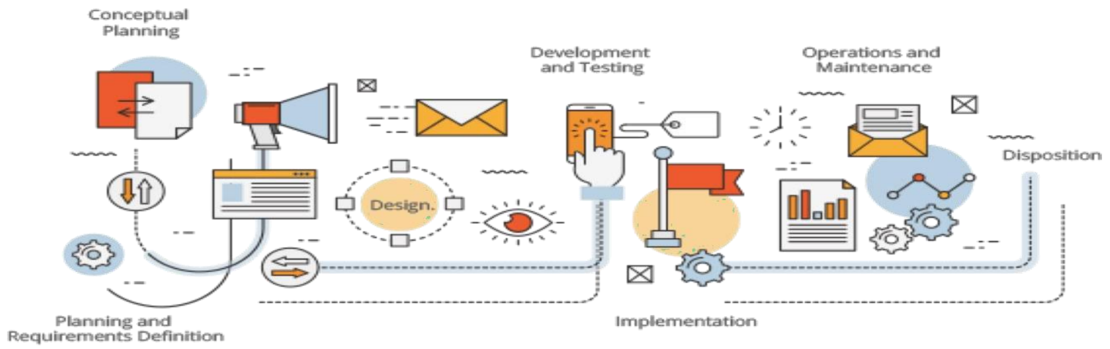
Software Engineering

Requirement Engineering



Requirement Document

The software requirements document

The software requirements document is **the official statement** of what is required of the system developers

Can include both a **definition of user requirements** and a specification of the **system requirements**

It is NOT a design document. As far as possible, it **should set of WHAT** the system should do rather than HOW it should do it

Requirements and design

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

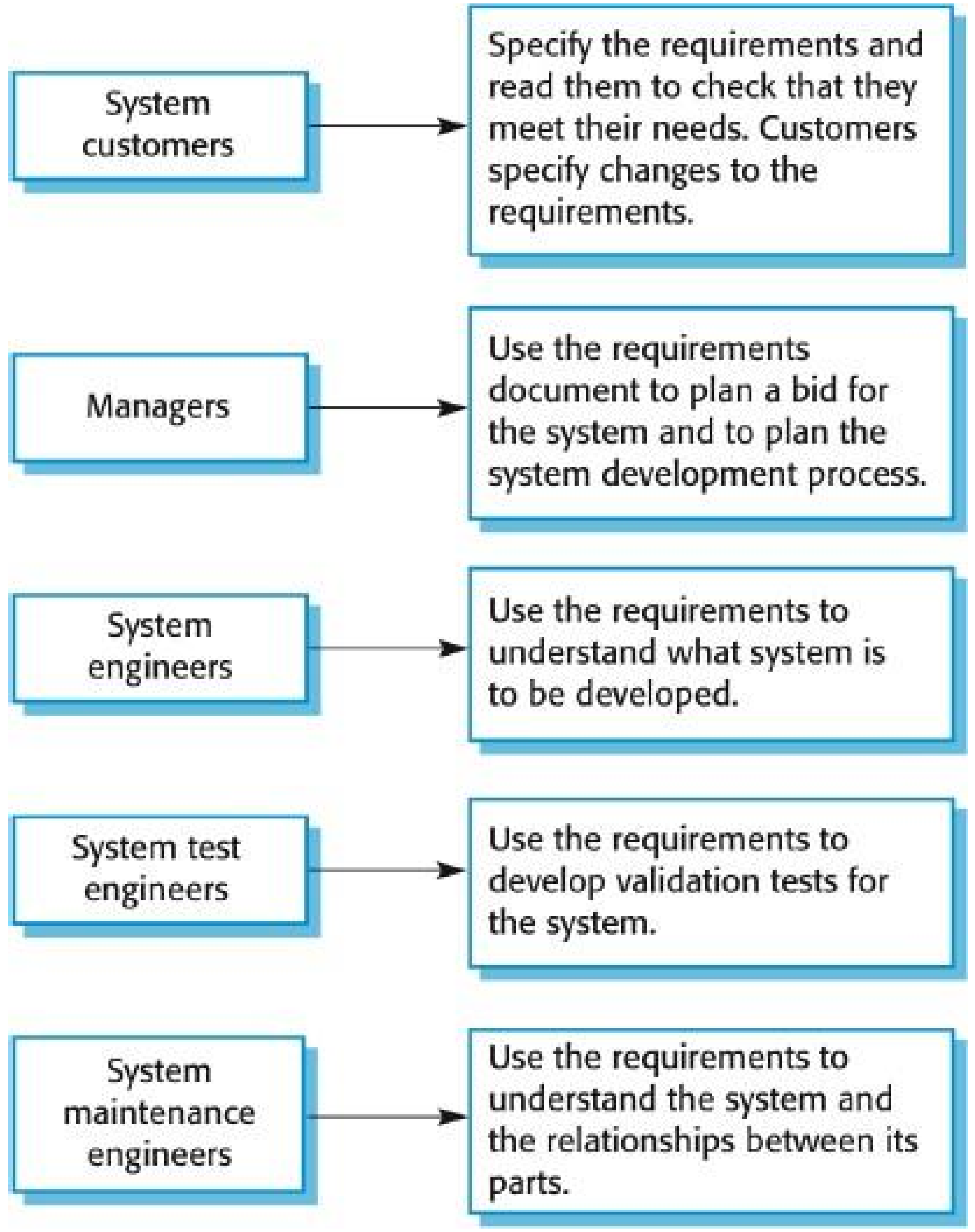
In practice, requirements and design are inseparable

- A system architecture may be designed to structure the requirements;
- The system may inter-operate with other systems that generate design requirements;
- The use of a specific architecture to satisfy non-functional requirements may be a domain requirement.
- This may be the consequence of a regulatory requirement.

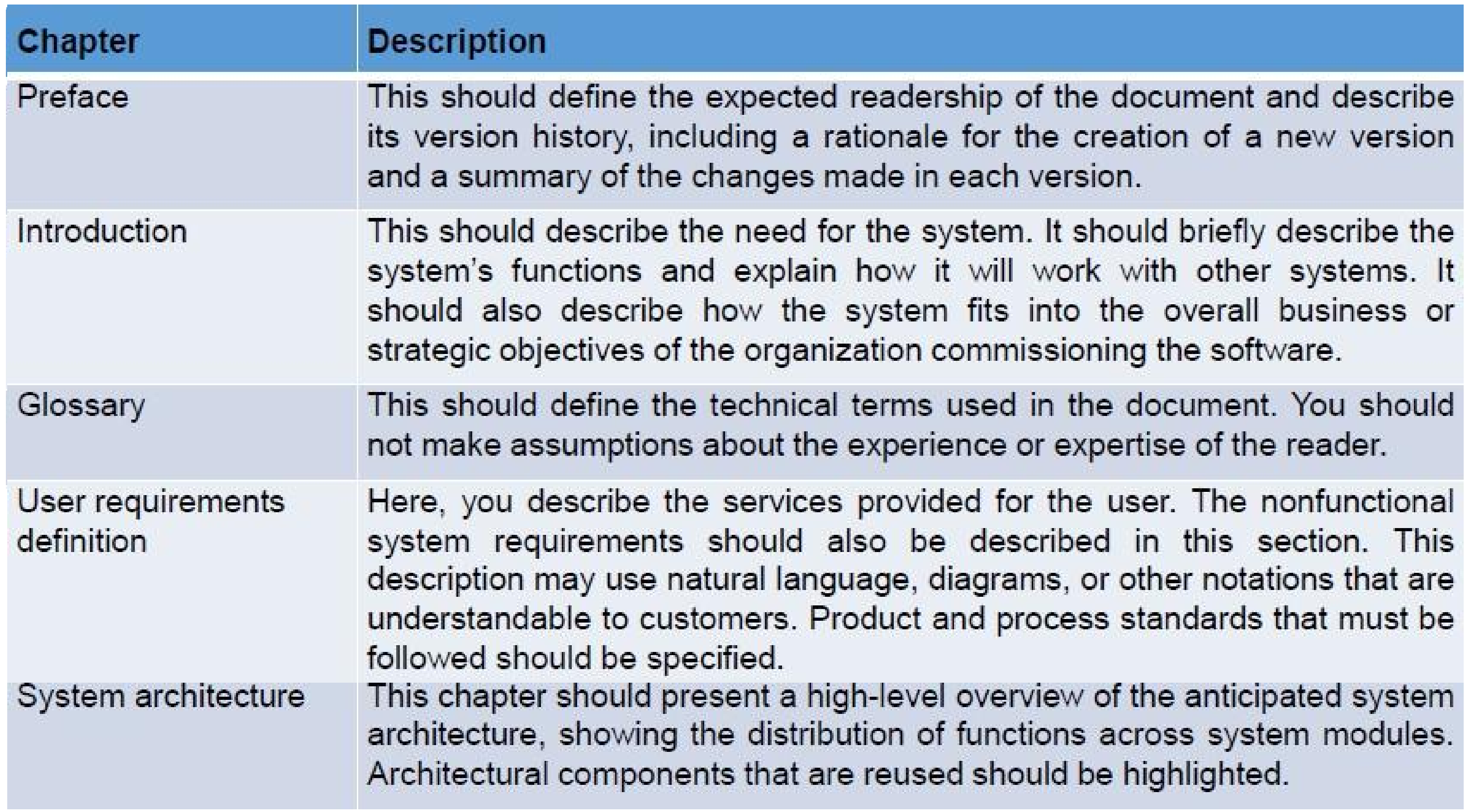
Agile methods and requirements

- Many agile methods argue that producing a **requirements document is a waste of time** as requirements change so quickly
- The document is therefore always out of date
- Methods such as XP use incremental requirements engineering and express requirements as ‘ user stories ’ This is practical for business systems but problematic for systems that require a lot of pre delivery analysis (e.g. critical systems) or systems developed by several teams

Users of a requirements document

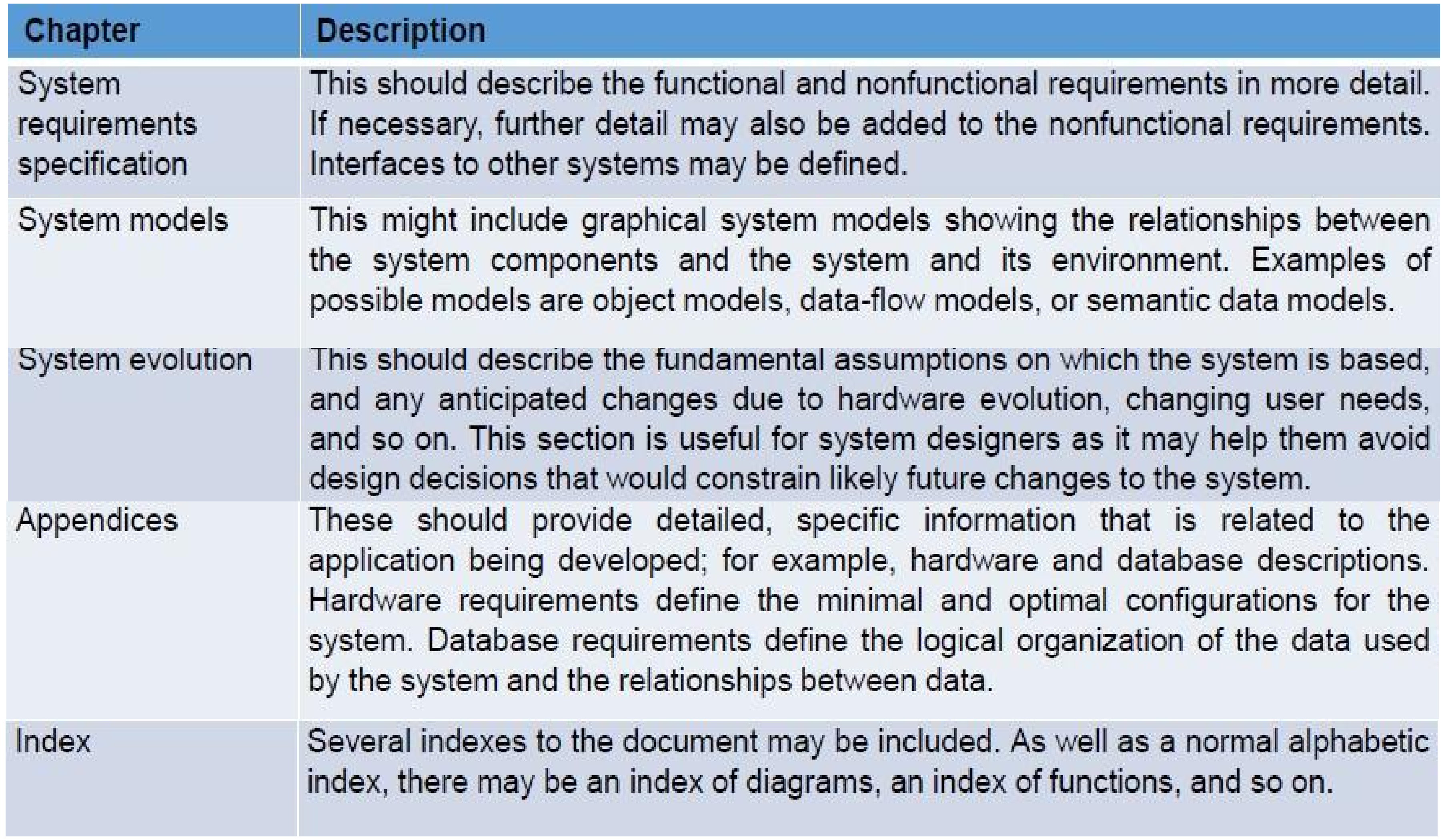

System customers
Specify the requirements and read them to check that they meet their needs. Customers specify changes to the requirements.
Managers
Use the requirements document to plan a bid for the system and to plan the system development process.
System engineers
Use the requirements to understand what system is to be developed.
System test engineers
Use the requirements to develop validation tests for the system.
System maintenance engineers
Use the requirements to understand the system and the relationships between its parts.

The structure of a requirements document



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

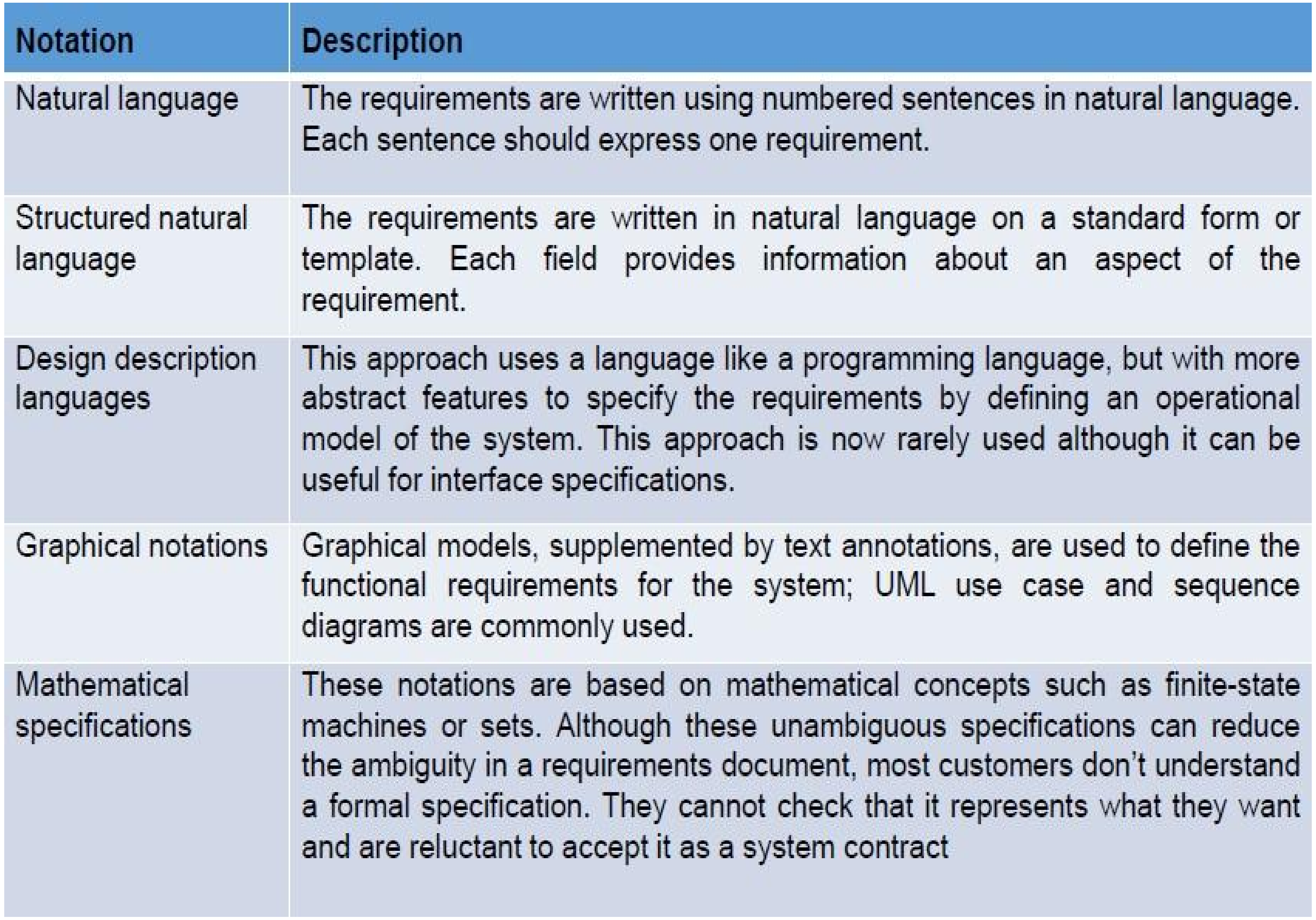


Chapter	Description
System requirements specification	This should describe the functional and nonfunctional requirements in more detail. If necessary, further detail may also be added to the nonfunctional requirements. Interfaces to other systems may be defined.
System models	This might include graphical 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.

Requirement's specification

- 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
- It is therefore important that these are as complete as possible

Ways of writing a 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 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

Natural language specification

- Requirements are written as natural language sentences supplemented by diagrams and tables
- Used for writing requirements because it is expressive, intuitive and universal. This means that the requirements can be understood by users and customers.

Guidelines for writing requirements

- Create a standard format and use it for all requirements
- Use language in a consistent way. Use **shall** for mandatory requirements, **should** for desirable requirements.
- Use text highlighting to identify key parts of the requirement
- Avoid the use of computer jargon
- Include an explanation (rationale) of why a requirement is necessary

Problems with natural language

Lack of clarity

Precision is difficult without making the document difficult to read.

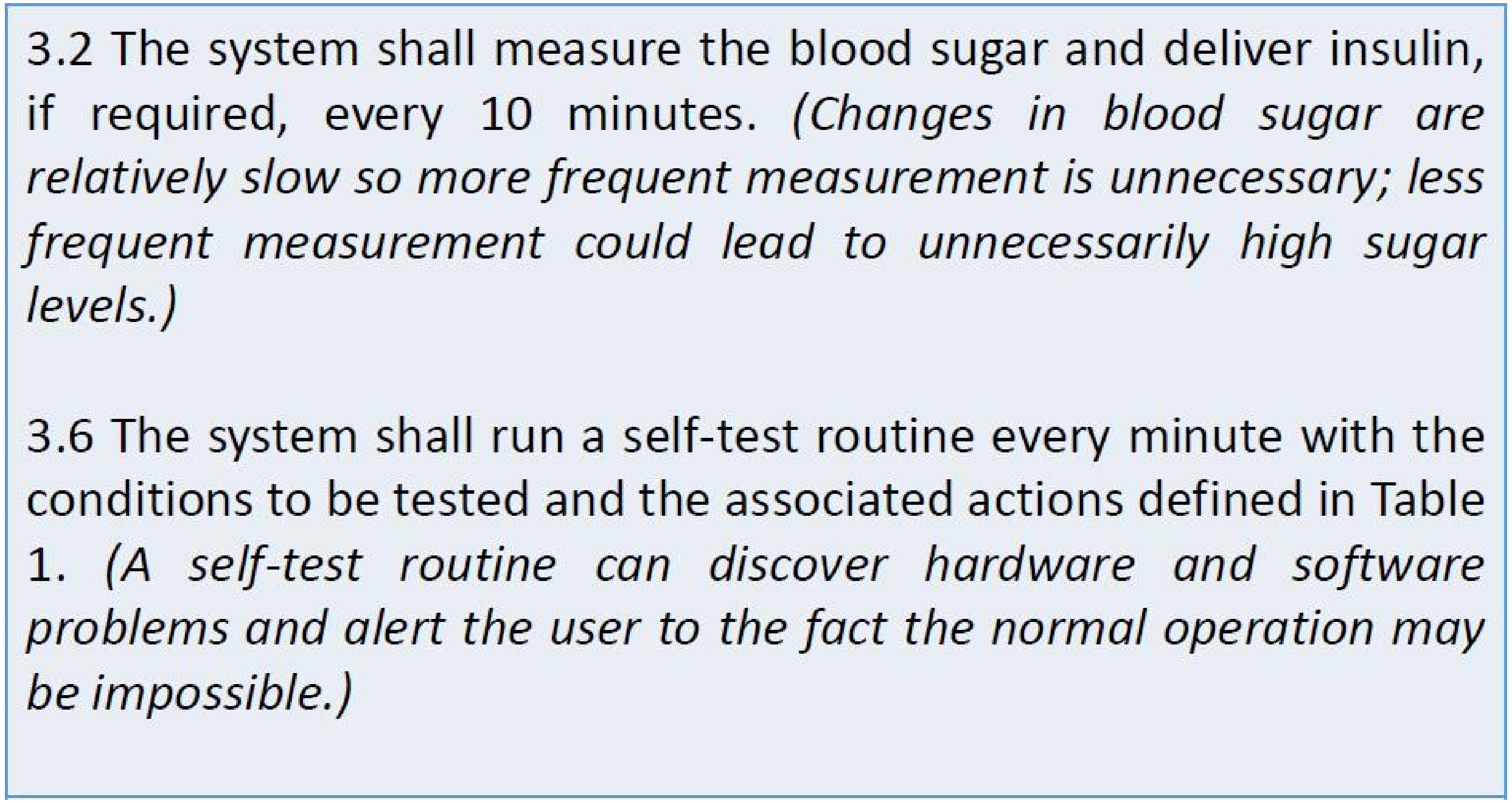
Requirements confusion

Functional and non-functional requirements tend to be mixed-up.

Requirements amalgamation

Several different requirements may be expressed together.

Example requirements for the insulin pump software system



3.2 The system shall measure the blood sugar and 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.)*

3.6 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.)*

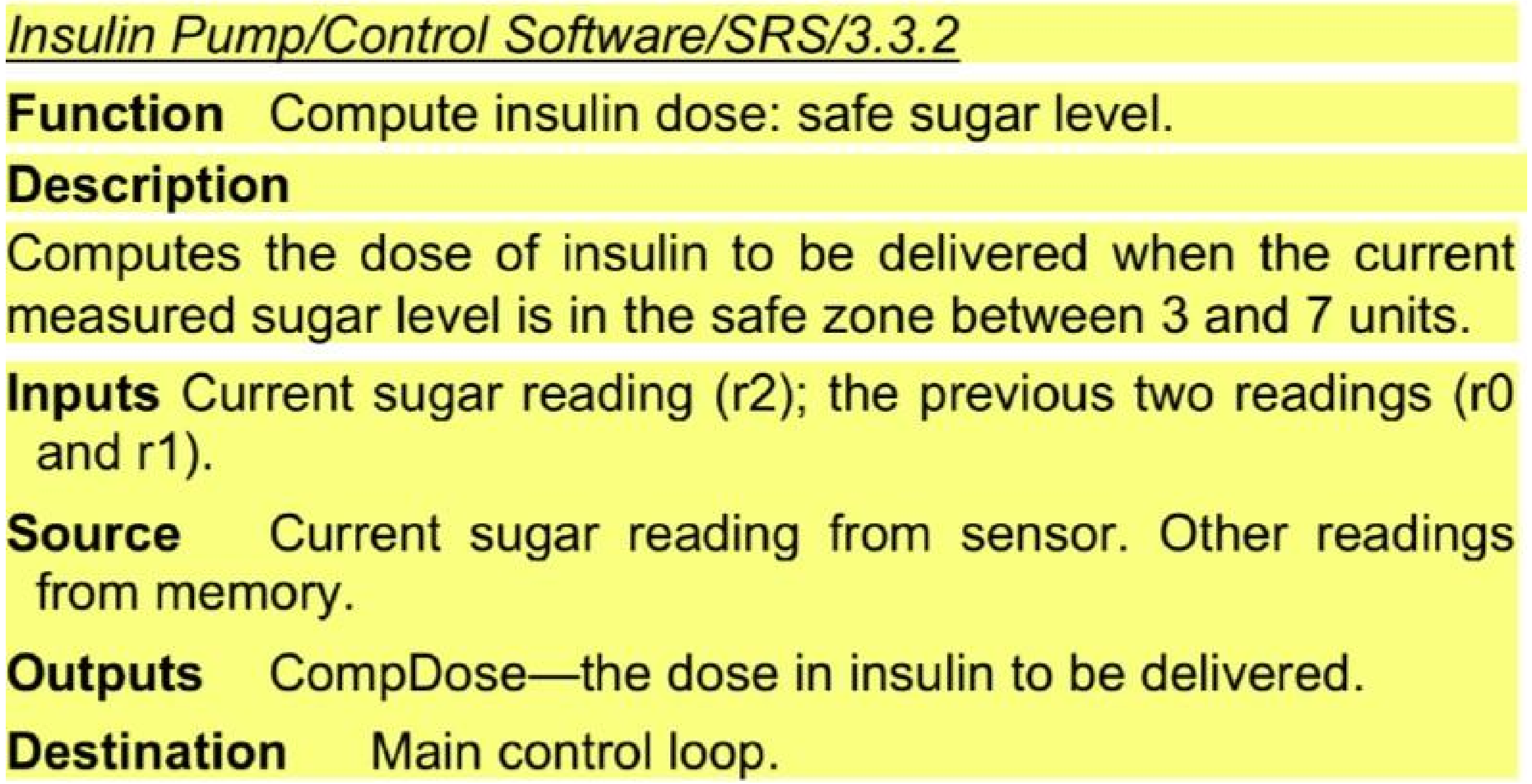
Tabular specification

- Used to supplement natural language
- Particularly useful when you have to define a number of possible alternative courses of action
- For example, the insulin pump systems bases its computations on the rate of change of blood sugar level and the tabular specification explains how to calculate the insulin requirement for different scenarios

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 $((r2 – r1) \geq (r1 – r0))$	CompDose = round ((r2 – r1)/4) If rounded result = 0 then CompDose = MinimumDose

A structured specification of a requirement for an insulin pump

Structured specifications



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.

Form based specifications

Definition of the function or entity

Description of inputs and where they come from

Description of outputs and where they go to

Information about the information needed for the computation and other entities used

Description of the action to be taken

Pre and post conditions (if appropriate)

The side effects (if any) of the function