Reasons to do requirements validations

- Checking if you are right
  - Actually define the system that the customer really wants
- Avoiding rework
  - The cost of fixing a requirements problem by making a system change is usually more greater than repairing design or coding errors
- Contractually agreeing
  - At some point you have to decide what **exactly do you want to build**
  - You want everyone to agree exactly what will be build, otherwise you may have different ideas from a customer. You calculate cost for your idea, but the customer doesn't pay until **their** idea is achieved.

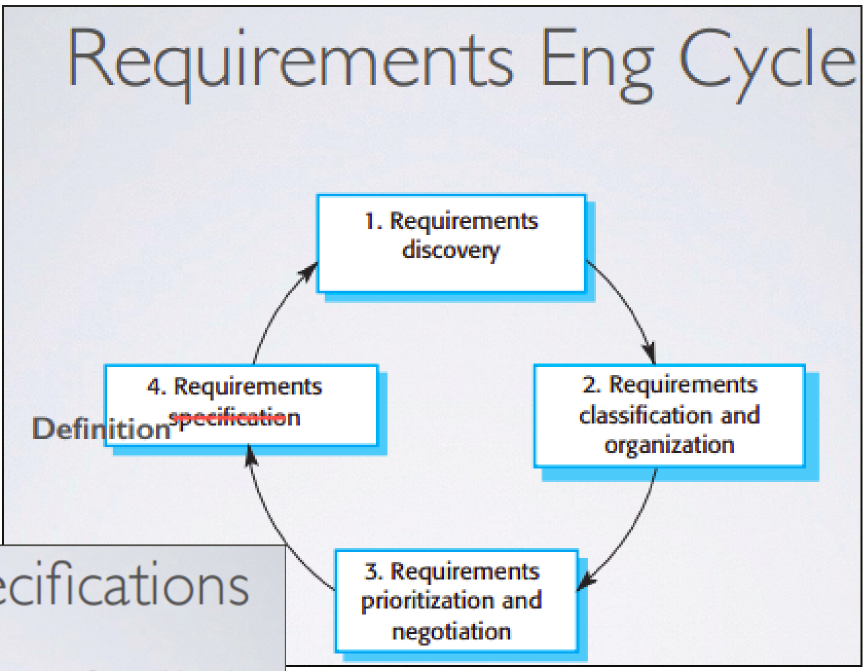
Requirements validation (with people)

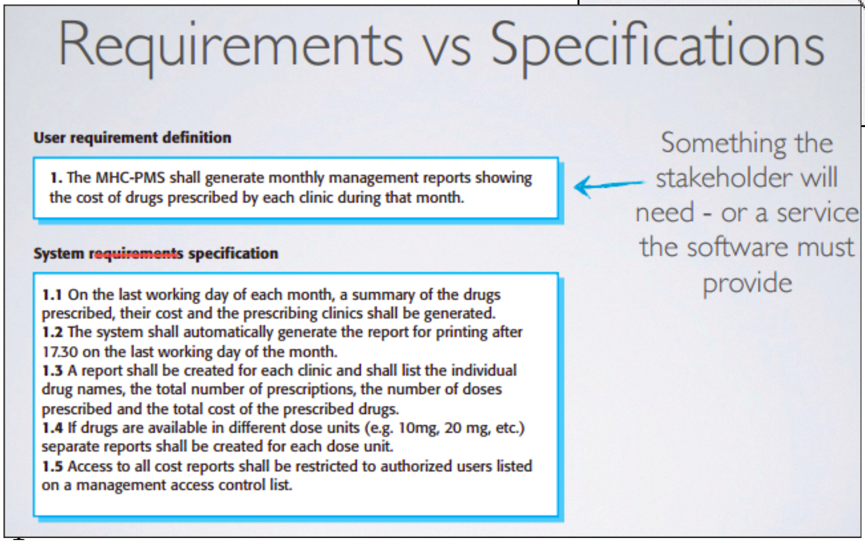
Internal : you present it to your "boss and colleagues". Having to explain things to an audience gives a first sanity check (does it make sense when you explain it to your "boss")

External : you present it back to participants/clients/users. Do they agree with your understanding? Do they agree with what you think is most important.

Requirements validation (with users)

- Using broad methods
  - This is like a survey to get large volume voting on issues. You could have had 250 tell you if issue X is important (or how important on a scale 1-5)
- Using focused methods a **Requirements Review**
  - More interviews with key stakeholders (discussing what you have found)


Requirements Eng Cycle
1. Requirements discovery
2. Requirements classification and organization
3. Requirements prioritization and negotiation
4. Requirements ~~specification~~
Definition



Requirements vs Specifications

User requirement definition

1. The MHC-PMS shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

Something the stakeholder will need - or a service the software must provide

System ~~requirements~~ specification

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
1.2 The system shall automatically generate the report for printing after 17.30 on the last working day of the month.
1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
1.4 If drugs are available in different dose units (e.g. 10mg, 20 mg, etc.) separate reports shall be created for each dose unit.
1.5 Access to all cost reports shall be restricted to authorized users listed on a management access control list.

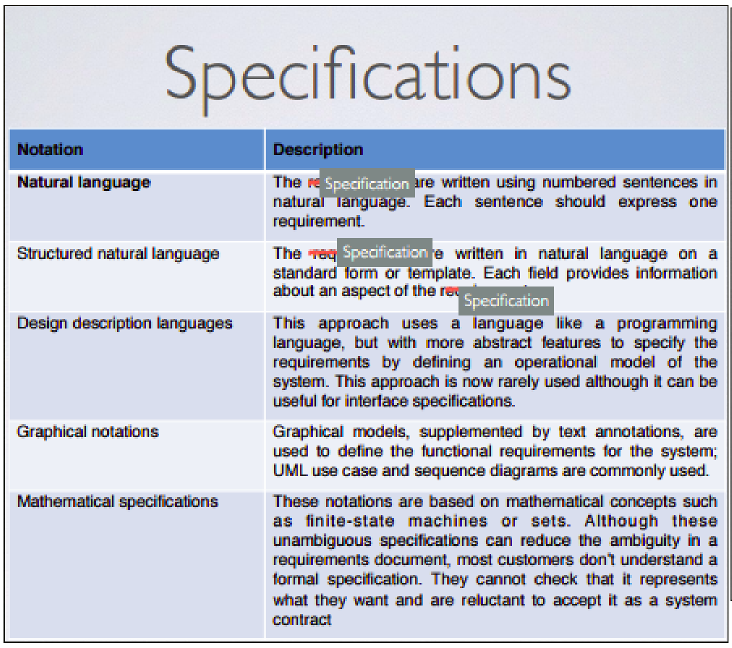
Specification & requirements validations

- When specifying how to met requirements you can often find conflicting ones
- If produced carefully, specifications help to validate requirements

Specifications

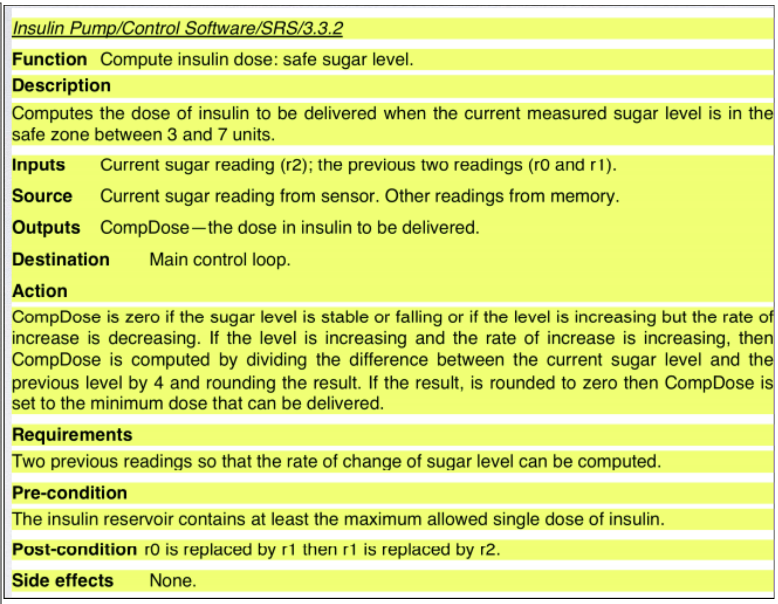
Detailed descriptions of the software system's functions, services, and operational constraints … they … should define exactly what is to be implemented

We are focusing on **what** should be built, not **how** it should be built. How a system will meet the user requirements. Which means specifications should be tied to a requirement (they are often tabulated with a column says which requirement ID it is for)



Specifications

Notation	Description
Natural language	The ~~re~~ Specification are written using numbered sentences in natural language. Each sentence should express one requirement.
Structured natural language	The ~~req~~ Specification e written in natural language on a standard form or template. Each field provides information about an aspect of the ~~re~~ Specification
Design description languages	This approach uses a language like a programming language, but with more abstract features to specify the requirements by defining an operational model of the system. This approach is now rarely used although it can be useful for interface specifications.
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML use case and sequence diagrams are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don't understand a formal specification. They cannot check that it represents what they want and are reluctant to accept it as a system contract



Insulin Pump/Control Software/SRS/3.3.2

Function	Compute insulin dose: safe sugar level.
Description	
Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.	
Inputs	Current sugar reading (r2); the previous two readings (r0 and r1).
Source	Current sugar reading from sensor. Other readings from memory.
Outputs	CompDose—the dose in insulin to be delivered.
Destination	Main control loop.
Action	
CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.	
Requirements	
Two previous readings so that the rate of change of sugar level can be computed.	
Pre-condition	
The insulin reservoir contains at least the maximum allowed single dose of insulin.	
Post-condition	r0 is replaced by r1 then r1 is replaced by r2.
Side effects	None.

Natural language specifications

- Can be expressive, intuitive , and universal
- Can also be ambiguous, vague, and interpreted differently
- Guidelines
  - Use standard format : 1 sentence, linked to a user requirement.
  - Distinguish between mandatory ("shall") and desirable ("should")
  - Emphasis important elements with bold,italic,etc.
  - Avoid jargon, unless clearly specified in a key words section

Structured specifications

Need to go further than natural language specifications. Tabulate specifications or put them in templates, that way it easier to enforce certain details. The tables can be used to specify additional information.

- Associated logic in the function
- Inputs and outputs
- An explanation
- Conditions
- Side effects or relations to other functions.

Tabulated specifications are used when **exact** conditions and calculations need to be known.

Graphical specifications

- UML model diagrams, prototypes
- At this stage the emphasis is on saying **how it will work** , **this is what you should build**
- It's often easier to see a UML sequence diagram, than to read 30 decision-dependent specifications. So when you find specifications are complex, **visualise** them.

Analysing the quality of specifications is also important. Good quality specifications have a few qualities. They should be :

- Correct,
- Complete
- Necessary
- Clear
- Feasible
- Consistent
- Unique
- Traceable
- Concise
- Verifiable

Traceability

It is important that all specifications can be **traced to user requirements**. In reports, you should, for every specification state which user requirement(s) it's **supporting**. You may also categorise them by importance, difficulty, etc.

In the end when you ask the customer to pay, they'll want to know you achieved what they asked for, you can only do this if specifications are **testable**. Can all the things you specify be **tested** and **proved**.

- It must load fast – unprovable
- It must load within 10s – testable and provable.

Specification review

There is formal review process you can go through. Several people n a room,reading each specification aloud. Systematically review the specifications :

- Validity checks (are the areas of functionality identified as necessary)
- Consistency checks (do specifications conflict with one another)
- Completeness checks (does it specify a coherent system or only parts of it)
- Realism checks (can specifications actually be implemented)
- Verifiability checks (can specifications be tested)

Are the system specifications: correct, necessary, important?

Can a **user/client** picture the system in operation with a large specification document. Even well structured specifications can be hard to imagine, that’s why a **prototype** is needed. It combines

- The UML modes or scenarios etc. from User requirements
- The specifics of the System requirements specifications

To demonstrate how they might work together in a system. They **envisage** the system requirements specifications