

Chapter 4 – Requirements Engineering

Lecture 2

4.2 The software requirements document

- ✧ The software requirements document (or called software requirements specification, SRS) is the official statement of what the system **developers** should implement.
- ✧ Should include both a definition of user requirements and a detailed 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.
 - ✧ Recall that use case diagrams never specify **how** software should work.
 - ✧ Step-by-Step descriptions somehow help you “imagine” how the system should work but still not as detailed as class or sequence diagrams

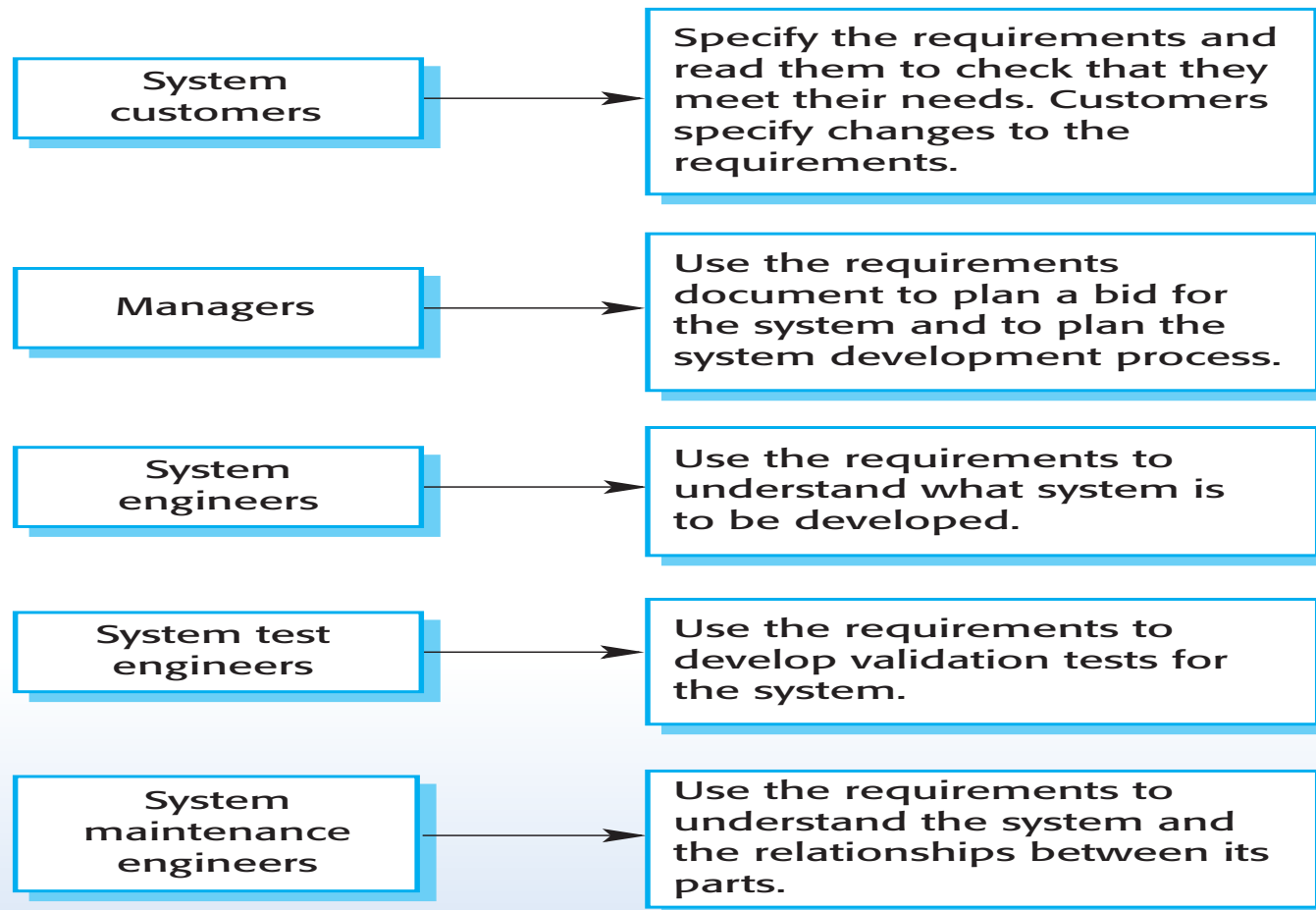
Agile methods and requirements

- ✧ 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- ✧ The document is therefore always out of date.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' (discussed in Chapter 3).
 - ✧ But still be better to have **business and dependability requirements**. (because people tend to forget after long time)
- ✧ 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.

Requirements document variability

- ✧ It is hard to balance the understandability of req. docs among **different types of users**.
 - ✧ For clients, for developers, for testers, for maintainers...etc. (e.g., clients vs. developers)
- ✧ Information in requirements document depends on **type of system** (e.g., business vs. critical) and the **development process** used
 - ✧ Systems developed incrementally will, typically, have less detail in the requirements document.
 - ✧ Focus on user req., high-level and **NFR**
- ✧ Requirements documents standards have been designed e.g., **IEEE standard**. These are mostly applicable to the requirements for large systems engineering projects.
- ✧ How details the spec should be also depends on the type of software

Figure 4.6 Users of a requirements document



The structure of a requirements document

Chapter	Description
Preface	This should define the expected readership of the document and describe its <u>version history</u> , 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 <u>need for the system</u> . It should briefly describe the <u>system's functions</u> and explain <u>how it will work with other systems</u> . 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 <u>technical terms</u> 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 <u>user</u> . The <u>nonfunctional system requirements</u> should also be described in this section. This description may use natural language, diagrams , or other notations that are <u>understandable to customers</u> . Product and process standards that must be followed should be specified.
System architecture	This chapter should present a <u>high-level overview</u> of the anticipated <u>system architecture</u> , showing the <u>distribution of functions across system modules</u> . Architectural components that are reused should be highlighted.

The structure of a requirements document

Chapter	Description
System requirements specification	This should describe the <u>functional and nonfunctional requirements</u> in more detail. If necessary, further detail may also be added to the nonfunctional requirements. <u>Interfaces</u> to other systems may be defined.
System models	This might include <u>graphical system models</u> showing the relationships between the system components and the system and its environment. Examples of possible models are <u>object models</u> , <u>data-flow models</u> , or <u>semantic data models</u> .
System evolution	This should describe the fundamental <u>assumptions</u> on which the system is based, and any <u>anticipated changes</u> 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, <u>hardware and database descriptions</u> . 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.

4.3 Requirements 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**. Specify **external** behavior only. Usually written in natural languages with intuitive diagrams or forms/tables (**don't use technical terms**).
- ✧ System requirements are more detailed requirements and may include more technical information. Can be written in natural languages/math models/diagrams...
- ✧ The requirements may be part of a **contract** for the system development
 - It is therefore important that these are as complete as possible.

Requirements and design

- ✧ In principle, **requirements** should state **what** the system should do (external behaviour and operational constraints) 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 (you need to know the dependencies and constraints from other systems);
 - The use of a specific **architecture** to satisfy non-functional requirements (e.g., reliability) may be a domain requirement.
 - This may be the consequence of a regulatory requirement (safety).

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 <u>interface specifications</u> .
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; <u>UML sequence diagrams and state chart diagrams</u> are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as <u>finite-state machines or sets</u> . 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**.
- ✧ But also vague, ambiguous, and the meaning depends on the background of the reader.

Guidelines for writing requirements

- ✧ Invent 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. (e.g., software architecture, modularization...etc)
- ✧ Include an explanation (rationale) of why a requirement is necessary.
 - ✧ This helps decide *what changes would be undesirable* if rationale has clear association with requirement

Problems with natural language

- ✧ Lack of clarity
 - Precision is difficult.
- ✧ Requirements confusion/tangling
 - 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.)*

Structured specifications

- ✧ An approach to writing requirements where the freedom of the requirements writer is limited and requirements are written in a standard way.
- ✧ This works well for some types of requirements e.g., requirements for embedded control system but is sometimes too rigid for writing business system requirements.

Form-based (structured) 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.

A structured specification of a requirement for an insulin pump

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 for an insulin pump

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 r_0 is replaced by r_1 then r_1 is replaced by r_2 .

Side effects None.


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.

Tabular specification of computation for an insulin pump

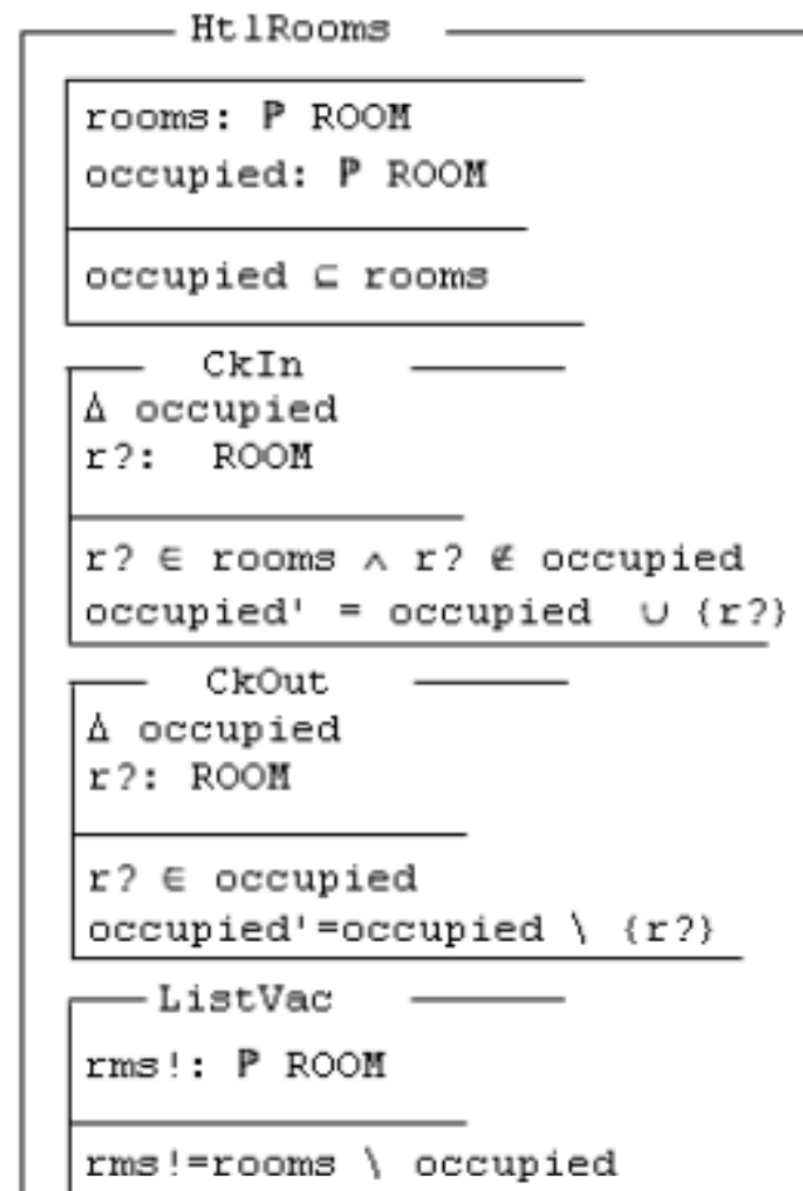
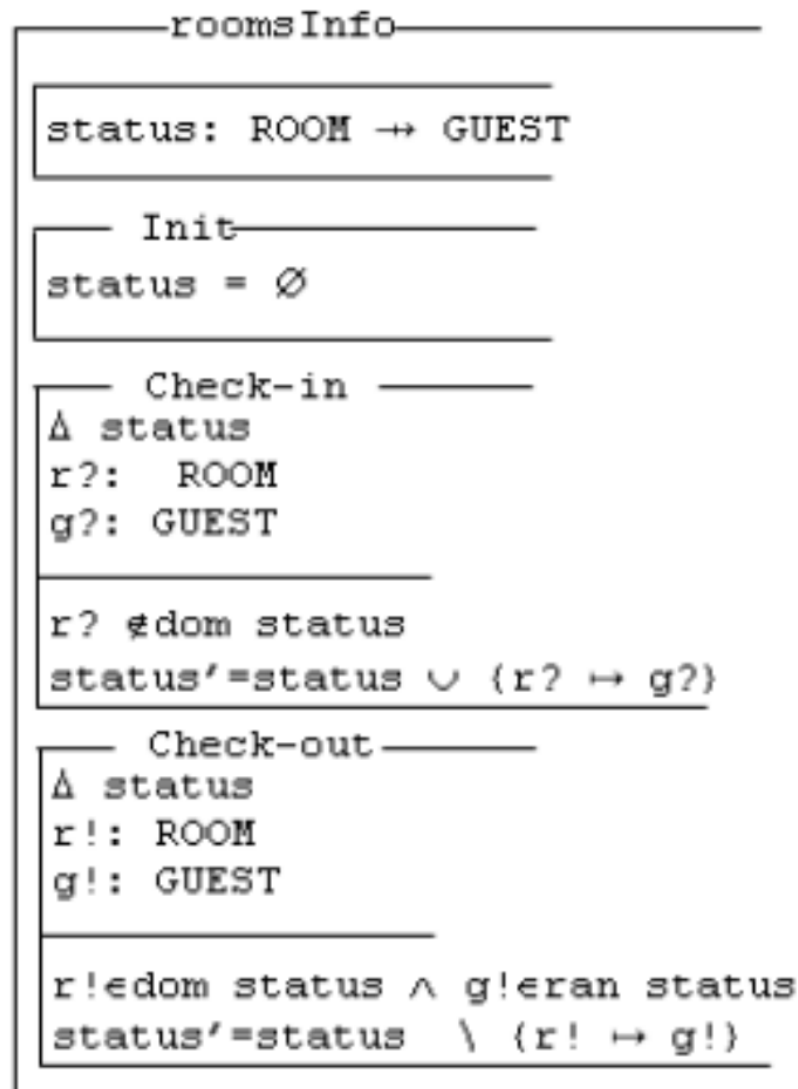
Condition	Action
Sugar level falling ($r_2 < r_1$)	CompDose = 0
Sugar level stable ($r_2 = r_1$)	CompDose = 0
Sugar level increasing and rate of increase decreasing ($(r_2 - r_1) < (r_1 - r_0)$)	CompDose = 0
Sugar level increasing and rate of increase stable or increasing ($(r_2 - r_1) \geq (r_1 - r_0)$)	CompDose = $\text{round}((r_2 - r_1)/4)$ If rounded result = 0 then CompDose = MinimumDose

List specification (Algebraic approach)

LIST (Elem)	
sort List imports INTEGER	A FIFO linear list 
Defines a list where elements are added at the end and removed from the front. The operations are Create, which brings an empty list into existence; Cons, which creates a new list with an added member; Length, which evaluates the list size; Head, which evaluates the front element of the list; and Tail, which creates a list by removing the head from its input list. Undefined represents an undefined value of type Elem.	
Create → List Cons (List, Elem) → List Head (List) → Elem Length (List) → Integer Tail (List) → List	Operator names + type info for argument(s) & result
<ol style="list-style-type: none">1 Head (Create) = Undefined exception (empty list)2 Head (Cons (L, v)) = if L=Create then v else Head (L)3 Length (Create) = 04 Length (Cons (L, v)) = Length (L) + 15 Tail (Create) = Create6 Tail (Cons (L, v)) = if L=Create then Create else Cons (Tail (L), v)	

Defines Tail in terms of
Create and Cons


goal: reading a specification (in Z)



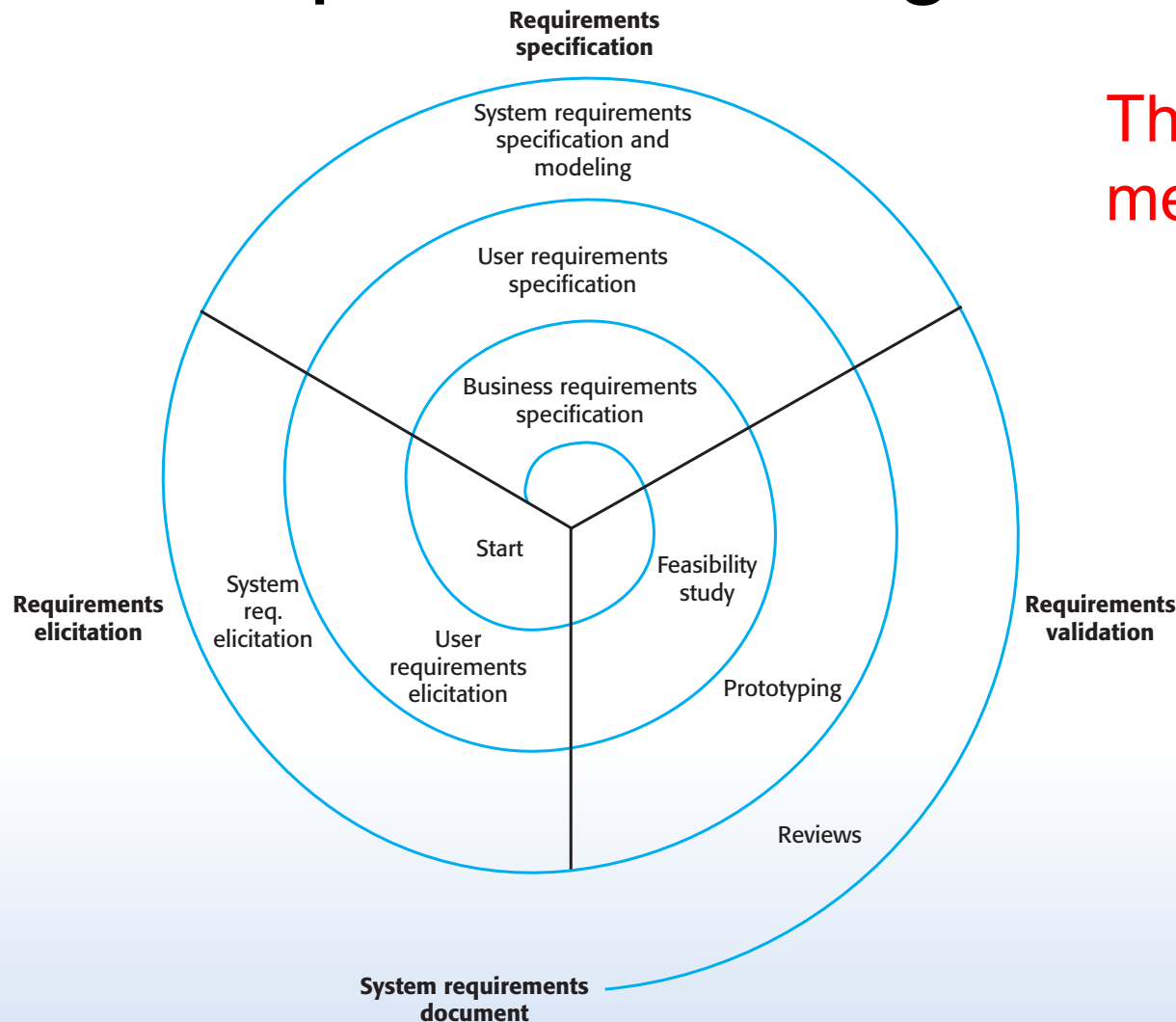
VDM++ examples

- <http://overturetool.org/download/examples/VDM++>

4.4 Requirements engineering processes

- ✧ The processes used for RE vary widely depending on the application domain, the people involved and the organisation developing the requirements.
- ✧ However, there are a number of generic activities common to all processes
 - Domain understanding and feasibility study;
 - Requirements elicitation and analysis;
 - Requirements validation;
 - Requirements management.
- ✧ In practice, RE is an **iterative** activity in which these processes are **interleaved**.

Figure 4.12 A **spiral** view of the requirements engineering process



This is not a spiral model mentioned in CH 2.

Agile process (E.g., XP) can replace prototyping so that req. and system impl. are developed together.

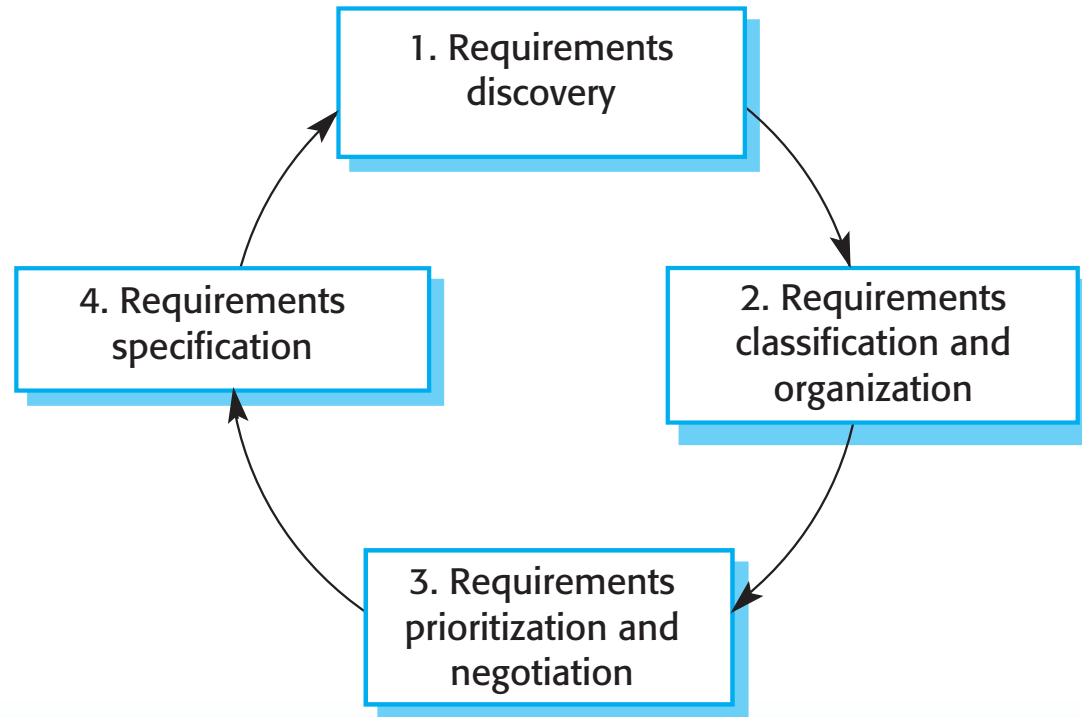
4.5 Requirements elicitation and analysis

- ✧ Sometimes called requirements elicitation or requirements discovery.
- ✧ Involves technical staff **working with customers** to find out about the application domain, the services that the system should provide and the system's operational constraints.
- ✧ May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called **stakeholders**.

Requirements elicitation and analysis

- ✧ Software engineers work with a range of system stakeholders to find out about the application domain, the services that the system should provide, the required system performance, hardware constraints, other systems, etc.
- ✧ Stages include:
 - Requirements discovery,
 - Requirements classification and organization,
 - Requirements prioritization and negotiation,
 - Requirements specification.

Figure 4.13 The requirements elicitation and analysis process



Process activities

- ✧ Requirements discovery
 - Interacting with stakeholders to discover their requirements. **Domain requirements** are also discovered at this stage.
- ✧ Requirements classification and organisation
 - Groups related requirements and organises them into coherent clusters.
- ✧ Prioritisation and negotiation (among stakeholders)
 - Prioritising requirements and resolving requirements conflicts.
- ✧ Requirements specification
 - Requirements are documented and input into the next round of the spiral.

Problems of requirements analysis

- ✧ Stakeholders don't know what they really want.
- ✧ Stakeholders express requirements in **their own terms**.
- ✧ Different stakeholders may have **conflicting** requirements.
- ✧ Organisational and political factors may influence the system requirements.
- ✧ The **requirements change** during the analysis process. New stakeholders may emerge and the business environment may change.

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.