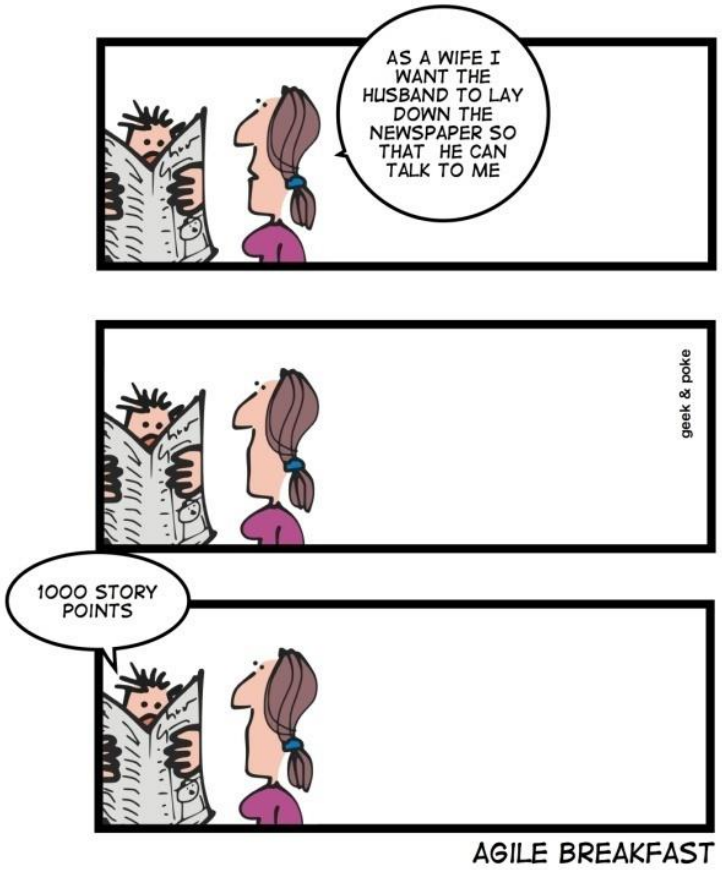


Software Requirements

Pr. Imane Fouad

Course Objectives

Learn	Software requirement process
Express	Formally express software requirements
Use	Use use case diagrams to model functional specifications

Course Plan

Section 1: Introduction

Section 2: Software requirement process

Section 3: Specification Model

Section 4: Use Case Model

Introduction

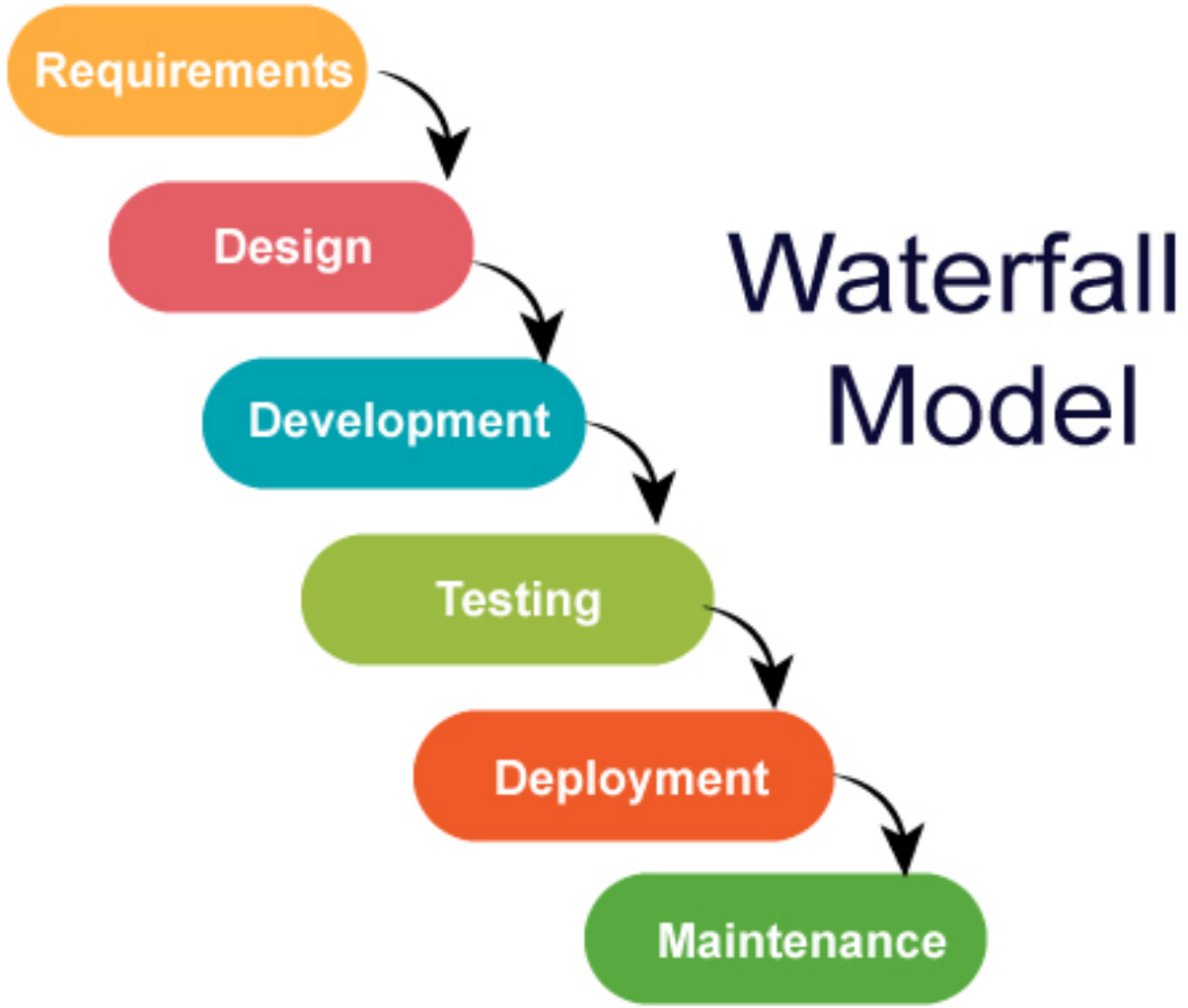
Section 1

What is a software requirement?

At its most basic, a **requirement** is a property that must be exhibited by a software in order to solve some problem in the real world.

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

Software Development Process



Motivation

What and why develop?

Need to **understand the client**

Need to **formalize client expectations**

The better requirements are **captured and formalized**, the higher the chance of project success

Help the client **express what they know and what they want**

What makes them so important?

According to a large scale empirical study (> 8000 projects) done by Standish in 1995 they are amongst the **top factors** that **cause a project to fail**.

Incomplete requirements (13.1%)
Lack of user involvement (12.4%)
Unrealistic expectations (9.9%)
Lack of executive support (9.3%)
Changing requirements and specifications (8.7 %)
Lack of planning (8.1%)
System no longer needed (7.5%)

Types of Requirements

Requirements can range from high-level abstract statements of a service or of a system constraints to detailed mathematical functional specification.

- **User Requirements** are high-level descriptions of the system's features and functionality (in natural language)
  - *Example*: *"The system must allow customers to search for products."*

- **System Requirements** are detailed descriptions of the system's functions, features, and constraints (from a technical perspective)
  - *Example: "*The system shall query the product database and return search results within 2 seconds, supporting both partial and full keyword matches*."*

Who Reads Different Types of Requirements

Requirement Type	Typical Readers / Audience	Purpose
User Requirements	Clients, end-users, managers, stakeholders	To communicate needs in an understandable form, verify system meets business goals
System Requirements	Developers, engineers, testers, architects	To guide design and implementation, ensure technical feasibility and correct construction

Types of Requirements

Functional Requirements:

What the system **must do**

Describes the system from the **user's perspective**

Answers the question: **"What?"**

Non-Functional Requirements:

Constraints or technical choices

Quality attributes like **performance, security, usability**

Answers the question: **"How?"**

What is a functional requirement?

A **functional requirement** describes a function/service that the software needs to execute/provide.

Sometimes known as capability or feature.

They can include:

- Creating, updating, and deleting records
- Searching, viewing, accessing data, and Transmitting data
- Performing calculations or processing information

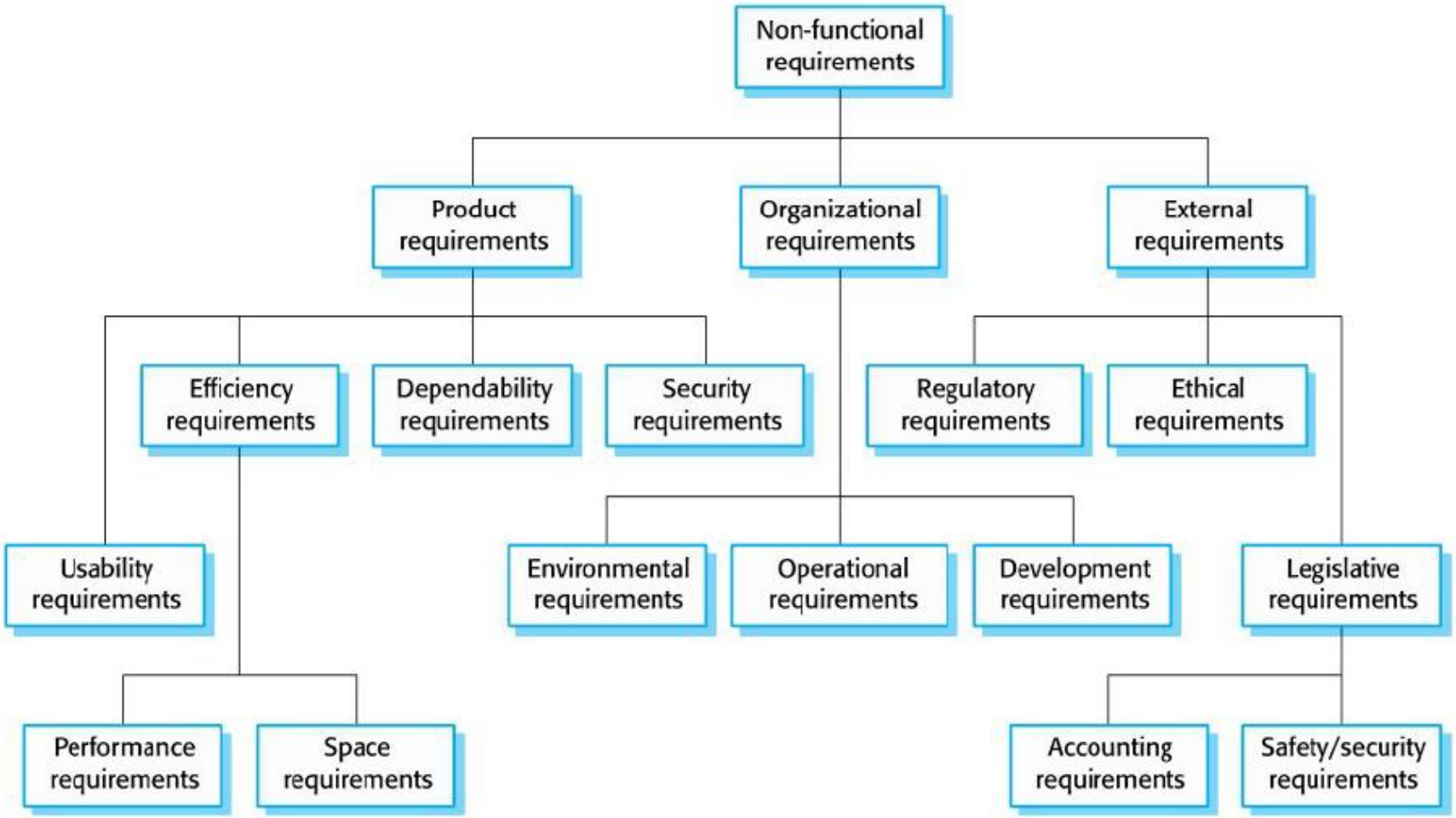
What is a nonfunctional requirement?

A **non-functional requirement** is a constraint on the provided functions/services.

Sometimes known as constraints or quality requirements.

Can be further classified according to whether they are performance requirements, maintainability requirements, safety requirements, reliability requirements, security requirements, availability requirements, interoperability requirements…. or any other quality attribute….

What is a nonfunctional requirement?



Exercise – Functional or Non-Functional?

Scenario: You are developing an **Online Shopping System**.
Decide whether each requirement is **Functional** or **Non-Functional**:

1. The system must allow users to add items to a shopping cart.
2. The system must process a payment within 3 seconds.
3. Only registered users can place an order.
4. The website should be available 24/7 with 99.9% uptime.
5. The system must generate an invoice after each order.
6. All user passwords must be encrypted.

Exercise – Functional or Non-Functional?

Scenario: You are developing an **Online Shopping System**.
Decide whether each requirement is **Functional** or **Non-Functional**:

1. Add items to a shopping cart → **Functional (F)**
2. Process a payment within 3 seconds → **Non-Functional (NF)**
3. Only registered users can place an order → **Functional (F)**
4. Website available 24/7 with 99.9% uptime → **Non-Functional (NF)**
5. Generate an invoice after each order → **Functional (F)**
6. User passwords must be encrypted → **Non-Functional (NF)**

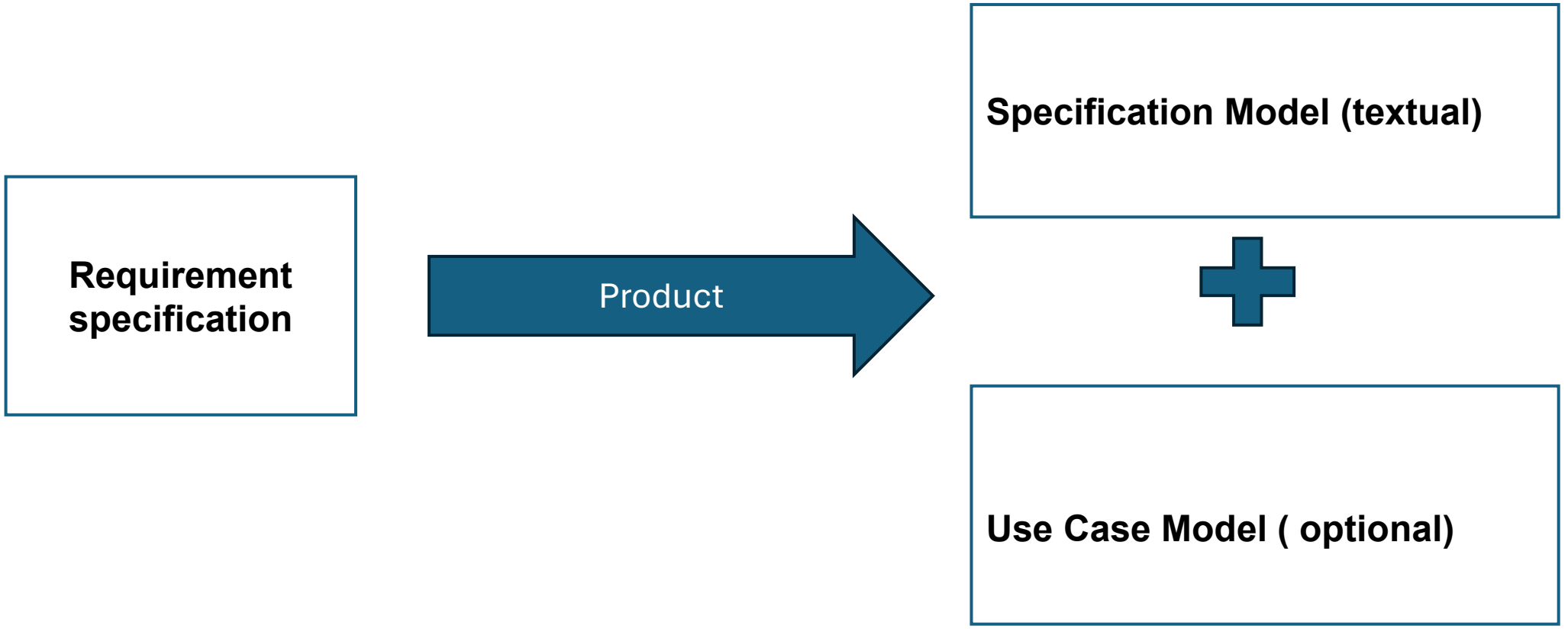
Expression of Requirements

- There are **two main ways** to express software requirements:
  - Specification Model
  - Use Case Model

- Specification Model:
  - Suitable for both functional and non-functional requirements
  - Can be created using a text editor, notepad, or dedicated tool

- Use Case Model:
  - Based on UML use case diagrams
  - Best suited for functional requirements

Requirement Models



Introduction

Section 1 - Discussion

Quiz

Software requirements process

Section 2

Where do we start from?

Feasibility study

What is Feasibility Study?

- First step of the Software Requirement Process
- Evaluates whether the proposed system is **practical and achievable**
- Checks **technical, economic, legal, operational, market, and schedule feasibility**

Outputs of Feasibility Study:

- **Feasibility Report / Study Report**
- Recommendation: **Proceed, Revise, or Reject** the project
- Identifies **major risks and constraints**

How to Generate the Output?

1 Collect stakeholder requirements and expectations

2 Analyze feasibility dimensions:

Technical (Can we build it?)	Economic (Is it cost-effective?)	Market (Is there demand or a user base for the system)	Operational (Will it fit into current operations?)	Schedule (Can it be delivered on time?)

3 Document findings clearly in a **Feasibility Report**

Project: Feasibility Report

Project assignment - Feasibility study

1. Technical Feasibility

- List all tools, software, and hardware needed for the project.
- Check if the team has the necessary skills to implement the solution.
- Identify potential technical challenges (e.g., learning a new programming language, integrating APIs).

Output: Short paragraph or table showing tools, technologies, and skills available vs. needed.

Project assignment - Feasibility study

2. Market / User Feasibility

- Identify target users and whether they need this solution.
- Research similar products or systems and see what they offer.
- Identify gaps or opportunities that your project can fill.
- Optional: Quick survey or interview with potential users for feedback.

Output:

Benchmark / Competitive Analysis

User Survey Results

Project assignment - Feasibility study

3. Schedule Feasibility

- Break the project into tasks and estimated durations.
- Check if it can be completed within the semester or project timeline.
- Highlight any tight deadlines or dependencies.

Output:

Timeline / Project Plan: A high-level project timeline (e.g., Gantt chart)

After the project is validated and approved, what happens next?

Requirements elicitation

"Yes, I'm a real Genie... but you're asking me to understand your client's requirements and even I can't do that!"

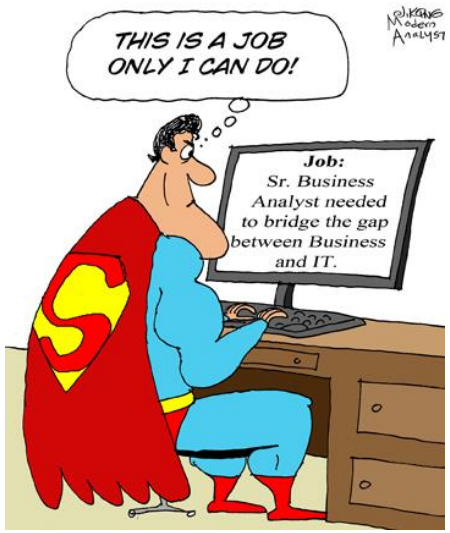
Requirements elicitation is the first stage in building an understanding of the problem the software is required to solve.

It is fundamentally a **human activity** and is where the stakeholders are identified and relationships established between the development team and the customer.

It is variously termed requirements capture, requirements discovery, and requirements acquisition.

How do we collect requirements?

Elicitation techniques



Interviews - Interviewing stakeholders is a "traditional" means of eliciting requirements.

Types of interviews

Closed interviews based on pre-determined list of questions

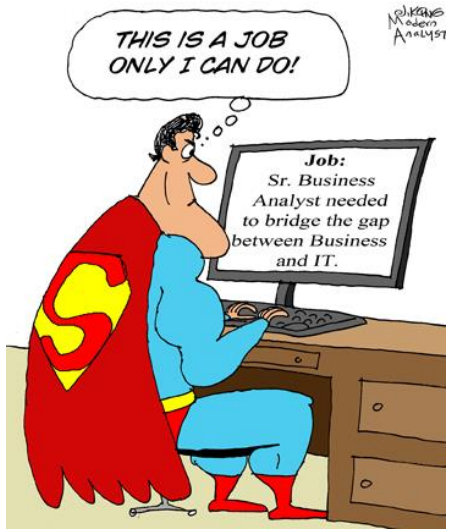
Open interviews where various issues are explored with stakeholders.

Effective interviewing

Be open-minded, avoid pre-conceived ideas and be willing to listen to stakeholders.

Prompt the interviewee using a springboard question, a requirements proposal, or by working together on a prototype system.

Elicitation techniques

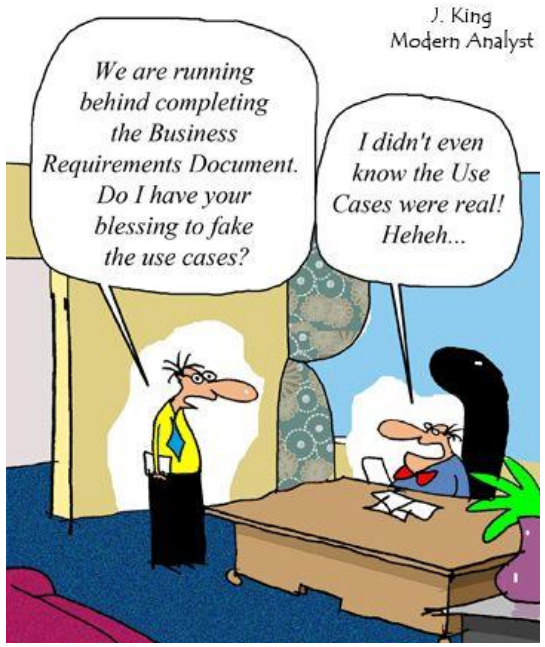


Interviews in practice

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.

Elicitation techniques



Scenarios

Scenarios are real-life descriptions 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;
- A description of the state when the scenario finishes.

Elicitation techniques

Prototypes

This technique is a valuable tool for clarifying **ambiguous** requirements and assess **alternatives**.

They provide users with a context within which they can better understand what information they need to provide.

Elicitation techniques

Observation - ethnography

The importance of software context within the organizational/operational environment has led to observational techniques.

> Software engineers learn about user tasks by observing how users perform their tasks by interacting with each other and with software tools and other resources.
>
> These techniques are relatively **expensive** but also **instructive** because they illustrate that many user tasks and business processes are too subtle and complex for their actors to describe.

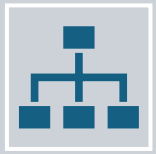
Ethnography is effective for understanding existing processes but **cannot identify new features**.

Outputs – Requirements Elicitation

1. Raw Data for Analysis

Tables, charts, or survey results collected from potential users

Example: Survey of 10 students



2. Initial User Stories (Optional but Recommended)

Short descriptions of how a user interacts with the system

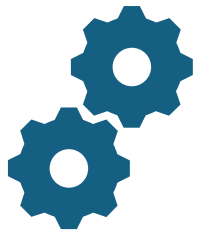
Example:"As a student, I want to reserve a book online so I can pick it up without waiting in line."

3. Prototypes (If Any) at this stage

Low-fidelity representations to help stakeholders visualize the system

Types: Paper sketches, wireframes, storyboards, clickable low-fidelity prototypes

What do we do once we have the requirements?



Requirements analysis

"We have been having a hard time guessing the business requirements. I'm hoping our new analyst can help."

Requirements Classification

Requirements can be classified on a number of dimensions:

- Functional/Non Functional
- Importance
- Stability
- Scope

Requirements analysis

"We have been having a hard time guessing the business requirements. I'm hoping our new analyst can help."

Conceptual modeling

Several kinds of models can be developed.

- Use case diagrams
- State models
- Process/Activity models
- Data models
- and many others.....

Many of these modeling notations are part of *the* **Unified Modeling Language (UML)**.

Outputs – Requirements Analysis

1. **Refined Requirements List**
   - Functional and Non-Functional requirements are clarified, consistent, and feasible
   - Requirements are specific, measurable, and testable

2. **Prioritized Requirements**
   - Requirements are ranked based on importance and feasibility
   - Common method: MoSCoW

3. **Optional Prototypes / Wireframes**
   - Updated or more detailed prototypes based on refined requirements
   - Helps visualize the final design and validate features

What do we do after the analysis?

Requirements specification

"Good news! He said he only needs a few more weeks to finish the first draft of the Requirements Document."

Requirements specification typically refers to the production of a document that can be systematically reviewed, evaluated, and approved.

Output - Requirements specification

Software Requirements Specification (SRS) Document:

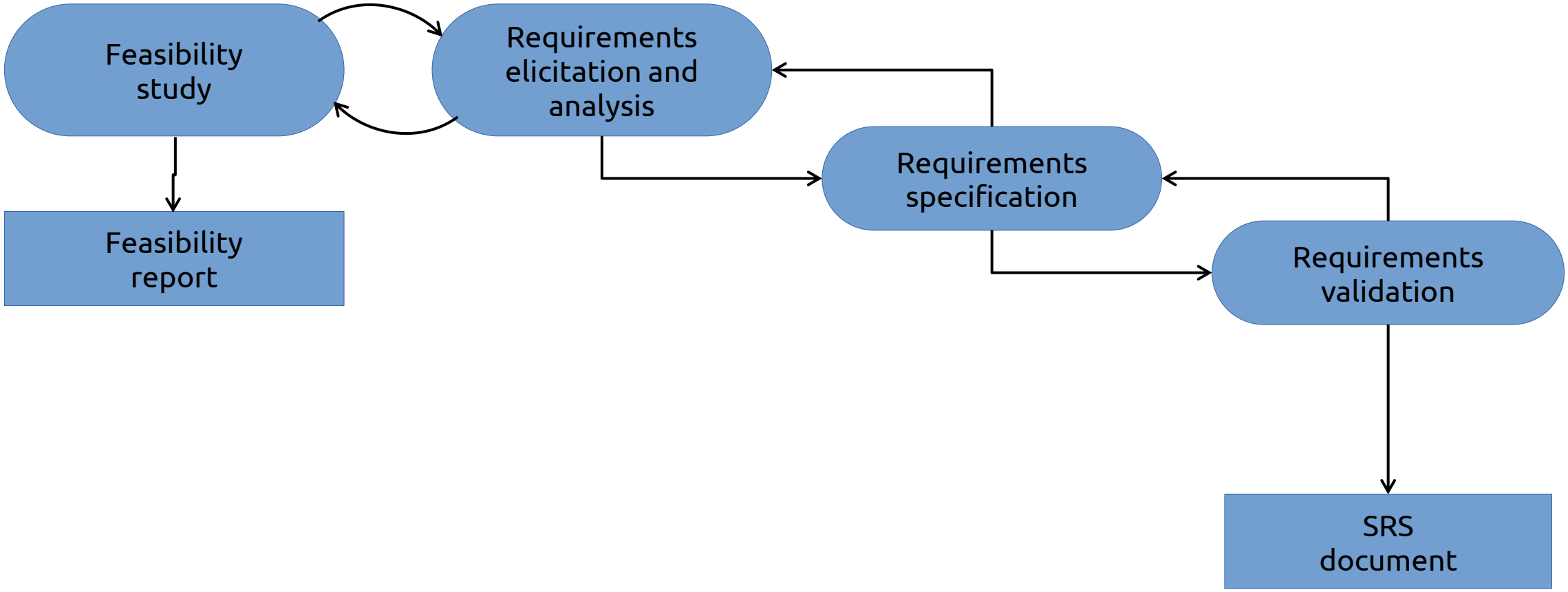
- A formal, structured document listing all system requirements
- Includes:
  - Functional requirements (what the system does)
  - Non-functional requirements (how it behaves)
  - Diagrams: use case, workflow, or data flow diagrams

Purpose: Provides a blueprint for design and development

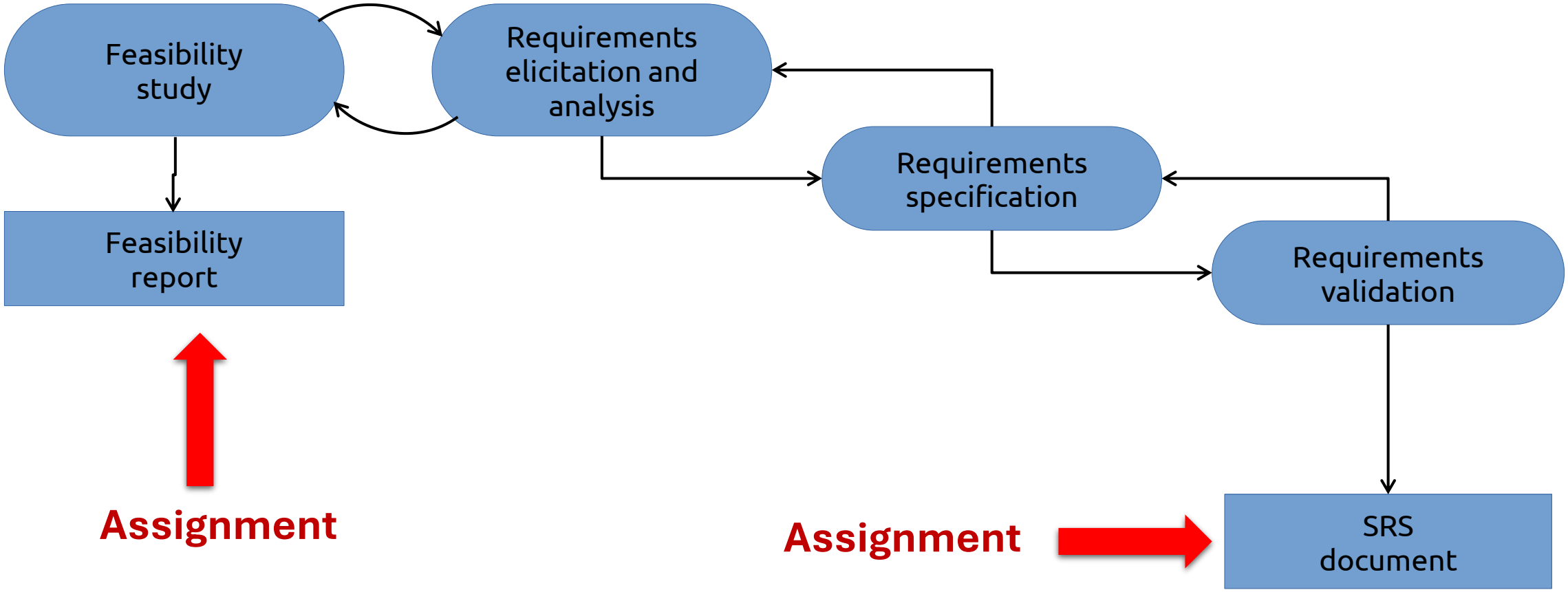
Requirements validation

The goal of this task is to check if the requirements specification is of **good quality** wrt required characteristics like correctness, completeness, unambiguity, traceability, verifiability, modifiability....

Software requirements process



Software requirements process



Assignment Instructions

How to Communicate Your Work

1. Use GitHub to submit your work.
2. Create a repository folder named SoftwareEng.
3. Give me access via email: imane.fouad@um6p.ma
4. Organize your folder:
   - TPs → for all practical exercises
   - Project → for your main project work
   - Other relevant subfolders as needed

Use latex for report

Assignment Instructions

What You Should Do

1- Update Your Project Incrementally

For every assignment, add the new work to your project.

Example: If you complete the feasibility analysis, update your project folder and slides to reflect it.

2- Update Your Slides

Include what was done in each stage of the project.

Examples:

- After the feasibility task → add a slide on feasibility.
- After the SRS task → add a slide summarizing functional and non-functional requirements.

3 - Maintain Versioning

Each update should be committed on GitHub with clear commit messages.

Keep your slides and documents synchronized with your project folder.

Assignment Instructions

Deadline: October 7th

Software requirements process

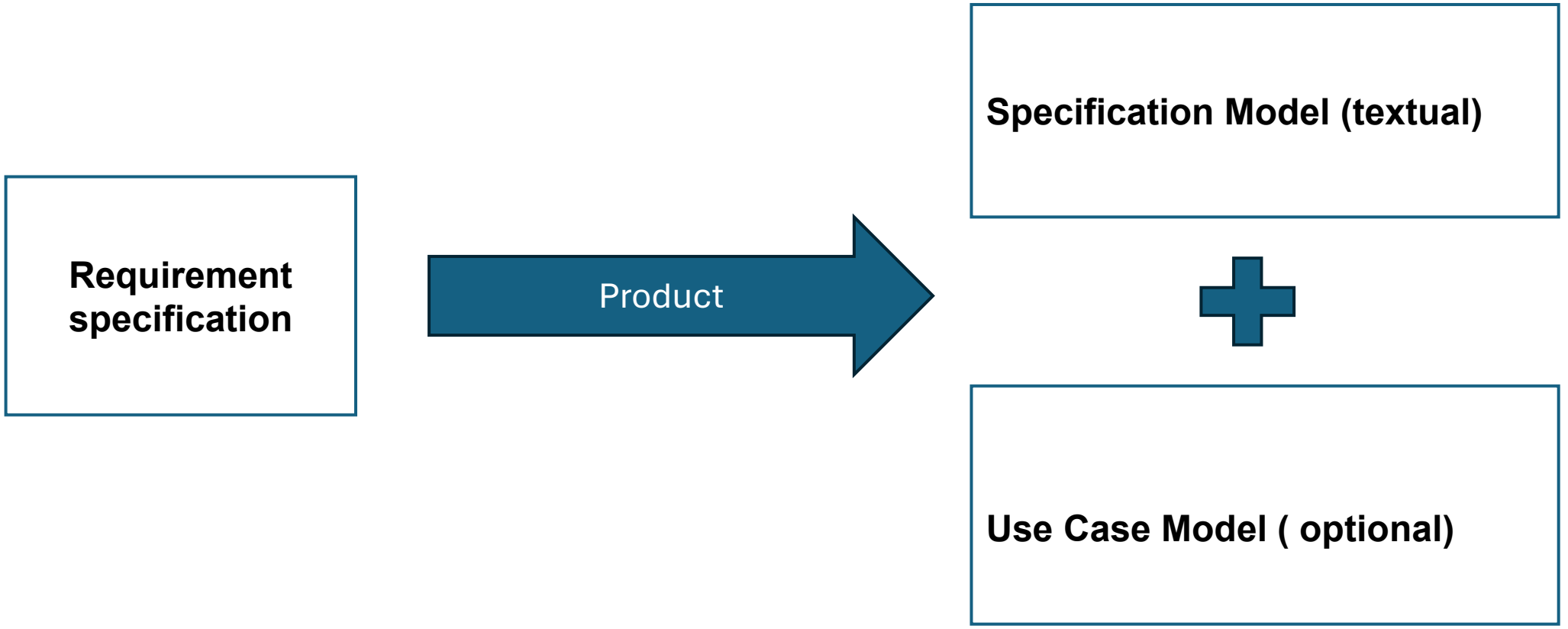
Section 2 - Discussion

Quiz

Specification Model

Section 3

Requirement Models



Specification Model

A **set of well-formed sentences**

Statements with a **uniform format**

Each specification has a **unique number**

Each specification describes **only one system function**

Specifications can be **Functional:** describing a business aspect or **Non-functional:** describing a technical aspect

Can be written using a **text editor**

Helps with **understanding, managing, and communicating requirements**

Formulation

Name of system or sub system

(ID) The **(system)** shall **<function>**

Unique identifier for each requirement

Action or behavior the system must perform

Example of Requirement Formulation

Functional Requirements:

- The ATM system shall verify the validity of the inserted card.
- The ATM system shall validate the PIN entered by the user.
- The ATM system shall allow a maximum withdrawal of 5000 DH per user.

Non-Functional Requirements:

- The ATM system shall be implemented in C++.
- The ATM system shall use 256-bit encryption for sensitive data.
- The ATM system shall verify the PIN in less than 3 seconds

Specification Priority

- Each specification must have a **priority**
- Priority helps identify the **most important requirements**
- Makes **planning and scheduling** easier
- Priority can be a **numeric scale** (e.g., 1 to 5)
- **MoSCoW method** can also be used to assign priorities

MoSCoW Method – Prioritization

Priority	Description
M (Must have)	Mandatory and fundamental requirement of the system
S (Should have)	Important, but can be omitted under certain conditions
C (Could have)	Optional, implemented if time/resources allow
W (Want to have)	Deferred requirements for later versions

Specification Attributes

Attribute	Description / Example
Status	**Proposed**: The requirement is still under discussion and not yet approved. **Approved**: The requirement is validated and ready to be implemented. **Rejected**: The requirement will not be implemented. **Implemented**: The requirement has already been implemented in a release.
Criticality	**Critical**: Must be implemented, system is not acceptable without it. **Important**: Can be skipped, but leaving it out affects usability significantly. **Useful**: Optional, skipping it has minimal impact on the system.

Specification Attributes

Attribute	Description
Effort	Approximate estimation of the resources and time required to implement the requirement.
Risk	Relative risk associated with this requirement: High, Medium, or Low.
Stability	Likelihood that the requirement will change over time.
Target	The product version or release in which the requirement is planned to be implemented.

Scenario - Online Shopping

“A customer browses a product catalog, searches for items, and adds desired products to their shopping cart. To complete a purchase, the customer provides payment and shipping information. The system verifies the payment securely and confirms the transaction. It should respond quickly to searches, and support multiple users simultaneously.”

Scenario - Online Shopping

ID	Requirement	State	Priority	Effort	Target
FR1	The system shall allow customers to browse the product catalog	Proposed	Must Have	5 days	V1
FR2	The system shall allow customers to search for products	Proposed	Must Have	3 days	V1
FR3	The system shall allow customers to add products to the shopping cart	Proposed	Must Have	4 days	V1
FR4	The system shall allow customers to remove items in the cart	Proposed	Must Have	3 days	V1
FR5	The system shall allow customers to provide payment and shipping information at checkout	Proposed	Must Have	5 days	V1
FR6	The system shall verify the customer’s payment	Proposed	Must Have	6 days	V1
FR7	The system shall confirm and validate the order	Proposed	Must Have	4 days	V1

Scenario - Online Shopping

ID	Requirement	State	Priority	Effort	Target
FR1	The system shall allow customers to browse the product catalog	Proposed	Must Have	5 days	V1
FR2	The system shall allow customers to search for products	Proposed	Must Have	3 days	V1
FR3	The system shall allow customers to add products to the shopping cart	Proposed	Must Have	4 days	V1
FR4	The system shall allow customers to remove items in the cart	Proposed	Must Have	3 days	V1
FR5	The system shall allow customers to provide payment and shipping information at checkout	Proposed	Must Have	5 days	V1
FR6	The system shall verify the customer’s payment	Proposed	Must Have	6 days	V1
FR7	The system shall confirm and validate the order	Proposed	Must Have	4 days	V1

Scenario - Online Shopping

ID	Requirement	State	Priority	Effort	Target
NFR1	The system shall respond to catalog searches within 2 seconds	Proposed	Must Have	3 days	V1
NFR2	All payment and personal data shall be encrypted	Proposed	Must Have	5 days	V1
NFR3	The system shall support multiple users simultaneously	Proposed	Should Have	4 days	V1

What makes a good software requirement specification (SRS)?

It should be correct

IEEE Recommended Practice for
Software Requirements Specifications

An SRS is correct if, and only if, every requirement stated therein is one that the software should/shall meet.

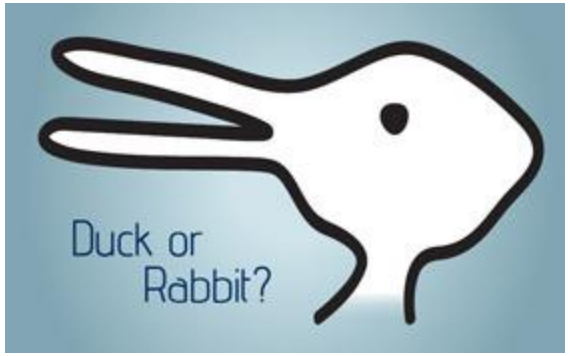
There is no tool or procedure that ensures correctness.

The SRS should be compared with other project documentation, and with other applicable standards, to ensure that it agrees.

Alternatively, the customer or user can determine if the SRS correctly reflects the actual needs.

It should be unambiguous

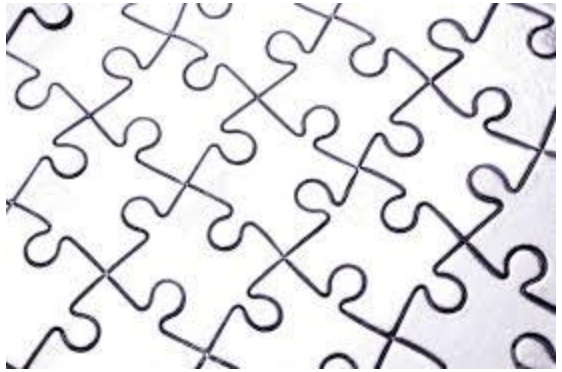
IEEE Recommended Practice for
Software Requirements Specifications



An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation.

As a minimum, this requires that each concept/characteristic of the product be described using a single unique term.

It should be complete



IEEE Recommended Practice for Software Requirements Specifications

An SRS is complete if, and only if it includes:

All significant requirements.

Definition of the responses of the software to all realizable classes of input data in all realizable classes of situations. Note that it is important to specify the responses to both valid and invalid input values.

Full labels and references to all figures, tables, and diagrams in the SRS and definition of all terms and units of measure.

It should be consistent

IEEE Recommended Practice for
Software Requirements Specifications

SRS is internally consistent if, and only if, no subset of individual requirements described in it conflict.

Consistency refers to internal consistency. If an SRS does not agree with some higher-level document, such as a system requirements specification, then it is not correct.

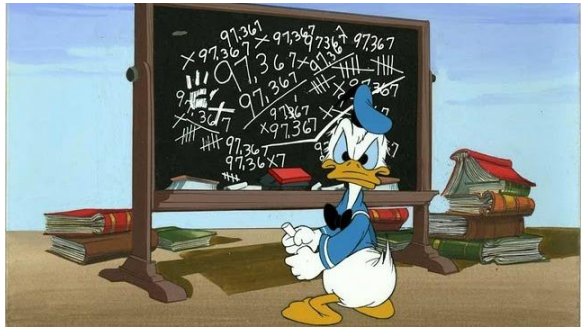
It should be consistent

IEEE Recommended Practice for
Software Requirements Specifications

Two or more requirements may describe the same real-world object but use different terms for that object.

For example, a program's request for a user input may be called a prompt in one requirement and a cue in another.

It should be verifiable



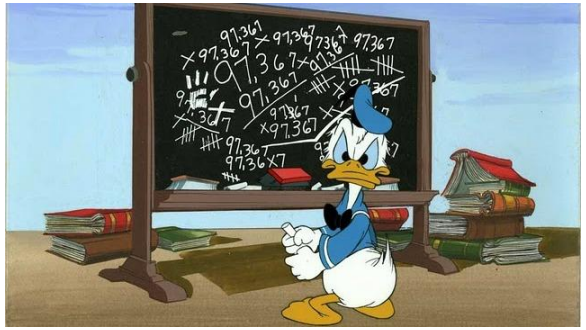
IEEE Recommended Practice for
Software Requirements Specifications

An SRS is verifiable if, and only if, every requirement stated therein is verifiable.

A requirement is verifiable if, and only if, there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement.

In general any ambiguous requirement is not verifiable.

It should be verifiable



IEEE Recommended Practice for
Software Requirements Specifications

Examples of non-verifiable requirements are statements such as:

"works well", "good human interface" and "shall usually happen"

An example of a verifiable statement is:

Output of the program shall be produced within 20 s of event X 60% of the time; and shall be produced within 30 s of event Y 100% of the time.

This statement can be verified because it uses concrete terms and measurable quantities.

It should be modifiable

IEEE Recommended Practice for
Software Requirements Specifications

An SRS is modifiable if, and only if, its structure and style are such that any changes to the requirements can be made easily.

SRS must have a coherent and easy-to-use organization with a table of contents, an index, and explicit cross-referencing.

Not be redundant (i.e., the same requirement should not appear in more than one place in the SRS).

Express each requirement separately, rather than intermixed with other requirements.

It should be traceable

IEEE Recommended Practice for
Software Requirements Specifications

An SRS is traceable if the origin of each of its requirements is clear and facilitates the referencing of each requirement in future development or documentation.

Backward traceability - each requirement must explicitly reference its source in earlier documents.

Forward traceability - each requirement in the SRS must have a unique name or reference number.

IEEE 830-1998 Standard – Objectives

- Help software customers to accurately describe what they wish to obtain
- Help software suppliers to understand exactly what the customer wants
- Help participants to:
  - Develop a template (format and content) for the software requirements specification (SRS) in their own organizations
  - Develop additional documents such as SRS quality checklists or an SRS writer's handbook

IEEE 830-1998 Standard – Benefits

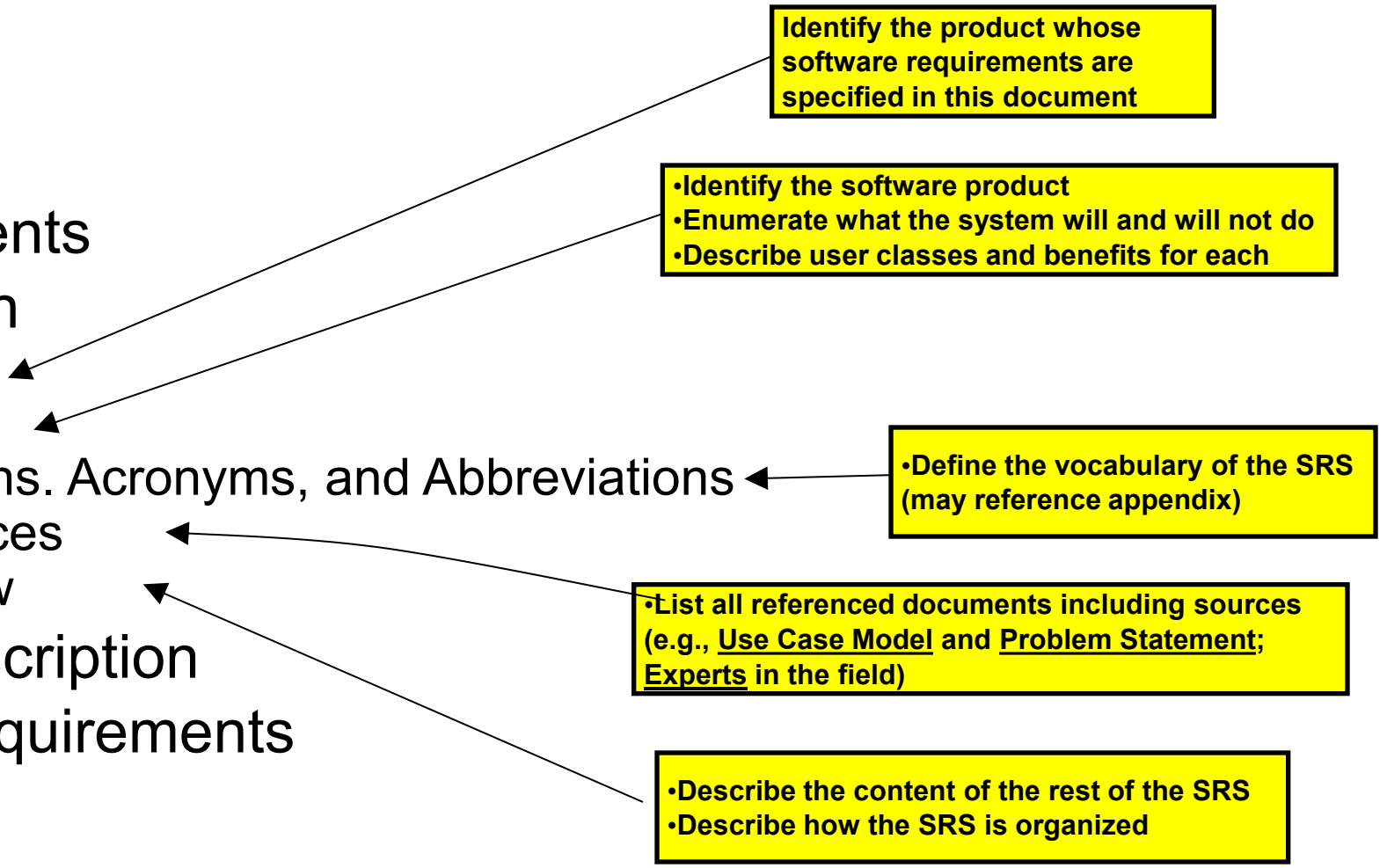
- Establish the basis for agreement between the customers and the suppliers on what the software product is to do

- Reduce the development effort
  - Forced to consider requirements early → reduces later redesign, recoding, retesting
- Provide a basis for realistic estimates of costs and schedules

- Provide a basis for validation and verification
- Facilitate transfer of the software product to new users or new machines
- Serve as a basis for enhancement requests

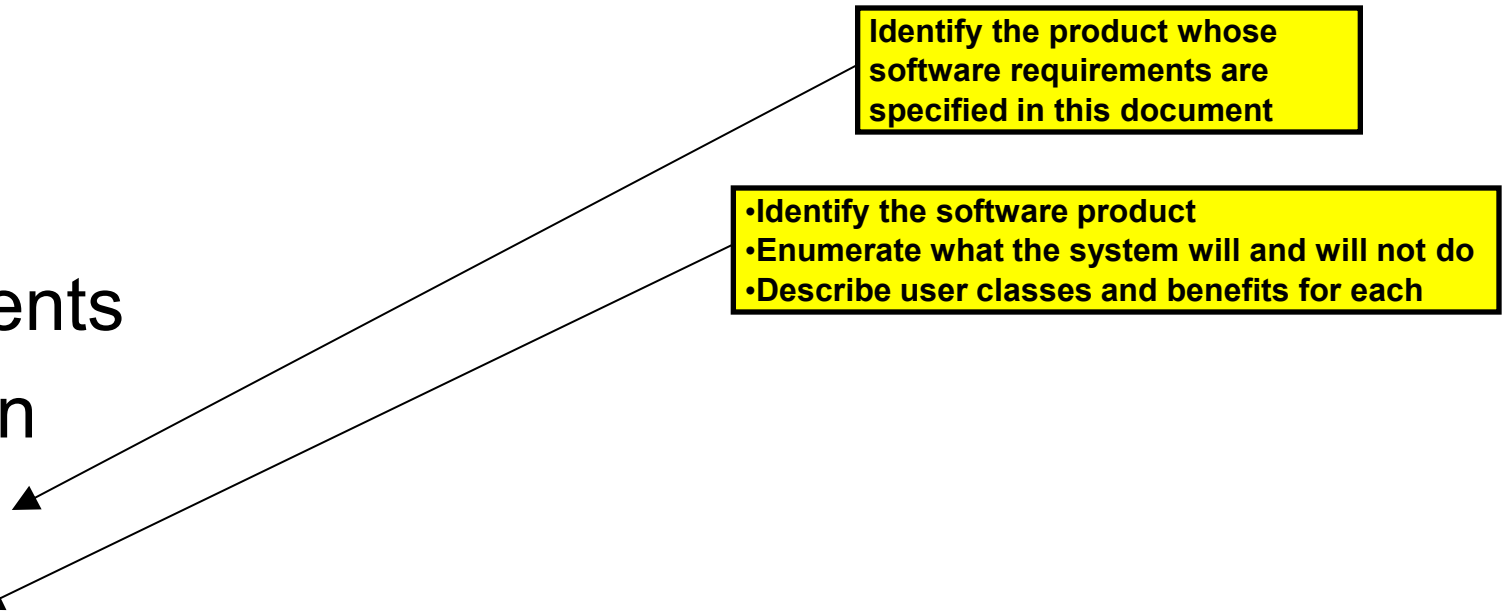
IEEE 830-1998 Standard – Section 1 of SRS

- Title
- Table of Contents
- 1. Introduction
  - 1.1 Purpose
  - 1.2 Scope
  - 1.3 Definitions. Acronyms, and Abbreviations
  - 1.4 References
  - 1.5 Overview
- 2. Overall Description
- 3. Specific Requirements
- Appendices
- Index



IEEE 830-1998 Standard – Section 1 of SRS

- Title
- Table of Contents
- 1. Introduction
  - 1.1 Purpose
  - 1.2 Scope
- 2. Overall Description
- 3. Specific Requirements
- Appendices
- Index



IEEE 830-1998 Standard – Section 2 of SRS

- Title
- Table of Contents
- 1. Introduction
- 2. Overall Description
  - 2.1 Product Perspective
  - 2.2 Product Functions
  - 2.3 User Characteristics
  - 2.4 Constraints
  - 2.5 Assumptions and Dependencies
- 3. Specific Requirements
- 4. Appendices
- 5. Index

•Present the business case and operational concept of the system
•Describe how the proposed system fits into the business context
•Describe external interfaces: system, user, hardware, software, communication
•Describe constraints: memory, operational, site adaptation

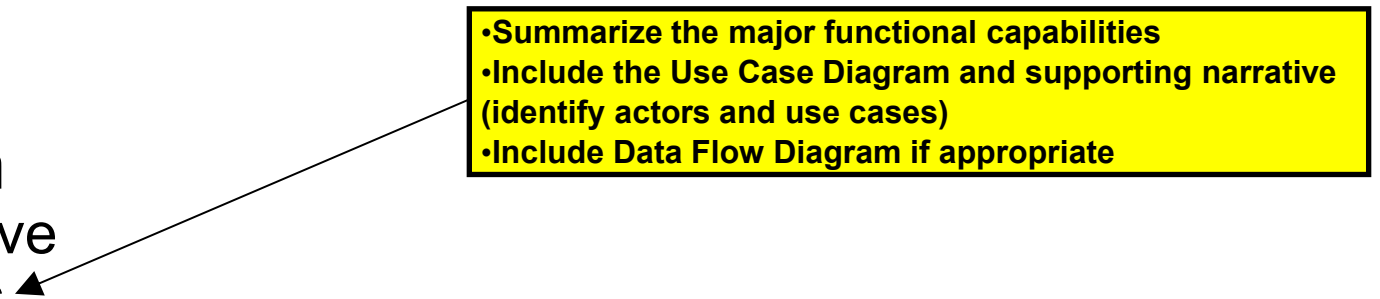
•Summarize the major functional capabilities
•Include the Use Case Diagram and supporting narrative (identify actors and use cases)
•Include Data Flow Diagram if appropriate

•Describe and justify technical skills and capabilities of each user class

•Describe other constraints that will limit developer's options; e.g., regulatory policies; target platform, database, network software and protocols, development standards requirements

IEEE 830-1998 Standard – Section 2 of SRS

- Title
- Table of Contents
- 1. Introduction
- 2. Overall Description
  - 2.1 Product Perspective
  - 2.2 Product Functions
  - 2.3 User Characteristics
  - 2.4 Constraints
  - 2.5 Assumptions and Dependencies
- 3. Specific Requirements
- 4. Appendices
- 5. Index



Overall Description (Product Functions) - University Library System

Section 2.2 Product Functions

- Catalog management (add, update, delete books).
- Search functionality (by title, author, ISBN).
- Borrow and return book processing.
- Report generation (monthly usage, overdue items).

IEEE 830-1998 Standard – Section 3 of SRS (1)

- …
- 1. Introduction
- 2. Overall Description
- 3. Specific Requirements
- 4. Appendices
- 5. Index

Specify software requirements in sufficient detail to enable designers to design a system to satisfy those requirements and testers to verify requirements

State requirements that are externally perceivable by users, operators, or externally connected systems

Requirements should include, at a minimum, a description of every input (stimulus) into the system, every output (response) from the system, and all functions performed by the system in response to an input or in support of an output

(a) Requirements should have characteristics of high quality requirements
(b) Requirements should be cross-referenced to their source.
(c) Requirements should be uniquely identifiable
(d) Requirements should be organized to maximize readability

Section 3 Overview

Specific Requirements

- Functional requirements (what the system shall do).
- Non-functional requirements (how).

3.1 Functional Requirements

Section 3.1.1: Search functionality

- FR1: The system shall allow students to search books by title.
- FR2: The system shall allow students to search books by author.
- FR3: The system shall allow students to search books by ISBN.

Functional Requirements (Borrowing & Cataloging)

Section 3.1.2: Borrowing & Cataloging

- FR4: The system shall allow students to borrow available books using their ID.
- FR5: The system shall allow students to return borrowed books.
- FR6: The system shall allow librarians to add, update, or delete book records.
- FR7: The system shall track borrowing history and fines.

Non-Functional Requirements (Performance & Reliability)

Section 3.2 Non-Functional Requirements

- NFR1: The system shall return search results within 2 seconds for up to 500 users.
- NFR2: The system shall be available 99% of the time.
- NFR3: The system shall recover from failures within 5 minutes.

Non-Functional Requirements (Security & Usability)

Section 3.2 Non-Functional Requirements (More)

- NFR4: The system shall require secure login with university credentials.
- NFR5: Only librarians and administrators shall have permission to modify catalog records.
- NFR6: New users shall be able to learn basic functions within 10 minutes without training

Specification Model

Section 3 - Discussion

Quiz