Software Engineering

Block 3: Software modeling
– Part I: Software analysis –
(Software Requirements Specification)

ESCI upf.
School of International Studies
BioInformatics-Software Engineering

Outline

*In this session we will discuss how to generate the **content** within the 'Software Requirements Specification' document.*
And we will also focus on the correct way to generate the Functional Requirements.

- Introduction to software requirements
  - Types of requirements
  - Functional and non-Functional requirements
- Understanding of the problem domain
  - Requirements gathering
  - Modeling methods
- Requirements specification
  - The S.R.S. document
  - Review and validation of the requirements

Why do we need an analysis phase?

*We need to create a **solution** (software) that resolves a certain problem.*

Analysis objectives

- As the complexity of the problems increases, it is necessary to perform a **software requirements analysis** phase to achieve the following two objectives:
  - Understanding the problem
  - Generate the requirements specification

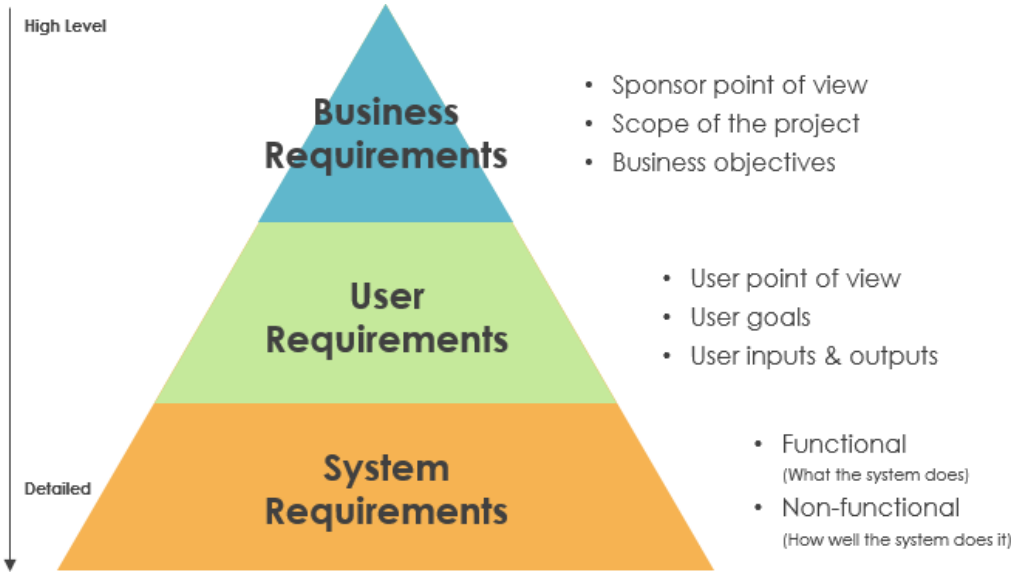
- And what is a software requirement?

  - **Requirements**: Set of ideas that the client has about what the software to be developed should be and its behavior. They are the characteristics of the system.

Are description of features and functionalities of the target system.

Therefore, we need to understand the problem and identify the requirements.

Types of requirements



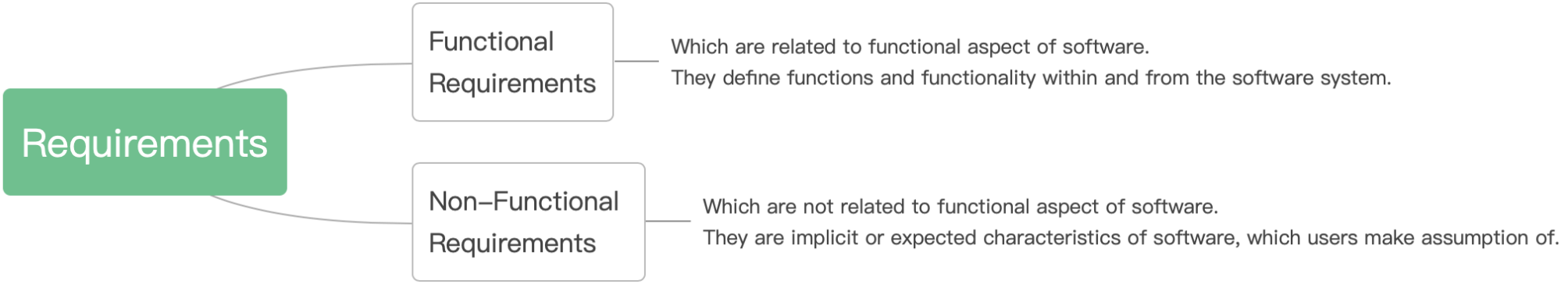
The requirements abstraction pyramid:

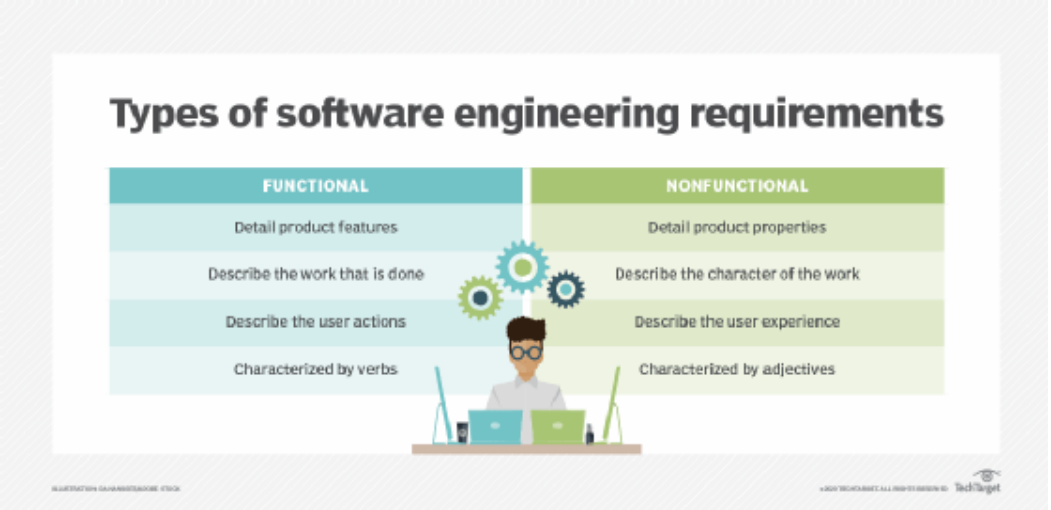
- Business Requirements:
  - Generic objectives of the client.
- User Requirements:
  - Natural language description of the services to be provided by the system and its operational limitations. Written by clients.
- System Requirements:
  - Structured document that provides a detailed description of the functions, services and operational constraints of the system. It defines what needs to be implemented so that it can be part of a contract between the customer and the developer. Written by engineers.

*From the user requirements **we** will generate the system requirements.*

System requirements

- The System Requirements are classified into two main categories:
  - **Functional Requirements**
  - **Non-Functional Requirements**





Functional vs. Non-Functional Req.

FUNCTIONAL REQUIREMENTS	NON-FUNCTIONAL REQUIREMENTS
WHAT THE SYSTEM SHOULD DO	HOW THE SYSTEM SHOULD DO IT
	
PRODUCT FEATURES	PRODUCT PROPERTIES
	
USER REQUIREMENTS	USER EXPECTATIONS
The system must do...	*The system should be...*

Functional Requirements (FR)

- Description of the desired **behavior** for the software.
- Each functional requirement describes, in general, the relationship between inputs and outputs → Given an input or condition, what is the process to be performed to produce a result.
- It is also necessary to describe the expected behavior when unexpected inputs or conditions (errors) occur.

Functional Requirements unambiguously describe the features of the system.

Example: "The system must allow the user to submit feedback through a contact form in the app."

Non-Functional Requirements (NFR)

- Non-functional requirements are **constraints** proposed by the customer or by the problem itself, related to the design to be implemented *later*.
- They are generally quantifiable (aka measurable).
- We can find several different categories:
  - Performance related
  - Design related (constraints, objectives, decisions)
  - Related to external interfaces

Non-Functional Requirements limit the freedom of design.

Example: "When the submit button is pressed, the confirmation screen must load within 2 seconds."

NFRs: Performance

There are two main groups of performance requirements:

- **Static requirements** (capacity):
  - Limits on execution characteristics (number of users, memory used, disk space, etc.)
  - Example: *The web server will work for 100 simultaneous users*.
- **Dynamic requirements**:
  - Limits on the behavior while the system is running (number of connected users, response time, performance, etc.)
  - Example: *The web server should provide a response in less than 3 seconds*.

NFRs: Design

There are three main Non-functional Requirements related to design:

- **Design constrains**: when design freedom is restricted in these areas:
  - Standards enforcement: formats, procedures, audition, etc.
  - Hardware limitations: computers/servers, O.S., language, licenses, storage, etc.
  - Errors management: tolerance to errors, recovery from errors, high availability, etc.
  - Security: access policies, activity logging, etc.
- **Design objectives**: general guidelines that apply to the entire design
  - Usually related to the quality of the product: usable, user friendly, maintainable, etc.
- **Design decisions**: when a specific design decision has already been made
  - For instance, implementation details about some algorithms, other low-level aspects…

Each group is completely different, so don't mix them up!

NFRs: External interfaces

Requirements on external interfaces define the interaction characteristics between **the system** and people, hardware and other software modules:

- With persons are the restrictions imposed on the U.I.
- Hardware access protocols are described for the equipment
- In the case of external software, specify the restrictions to be complied with

Summary Note:

Five types of NFRs *→ Performance, Design Constrains, External Interfaces, Design Objectives & Design Decisions*

NFRs: The metrics

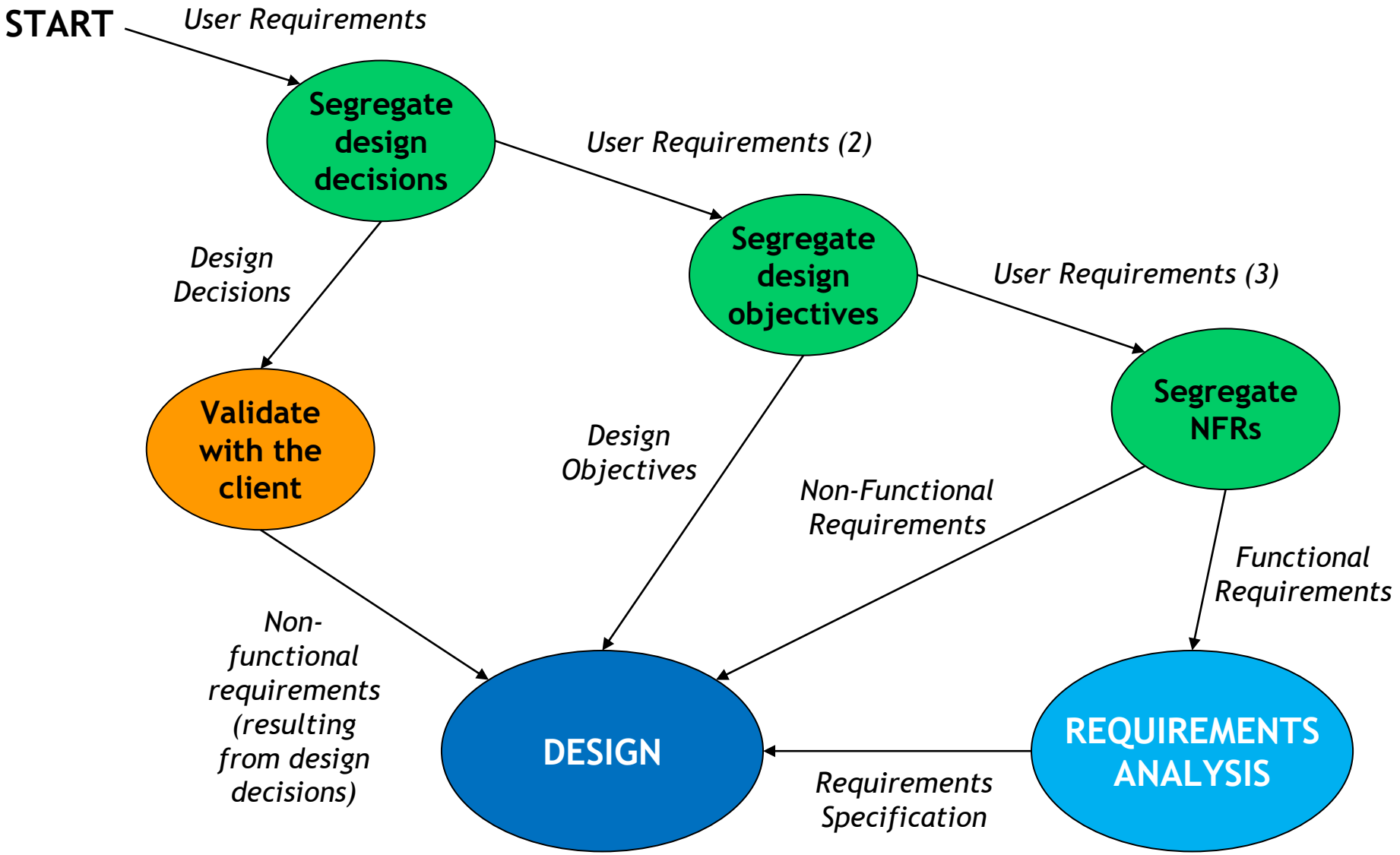
Non-functional requirements are usually quantifiable:

- They are only well defined when we can expose a metric for them.

- Examples of common metrics:

Property	Metric
Speed	Transactions/sec. ; Response time/requests ; screen refresh time
Length	MBytes ; Number of files
Easy to use	Time to learn ; Number of help pages
Reliability	Mean time between failures ; Error ratio ; Availability factor
Robustness	Recovery time ; Probability of data corruption in failures
Portability	Number of supported platforms ; Percentage of target dependencies

Requirements: Identification process



Requirements: Creation process

- The process is comprised of four distinct stages, as outlined below:
    - **Feasibility Study**: The output of this initial phase should be a feasibility study report that should contain adequate comments and recommendations for management about whether or not the project should be undertaken.
    - **Requirement Gathering**: This phase involves the identification of all the necessary elements that must be implemented.
    - **Software Requirement Specification**: This document categorizes and provides a detailed description of all the requirements, both functional and non-functional.
    - **Software Requirement Validation**: A review of the internal consistency of the requirements.

Requirements gathering

Requirements elicitation (**Req. Gathering or Capture**) is the process of generating a list of all requirements from different sources (clients, users, etc.). This task involves gaining a deep understanding of the problem domain.

- Some of the methods to obtain the requirements are:
  - Interviews
  - Surveys
  - Documentation review
  - Observation of use
  - Prototyping
  - Brainstorming

Several different methods can be used at the same time.

Interviews

Direct interviews with the client are a primary and direct source of information:

- *From the customer's point of view, we obtain the user requirements that we will transform into system requirements.*

First action to be taken. It includes:

- Talk with the client and write down the problem, the wish list of solutions, doubts, questions, etc.
- After the first iterations, at an advanced stage, we can enter the *"Joint Application Development"* (iterative meetings).
- These are several meetings every 3 or 4 consecutive days with all customers (stakeholders), some end users and some development teams in order to improve the analysis and design.

Surveys

Another direct way to receive customer feedback is through surveys:

- *Surveys are well-structured written documents used to obtain specific information.*

If this task is carried out, it should always be done after the first interviews. Particularities:

- Useful for retrieving information from many people in a short time.
- Useful in situations where it is not possible to meet with people (geographic dispersion).
- It will require a lot of effort and time.

Documentation review

The documentation review is performed by the development team itself:

- *It is the internal task of searching for information about the problem domain.*

The research of pertinent data is carried out in the following fields:

- User manuals of previous solutions.
- Standards and regulations.
- Experience gained from other projects
- Market studies.
- …

Observation of use

Live observation of the problem domain is another possible source of information:

- *The task is to externally investigate how the system currently works, without the influence of the client's thinking.*

This task includes:

- Experience the actual operations on the customer side.
- Live the problem from the users' point of view and not from the managers' point of view.
- It is important to consider that sometimes the actual operation does not conform to the protocols established by the company.

Prototyping

Build an abstract model of the application that allows the customer to ‘play’ with it and better detail their needs:

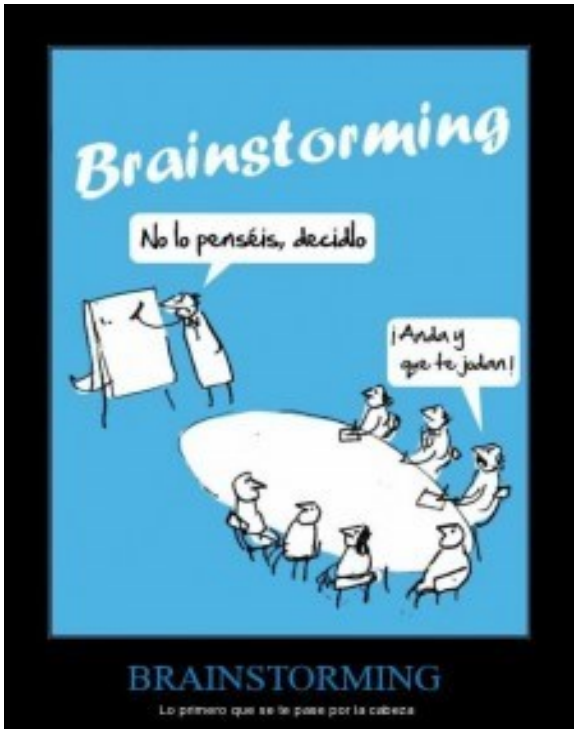
- *The prototype is an example, not a final product.*

Two types or phases with prototypes:

- Paper schematic, describing the human-machine interaction.
- Executables, in two subtypes:
  - In with: All functionalities covered with limited or not complete implementation with dummy data and parameters.
  - In depth : A subset of functionalities covered with high implementation, in general the most problematic functionalities.

Warning: It is also important to be cautious so that the client does not perceive the "prototype" as version 0 of the software. To avoid wasting time, the prototype should be built quickly and *not be reused* for the final version of the software.

Brainstorming



Brainstorming is a method used by design teams to generate ideas to solve clearly defined design problems:

- *A wide range of ideas are generated and links are made between them to find workable solutions.*

Things to consider when brainstorming:

- Usable for new products or new functionalities.
- Sessions must be managed and led for them to be profitable.
- At the end of the session, the results should be organized, classified and prioritized so that they can be used.

Tools such as whiteboards or mind-mapping can be used...

Requirements modelling methods

Understanding the problem

Problem specification

1. Problem analysis
   - Understanding the software in its context.
   - Analyze the problem from the user's point of view.

Requirements capture (*Req's elicitation*)

2. Evaluation and synthesis
   - Define software functions.
   - Focus on information flow and structure.
   - Characterize the user interface.
   - Discover design constraints.

3. Modeling
   - A description that facilitates the understanding of the problem.

Specification: Formal Description

4. Specification
   - It collects the desired functional behavior, without considering how to achieve it.

5. Review
   - Verify that both the client and the analyst have 'understood' the problem.
   - Agreement on the scope of the problem.

Validation

6. Management

Management

User stories and Scenarios

- Scenarios and user stories are **real-world examples** of how a system can be used.
- They are a description of how a system can be used for a particular task.
- Since they are based on a practical situation, stakeholders can relate to them and comment on their situation with respect to the story.
- SCRUM format of user stories:

The scenario is a formal description structured in the form of a diagram.

Principles of analysis

Each analysis method has its own notation and point of view. However, there are principles that all modeling tools associated with these methods must comply with:

- **Partition**:
  - A complex system can only be understood if it is structured in inter-related parts.
  - Therefore, the tools we use must be able to describe a problem from its parts.
- **Abstraction**:
  - We need to be able to define an entity or a problem in general terms. Removing all the details (referencing them separately) to simplify understanding.
  - Consequently, the tools used must allow us to find the details progressively.
- **Projection**:
  - It is necessary to define the system from different points of view (projections).
  - The combination of points of view ensures a complete understanding of the system, which must be supported by the tools.

The result of the Analysis:

The Software Requirement Specification is delivered as a technical document.

Requirements specification: Document

- Once requirements are clear, they shall be covered by the ***"Software Requirements Specification"*** **(SRS)**.
- The purpose of the SRS is multiple:
  - Contractual: Development will implement this and nothing more.
  - Documentation: Working baseline (reference) to be used by the designer and development team.

In some contexts, instead of 'Software Requirements Specification' (SRS) one speaks of 'Software Requirements Document' (SRD). But both refer to the same content and objective.

Specification properties

- The SRS specifies the expected behavior, not the actual implementation.

- Therefore, it must satisfy the following properties:
  - Correctness
  - Comprehensible
  - No ambiguities
  - Completeness
  - Consistent
  - Ordered
  - Verifiable
  - Modifiable
  - Traceable

Specification properties

- **Correctness**: The SRS only lists the requirements that shall be implemented.
- **No ambiguities**: The requirements shall be interpretable in a unique way, two developers shall understand the same:
  - Natural language is ambiguous by default.
  - Formal language is preferred
- **Completeness**: Only if SDR includes:
  - All the requirements related with: functionality, performance, external interfaces, and design restrictions.
  - Defines all the possible answers again all the possible inputs, and in all possible scenarios.
  - Define all the terms and used units.

Specification properties

- **Consistent**: If we cannot find any subset of requirements conflicting with other requirements.
  - Inconsistency in the characteristics of the same element (i.e. format).
  - Logical inconsistency between two processes (i.e. sequence vs. concurrency).
  - Inconsistency in the word and names (i.e. use of synonyms).
- **Ordered**: Group requirements by categories and relevance.
- **Verifiable**: Each requirement must be verifiable, there must be a process (finite with an affordable cost) that guarantees the correct implementation of the requirement. Three types of verification:
  - By Test: Software that test software.
  - By Design: The design ensures the implementation.
  - By Code Review: A manual review shall be executed.

Specification properties

- **Modifiable**: The structure and style of the document should allow for uncomplicated changes, so it is important not to be redundant in concepts and to cover all requirements without repetition.
- **Traceable**: In two directions:
  - Backward: The origin of the requirement shall be clear.
  - Forward: Each requirement shall have a unique identification to be referenced in any point in the development.

Document format

- The format of the "Software Requirements Specification" must be **easy to read** and well **structured**. The requirements should be clear, easy to understand, complete, and consistent.
- There are several standards that defines how the Specification Document shall be.
  - As an example, the ANSI/IEEE organization has established the IEEE 830 standard.
  - This standard proposes the following sections:

1. **Introduction**
   1.1 Aim of the product
   1.2 Scope
   1.3 Acronyms
   1.4 References
   1.5 General view / Introduction
2. **General Description**
   2.1 Product perspective
   2.2 Product functions
   2.3 User characteristics
   2.4 General Restrictions
   2.5 Assumptions and dependencies
3. **Requirements specification**
4. **Appendixes**
5. **Index**

- *User Requirements are expressed in natural language.*
- *Technical requirements are expressed in structured language.*

Document format

- And main section of the document should cover:

```
3. Requirements specification
 3.1 Functional requirements
  3.1.1  Functional requirement - 1
   3.1.1.1 Introduction
  3.1.1.2 Inputs
  3.1.1.3 Process
  3.1.1.4 Outputs
  3.1.2 Functional requirement - 1
   ...
 3.2  Non-Functional requirements
  3.2.1  Extern interfaces requirements
   3.2.1.1 Of Users
   3.2.1.2 Hardware
   3.2.1.3 Software
   3.2.1.4 Communications
  3.2.2 Performance requirements
  3.2.3 Design restrictions
   …
  3.2.6 Other NF requirements
```

Requirement tables

- Each requirement should be formatted to cover all basic information.
- Typical tables used for this purpose are shown below:

Code : FR-01-001	Users	Verification: D, R
The system shall manage a user access.		
Parent:		

Code : FR-01-002	User information	Verification: D, R
User profile shall contain: •Username : Alphanumeric string from 3 to 8 characters. •Password: Alphanumeric string from 6 to 8 characters. •Email •DNI •Photography : 300x300 pixels in color		
Parent: FR-01-001		

Requirement Validation

Validation is the process of ensuring that the specified requirements align with the customer's needs. It involves identifying inconsistencies within the requirements.

Techniques for validating requirements include:

- Requirements Reviews: interact with the customer to gather requirements, and the system developers read the requirements in the document and investigate in great detail to check for errors, inconsistency, conflicts, and any ambiguity.
- Prototyping: used when the requirements are not yet fully defined. So, we make a quick design of the system to validate the requirements. If it fails, we then refine it, and check again, until it meets the customer's needs.
- Test-case Generation: logical check of actions.

Date run	Time run	Tester	Pass/Fail	Summary of Failure

Assumptions:			
Here you write your assumptions; the pre-conditions; or the initial state.			
System Requirement Covered:			
The number of the function that's being tested			
Exclusions:			
Exclusions, like any un-desired values.			
Setup: Steps taken			
Write the steps that should be taken by the user for this function			
User Action	Value(s)	Expected Results	Pass/Fall/Untested
What are the inputs	List all the values that will be entered by the user	The expected output	

The term "tests" here doesn't mean to write and run some code for every function. It means to write a textual description of the "inputs", "expected value", and "steps taken" to perform each function.

Validation: The verification matrix

- Another important element related to requirements is the **verification matrix**.
  - This matrix links the requirements to the other elements developed during the project.
  - The objective is to trace the relationships derived from a requirement to other elements, and it helps to trace requirements that are only defined in the SRS but are not present elsewhere.
- The verification matrix can have any needed column:
  - Requirement: The requirement ID to track
  - Use Case: The Use case ID (next block in the subject) that defines behavior related with the requirement
  - Test: ID of the test proposed to verify that requirement
  - Test Report: Document that shows the results of the test
  - Code: Module, Source code, that implements things related with the requirement

Verification Matrix

- Example of a verification matrix:

Requirement	Use Case	Test
REQ-F-1-01	UC-01	TP-01

Any requirement with no linked items is a requirement that is not analyzed, not implemented and not tested. So, it is a RED flag!

CONCLUSIONS

- Main objective of the analysis: to understand the scope of the problem and gather System Requirements.
- User Histories: a useful tool to obtain the Functional Requirements.
- All requirements must be well classified according to their type.
- A good SRS document should cover the entire problem domain.

Recommended reading to find out more:
https://www.projectmanager.com/blog/requirements-gathering-guide