Requirements Validation Techniques

How to valid that captured requirements do define correctly the system that the customer really wants?

Requirements validation

Concerned with demonstrating that the requirements define the system the customer really wants.

Requirements error costs are high so validation is very important

Fixing a **requirements error** after delivery may cost up to **100 times** the cost of fixing an implementation error.

Requirements checking

Correctness/Validity: Does the system provide the functions that best support the actual business/customer's needs?

Precision/unambiguous: Can any of the requirements be interpreted in more than one way?

Consistency: Are there requirements that may conflict?

Completeness: Are ALL functions required by the customer included?

Verifiability: Can requirements be verified and validated?

Traceability: Is each one of the requirements traceable, i.e., uniquely identified?

Realism/Feasibility. Can the requirements be implemented within specified time and budget given available team skills and technology?

Requirements validation techniques

Requirements reviews

Systematic manual analysis of the requirements.

Prototyping

Using an “executable model” of the system to check requirements.

Test-case generation

Developing tests for requirements to check testability.

Documentation of Requirements

Requirements document requirements

Specify external system behaviour
Specify implementation constraints
Easy to change
Serve as reference tool for maintenance
Record forethought about the life cycle of the system i.e. predict changes
Characterise responses to unexpected events

IEEE requirements standard

Introduction
General description
Specific requirements
Appendices
Index

This is a generic structure that must be instantiated for specific systems

Requirements document structure

Introduction
Glossary
User requirements definition
System requirements specification
System architecture
System models
System evolution
Appendices
Index

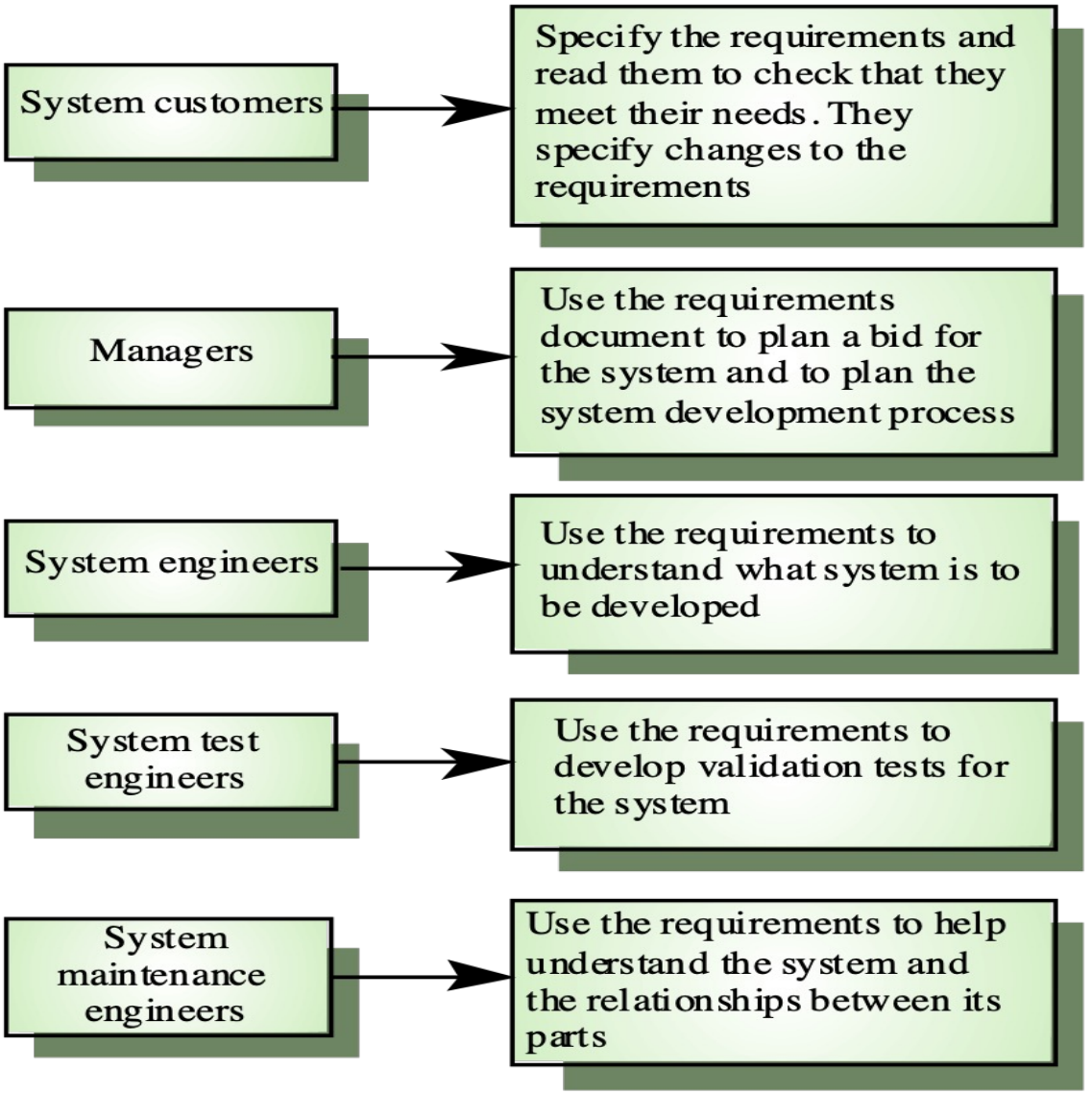
The structure of a requirements document-1

Chapter	Description
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	Here, you describe the services provided for the user. The nonfunctional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.
System architecture	This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.

The structure of a requirements document-2

Chapter	Description
System requirements specification	This should describe the functional and nonfunctional requirements in more detail. If necessary, further detail may also be added to the nonfunctional requirements. Interfaces to other systems may be defined.
System models	This might include graphical system models showing the relationships between the system components and the system and its environment. Examples of possible models are object models, data-flow models, or semantic data models.
System evolution	This should describe the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on. This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system.
Appendices	These should provide detailed, specific information that is related to the application being developed; for example, hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organization of the data used by the system and the relationships between data.
Index	Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, and so on.

Users of a requirements document



Guidelines for writing requirements

Invent a standard format and use it for all requirements

Use language in a consistent way. Use
shall for mandatory (or forceful) requirements,
should for desirable (auxiliary) requirements

Use text highlighting to identify key parts of the requirement

Include an explanation (rationale) of why a requirement is necessary

Avoid the use of computer jargon !!!

Requirement Phrasing Example:
different phrasing of the same requirement

The same requirement can be phrased differently, although they carry the same meaning. The example below- two different phrasing of the same User Requirement.

UR2.0: The user, as a member of staff, shall be able to login into the PMS system, using a preregistered username and a password.

OR

UR2.0: The PMS system shall enable users, members of staff, to login into the system, using a preregistered username and a password.

Ways of writing system requirements specification

Notation	Description
Natural language	The requirements are written using numbered sentences in natural language. Each sentence should express one requirement.
Structured natural language	The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement.
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 activity 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

Problems with natural language

Lack of clarity

Precision is difficult without making the document difficult to read

Requirements confusion

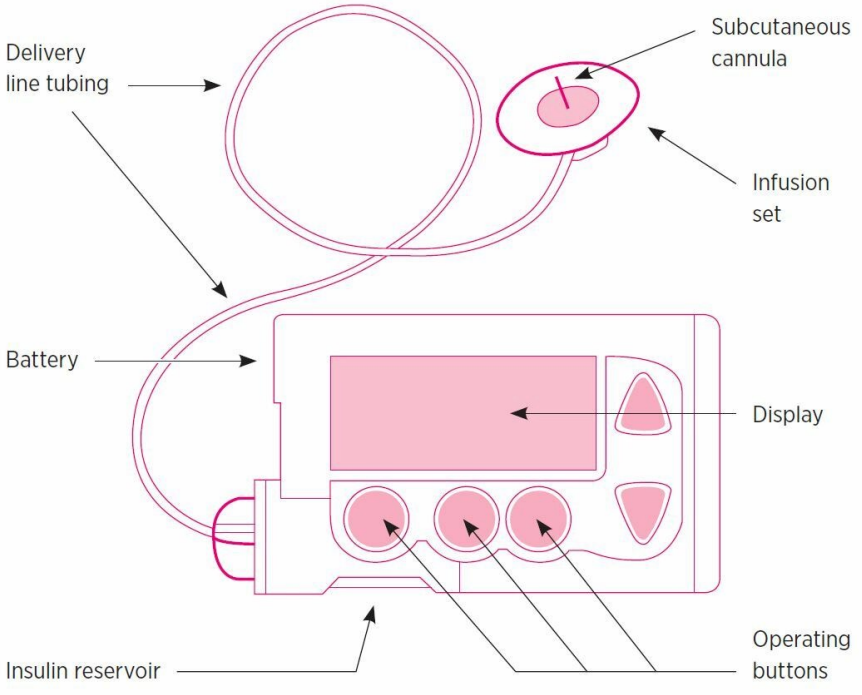
Functional and non-functional requirements tend to be mixed-up or confused

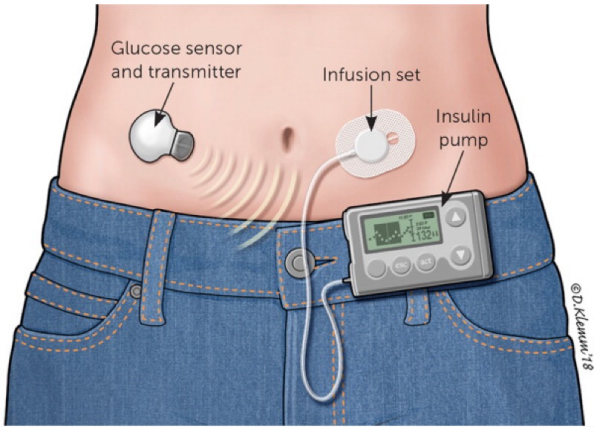
Requirements amalgamation

Several different requirements may be expressed together

Requirements-Example: Insulin Pump

Insulin pump is a machine that helps to control sugar level by delivering a correct dose of **insulin,** often automatically, to diabetic patients. (nicknamed as virtual Pancreas)





A requirement in Natural Language: Example

R3.2 The system shall measure the blood sugar and shall deliver insulin, if required, every 10 minutes. *(Changes in blood sugar are relatively slow so more frequent measurement is unnecessary; less frequent measurement could lead to unnecessarily high sugar levels.)*

R3.6 The system shall run a self-test routine every minute with the conditions to be tested and the associated actions defined in Table 1. *(A self-test routine can discover hardware and software problems and alert the user to the fact the normal operation may be impossible.)*

A structured specification of a requirement: Example 1

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.

A structured specification of a requirement: Example 2

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.

A structured specification of a requirement: Example 3

Title	**Compute Insulin Dose (CompDose)**
Purpose	To compute insulin dose based on the measured sugar level.
Description	Computes the dose of insulin to be delivered when the current measured level of sugar is in the safe zone between 3 and 7 units.
Actors	SystemTimer (actors that interact with this requirement)
Trigger	Automatic (triggered automatically every 10 minutes by SystemTimer)
Pre-condition	The insulin reservoir contains at least the maximum allowed single dose of insulin
Workflow	1. obtain current sugar level reading **r2** 2. read stored previous two sugar level readings, **r0** and **r1** 3. compute increasing/decreasing level of sugar within safe zone 4. compute a single dose of insulin based on sugar level 5. if dose of insulin is within allowed limits (5-15 mg), deliver insulin dose else generate beep sound 6. replace previous readings with current readings r0=r1 and r1=r2.
Alternative Workflow	
Post-condition	Previous readings replaced and stored

Tabular specification: Example

Condition	Action
Sugar level falling (r2 < r1)	CompDose = 0
Sugar level stable (r2 = r1)	CompDose = 0
Sugar level increasing and rate of increase decreasing ((r2 – r1) < (r1 – r0))	CompDose = 0
Sugar level increasing and rate of increase stable or increasing ((r2 – r1) ≥ (r1 – r0))	CompDose = round ((r2 – r1)/4) If rounded result = 0 then CompDose = MinimumDose

Key points

Requirements for a software system set out what the system should do and define constraints on its operation and implementation.

Functional requirements are statements of the services that the system must provide or are descriptions of how some computations must be carried out. They describe WHAT the system should undertake.

Non-functional requirements often constrain the system being developed and the development process being used. They often describe HOW WELL the system should undertake its functional requirements

Requirements often relate to the emergent properties of the system and therefore apply to the system as a whole.

Key points

The software requirements document is an agreed statement of the system requirements. It should be organized so that both system customers and software developers can use it.

The requirements engineering process is an iterative process including requirements elicitation, specification and validation.

Requirements elicitation and analysis is an iterative process that can be represented as a spiral of activities – requirements discovery, requirements classification and organization, requirements negotiation and requirements documentation.

Key points

You can use a range of techniques for requirements Engineering or elicitation including interviews, scenarios, use-cases and ethnography.

Requirements validation is the process of checking the requirements for validity, consistency, completeness, correctness and realism, unambiguity and verifiability.

Business, organizational and technical changes inevitably lead to changes to the requirements for a software system. Requirements management is the process of managing and controlling these changes.