What is the difference between a Requirement and a Specification?

The sound-bite answer is that requirements are what your program **should** do, the specifications are **how** you plan to do it.

Another way to look at it is that the requirements represent the application from the perspective of the **user**, or the business as a whole. The specification represents the application from the perspective of the **technical** team. Specifications and requirements roughly communicate the same information, but to two completely different audiences.

User requirements

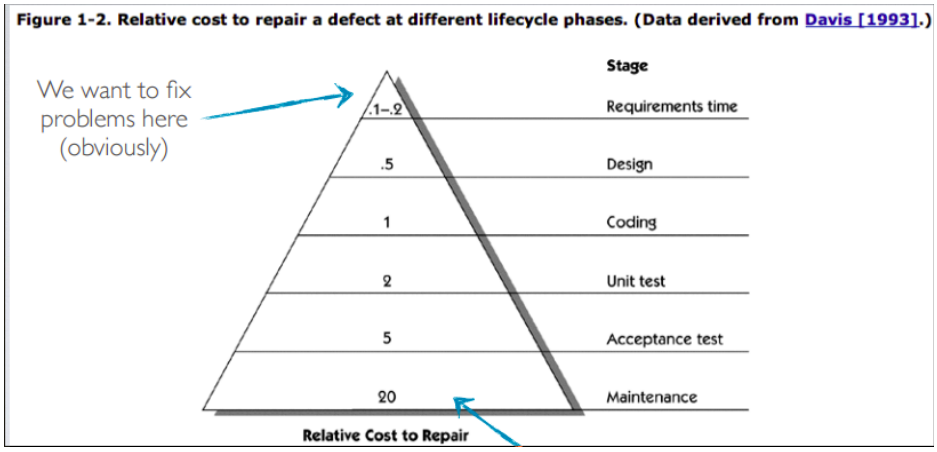
- What the stakeholders want/have to do
- High level descriptions
- Natural language statements
- Diagrams to show things

System specifications

- What the system should actually do
- Specific detail for someone to build them
- **What** but not **how**
- Can be ticked/marked as done

Stages where you are likely to fail

- Requirement specifications
- Managing customer requirements
- Documentation
- Software and testing
- Project management



To build software you need to :

- Figuring out what "stakeholder" needs and what services a product will need to provide to meet those needs
- Know how to identify the requirements of a systems project
  - Need to be able to determine them, by working with clients

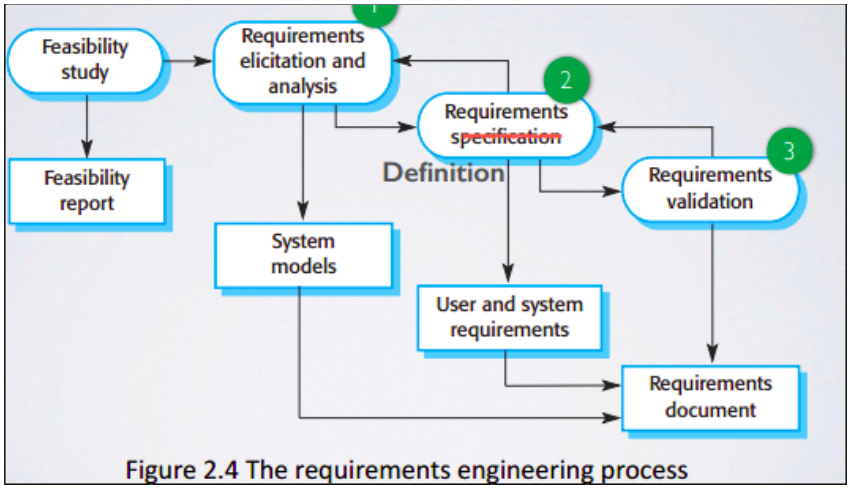


Figure 2.4 The requirements engineering process

1. Requirements elicitation & analysis
   1. Find out what stakeholders require from the system
   2. Define the requirements in a form understandable to the customer
2. Requirements definition
   1. Define the requirements in detail
3. Requirements validation
   1. Check that they don't conflict with each other
   2. Check requirements & designs with the customer

Functional & Non Functional Requirements.

A functional requirement describes **what** a software system should do, while non-functional requirements place constraints on **how** the system will do so.

Example :

A system must send an email whenever a certain condition is met (e.g. an order is placed, a customer signs up, etc) – f**unctional**

Emails should be sent with a latency of no greater than 12 hours from such an activity. - **non-functional**

The functional requirement is describing the **behaviour** of the system as it relates to the system's functionality. The non-functional requirement elaborates a performance **characteristic** of the system.

Risks of requirements engineering

Overlooking a crucial requirement

If you overlook an important user class, you'll probably have to put in a lot of work to add in what that user needs. Missing a critical quality or performance attribute is even worse. Often the only way to adapt a software-based system to important strategical changes is to re-architect.

One of the common examples is scalability in e-commerce. If designers don't keep scalability in mind when choosing their architectures, they find themselves in a tough position when the usage load goes beyond their expectations. System performance is inextricably tied to system architecture. In most instances, the only way to improve performance is to start over from scratch.

Inadequate customer representation

One of the central activities in RE is negotiating agreement on requirements. To achieve this, you must find out what your customers really need. Not much of a negotiation will take place if you never actually interact with them. "Take it or leave it" happens all too frequently when we assume our design ideas suit our customers and don't bother to check if this assumption is really true.

Modelling only functional requirements

Functional requirements are the most obvious ones to the user, so most elicitation(collecting the requirements) discussions focus on them. Sometimes it's more important gaining agreement on quality attribute requirements(non functional)—the characteristics you intend your software to exhibit. Reliability, performance, security, robustness, ease of use, scalability, innovation.

Not inspecting requirements

Cost to remove defects in requirements increases geometrically with time. Once your software hits the field, removing a requirements defect costs at least a hundred times as much, assuming you can fix it at all. That's why inspecting your requirement model is extremely important. To hold an inspection, you must have something inspectable and believe that your set of requirements has defects that need to be identified.

Attempting to perfect requirements before beginning construction

We can't possibly know everything we'd like to know before we start development. It's safer to assume that our requirements are going to change than that they won't. You need to do some design and construction before you can tell how hard the job will be and how much each part will cost. As this kind of information becomes evident, it could well affect your views about your requirements, further changing them. Do the best job you can early to get a good set of requirements, but don't be discouraged if everything isn't absolutely certain. Identify those areas of uncertainty and move on, ensuring that some is responsible for closing those gaps. Be careful to track uncertain requirements as the project proceeds.

Representing requirements in the form of designs

Possibly the subtlest risk in requirements engineering is letting designs creep into, and then remain in, your requirements specifications. You run the risk of choosing a particular solution that might not be the best one to implement. Also, you undermine your ability to validate your system, because you are specifying the problem you hope to solve in terms of how you intend to solve it.

Using designs as requirements is a subtle risk because although you might have specific information about what you want, it doesn't represent the underlying need and is consequently vulnerable to mistaken assumptions.

Requirements

Functional : Exact functions the system should provide | Non-functional : Constraints on the system. Performance, security, uptime, number of users,etc.