UNIVERSITÀ DEGLI STUDI DI TRIESTE

Class 10: Introduction to Requirements Engineering

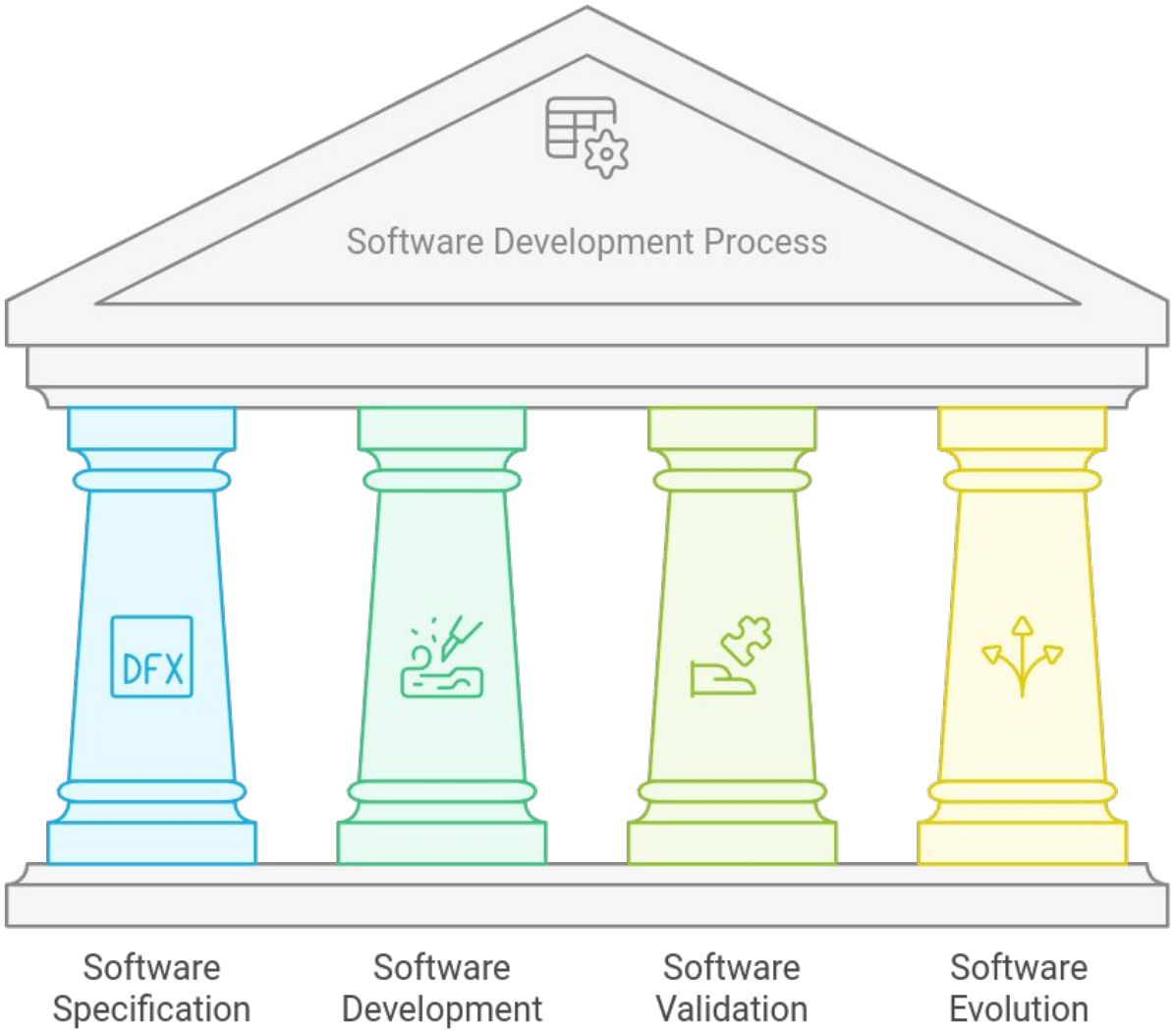
Master Course:

Data-driven Systems Engineering (ML Operations)

440MI and 305SM

Today's goal:

- Part 1
  - Introduction to Requirements Engineering
  - Functional requirements
  - Non-functional requirements
- Part 2
  - The software requirements document
  - Requirements specification
  - Requirements engineering processes
  - Requirements validation
- Part 3
  - Requirements management
  - Some Practice

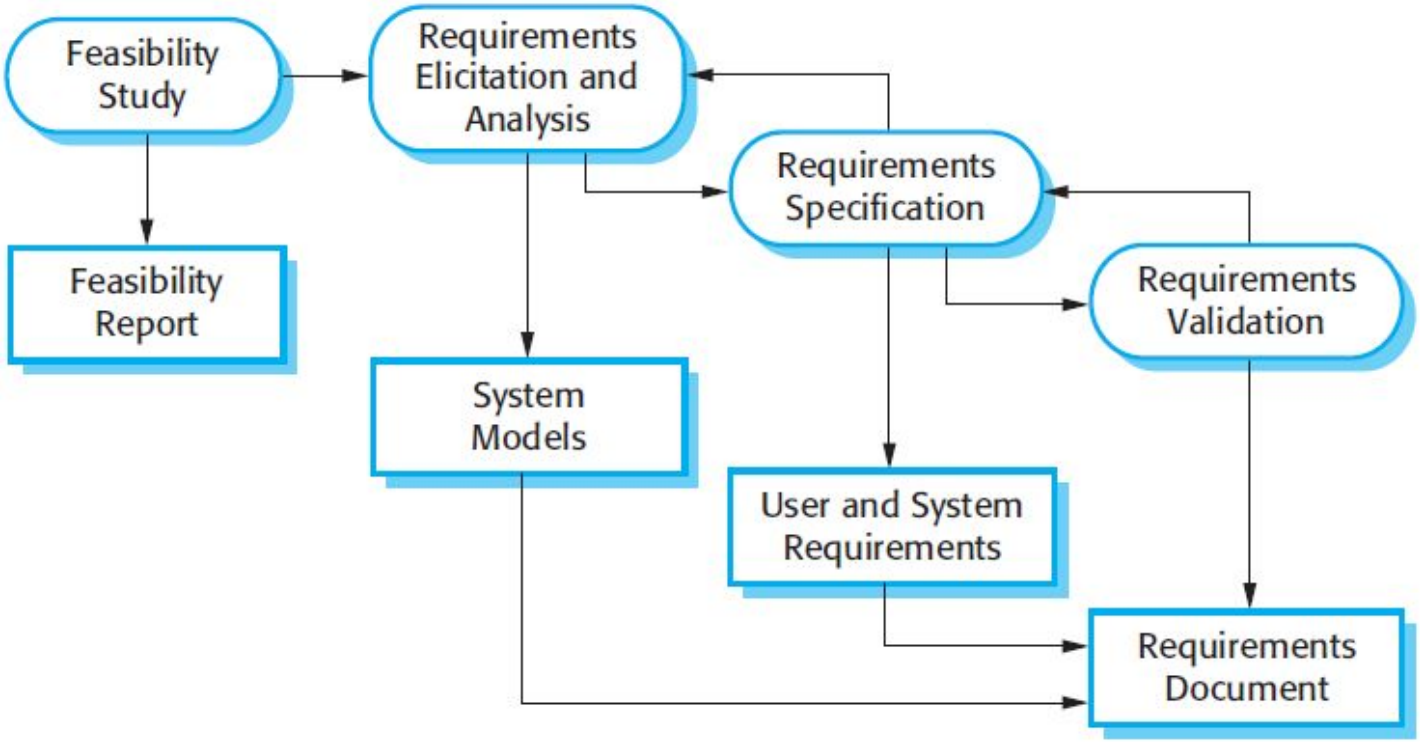

Software Development Process
DFX
Software Specification
Software Development
Software Validation
Software Evolution

Process activities - specification

The four basic **process activities** of:

- specification;
- development (design and implementation);
- validation;
- evolution;

Process activities - specification



Requirements engineering

The **process of establishing the services** that **the customer requires** from a system and **the constraints** under which it operates and is developed.

The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process.

What is a requirement?

- It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.

- **This is inevitable** as requirements may serve a dual function
  - May be the basis for a **bid for a contract** - therefore must be open to interpretation;
  - May be the basis for the **contract** itself - therefore must be defined in detail;
  - Both these statements may be called requirements.

Types of requirement

User requirements:

Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.

System requirements:

A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.

User and system requirements

Mental Health Care Patient Management System.

MHC-PMS

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.

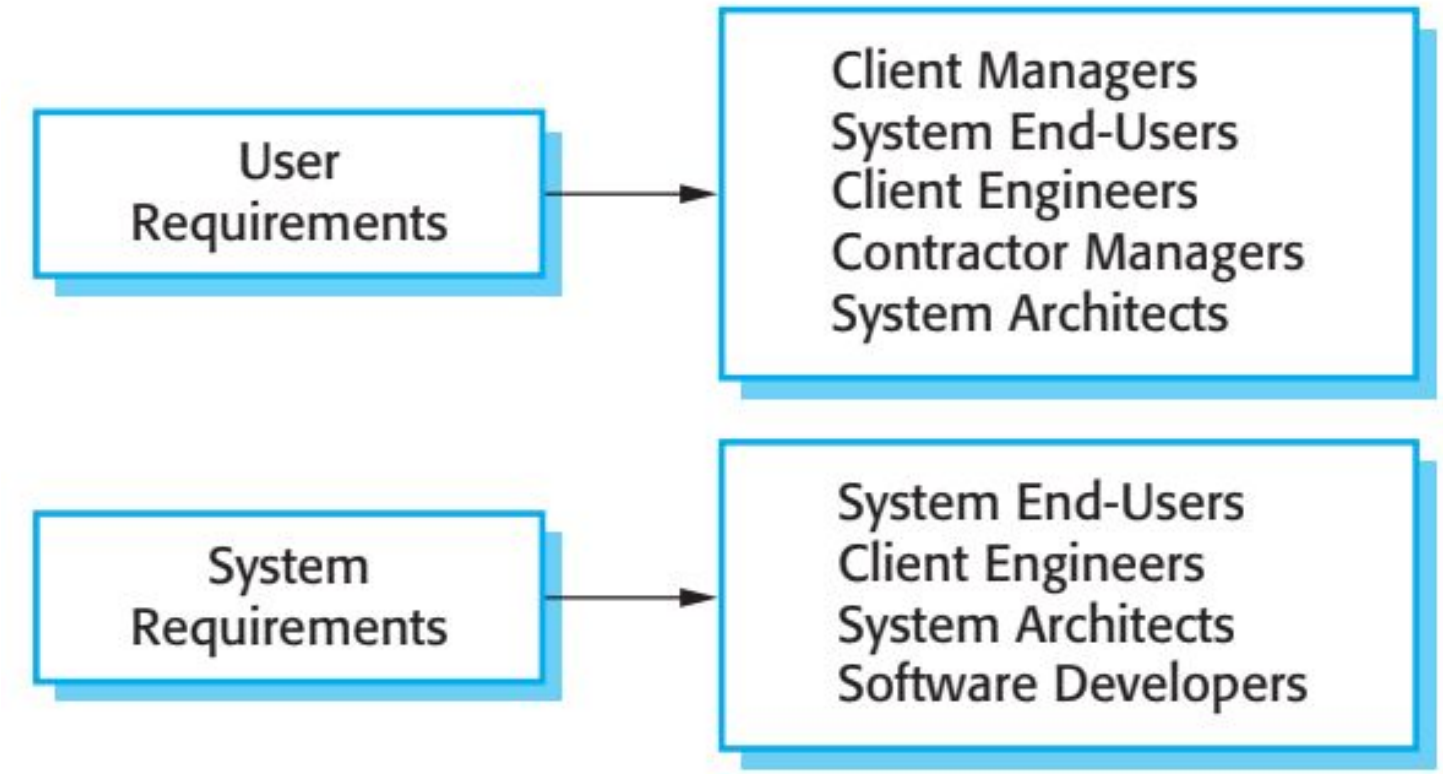
1.2 The system shall automatically generate the report for printing after 17.30 on the last working day of the month.

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.

Readers of different types of requirements specification



Functional and non-functional requirements

- **Functional requirements**
  - Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
  - May state what the system should not do.
- **Non-functional requirements**
  - Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
  - Often apply to the system as a whole rather than individual features or services.
- Domain requirements
  - Constraints on the system from the domain of operation

Functional requirements

- Describe functionality or system services.
- Depend on the type of software, expected users and the type of system where the software is used.
- Functional user requirements may be high-level statements of what the system should do.
- Functional system requirements should describe the system services in detail.

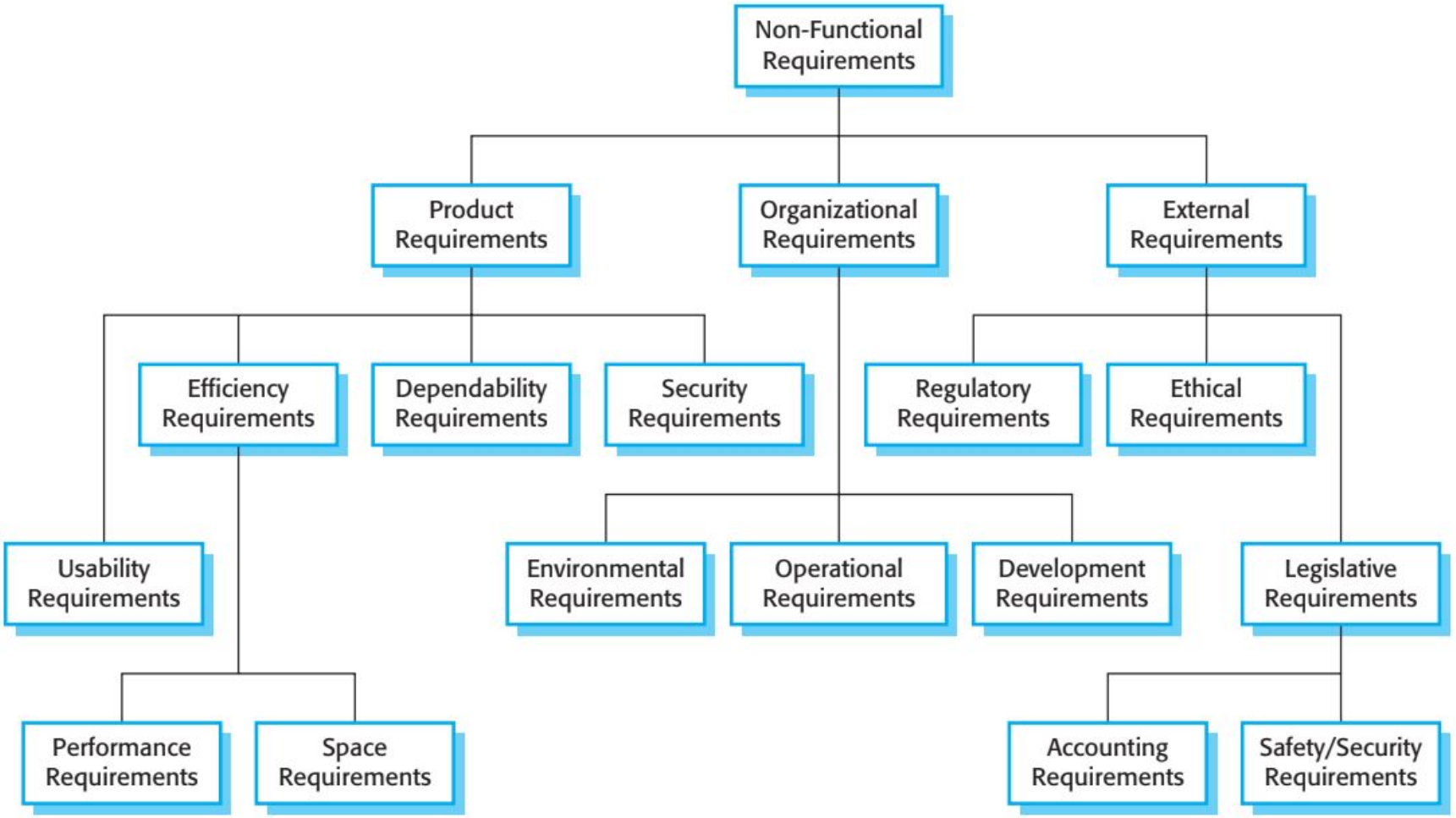
Functional requirements for the MHC-PMS

- A user shall be able to search the appointments lists for all clinics.
- The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day.
- Each staff member using the system shall be uniquely identified by his or her 8-digit employee number.

Non-functional requirements

- These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
- Process requirements may also be specified mandating a particular IDE, programming language or development method.
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.

Types of non-functional requirement



Non-functional requirements implementation

- Non-functional requirements may affect the overall architecture of a system rather than the individual components.
  - For example, to ensure that performance requirements are met, you may have to organize the system to minimize communications between components.
- A single non-functional requirement, such as a security requirement, may generate a number of related functional requirements that define system services that are required.
  - It may also generate requirements that restrict existing requirements.

Non-functional classifications

- Product requirements
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
- Organisational requirements
  - Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.
- External requirements
  - Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

Examples of nonfunctional requirements in the MHC-PMS

Product requirement
The MHC-PMS shall be available to all clinics during normal working hours (Mon–Fri, 0830–17.30). 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.

Usability requirements

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

- 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 experienced users shall not exceed two per hour of system use. (Testable non-functional requirement)

Metrics for specifying **non-functional** requirements

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

Domain requirements

- The system's operational domain imposes requirements on the system.
  - For example, a train control system has to take into account the braking characteristics in different weather conditions.
- Domain requirements be new functional requirements, constraints on existing requirements or define specific computations.
- If domain requirements are not satisfied, the system may be unworkable.

Domain requirements **problems**

Understandability

- Requirements are expressed in the language of the application domain;
- This is often not understood by software engineers developing the system.

Implicitness

- Domain specialists understand the area so well that they do not think of making the domain requirements explicit.

Key points

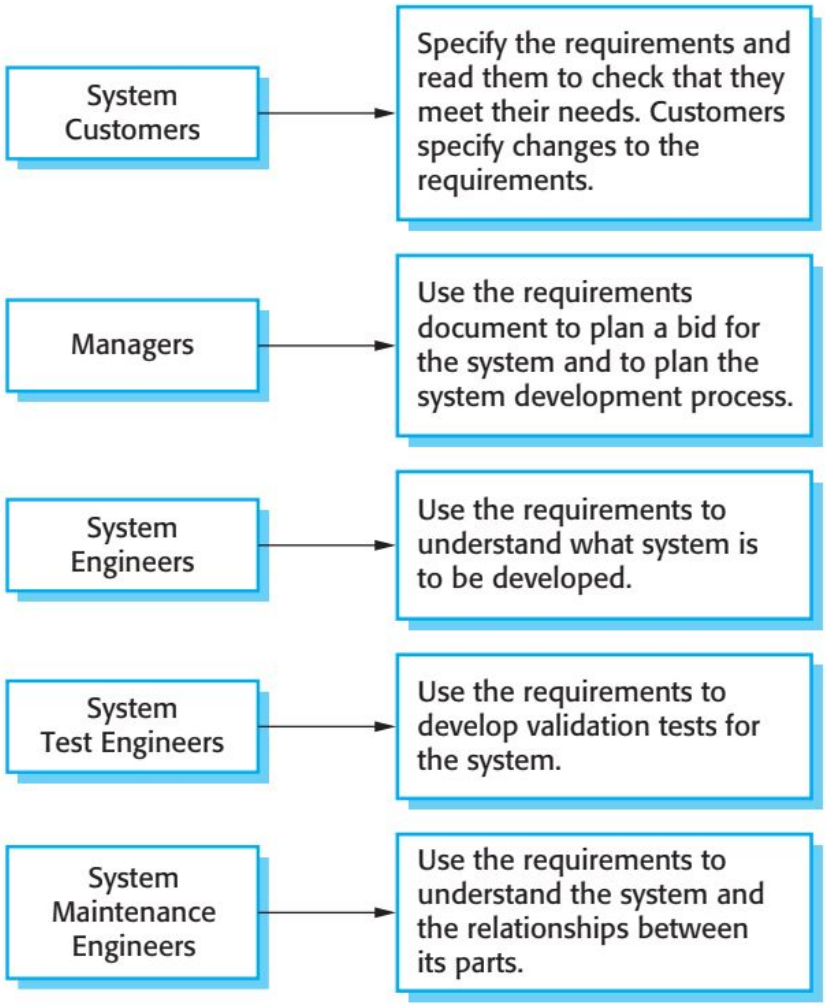
- Requirements for a software system set out what the system should do and define constraints on its operation and implementation.
- Functional requirements are statements of the services that the system must provide or are descriptions of how some computations must be carried out.
- Non-functional requirements often constrain the system being developed and the development process being used.
- They often relate to the emergent properties of the system and therefore apply to the system as a whole.

PART 2

The software requirements document

- The software requirements document is the official statement of what is required of the system developers.
- Should include both a definition of user requirements and a specification of the system requirements.
- It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it.

Users of a requirements document



Requirements document variability

- Information in requirements document depends on type of system and the approach to development used.
- Systems developed incrementally will, typically, have less detail in the requirements document.
- Requirements documents standards have been designed e.g. IEEE standard. These are mostly applicable to the requirements for large systems engineering projects.

The IEEE standard for requirements documents

The most widely known requirements document standard is IEEE/ANSI 830-1998 (IEEE, 1998). This IEEE standard suggests the following structure for requirements documents:

1. Introduction
1.1 Purpose of the requirements document
1.2 Scope of the product
1.3 Definitions, acronyms and abbreviations
1.4 References
1.5 Overview of the remainder of the document

2. General description
2.1 Product perspective
2.2 Product functions
2.3 User characteristics
2.4 General constraints
2.5 Assumptions and dependencies

3.**Specific requirements**, covering functional, non-functional and interface requirements. This is obviously the most substantial part of the document but because of the wide variability in organisational practice, it is not appropriate to define a standard structure for this section. The requirements may document external interfaces, describe system functionality and performance, and specify logical database requirements, design constraints, emergent system properties and quality characteristics.

4.**Appendices**

5.**Index**

Although the IEEE standard is not ideal, it contains a great deal of good advice on how to write requirements and how to avoid problems. It is too general to be an organisational standard in its own right. It is a general framework that can be tailored and adapted to define a standard geared to the needs of a particular organisation.

https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=502838

Ways of writing a system requirements specification

Notation	Description
Natural language	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

Problems with natural language

- Lack of clarity
  - Precision is difficult without making the document difficult to read.
- Requirements confusion
  - Functional and non-functional requirements tend to be mixed-up.
- Requirements amalgamation
  - Several different requirements may be expressed together.

Example requirements for the insulin pump software system

3.2 The system shall measure the blood sugar and deliver insulin, if required, every 10 minutes. *(Changes in blood sugar are relatively slow so more frequent measurement is unnecessary; less frequent measurement could lead to unnecessarily high sugar levels.)*

3.6 The system shall run a self-test routine every minute with the conditions to be tested and the associated actions defined in Table 1. *(A self-test routine can discover hardware and software problems and alert the user to the fact the normal operation may be impossible.)*

Structured specifications

- An approach to writing requirements where the freedom of the requirements writer is limited and requirements are written in a standard way.

- This works well for some types of requirements e.g. requirements for embedded control system but is sometimes too rigid for writing business system requirements.

Form-based specifications

- Definition of the function or entity.
- Description of inputs and where they come from.
- Description of outputs and where they go to.
- Information about the information needed for the computation and other entities used.
- Description of the action to be taken.
- Pre and post conditions (if appropriate).
- The side effects (if any) of the function.

A structured specification of a requirement for an insulin pump

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.

A structured specification of a requirement for an insulin pump

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 engineering processes

- The processes used for RE vary widely depending on the application domain, the people involved and the organisation developing the requirements.
- However, there are a number of generic activities common to all processes
  - Requirements elicitation;
  - Requirements analysis;
  - Requirements validation;
  - Requirements management.
- In practice, RE is an iterative activity in which these processes are interleaved.

PART 3

A spiral view of the requirements engineering process

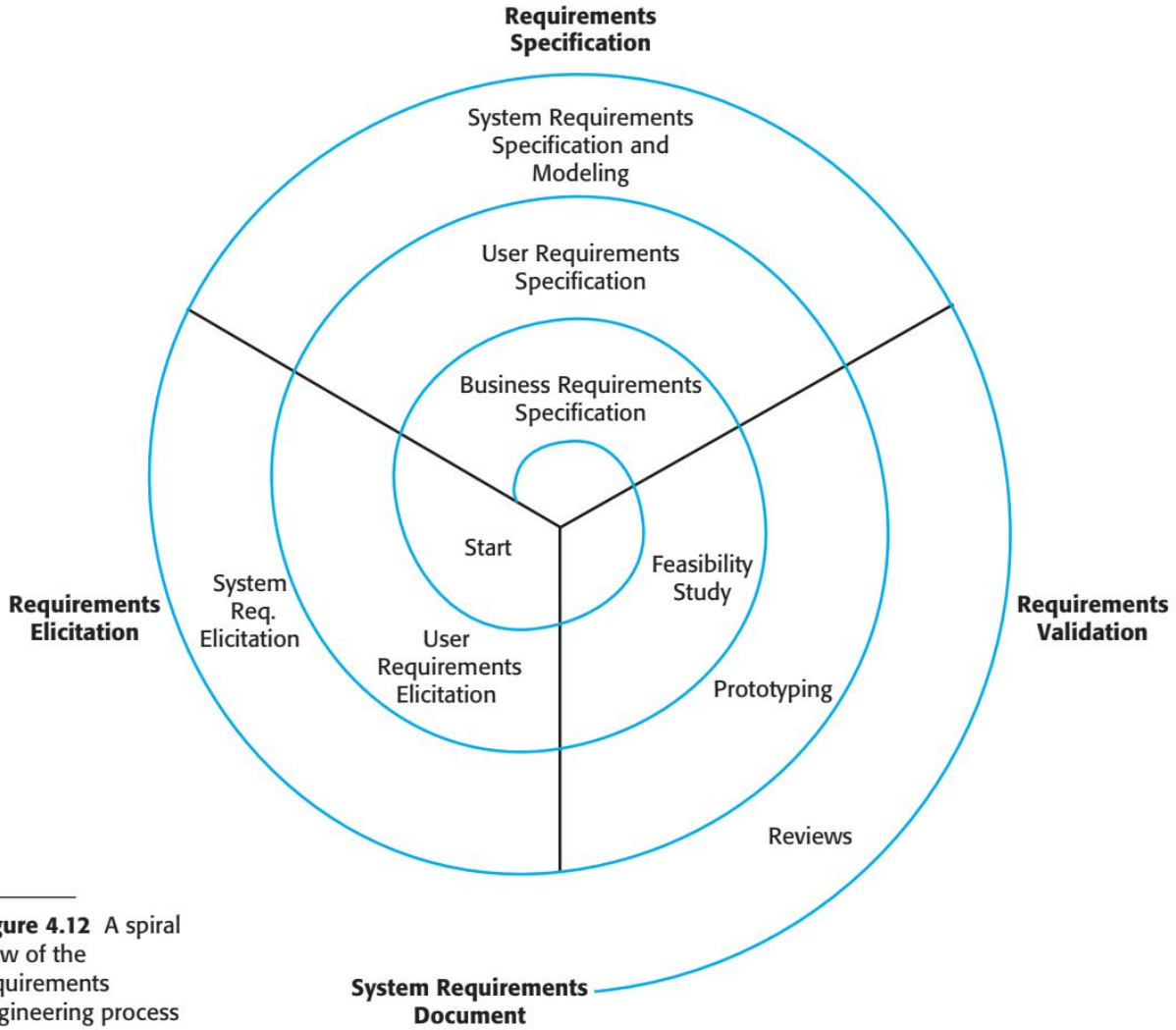
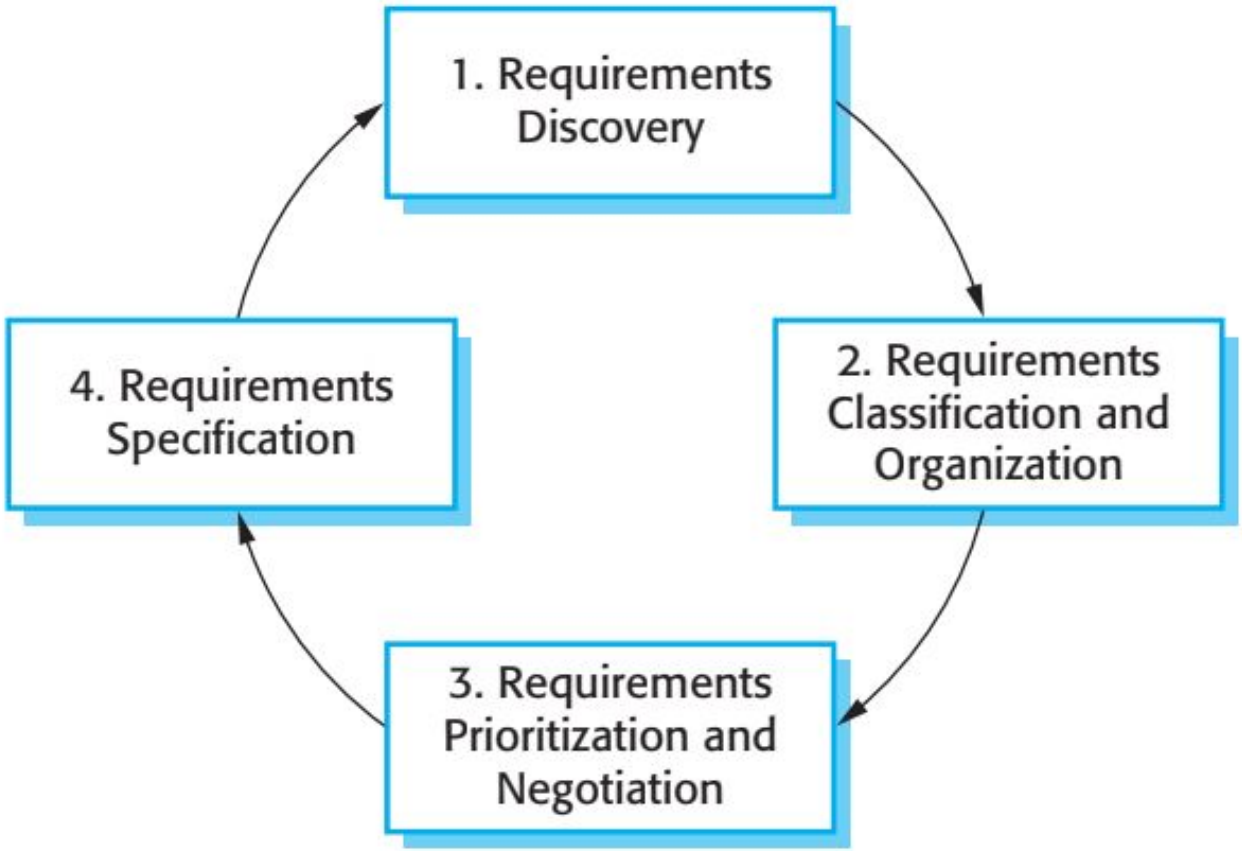


Figure 4.12 A spiral view of the requirements engineering process

The requirements elicitation and analysis process



Requirements discovery

- The process of **gathering** information about the required and existing systems and **distilling** the user and system requirements from this information.
- **Interaction** is with system **stakeholders** from managers to external regulators.
- Systems normally have a range of stakeholders.

Stakeholders in the MHC-PMS

- **Patients** whose information is recorded in the system.
- **Doctors** who are responsible for assessing and treating patients.
- **Nurses** who coordinate the consultations with doctors and administer some treatments.
- **Medical** receptionists who manage patients' appointments.
- **IT staff** who are responsible for installing and maintaining the system.

Stakeholders in the MHC-PMS

- A **medical ethics manager** who must ensure that the system meets current ethical guidelines for patient care.
- **Healthcare managers** who obtain management information from the system.
- **Medical records staff** who are responsible for ensuring that system information can be maintained and preserved, and that record keeping procedures have been properly implemented.

Interviewing

- Formal or informal interviews with stakeholders are part of most RE processes.
- Types of interview
  - Closed interviews based on pre-determined list of questions
  - Open interviews where various issues are explored with stakeholders.
- Effective interviewing
  - Be open-minded, avoid preconceived ideas about the requirements and are willing to listen to stakeholders.
  - Prompt the interviewee to get discussions going using a springboard question, a requirements proposal, or by working together on a prototype system.

Interviews in practice

- Normally a mix of closed and open-ended interviewing.
- Interviews are good for getting an overall understanding of what stakeholders do and how they might interact with the system.
- Interviews are not good for understanding domain requirements
  - Requirements engineers cannot understand specific domain terminology;
  - Some domain knowledge is so familiar that people find it hard to articulate or think that it isn't worth articulating.

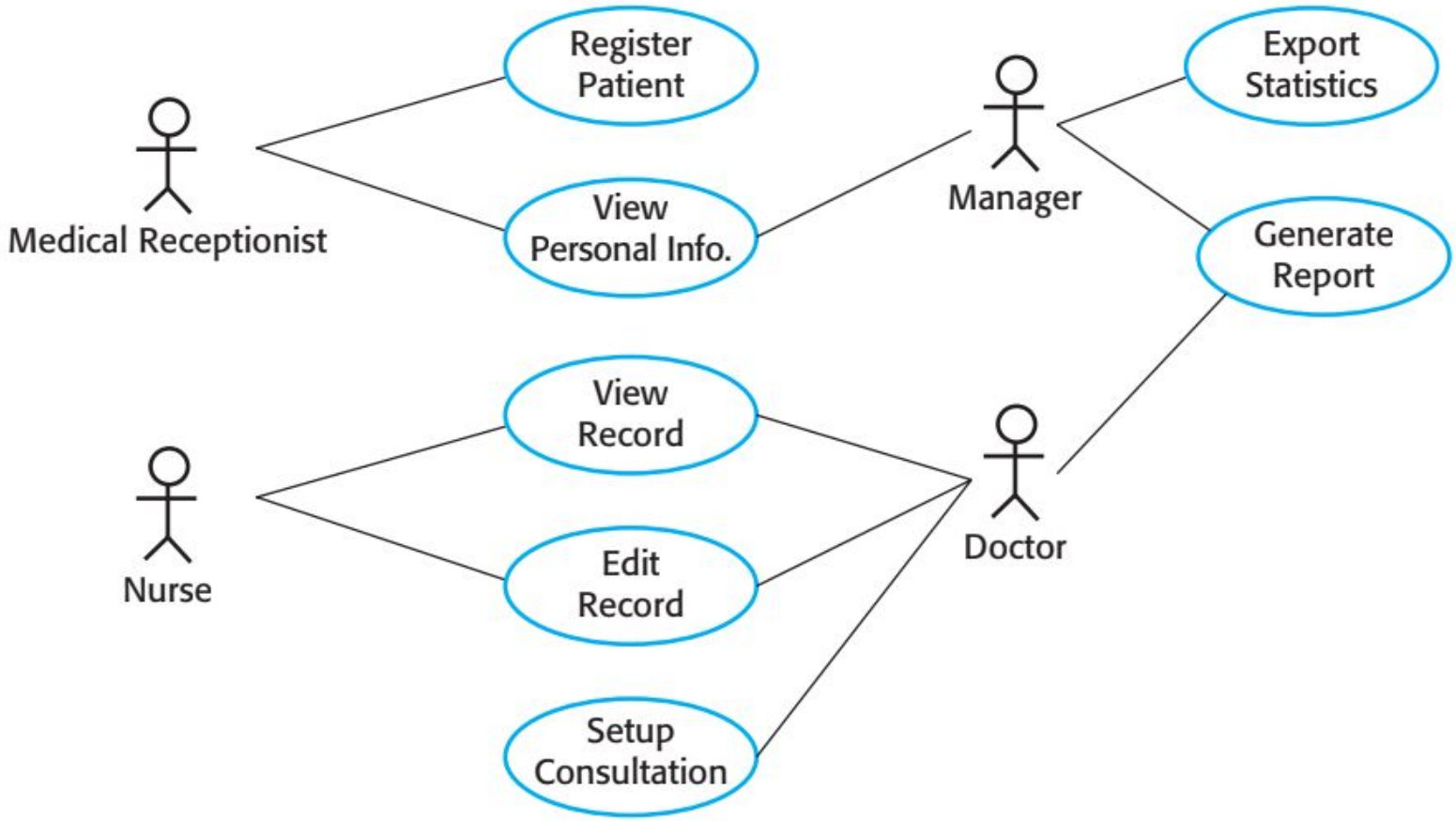
Scenarios Strategies

- Scenarios are real-life examples of how a system can be used.
- They should include
  - A description of the starting situation;
  - A description of the normal flow of events;
  - A description of what can go wrong;
  - Information about other concurrent activities;
  - A description of the state when the scenario finishes.

Use cases

- Use-cases are a scenario based technique in the UML which identify the actors in an interaction and which describe the interaction itself.
- A set of use cases should describe all possible interactions with the system.
- High-level graphical model supplemented by more detailed tabular description.
- Sequence diagrams may be used to add detail to use-cases by showing the sequence of event processing in the system.

Use cases for the MHC-PMS



Requirements validation

- Concerned with demonstrating that the requirements define the system that the customer really wants.
- Requirements error costs are high so validation is very important
  - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

Requirements validation techniques

- Requirements reviews
  - Systematic manual analysis of the requirements.
- Prototyping
  - Using an executable model of the system to check requirements. Covered in Chapter 2.
- Test-case generation
  - Developing tests for requirements to check testability.

Changing requirements

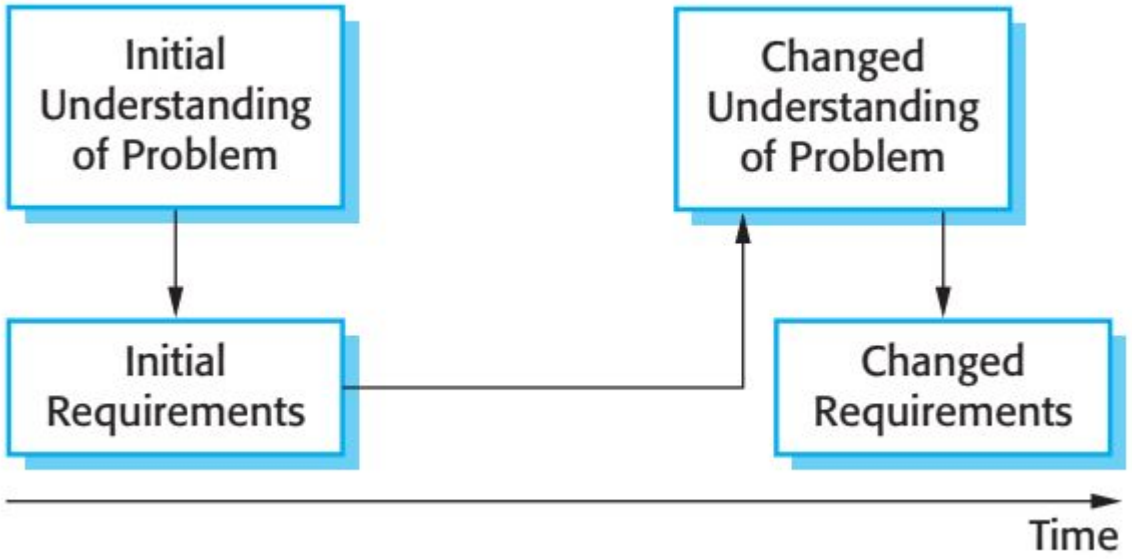
- The business and technical environment of the system always changes after installation.
  - New hardware may be introduced, it may be necessary to interface the system with other systems, business priorities may change (with consequent changes in the system support required), and new legislation and regulations may be introduced that the system must necessarily abide by.
- The people who pay for a system and the users of that system are rarely the same people.
  - System customers impose requirements because of organizational and budgetary constraints. These may conflict with end-user requirements and, after delivery, new features may have to be added for user support if the system is to meet its goals.

Changing requirements

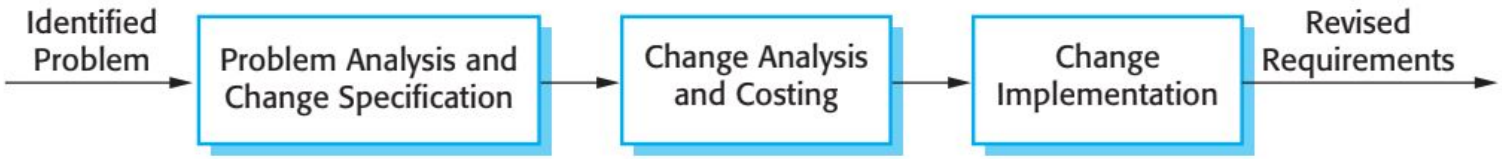
- Large systems usually have a diverse user community, with many users having different requirements and priorities that may be conflicting or contradictory.

- The final system requirements are inevitably a compromise between them and, with experience, it is often discovered that the balance of support given to different users has to be changed.

Requirements evolution



Requirements change management

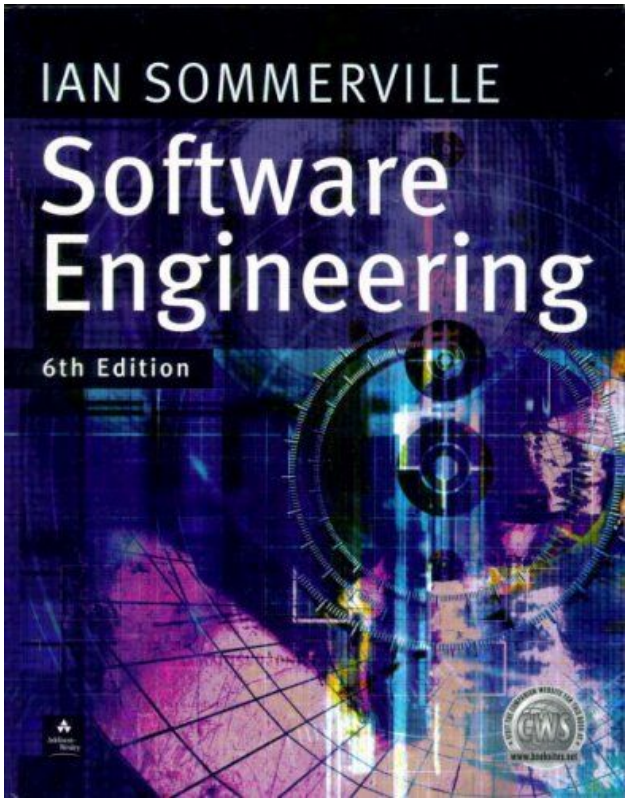



coggle

Key points

- You can use a range of techniques for requirements elicitation including interviews, scenarios, use-cases and ethnography.
- Requirements validation is the process of checking the requirements for validity, consistency, completeness, realism and verifiability.
- Business, organizational and technical changes inevitably lead to changes to the requirements for a software system. Requirements management is the process of managing and controlling these changes.

Textbook



Practical Project (on Teams)

Issues to Identify:

- **Lack of clarity**: vague terms like 'stuff', 'maybe', 'nice', and 'good graphics'.
- **Requirements confusion**: mixes functional, non-functional, and design ideas.
- **Amalgamation**: multiple unrelated requirements in a single paragraph.
- **No numbering or structure**: difficult to trace and verify.
- **Missing rationale, dependencies, and acceptance criteria**.
- No **graphical models** or **formal** specification provided.

Scenario 1 – Audio Streaming Application (Expected Standard), Strengths / Good Practices:

Property	Strengths and Observations
Natural Language	Requirements are written as short, numbered sentences; each statement expresses one idea clearly and unambiguously.
Structured Natural Language	Uses a consistent template with fields such as *Requirement ID*, *Description*, *Rationale*, and *Dependencies*. This improves traceability and clarity.
Design Description Languages	Mentions the system's architecture (client-server model) using descriptive but precise language. Technical enough for developers while still readable.
Graphical Notations	UML Use Case diagram is referenced and clearly described, showing main actors and interactions. Aids communication between stakeholders.
Mathematical Specifications	Includes a simple, understandable state transition model, adding rigor and reducing ambiguity.
Clarity / Precision	Each requirement is atomic, measurable, and testable. Avoids vague words like *fast*, *good*, or *nice*.
Requirement Confusion	Functional and non-functional requirements are separated into distinct sections (FRs vs. NFRs).
Requirement Amalgamation	Each requirement covers only one concept or feature. No overlapping or combined ideas.
Traceability	Requirements have identifiers (e.g., FR-03) allowing linkage to design, implementation, and tests.
Completeness	Covers introduction, overview, functional and non-functional parts, diagrams, and acceptance criteria.

Scenario 2 – Audio Streaming Application (Expected Standard), Weaknesses / Poor Practices:

Property	Weaknesses and Observations
Natural Language	Written in vague, informal terms (“nice to use”, “stuff”, “maybe”, “good graphics”). No numbering; difficult to reference or test.
Structured Natural Language	No template or requirement form; requirements are written as unstructured paragraphs. Missing fields such as ID, rationale, or dependencies.
Design Description Languages	None provided — mentions technical aspirations (“AI”, “fast”) without defining implementation details.
Graphical Notations	Missing; the text only says diagrams “might be added later”. No visual support for understanding the system.
Mathematical Specifications	Absent — mentions that they “will use math later,” showing poor planning and incomplete specification.
Clarity / Precision	Low precision — requirements are subjective and not measurable (“should be beautiful”, “should be perfect”).
Requirement Confusion	Functional, non-functional, and design requirements are mixed together (e.g., “The app should be fast and have good graphics”).
Requirement Amalgamation	Many sentences mix unrelated ideas (“users buy tickets and see bus times and it should work on all phones and be reliable”).
Traceability	Lacks numbering, labels, or structure — making traceability impossible.
Completeness	Incomplete — lacks rationale, dependencies, acceptance criteria, and formal sections.

Conclusion

This low-quality SRD lacks formal structure, measurable statements, and traceability. It illustrates how poor documentation can create confusion between developers, stakeholders, and end-users. Students should identify all weaknesses and rewrite the document using best practices.