# CS 410/510 - Software Engineering

# Requirements Engineering

Reference: Sommerville, Software Engineering, 10 ed., Chapter 4

## The big picture

**Requirements engineering** (RE) is 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. Requirements may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification. As much as possible, requirements should describe **what** the system should do, but **not how** it should do it.

Two kinds of requirements based on the intended purpose and target audience:

**User requirements**
: High-level abstract requirements written as statements, in a natural language plus diagrams, of what services the system is expected to provide to system users and the constraints under which it must operate.

**System requirements**
: Detailed description of what the system should do including the software system's functions, services, and operational constraints. The system requirements document (sometimes called a functional specification) should define exactly what is to be implemented. It may be part of the contract between the system buyer and the software developers.

Three classes of 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

Functional requirements describe **functionality** or system services. They 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.

Problems arise when requirements are not precisely stated. Ambiguous requirements may be interpreted in different ways by developers and users. In principle, requirements should be both

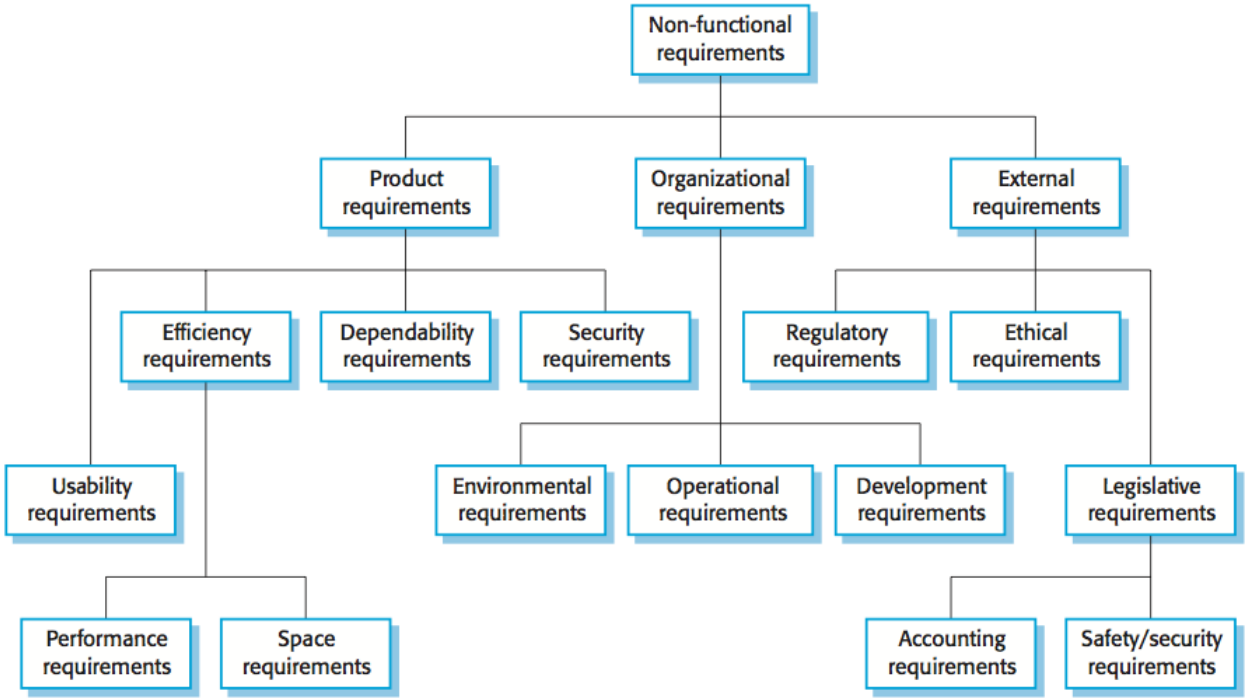
- **Complete**: they should include descriptions of all facilities required, and
- **Consistent**: there should be no conflicts or contradictions in the descriptions of the system facilities.

In practice, it is impossible to produce a complete and consistent requirements document.

## Non-functional requirements

Non-functional requirements 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.

Non-functional requirements may affect the overall architecture of a system rather than the individual 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.




Three classes of non-functional requirements:

**Product requirements**
: Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.

**Organizational requirements**
: Requirements which are a consequence of organizational 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.

Non-functional requirements may be very difficult to state precisely and imprecise requirements may be difficult to verify. If they are stated as a **goal** (a general intention of the user such as ease of use), they should be rewritten as a **verifiable** non-functional requirement (a statement using some **quantifiable metric** that can be objectively tested). Goals are helpful to developers as they convey the intentions of the system users.

## Domain requirements

The system's operational domain imposes requirements on the system. Domain requirements may be new functional requirements, constraints on existing requirements or define specific computations. If domain requirements are not satisfied, the system may be unworkable. Two main **problems** with domain requirements:

**Understandability**
: Requirements are expressed in the language of the application domain, which is not always 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.

## Requirements engineering process

Processes vary widely depending on the application domain, the people involved and the organization developing the requirements. In practice, requirements engineering is an **iterative process**, in which the following generic activities are interleaved:

- Requirements **elicitation**;
- Requirements **analysis**;
- Requirements **validation**;
- Requirements **management**.

## Requirements elicitation and analysis

Software engineers work with a range of system **stakeholders** to find out about the application domain, the services that the system should provide, the required system performance, hardware constraints, other systems, etc. Stages include:

**Requirements discovery**
: Interacting with stakeholders to discover their requirements. Domain requirements are also discovered at this stage.

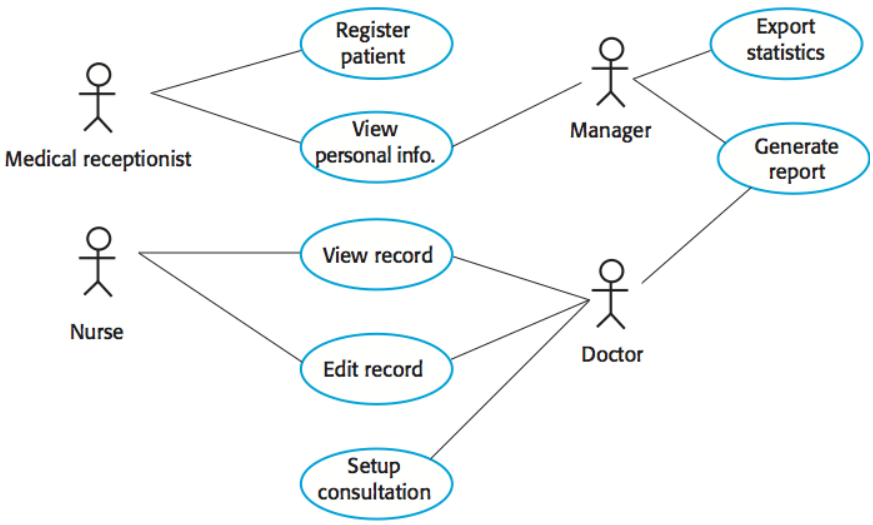
**Requirements classification and organization**
: Groups related requirements and organizes them into coherent clusters.

**Prioritization and negotiation**
: Prioritizing requirements and resolving requirements conflicts.

**Requirements specification**
: Requirements are documented and input into the next round of the spiral.

Closed (based on pre-determined list of questions) and open **interviews** with stakeholders are a part of the RE process. **User stories** and **scenarios** are real-life examples of how a system can be used, which are usually easy for stakeholders to understand. Scenarios should include descriptions of the starting situation, normal flow of events, what can go wrong, other concurrent activities, the state of the system when the scenario finishes.

**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.




**Problems** to look for during requirements elicitation and analysis:

- **Stakeholders don't know what they really want.**
- Stakeholders express requirements in their own terms.
- Different stakeholders may have conflicting requirements.
- Organizational and political factors may influence the system requirements.
- The requirements change during the analysis process.
- New stakeholders may emerge and the business environment may change.

## Requirements specification

Requirements specification is the process of **writing down the user and system requirements** in a requirements document. User requirements have to be understandable by end-users and customers who do not have a technical background. System requirements are more detailed requirements and may include more technical information. The requirements may be part of a contract for the system development and it is important that these are as complete as possible.

In principle, requirements should state what the system should do and the design should describe how it does this. In practice, **requirements and design are inseparable**.

User requirements are almost always written in **natural language** supplemented by appropriate diagrams and tables in the requirements document. System requirements may also be written in natural language but other notations based on forms, graphical system models, or mathematical system models can also be used. Natural language is expressive, intuitive and universal. This means that the requirements can be understood by users and customers.

**Structured natural language** is a way of writing system requirements where the freedom of the requirements writer is limited and all requirements are written in a standard way. This approach maintains most of the expressiveness and understand-ability of natural language but ensures that some uniformity is imposed on the specification.

## Requirements validation

Requirements validation is concerned with demonstrating that the requirements define the system that the customer really wants. Requirements error costs are high so validation is very important.

What **problems** to look for:

- **Validity**: does the system provide the functions which best support the customer's needs?
- **Consistency**: are there any requirements conflicts?
- **Completeness**: are all functions required by the customer included?
- **Realism**: can the requirements be implemented given available budget and technology?
- **Verifiability**: can the requirements be checked?

Requirements validation **techniques**:

**Requirements reviews**

Systematic manual analysis of the requirements. Regular reviews should be held while the requirements definition is being formulated. What to look for:

- **Verifiability**: is the requirement realistically testable?
- **Comprehensibility**: is the requirement properly understood?
- **Traceability**: is the origin of the requirement clearly stated?
- **Adaptability**: can the requirement be changed without a large impact on other requirements?

**Prototyping**

Using an executable model of the system to check requirements.

**Test-case generation**

Developing tests for requirements to check testability.

## Requirements change

Requirements management is the process of managing changing requirements during the requirements engineering process and system development. New requirements emerge as a system is being developed and after it has gone into use. Reasons why requirements change after the system's deployment:

- The business and technical environment of the system always changes after installation.
- The people who pay for a system and the users of that system are rarely the same people.
- Large systems usually have a diverse user community, with many users having different requirements and priorities that may be conflicting or contradictory.