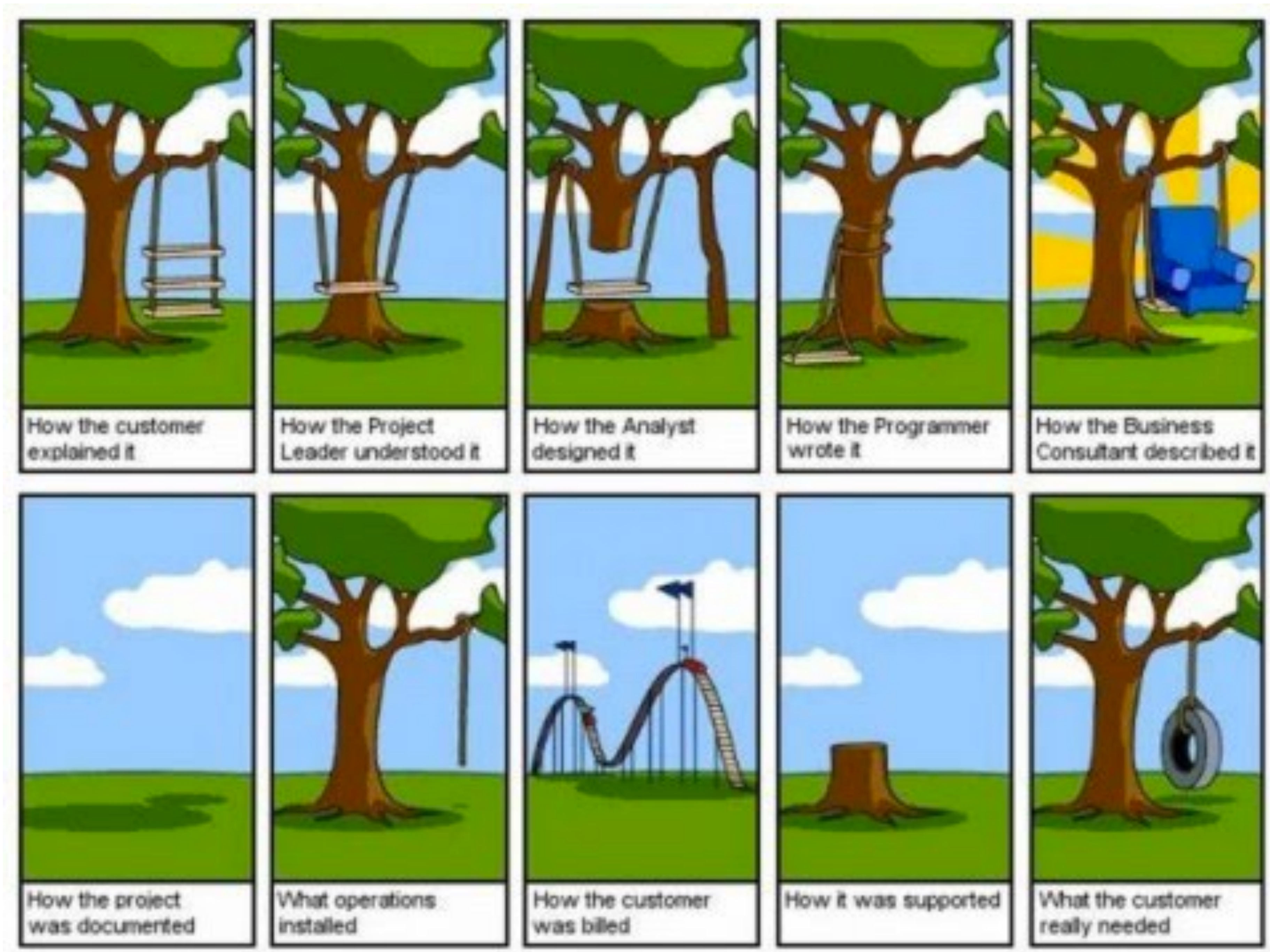
Software Engineering @ LEIC/LETI

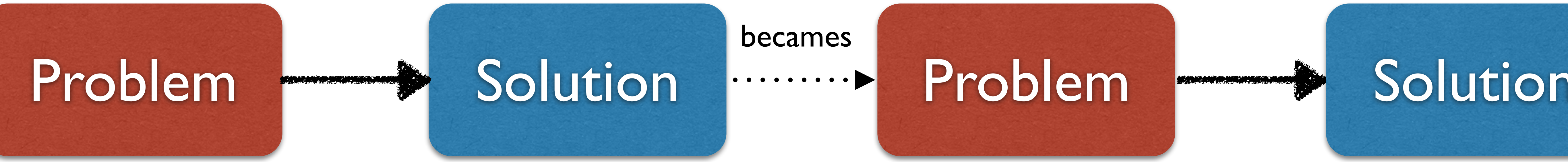
Requirements Engineering

Requirements Engineering

no man’s land

Understand what needs to be solved



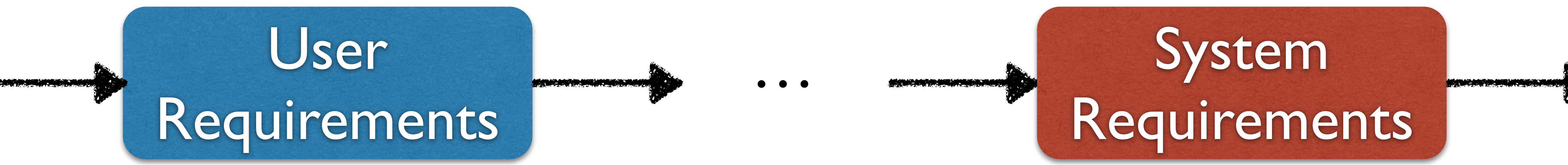


Problem
Solution
becames
Problem
Solution

The scope

the solution from
the user perspective

the problem from
the developers perspective



The Problem Space

User Requirement Definition

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

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.

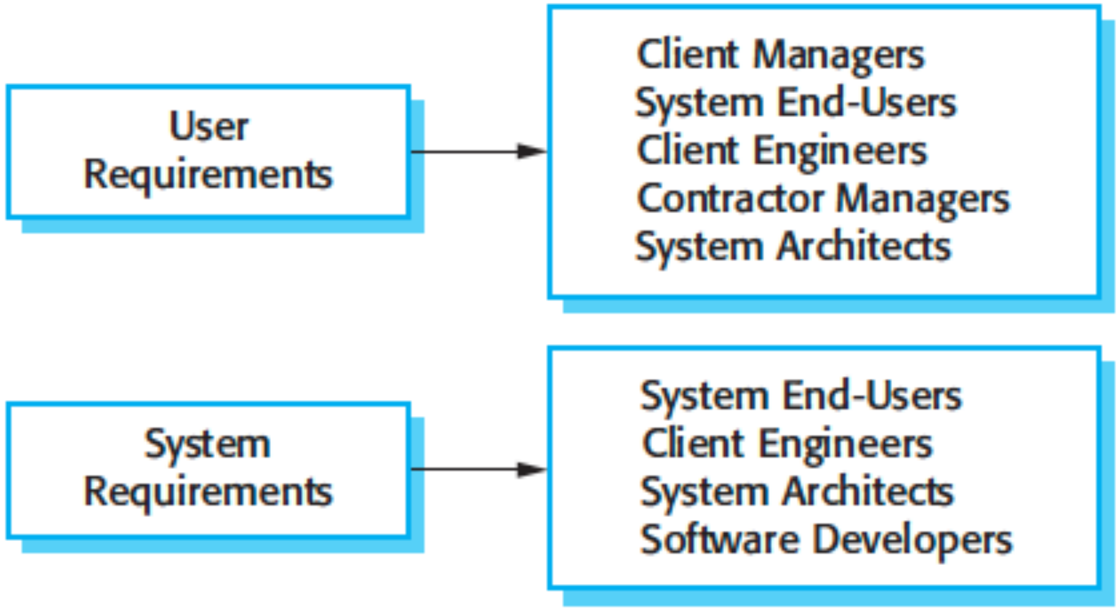
1.2 The system shall automatically generate the report for printing after 17.30 on the last working day of the month.

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., 10 mg, 20 mg) 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.

(Fig 4.1, Sommerville)



(Fig 4.2, Sommerville)

Functional and Non-functional Requirements

system behaviour and its constraints

create an account

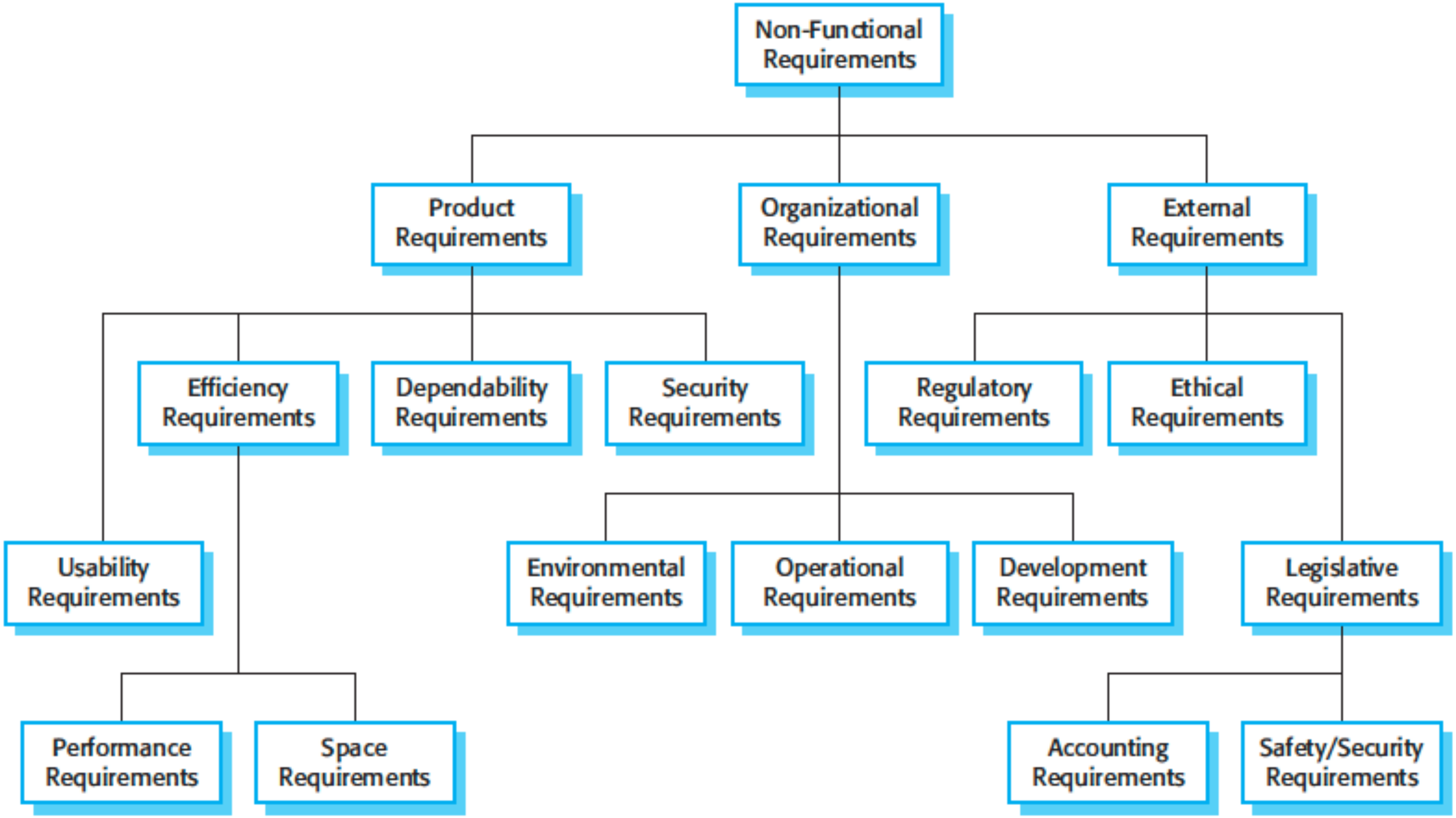
transactions should
execute in less than 1 second

process an adventure

system is unavailable 5
minutes per day maximum

only authorized users
can use the system

Is Login functional or non-functional?



(Fig 4.3, Sommerville)

PRODUCT REQUIREMENT
The MHC-PMS shall be available to all clinics during normal working hours (Mon–Fri, 08.30–17.30). Downtime within normal working hours shall not exceed five seconds in any one day.

ORGANIZATIONAL REQUIREMENT
Users of the MHC-PMS system shall authenticate themselves using their health authority identity card.

EXTERNAL REQUIREMENT
The system shall implement patient privacy provisions as set out in HStan-03-2006-priv.

(Fig 4.4, Sommerville)

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

(Fig 4.5, Sommerville)

complete

consistent

measurable

Qualities of Requirements

Is it easy to achieve the qualities aimed for requirements?

Completeness

do they capture all relevant aspects

Consistent

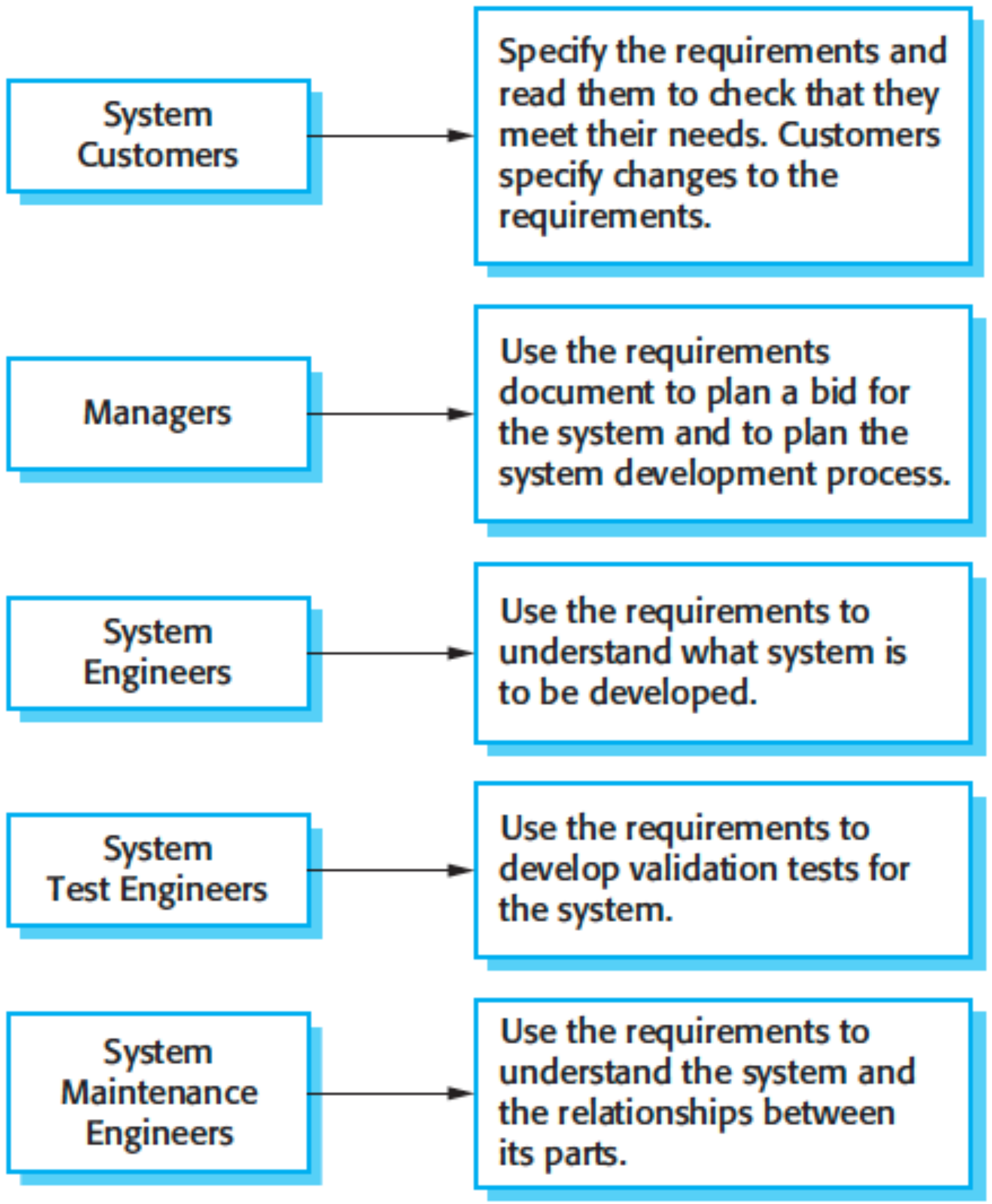
different stakeholders

security vs availability

Measurable

the system should resist attacks

Software Requirements Document



(Fig 4.16, Sommerville)

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 non-functional 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.
System requirements specification	This should describe the functional and non-functional requirements in more detail. If necessary, further detail may also be added to the non-functional requirements. Interfaces to other systems may be defined.
System models	This might include graphical system models showing the relationships between the system components, 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.

(Fig 4.17, Sommerville)

How to describe requirements?

Notation	Description
Natural language sentences	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.

what about test cases?

(Fig 4.11, Sommerville)

natural language

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.*)

(Fig 4.12, Sommerville)

structured natural language

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.

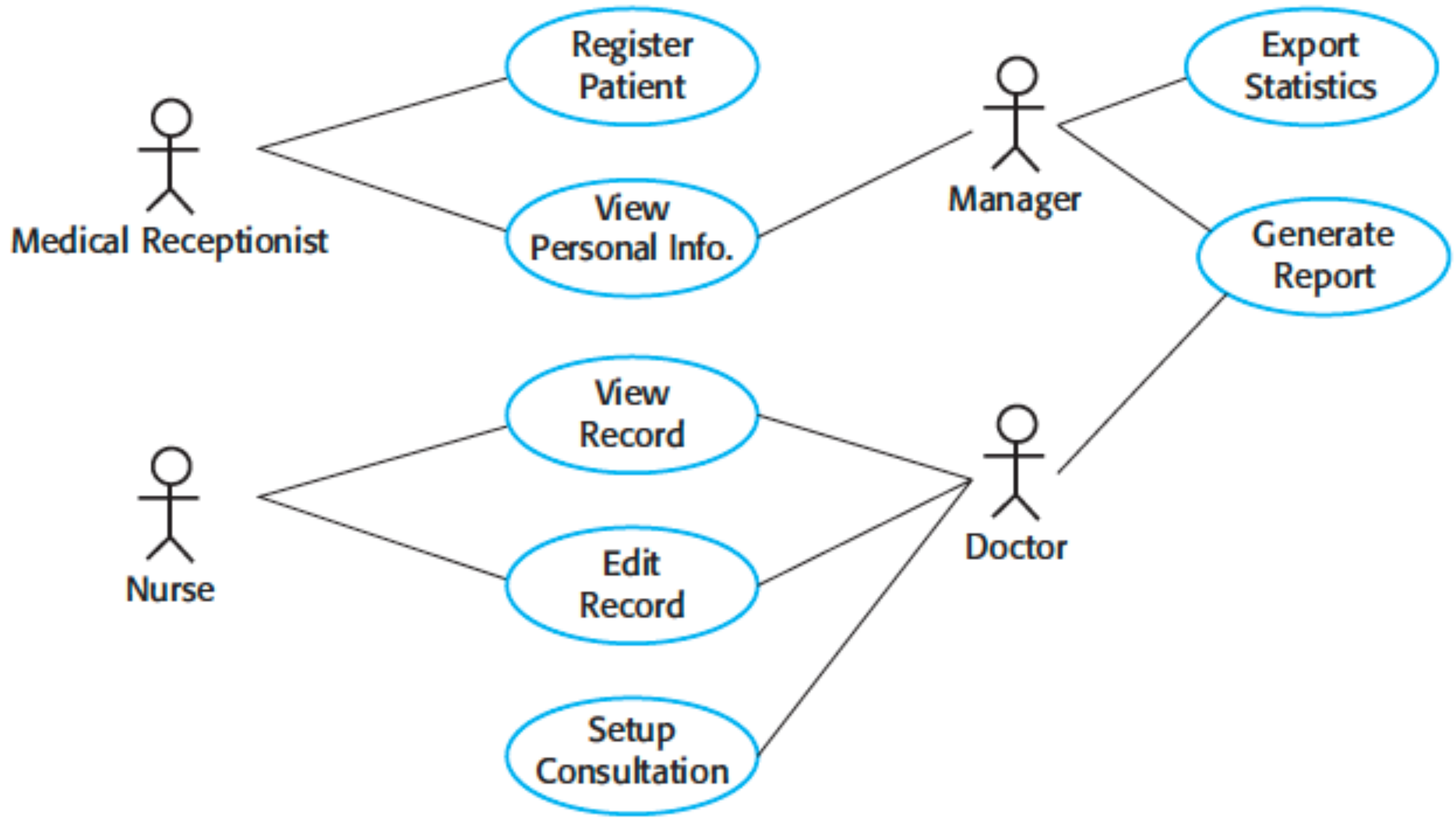
(Fig 4.13, Sommerville)

tabular specification

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

(Fig 4.14, Sommerville)

graphical notation



(Fig 4.15, Sommerville)

formal specification

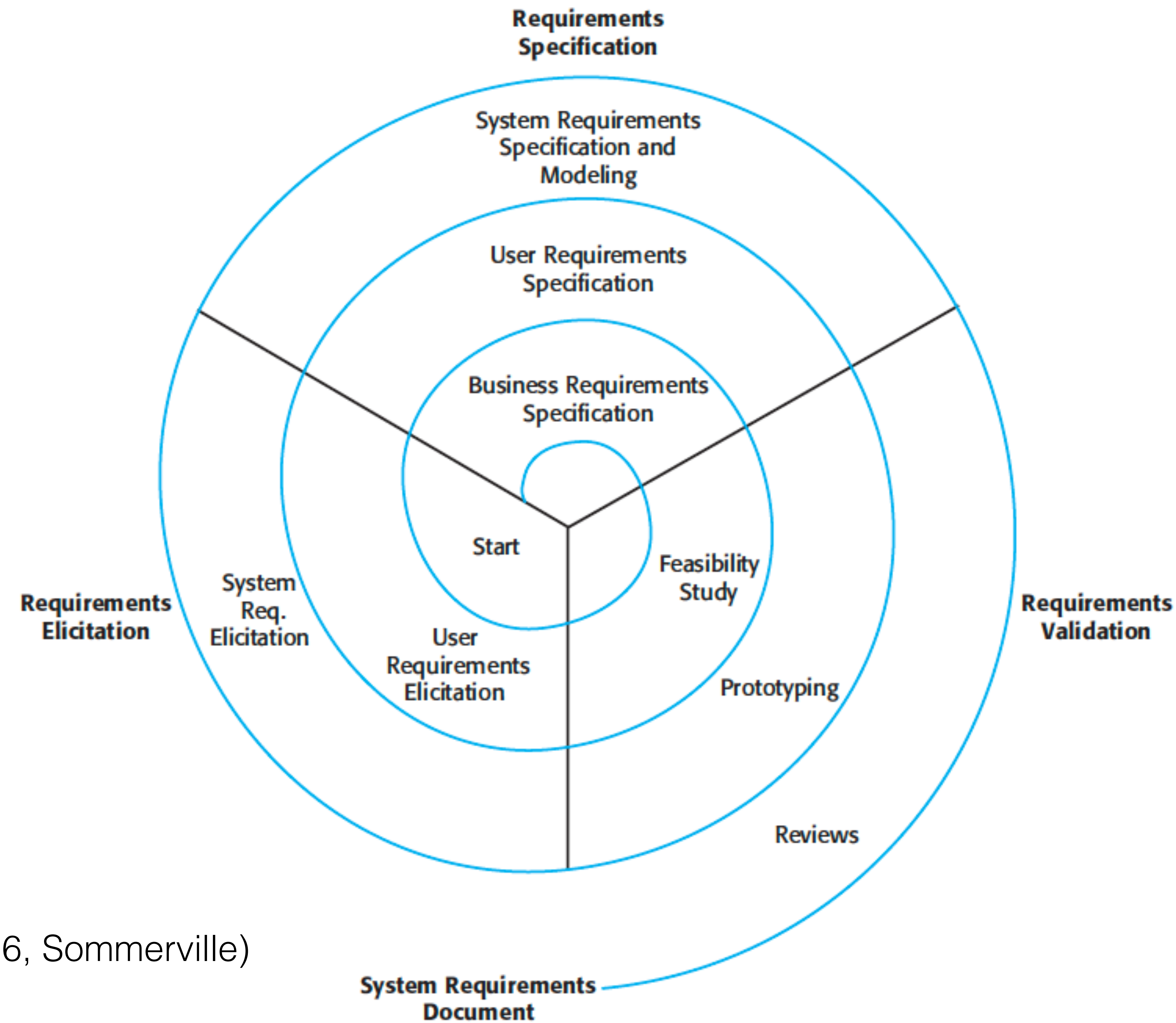
SUGAR_OK

$r2 \geq safemin \land r2 \leq safemax$
// sugar level stable or falling
$r2 \leq r1 \Rightarrow CompDose = 0$
$\lor$
// sugar level increasing but rate of increase falling
$r2 > r1 \land (r2-r1) < (r1-r0) \Rightarrow CompDose = 0$
$\lor$
// sugar level increasing and rate of increase increasing compute dose
// a minimum dose must be delivered if rounded to zero
$r2 > r1 \land (r2-r1) \geq (r1-r0) \land (round((r2-r1)/4) = 0) \Rightarrow$
$CompDose = minimum\_dose$
$\lor$
$r2 > r1 \land (r2-r1) \geq (r1-r0) \land (round((r2-r1)/4) > 0) \Rightarrow$
$CompDose = round((r2-r1)/4)$

what is the difference from a tabular specification?

wait for Dafny and see ;)

Requirements Engineering Process



(Fig 4.6, Sommerville)

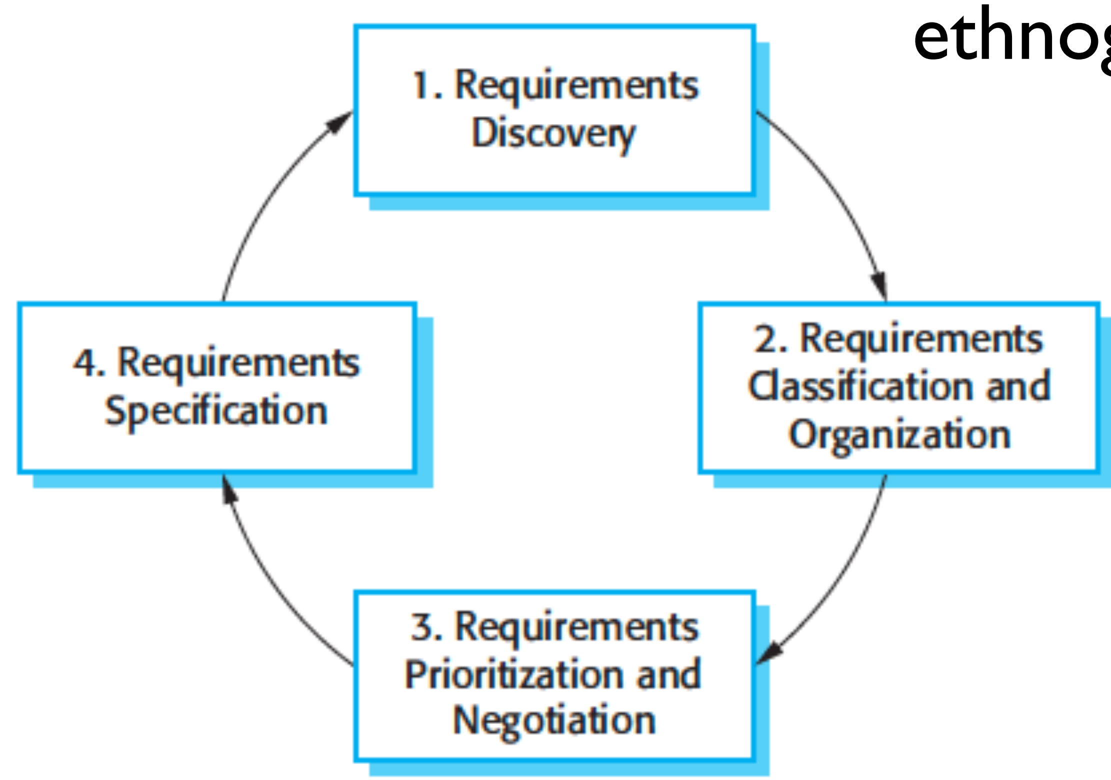
Requirements Elicitation and Analysis

interviewing

ethnography

stories

scenarios



(Fig 4.7, Sommerville)

Requirements Validation

TYPES

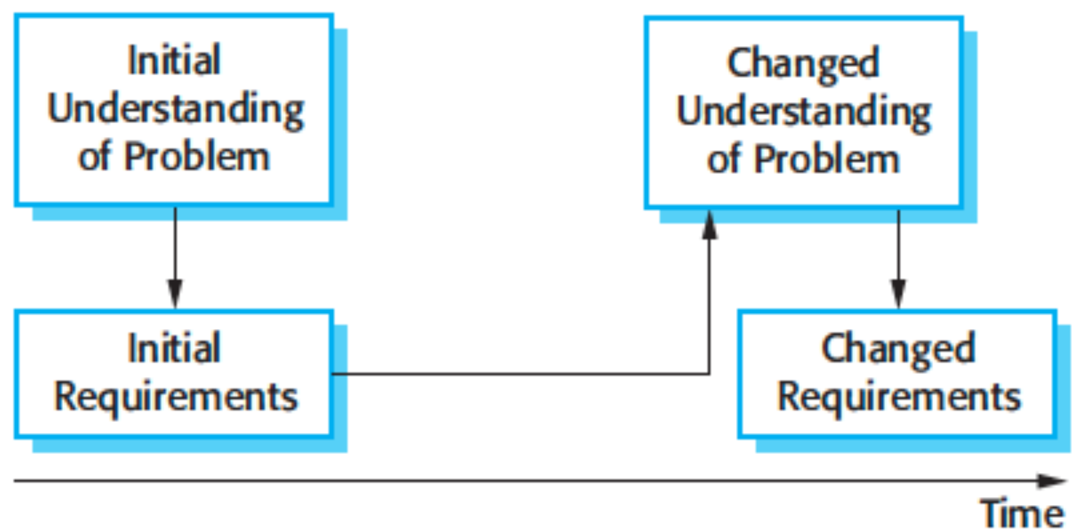
validity checks (real needs)

consistency checks

completeness checks

realism checks

verifiability (measure)



(Fig 4.18, Sommerville)

VALIDATION

reviews

prototyping

test-case generation

Requirements Management

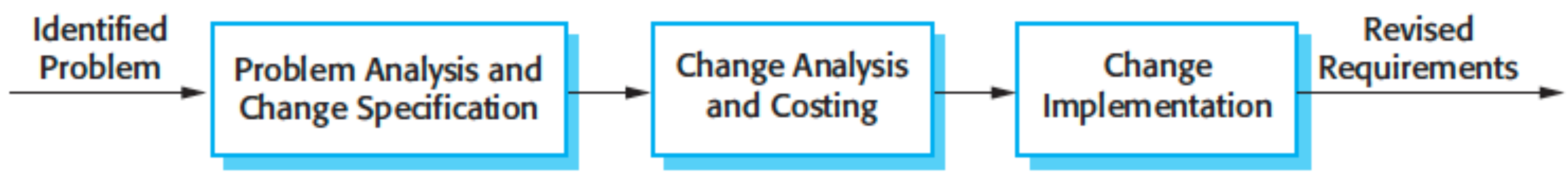
Planning

identification

large number of requirements → change management

traceability

tool support



(Fig 4.19, Sommerville)

Requirements Change Management

stories

value
and
dimension

Agile Requirements

product backlog

can be changed

sprint backlog

preserve goal

Complete specifications are not always possible

¿ S, P, and E systems ?

Lehman’s Law

Different classifications of software systems

In S-systems the problem is formally defined

math library

Implement the specification

P-systems implement a model of the problem

chess game

Define the abstraction and implement it

E-systems implements a model of the problem, which becomes part of the reality

automated stock trading

Define the abstraction,
implement it,
and …
change the world

The impact of Lehman's Law on Requirements Engineering?

the requirements are a model of reality

and models change