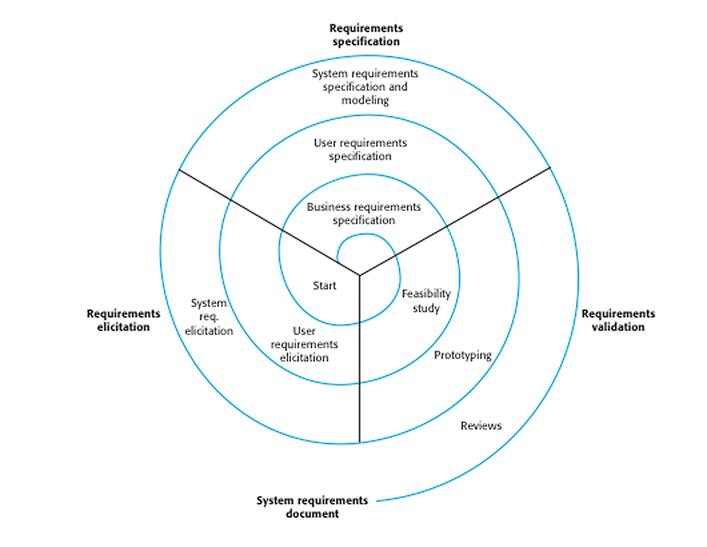
Requirements engineering is a process of gathering and defining of what the services should be provided by the system.

It focuses on assessing if the system is useful to the business (feasibility study), discovering requirements (elicitation and analysis), converting these requirements into some standard format (specification), and checking that the requirements define the system that the customer wants (validation).

In practice, requirements engineering isn’t sequential process, it’s an iterative process in which activities are interleaved.

For example, you iterate first on the user requirements; elicitation, specification, and validation, and repeat the same steps for the system requirements.



The process of requirements engineering

Early in the process, most effort will be spent on understanding high-level business and *user* requirements. Later in the process, more efforts will be spent on elicitation and understanding detailed *system* requirements.

*Some people consider requirements engineering to be the process of applying structured analysis method, such as object-oriented analysis. This involves analyzing the system and developing a set of graphical system models such as use case models, which then serve as a system specification.*

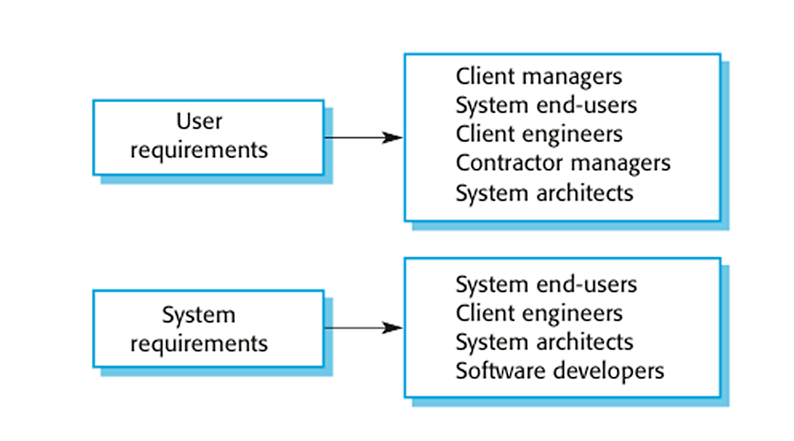
*Although structured methods have a role to play in the requirements engineering process, there’s much more to the requirements engineering than that’s covered by these methods.*

[*Object-oriented analysis and design*](https://medium.com/omarelgabrys-blog/analysis-and-design-identifying-objects-part-1-79b825d169a4)*will be discussed in another series of tutorials.*

**User and System Requirements**

Typically, requirements are presented into two level of detail; user and system requirements, where user need a high-level statements of the requirements, while system developers need a more detailed system specification. So, user and system requirements are just refer to different level of detail.

Having different level of details is useful because it communicates information about the system being developed for different types of readers.



Readers of different types of requirements specification

So, end-users will not be concerned with the detail, they need a generic, abstracted written requirement.

While the people who are involved in the development, they need what exactly they system should do.

You will probably will end up with a lot of problems and misunderstandings if you didn’t have a clear separation between the different level detail.

**User Requirements**

It describes the services that the system should provide and the constrains under which it must operate. We don’t expect to see any level of detail, or what exactly the system will do, It’s more of generic requirements.

It’s usually written in a natural language and supplied by diagrams.

*We’ll discuss different ways of specifying the requirements later in this series.*

**System Requirements**

The system requirements mean a more detailed description of the system services and the operational constrains such as how the system will be used, and development constrains such as the programming languages.

This level of detail is needed by those who are involved in the system development, like engineers, system architects, testers, etc.

**Functional & Non-Functional Requirements**

The software requirements are classified into functional requirements and non-functional requirements.

**Functional Requirements**

It covers the main functions that should be provided by the system. When expressed as *user* requirement, they are usually descried in an abstract way.

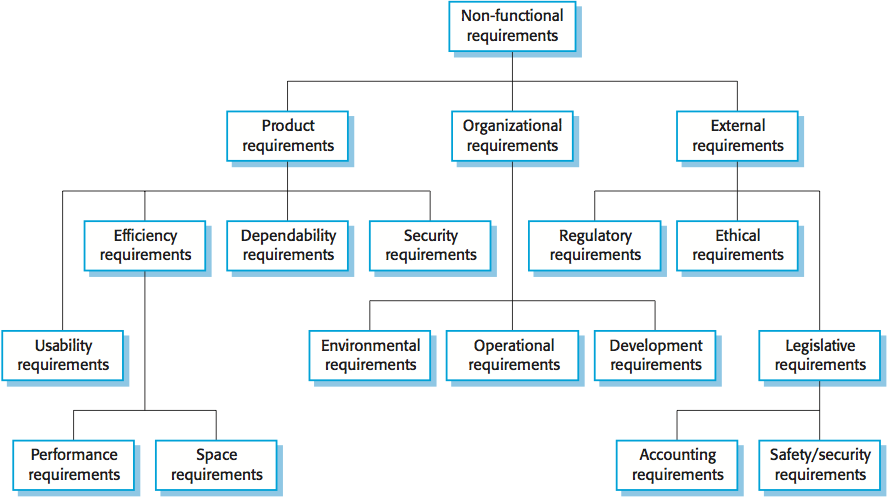
However, more specific functional *system* requirement describe the system functions, it’s inputs, processing; how it’s going to react to a particular input, and what’s the expected output.

**Non-Functional Requirements**

These are the constrains on the functions provided by the system.

The constrains, like how many process the system can handle (performance), what are the (security) issues the system needs to take care of such as SQL injections …

The rate of failure (reliability), what are the languages and tools will be used (development), what are the rules you need to follow to ensure the system operates within the law of the organization (legislative).



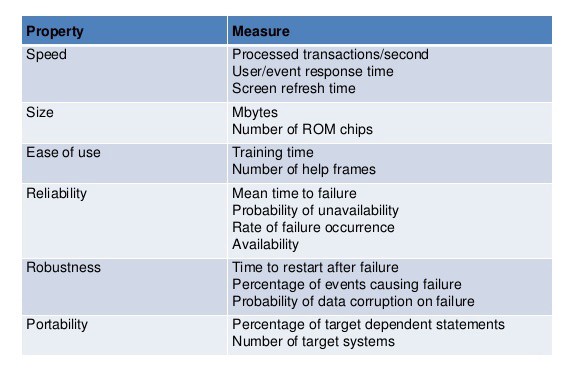
Non-Functional Requirements

Non-functional requirements are often critical than individual functional requirements. Users can usually find ways to work around a system function that doesn’t really meet their needs. However, failing to meet a non-functional requirement can mean that the whole system is unusable.

For example, if an aircraft doesn’t mean meet it’s reliability requirements, it won’t be safe for operation, or if an embedded control system fails to meet it’s performance requirements, the control functions won’t operate correctly.

**Non-functional requirements should be measurable**

Whenever possible, we should write non-functional requirements quantitatively, so that they can be tested. You can measure them when the system being tested to check whether the system meet it’s non-functional requirements.



Metrics for non-functional requirements

In practice, customers for a system often find it difficult to translate their goals into measurable requirements. They don’t understand what some number defining the required speed or reliability. For some goals, such as maintainability, there’re no metrics that can be used.

The cost of verifying measurable non-functional requirements can be very high and the customers may not think that these costs are justified.

**Non-functional and functional requirements are dependent**

Non-functional requirements often conflict, interact, or even generate other functional or non-functional requirements.

A user requirement concerned with security, such as limiting access to authorized users, may generate other requirements that are functional, such as the need to include user authentication facilities in the system.

**The distinction between functional and non-functional requirements**

In practice, it’s difficult to separate functional and non-functional requirements. The distinction is not clear as their definitions suggest.

**Separate between functional and non-functional requirements**

If the non-functional requirements are stated separately from the functional requirements, the relationship between them may be hard to understand.

However, we should explicitly highlight requirements that are clearly related to emergent system properties such as performance or reliability.

*Emergent properties are properties of the system as a whole rather than properties that can be derived from the properties of the individual system components.*

**Feasibility Report**

Before getting started with the software, you need to make a study to identify of whether the system is worth implementing and if it can be implemented under the current the current budget, technical skills, schedule, and if it does contribute to the whole organization objectives or not, etc.

The input to the feasibility study is a set of preliminary business requirements, an outline description of the system and how the system is intended to support business processes.

*The business requirements are the need of the customer or the developing organization; why the software is being developed, that must be fulfilled to achieve a high-level objective.*

The source for information may be the managers of departments where the system will be used, software engineers who are familiar with the type of proposed system, technology experts, end-users of the system, etc. Normally, we should try and complete a feasibility study in two or three weeks.

The results of the feasibility study should be a report that recommends whether to go forward to the next process or you won’t be able to implement the software at all.