficiency Requirements
Dependability Requirements
Security Requirements
Regulatory Requirements
Ethical Requirements
Usability Requirements
Environmental Requirements
Operational Requirements
Development Requirements
Legislative Requirements
Performance Requirements
Space Requirements
Accounting Requirements
Safety/Security Requirements

Classification of Non-functional Requirements

- *Product requirements* - These requirements specify or constrain the behavior of the software.
- *Organizational requirements* - These requirements are broad system requirements derived from policies and procedures in the customer's and developer's organization.
- *External requirements* - This broad heading covers all requirements that are derived from factors external to the system and its development process

Examples

PRODUCT REQUIREMENT
The MHC-PMS shall be available to all clinics during normal working hours (Mon–Fri, 08.30–17.30). Downtime within normal working hours shall not exceed five seconds in any one day.

ORGANIZATIONAL REQUIREMENT
Users of the MHC-PMS system shall authenticate themselves using their health authority identity card.

EXTERNAL REQUIREMENT
The system shall implement patient privacy provisions as set out in HStan-03-2006-priv.

Non-functional Requirements - Testability

- A common problem with non-functional requirements is that users or customers often propose these requirements as general goals, such as …

The system should be easy to use by medical staff and should be organized in such a way that user errors are minimized.

vs

Medical staff shall be able to use all the system functions after four hours of training. After this training, the average number of errors made by the trained users shall not exceed two per hour of system use.

Non-functional Requirements - Testability

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Characteristics of Testable Non-Functional Requirements

- **Specific**: Clearly defines what is expected.
- **Measurable**: Provides metrics that can be quantified.
- **Achievable**: Realistic and attainable within the project constraints.
- **Relevant**: Directly related to the system’s goals and objectives.
- **Context**: Specifies the context where the requirement should be met.

Example of a Testable Non-Functional Requirement

- **Performance**: “The system should process 1,000 transactions per second with a response time of less than 2 seconds under peak load conditions.”
- **Testability**: This requirement is specific, measurable, achievable, context, and can be tested using performance testing tools.

Example of a Non-Testable Non-Functional Requirement

- **Usability**: “The system should be user-friendly and intuitive.”
- **Testability**: This requirement is vague and subjective. It does not provide specific criteria or metrics that can be tested.
  - **Vagueness**: The term “user-friendly and intuitive” is subjective and open to interpretation. Different users may have different opinions on what constitutes user-friendliness.
  - **Lack of Specific Metrics**: The requirement does not provide specific, measurable criteria that can be used to evaluate usability.
  - **Ambiguity**: The requirement is not clear about what aspects of the system should be user-friendly or how intuitiveness should be measured.

Improved, Testable Non-Functional Requirement

- **Original Statement**: “The system should be user-friendly and intuitive.”
- **Refined Statement**: “The system should achieve a System Usability Scale (SUS) score of at least 80% based on user feedback surveys conducted with a sample size of at least 50 users.”
  - **Specific**: Defines the use of usability metric (SUS score), user feedback surveys.
  - **Measurable and achievable**: Provides a target score (80%).
  - **Context**: Sample size of at least 50 users.

Improved, Testable Non-Functional Requirement

- **Original Statement**: “The system should be user-friendly and intuitive.”
- **Refined Statement**: “Users should be able to complete the primary tasks (e.g., placing an order) within 3 minutes on their first attempt, with no more than two errors.”
  - **Specific**: Complete the primary tasks (e.g., placing an order).
  - **Measurable and achievable**: Specify the limit of 3 minutes and no more than 2 errors.
  - **Context**: Complete the primary tasks on their first attempt.

Emergent System Properties

- **Definition**: Emergent system properties are characteristics that arise from the interactions and collective behavior of the system components. These properties are not attributable to any single component but emerge from the system as a whole.
- **Examples**:
  - **Performance**: Overall speed and efficiency of the system.
  - **Reliability**: The system’s ability to function correctly over time and under varying conditions.
  - **Security**: The system’s ability to protect data and resist attacks.
  - **Scalability**: The system’s ability to handle increased load without degrading performance.

Importance of Highlighting Emergent System Properties

- **Highlighting Non-Functional Requirements Related to Emergent System Properties**
  - **System-Wide Impact**: Emergent properties affect the entire system and are critical to its overall success.
  - **Visibility and Awareness**: Explicitly highlighting these requirements ensures that all stakeholders are aware of their importance and impact.
  - **Design Considerations**: Non-functional requirements related to emergent properties often influence architectural and design decisions.
  - **Testing and Validation**: These requirements need specific testing strategies to ensure that the system meets the desired performance and reliability standards.
  - **Risk Management**: Identifying and addressing these requirements early helps mitigate risks associated with system performance and reliability.

Feasibility Study

- **Technical Feasibility**:
  - **Assessment**: Evaluate the technology stack, technical resources, and skills required.
  - **Challenges**: Identify potential technical challenges and risks.
  - **Outcome**: Determine if the project is technically achievable.

- **Economic Feasibility**:
  - **Cost-Benefit Analysis**: Compare the estimated costs with the expected benefits.
  - **Budgeting**: Assess the financial resources required and available.
  - **Outcome**: Determine if the project is financially viable.

- **Operational Feasibility**:
  - **Organizational Fit**: Evaluate how well the system fits within the organization's operations.
  - **Support and Maintenance**: Assess the organization's ability to support and maintain the system.
  - **Outcome**: Determine if the project is operationally feasible.

Requirements Elicitation and Analysis

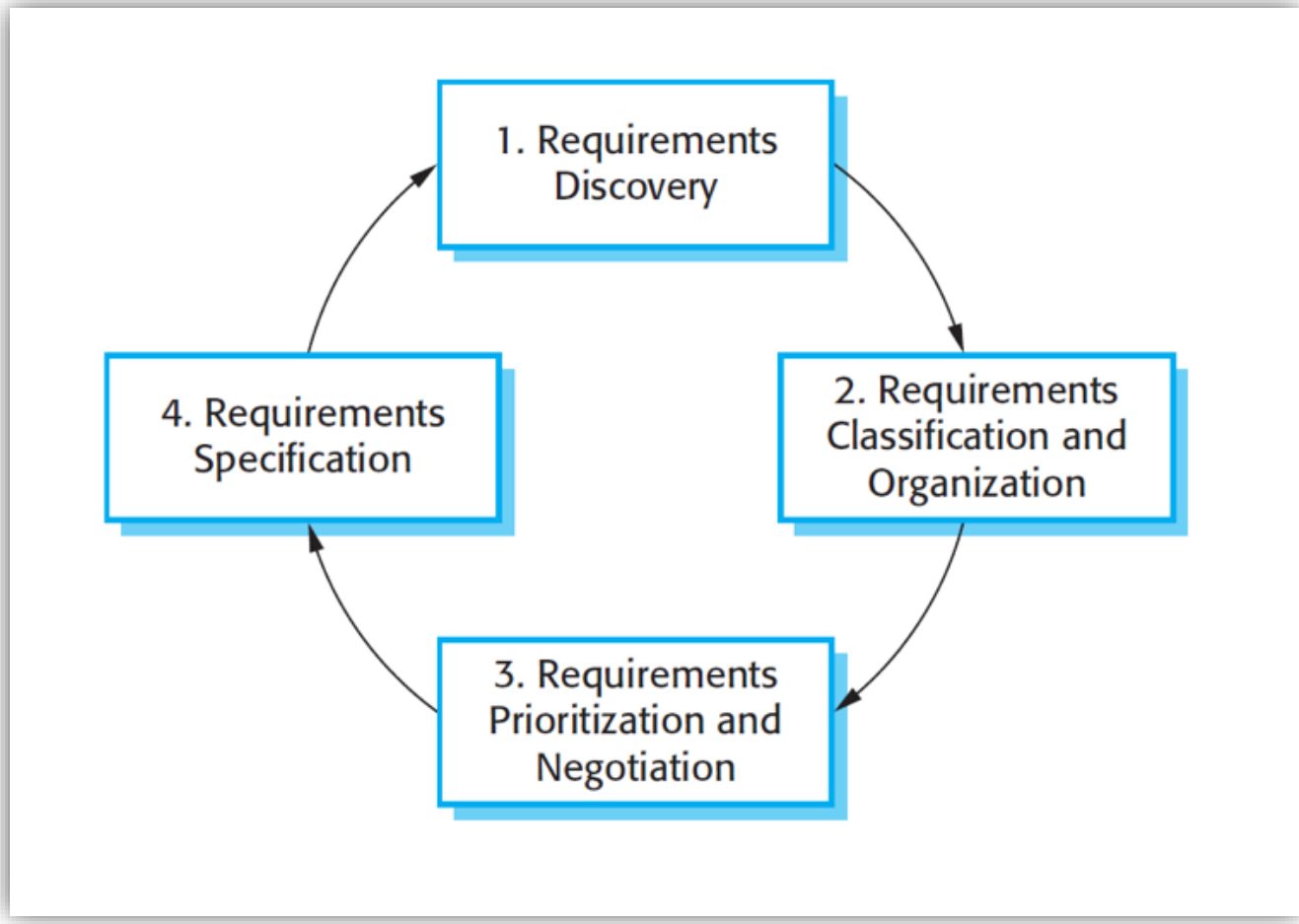
- Software engineers work with customers and system end-users to find out about
  - the application domain
  - what services the system should provide
  - the required performance of the system
  - hardware constraints, and so on.

Requirements Elicitation and Analysis

- may involve a variety of different kinds of people in an organization include: -
  - end users who will interact with the system
  - anyone else in an organization who will be affected by it
  - engineers who are developing or maintaining other related systems
  - business managers
  - domain experts
  - trade union representatives.

Requirements Elicitation and Analysis

- Requirements elicitation and analysis is an iterative process with continual feedback from each activity to other activities



Requirements Elicitation and Analysis

- Requirements discovery - This is the process of interacting with stakeholders of the system to discover their requirements
- Requirements classification and organization - This activity takes the unstructured collection of requirements, groups related requirements, and organizes them into coherent clusters.
  - The most common way of grouping requirements is to use a model of the system architecture to identify sub-systems and to associate requirements with each sub-system

Requirements Elicitation and Analysis

- Requirements prioritization and negotiation - When multiple stakeholders are involved, requirements will conflict. This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiation.
- Requirements specification - The requirements are documented and input into the next round of the spiral.

Requirements Elicitation and Analysis
Challenges

- Eliciting and understanding requirements from system stakeholders is a difficult process for several reasons:
  - Stakeholders often don’t know what they want from a computer system except in the most general terms; they may find it difficult to articulate the requirements; they may make unrealistic demands because they don’t know what is and isn’t feasible.
  - Stakeholders in a system express requirements in their own terms and with implicit knowledge of their own work.

Requirements Elicitation and Analysis
Challenges

- Eliciting and understanding requirements from system stakeholders is a difficult process for several reasons:
  - Different stakeholders have different requirements and priorities, and they may express these in different ways, some of these requirements have commonalities and conflict.
  - Political factors may influence the requirements of a system.
  - The economic and business environment in which the analysis takes place is dynamic. It changes during the analysis process.

Requirements Elicitation and Analysis

Requirements Discovery

- Requirements discovery is the process of gathering information about the required system and existing systems, and distilling the user and system requirements from this information.
- Sources of information during the requirements discovery phase include documentation, stakeholders, and specifications of similar systems.
  - interviews
  - observation
  - scenarios
  - prototypes

Requirements Elicitation and Analysis

Requirements Discovery

- Stakeholders range from end-users of a system through managers to external stakeholders such as regulators
- Stakeholders for the mental healthcare patient information system include:
  - Patients
  - Doctors
  - Nurses
  - Medical receptionists
  - IT staff
  - Medical ethics manager
  - Healthcare managers
  - Medical records staff

requirements may also come from the application domain and from other systems that interact with the system

Requirements Elicitation and Analysis

Requirements Discovery - Interview

- Formal or informal interviews with system stakeholders
- The requirements engineering team puts questions to stakeholders about the system that they currently use and the system to be developed.
- Requirements are derived from the answers to these questions.
- Interviews may be of two types:
  - Closed interviews - where the stakeholder answers a pre-defined set of questions.
  - Open interviews - no pre-defined agenda. The requirements engineering team explores a range of issues with system stakeholders

Requirements Elicitation and Analysis

Requirements Discovery - Interview

- Interviews are good for: -
  - getting an overall understanding of what stakeholders do
  - how they might interact with the new system
  - the difficulties that they face with current systems

Requirements Elicitation and Analysis

Requirements Discovery - Interview

- It can be difficult to elicit domain knowledge through interviews for two reasons:
  - All application specialists use terminology and jargon that are specific to a domain. They normally use terminology in a precise and subtle way that is easy for requirements engineers to misunderstand.
  - Some domain knowledge is so familiar to stakeholders that they either find it difficult to explain or they think it is so fundamental that it isn’t worth mentioning

Requirements Elicitation and Analysis

Requirements Discovery - Interview

- Not an effective technique for eliciting knowledge about organizational requirements and constraints because there are subtle power relationships between the different people in the organization.
  - most people are generally reluctant to discuss political and organizational issues that may affect the requirements

Requirements Elicitation and Analysis

Requirements Discovery - Interview

- Effective interviewers have two characteristics:
- They are open-minded, avoid pre-conceived ideas about the requirements, and are willing to listen to stakeholders.
- They prompt the interviewee to get discussions going using a springboard question, a requirements proposal, or by working together on a prototype system.

Information from interviews supplements other information about the system from documentation describing business processes or existing systems, user observations, etc.

Requirements Elicitation and Analysis

Requirements Discovery - Scenarios

- System user can understand and criticize a scenario of how they might interact with a software system.
- Use the information gained from the discussion based on a scenario to formulate the actual system requirements.
- A scenario is the descriptions of example interaction sessions. Each scenario usually covers one or a small number of possible interactions
- A scenario starts with an outline of the interaction. During the elicitation process, details are added to this to create a complete description of that interaction.

Requirements Elicitation and Analysis

Requirements Discovery – Scenarios

Collecting medical history

INITIAL ASSUMPTION:
The patient has seen a medical receptionist who has created a record in the system and collected the patient's personal information (name, address, age, etc.). A nurse is logged on to the system and is collecting medical history.

NORMAL:
The nurse searches for the patient by family name. If there is more than one patient with the same surname, the given name (first name in English) and date of birth are used to identify the patient.

The nurse chooses the menu option to add medical history.

The nurse then follows a series of prompts from the system to enter information about consultations elsewhere on mental health problems (free text input), existing medical conditions (nurse selects conditions from menu), medication currently taken (selected from menu), allergies (free text), and home life (form).

WHAT CAN GO WRONG:
The patient's record does not exist or cannot be found. The nurse should create a new record and record personal information.

Patient conditions or medication are not entered in the menu. The nurse should choose the 'other' option and enter free text describing the condition/medication.

Patient cannot/will not provide information on medical history. The nurse should enter free text recording the patient's inability/unwillingness to provide information. The system should print the standard exclusion form stating that the lack of information may mean that treatment will be limited or delayed. This should be signed and handed to the patient.

OTHER ACTIVITIES:
Record may be consulted but not edited by other staff while information is being entered.

SYSTEM STATE ON COMPLETION:
User is logged on. The patient record including medical history is entered in the database, a record is added to the system log showing the start and end time of the session and the nurse involved.

Requirements Elicitation and Analysis

Requirements Discovery - Scenarios

- Scenario-based elicitation involves working with stakeholders to identify scenarios and to capture details to be included in these scenarios.
- Scenarios may be written as text, supplemented by diagrams, screen shots, etc.
- Alternatively, a more structured approach such as event scenarios or use cases may be used.

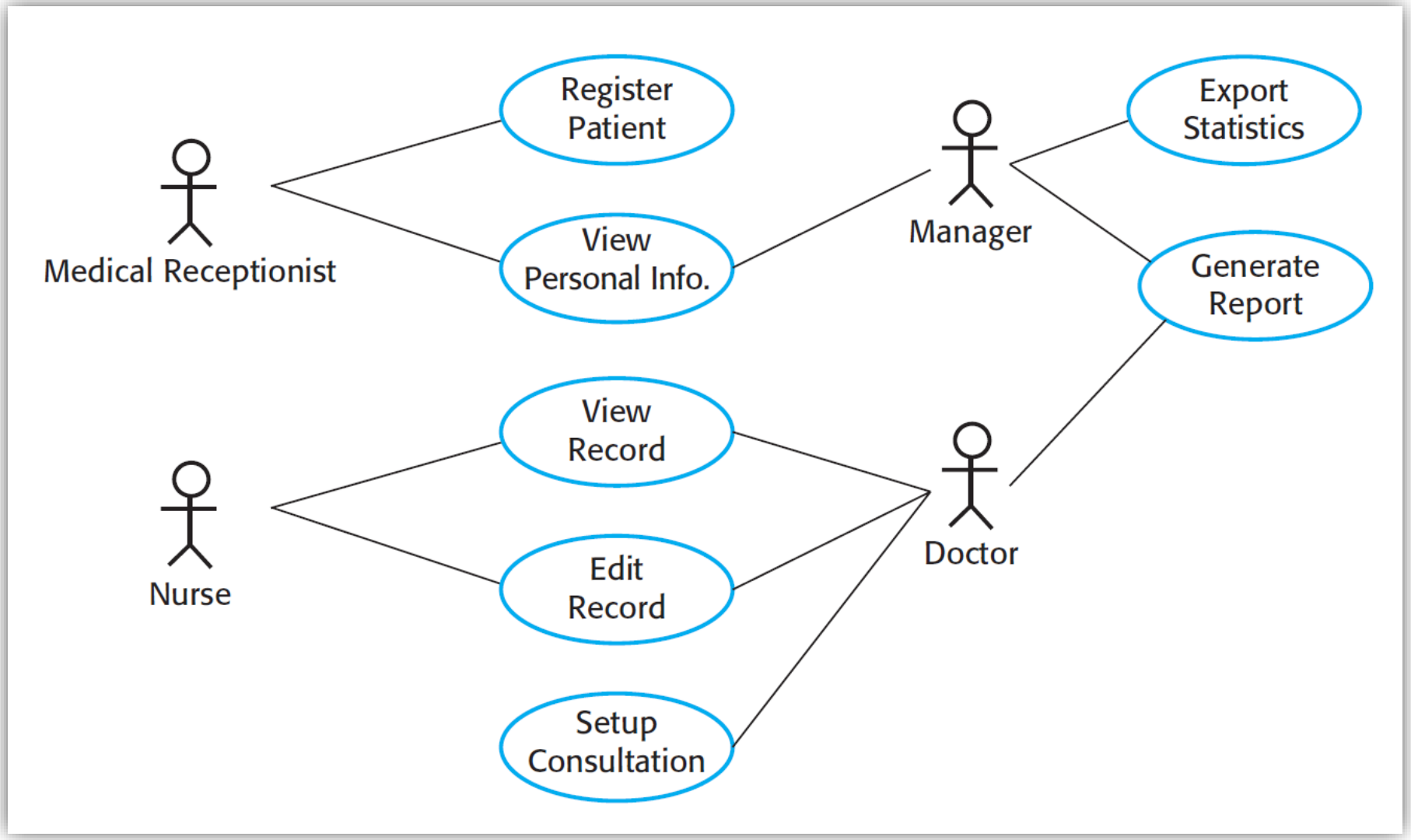
Requirements Elicitation and Analysis

Requirements Discovery – Use cases

- Use cases are a requirements discovery technique. In their simplest form, a use case identifies the actors involved in an interaction and names the type of interaction
- It is supplemented by additional information describing the interaction with the system.
  - Textual description
  - one or more graphical models
- Use cases are documented using a high-level use case diagram. The set of use cases represents all of the possible interactions that will be described in the system requirements.
  - each use case is a single scenario;
  - encapsulate a set of scenarios in a single use case diagram.

Requirements Elicitation and Analysis

Requirements Discovery – Use cases



Requirements Elicitation and Analysis

Requirements Discovery – Use cases

- Use cases identify the individual interactions between the system and its users or other systems.
- Each use case should be documented with a textual description.
- A use case can then be linked to other models in the UML that will develop the scenario in more detail

Requirements Elicitation and Analysis

Requirements Discovery – Use cases

Setup Consultation Use Case

Setup consultation allows two or more doctors, working in different offices, to view the same record at the same time. One doctor initiates the consultation by choosing the people involved from a drop-down menu of doctors who are online. The patient record is then displayed on their screens but only the initiating doctor can edit the record. In addition, a text chat window is created to help coordinate actions. It is assumed that a phone conference for voice communication will be separately set up.

Requirements Elicitation and Analysis

Requirements Discovery – Use cases

- Use case focus on interactions with the system, they are not as effective for eliciting: -
  - constraints
  - high-level business
  - nonfunctional requirements
  - domain requirements.

Requirements Elicitation and Analysis

Requirements Discovery – Ethnography

- Systems are used in a social and organizational context and software system requirements may be derived or constrained by that context
- Satisfying these social and organizational requirements is often critical for the success of the system.
- Ethnography is an observational technique that can be used to understand operational processes and help derive support requirements for these processes.
- An analyst immerses himself or herself in the working environment to observe the actual tasks in which participants are involved.
- Helps discover implicit system requirements that reflect the actual ways that people work

Requirements Elicitation and Analysis

Requirements Discovery – Ethnography

- Ethnography is particularly effective for discovering two types of requirements: -
  - Requirements that are derived from the way in which people actually work, rather than the way in which process definitions say they ought to work
  - Requirements that are derived from cooperation and awareness of other people's activities.
- Ethnography focuses on the end-user, this approach: -
  - is not always appropriate for discovering organizational or domain requirements
  - cannot always identify new features that should be added to a system.

Requirements Specification

- The process of writing down the **user and system requirements** in a requirements document.
- The user and system requirements should be clear, unambiguous, easy to understand, complete, and consistent.
- Stakeholders interpret the requirements in different ways and there are often inherent conflicts and inconsistencies in the requirements.
- The user requirements for a system should describe the **functional and nonfunctional requirements** in a way they are understandable by system users who don’t have detailed technical knowledge

Requirements Specification

- Specify only the external behavior of the system.
- Should not include details of the system architecture or design
- Write user requirements in natural language, with simple tables, forms, and intuitive diagrams.
- System requirements are expanded versions of the user requirements
- System requirements add detail and explain how the user requirements should be provided by the system

Requirements Specification

- System requirements may be used as part of the contract for the implementation of the system
- Describe the external behavior of the system and its operational constraints. They should not be concerned with how the system should be designed or implemented
- System requirements may be written in natural language, but other notations based on forms, graphical system models, or mathematical system models can also be used.
- Graphical models are most useful when you need to show how a state changes or when you need to describe a sequence of actions.

Question

Which of the following are the external behaviours of the systems?

- The system must generate a monthly reports in a timely manner.
- The monthly commission must be calculated on the last day of the month.
- To improve respond time, the configuration data should be pre-loaded into the in-memory cache.

Requirements Specification

- it is practically impossible to exclude all design information. There are several reasons for this:
  - You may have to design an initial architecture of the system to help structure the requirements specification. The system requirements are organized according to the different sub-systems that make up the system.
  - Systems may interoperate with existing systems, which constrain the design and impose requirements on the new system.
  - The use of a specific architecture to satisfy non-functional requirements may be necessary.

Requirements Specification

Natural Language Specification

- It is expressive, intuitive, and universal.
- It is also potentially vague, ambiguous, and its meaning depends on the background of the reader

Requirements Specification

Natural Language Specification

- To minimize misunderstandings when writing natural language requirements:
  - Invent a standard format and ensure that all requirement definitions adhere to that format. Standardizing the format makes omissions less likely and requirements easier to check
  - Use language consistently to distinguish between mandatory and desirable requirements
  - Use text highlighting to pick out key parts of the requirement.
  - Do not assume that readers understand technical software engineering language.
  - Associate a rationale with each user requirement. The rationale should explain why the requirement has been included

Requirements Specification

Structured Specification

- Structured natural language is a way of writing system requirements in a predefined structure or template using natural language
- user requirements can be written on cards, one requirement per card.
  - a number of fields on each card, such as:
  - requirements rationale
  - dependencies on other requirements
  - the source of the requirements
  - supporting materials, and so on

Requirements Specification

Structured Specification - Sample

Insulin Pump/Control Software/SRS/3.3.2

Function	Compute insulin dose: Safe sugar level.
Description	Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.
Inputs	Current sugar reading (r2), the previous two readings (r0 and r1).
Source	Current sugar reading from sensor. Other readings from memory.
Outputs	CompDose—the dose in insulin to be delivered.
Destination	Main control loop.
Action	CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.
Requirements	Two previous readings so that the rate of change of sugar level can be computed.
Pre-condition	The insulin reservoir contains at least the maximum allowed single dose of insulin.
Post-condition	r0 is replaced by r1 then r1 is replaced by r2.
Side effects	None.

Requirements Specification

Structured Specification

- Overcome the limitation of natural language using tables or graphical models to show computations, system state changes, system interaction, execution sequences
- Tables are particularly useful when there are a number of possible alternative situations

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) If rounded result = 0 then CompDose = MinimumDose

Software Requirements Document

- **Definition**: A Software Requirements Document (SRD) is a formal document that captures and specifies the functional and non-functional requirements of a software system. It serves as a blueprint for the development team and other stakeholders.
- **Purpose**:
  - **Communication**: Facilitates clear communication between stakeholders, including clients, developers, and testers.
  - **Guidance**: Provides a detailed guide for the design, development, and testing of the software.
  - **Agreement**: Acts as a contractual agreement between stakeholders on the expected functionalities and constraints of the system.

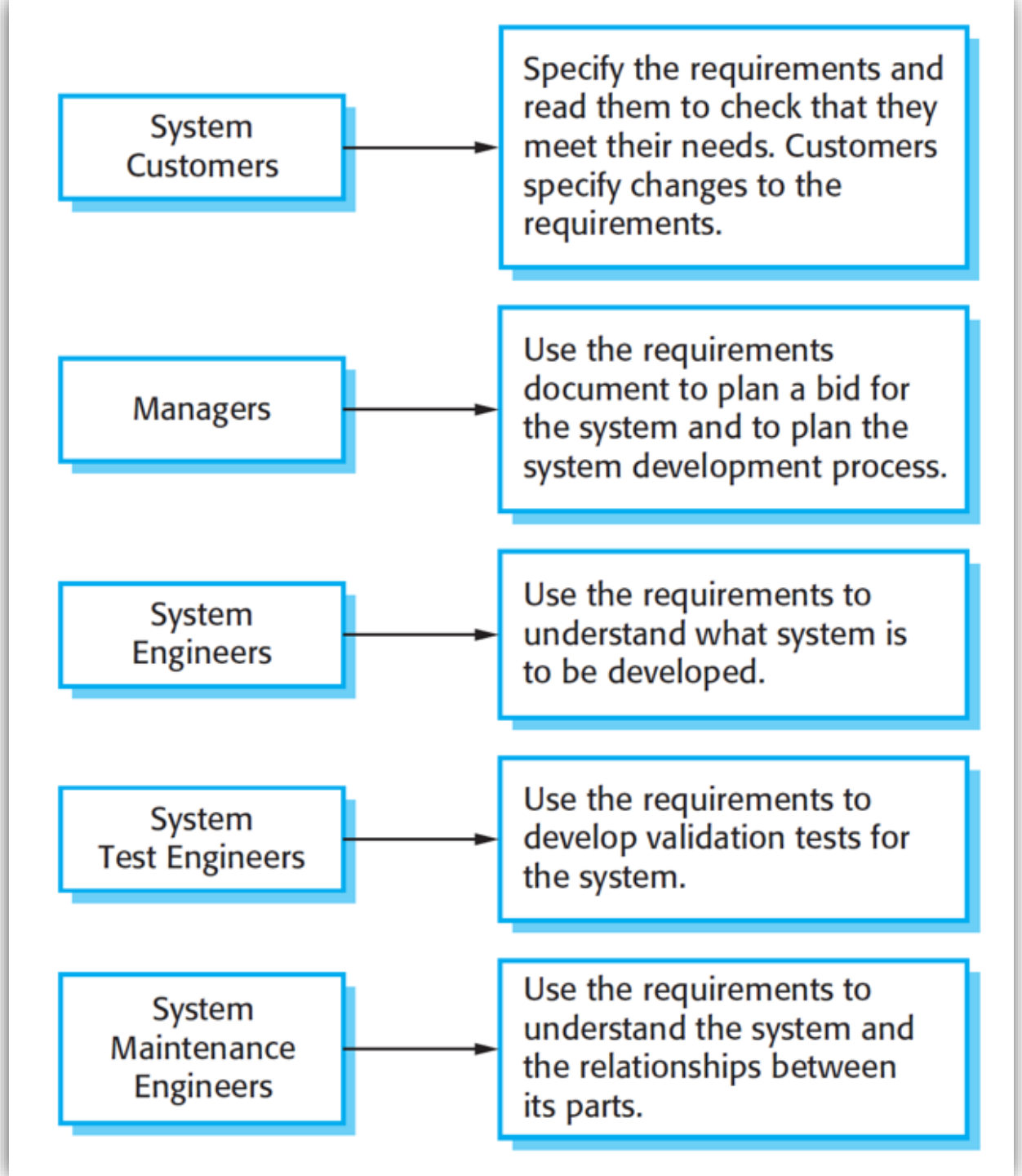
Software Requirements Document

- Requirements documents are essential when an outside contractor is developing the software system
- Suitable for business systems where requirements are stable;
- Agile development methods argue that requirements change so rapidly that a requirements document is out of date as soon as it is written.
  - However, it is still useful to write a short supporting document that defines the business and dependability requirements for the system;

Importance of the Software Requirements Document

- **Clarity and Understanding**: Ensures that all stakeholders have a clear and shared understanding of the system requirements.
- **Project Planning**: Aids in project planning, including resource allocation, timeline estimation, and risk management.
- **Scope Management**: Helps in managing project scope and preventing scope creep by clearly defining what is included and excluded.
- **Basis for Testing**: Provides a foundation for creating test cases and validating that the system meets the specified requirements.
- **Change Management**: Facilitates the management of changes to requirements throughout the project lifecycle.

The Software Requirements Document



Best Practices for Creating a Software Requirements Document

- **Engage Stakeholders**: Involve all relevant stakeholders in the requirements gathering process to ensure comprehensive coverage.
- **Be Clear and Unambiguous**: Use clear and precise language to avoid misunderstandings.
- **Prioritize Requirements**: Identify and prioritize critical requirements to ensure that the most important aspects are addressed first.
- **Use Multiple Techniques**: Combine different techniques (e.g., interviews, surveys, workshops) to gather requirements comprehensively.
- **Validate and Verify**: Continuously validate and verify requirements with stakeholders to ensure accuracy and completeness.
- **Maintain Traceability**: Ensure that each requirement can be traced back to its source and throughout the development process.
- **Iterative Refinement**: Refine the requirements iteratively based on feedback and testing results.

Level of detail

- Specifying the right level of detail in a Software Requirements Document (SRD) is critical for ensuring clarity, effective communication, and successful project outcomes. The SRD should provide sufficient detail to guide development while remaining clear and manageable.
- Balancing Detail and Manageability
  - **Avoid Over-Specification**: Providing too much detail can make the document cumbersome and difficult to manage.
  - **Avoid Under-Specification**: Providing too little detail can lead to misunderstandings and gaps in the requirements.
  - **Use Visual Aids**: Use diagrams, models, and tables to convey complex information more clearly.
  - **Iterative Refinement**: Continuously refine the requirements based on feedback and testing results.

Factors Influencing the Level of Detail

- **Project Complexity**: More complex projects typically require more detailed requirements.
- **Stakeholder Needs**: Different stakeholders may require varying levels of detail based on their roles and responsibilities.
- **Development Methodology**: The chosen development methodology (e.g., Agile, Waterfall) can influence the level of detail needed.
- **Regulatory and Compliance Requirements**: Projects subject to regulatory requirements may need more detailed documentation.
- **Risk Management**: High-risk projects may require more detailed requirements to mitigate potential issues.

Question

- For each of the following project, MORE detailed or LESS detailed is required?
  - Mission critical systems
  - Outsource baby sitter hiring web app project
  - In-house software development
  - Mobile app using Agile development process
  - Medical instrument control system
  - Social media platform

Requirements Validation

- Requirements validation is the process of checking that requirements correctly define the system that the customer really wants
- It is important because errors in the Software Requirements Document (SRD) can lead to extensive rework costs when these problems are discovered during development or after the system is in service.

Requirements Validation

- During the requirements validation process, different types of checks should be carried out on the requirements in the SRD. These checks include:
  - *Validity checks* - The functions proposed by stakeholders should be aligned with what the system needs to perform. You may find later that there are additional or different functions are required instead.
  - *Consistency checks* - Requirements in the document should not conflict. That is, there should not be contradictory constraints or different descriptions of the same system function.

Requirements Validation

- *Completeness checks* - The requirements document should include requirements that define all functions and the constraints intended by the system user.
- *Realism checks* - Using knowledge of existing technology, the requirements should be checked to ensure that they can be implemented realistically.
- *Verifiability* - To reduce the potential for dispute between customer and contractor, system requirements should always be written so that they are verifiable.

Validation Techniques

- There are a number of requirements validation techniques that can be used individually or in conjunction with one another:
  - *Requirements reviews* - The requirements are analyzed systematically by a team of reviewers who check for errors and inconsistencies.
  - *Prototyping* - In this approach to validation, an executable model of the system in question is demonstrated to end-users and customers
  - *Test-case generation* - Requirements should be testable. If the tests for the requirements are devised as part of the validation process, this often reveals requirements problems. If a test is difficult or impossible to design, this usually means that the requirements will be difficult to implement and should be reconsidered

Best Practices for Software Specification

- **Engage Stakeholders**: Involve all relevant stakeholders throughout the process to ensure comprehensive coverage and buy-in.
- **Use Multiple Techniques**: Combine different elicitation and analysis techniques to gather and refine requirements.
- **Communicate Clearly**: Use clear and precise language to avoid misunderstandings.
- **Prioritize Requirements**: Identify and prioritize critical requirements to ensure that the most important aspects are addressed first.
- **Validate Continuously**: Continuously validate requirements with stakeholders to ensure accuracy and completeness.
- **Document Thoroughly**: Maintain comprehensive and clear documentation of all requirements and analysis activities.
- **Iterate and Refine**: Refine requirements iteratively based on feedback and testing results.

Appendix 1 - Structure of the Software Requirements Document

- **Introduction**
  - **Purpose**: States the purpose of the SRD.
  - **Scope**: Defines the scope of the software system.
  - **Definitions, Acronyms, and Abbreviations**: Provides definitions of terms, acronyms, and abbreviations used in the document.
  - **References**: Lists any reference documents or standards.
  - **Overview**: Provides an overview of the document structure.
- **Overall Description**
  - **Product Perspective**: Describes the context and environment in which the system operates.
  - **Product Functions**: Summarizes the major functions of the system.
  - **User Characteristics**: Describes the characteristics of the intended users.
  - **Constraints**: Lists any constraints that affect the system.
  - **Assumptions and Dependencies**: States any assumptions and dependencies related to the system.
- **Specific Requirements**
  - **Functional Requirements**: Details the specific functionalities that the system must provide.
  - **Non-Functional Requirements**: Specifies the system's operational qualities and constraints (e.g., performance, reliability, security).
  - **External Interface Requirements**: Describes how the system interacts with external systems, hardware, and software.
  - **System Features**: Provides detailed descriptions of the system features, including use cases and user stories.
- **Appendices**
  - **Appendix A: Glossary**: Defines terms and concepts used in the SRD.
  - **Appendix B: Analysis Models**: Includes models and diagrams (e.g., UML diagrams) that support the requirements.
  - **Appendix C: Issues List**: Tracks any outstanding issues or open questions related to the requirements.

The end
Thank you