



Requirements Engineering

Requirements Engineering

- ◊ The **process of establishing the services** that the customer requires from a system **and the constraints** under which it operates and is developed.
- ◊ The requirements themselves are the **descriptions of the system services and constraints** that are generated during the requirements engineering process.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

3



Requirements Abstraction (Davis)

"If a company wishes to let a contract for a large software development project, it must define its needs in a sufficiently **abstract** way that a solution is not pre-defined. The requirements must be written so that several contractors can bid for the contract, offering, perhaps, different ways of meeting the client organisation's needs. Once a contract has been awarded, the contractor must write a system definition for the client in **more detail** so that the client understands and can validate what the software will do. Both of these documents may be called the requirements document for the system."

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

5



Objectives

◊ To learn about

- Functional and non-functional requirements
- The software requirements document
- Requirements specification
- Requirements engineering processes
- Requirements elicitation and analysis
- Requirements validation
- Requirements management

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

2



What is a Requirement?

- ◊ It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.
- ◊ This is inevitable as requirements may serve a dual function
 - May be the basis for a bid for a contract - therefore must be open to interpretation;
 - May be the basis for the contract itself - therefore must be defined in detail;
- ◊ Both these statements may be called requirements.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

4



Types of requirement

◊ User requirements

- Statements in **natural language plus diagrams** of the **services** the system provides and its **operational constraints**.
- Written for customers.

◊ System requirements

- A structured document setting out **detailed descriptions of the system's functions, services and operational constraints**.
- Defines what should be implemented so may be part of a contract between client and contractor.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

6

Example**The Mental Health Care Patient Management System (MHC-PMS) Case Study**

- ◊ A system used to maintain records of people receiving care for mental health problems.
- A patient information system to support mental health care
 - A medical information system that maintains information about patients suffering from mental health problems and the treatments that they have received.
- ◊ Most mental health patients do not require dedicated hospital treatment but need to attend specialist clinics regularly where they can meet a doctor who has detailed knowledge of their problems.
- ◊ To make it easier for patients to attend, these clinics are not just run in hospitals.
 - They may also be held in local medical practices or community centres.

Chapter 1 Introduction

7

The MHC-PMS (Mental Health Care-Patient Management System)

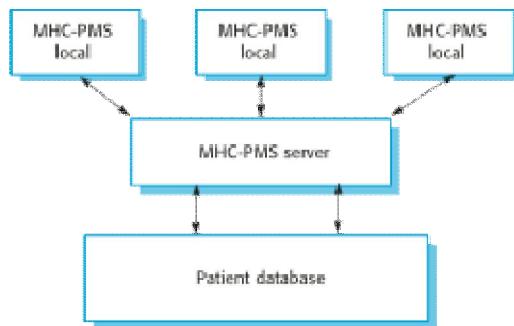
- ◊ An information system that is intended for use in clinics.
- ◊ It makes use of a **centralised database** of patient information but has also been designed to run on a PC, so that it may be accessed and used from sites that do not have secure network connectivity.
- ◊ When the local systems have secure network access, they use patient information in the database but they can download and use local copies of patient records when they are disconnected.

MHC-PMS Goals

- ◊ To generate management information that allows health service managers to assess performance against local and government targets.
- ◊ To provide medical staff with timely information to support the treatment of patients.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

8

The Organisation of the MHC-PMSSoftware Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

9

MHC-PMS Key Features

- ◊ Individual care management
 - Clinicians can create records for patients, edit the information in the system, view patient history, etc.
 - The system supports data summaries so that doctors can quickly learn about the key problems and treatments that have been prescribed.
- ◊ Patient monitoring
 - The system monitors the records of patients that are involved in treatment and issues warnings if possible problems are detected.
- ◊ Administrative reporting
 - The system generates monthly management reports showing the number of patients treated at each clinic, the number of patients who have entered and left the care system, number of patients sectioned, the drugs prescribed and their costs, etc.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

10

MHC-PMS Concerns

- ◊ Privacy
 - It is essential that patient information is confidential and is never disclosed to anyone apart from authorised medical staff and the patient themselves.
- ◊ Safety
 - Some mental illnesses cause patients to become suicidal or a danger to other people. Wherever possible, the system should warn medical staff about potentially suicidal or dangerous patients.
 - The system must be available when needed otherwise safety may be compromised and it may be impossible to prescribe the correct medication to patients.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

11

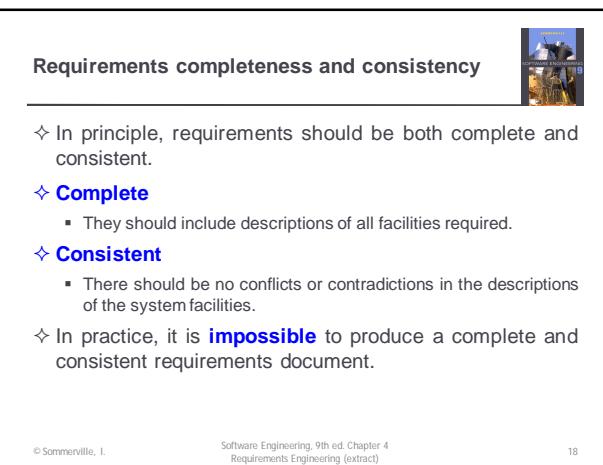
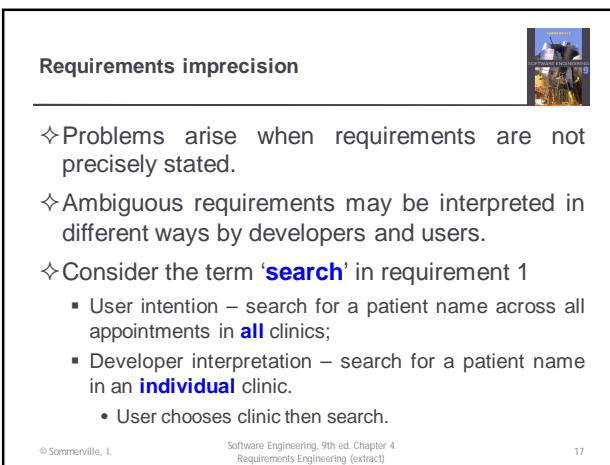
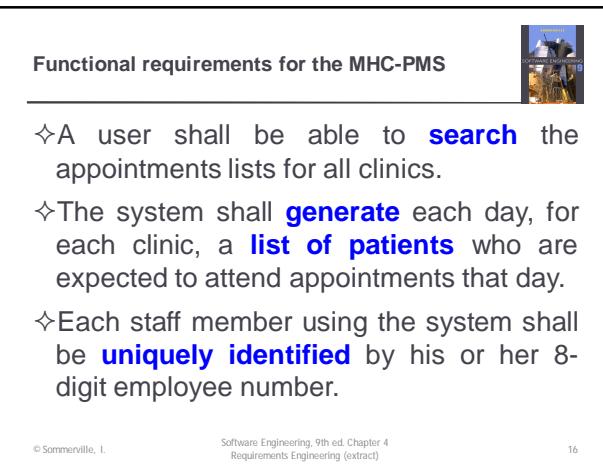
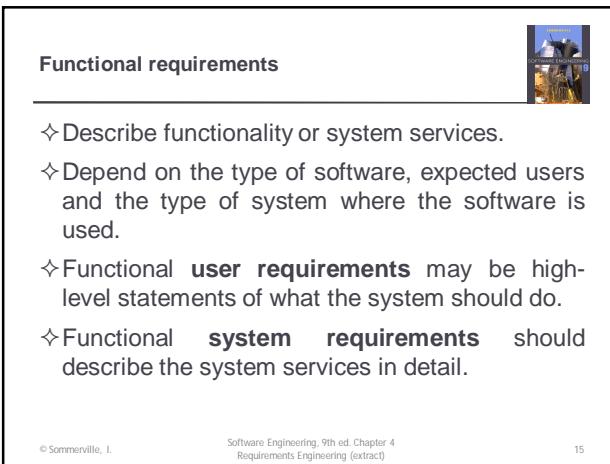
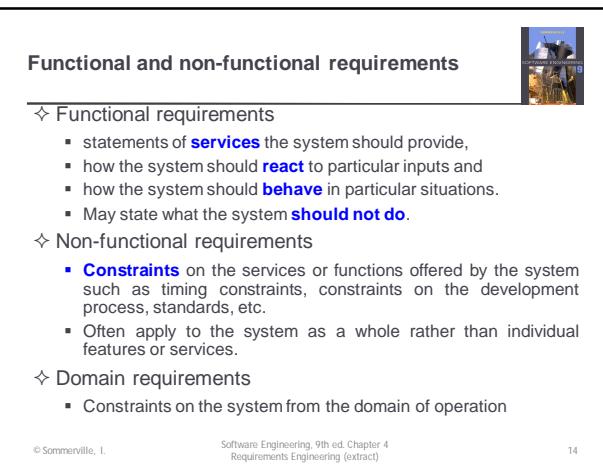
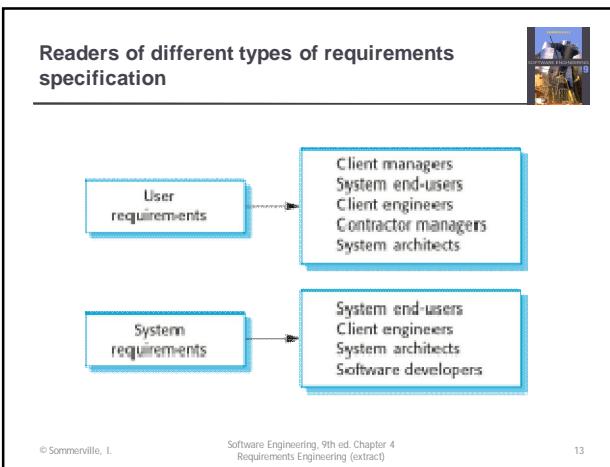
User And System Requirements**User requirement definition**

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. 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.

12



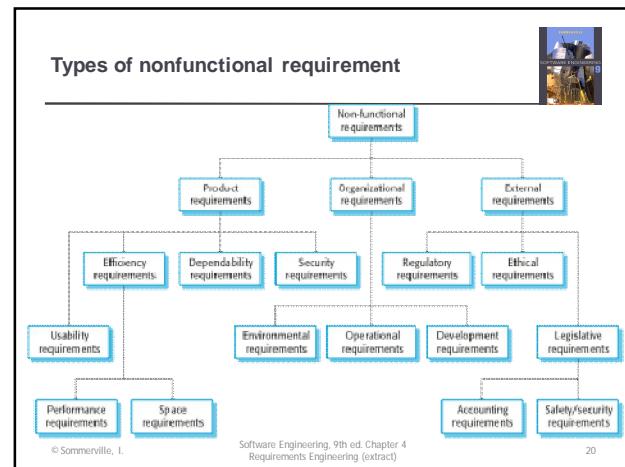
Non-functional requirements



- ◊ These define system **properties and constraints**
 - e.g. reliability, response time and storage requirements.
 - Constraints are I/O device capability etc.
- ◊ Process requirements may also be specified mandating a particular IDE, programming language or development method.
- ◊ Non-functional requirements may be more critical than functional requirements.
 - If these are not met, the system may be useless.

© Sommerville, I.
Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

19



Non-functional requirements implementation



- ◊ Non-functional requirements may affect the overall architecture of a system rather than the individual components.
 - For example, to ensure that performance requirements are met, you may have to organise the system to minimise communications between components.
- ◊ A single **non-functional requirement**, such as a security requirement, may generate a number of related **functional requirements** that define system services that are required.
 - It may also generate requirements that restrict existing requirements.

© Sommerville, I.
Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

21

Non-functional classifications



- ◊ Product requirements
 - Requirements which specify that the delivered product must behave in a particular way
 - e.g. execution speed, reliability, etc.
- ◊ Organisational requirements
 - Requirements which are a consequence of organisational policies and procedures
 - e.g. process standards used, implementation requirements, etc.
- ◊ External requirements
 - Requirements which arise from factors which are external to the system and its development process
 - e.g. interoperability requirements, legislative requirements, etc.

© Sommerville, I.
Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

22

Examples of nonfunctional requirements in the MHC-PMS



Product requirement
The MHC-PMS shall be available to all clinics during normal working hours (Mon-Fri, 0830-17.30). Downtime within normal working hours shall not exceed five seconds in any one day.

organisational 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.

23

Goals and requirements



- ◊ Non-functional requirements may be very difficult to state precisely and imprecise requirements may be difficult to **verify**.
- ◊ Goal
 - A general intention of the user such as **ease of use**.
- ◊ Verifiable non-functional requirement
 - A statement using some **measure** that can be objectively tested.
- ◊ Goals are helpful to developers as they convey the intentions of the system users.

© Sommerville, I.
Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

24

Usability requirements



- ✧ The system should be **easy to use** by medical staff and should be organised in such a way that **user errors are minimised**. (**Goal**)
- ✧ Medical staff shall be able to use all the system functions after **four hours of training**. After this training, the average number of errors made by experienced users shall not exceed **two per hour** of system use. (**Testable non-functional requirement**)

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

25

Metrics for specifying nonfunctional requirements



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

Domain requirements



- ✧ The system's **operational domain** imposes requirements on the system.
 - For example, a train control system has to take into account the braking characteristics in different weather conditions.
- ✧ Domain requirements be new functional requirements, constraints on existing requirements or define specific computations.
- ✧ If domain requirements are not satisfied, the system may be unworkable.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

27

Train protection system



- ✧ This is a domain requirement for a train protection system:
- ✧ The deceleration of the train shall be computed as:
 - $D_{train} = D_{control} + D_{gradient}$
 - where $D_{gradient}$ is $9.81\text{ms}^2 * \text{compensated gradient}/\alpha$ and where the values of $9.81\text{ms}^2/\alpha$ are known for different types of train.
- ✧ It is difficult for a non-specialist to understand the implications of this and how it interacts with other requirements.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

28

Domain Requirements Problems



Understandability

- Requirements are expressed in the language of the application domain;
- This is often not understood by software engineers developing the system.

Implicitness

- Domain specialists understand the area so well that they do not think of making the domain requirements explicit.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

29

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.
- ✧ Non-functional requirements often **constrain the system** being developed and the development process being used.
 - They often relate to the **emergent properties** of the system and therefore apply to the system as a whole.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

30

The Software Requirements Document



- ◊ The software requirements document is the official statement of what is required of the system developers.
- ◊ Should 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 out **WHAT** the system should do rather than **HOW** it should do it.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

31

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
 - they express requirements as 'user stories' or scenarios.
 - These are written on cards and the development team break them down into implementation tasks. These tasks are the basis of schedule and cost estimates. ([See next 2 slides](#))
 - The customer chooses the stories for inclusion in the next release based on their priorities and the schedule estimates.
- ◊ 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.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

32

A 'prescribing medication' story



Prescribing medication

The record of the patient must be open for input. Click on the medication field and select either 'current medication', 'new medication' or 'formulary'.

If you select 'current medication', you will be asked to check the dose; if you wish to change the dose, enter the new dose then confirm the prescription.

If you choose 'new medication', the system assumes that you know which medication you wish to prescribe. Type the first few letters of the drug name. You will then see a list of possible drugs starting with these letters. Choose the required medication. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

If you choose 'formulary', you will be presented with a search box for the approved formulary. Search for the drug required then select it. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

In all cases, the system will check that the dose is within the approved range and will ask you to change it if it is outside the range of recommended doses.

After you have confirmed the prescription, it will be displayed for checking. Either click 'OK' or 'Change'. If you click 'OK', your prescription will be recorded on the audit database. If you click 'Change', you re-enter the 'Prescribing medication' process.

33

Examples of task cards for prescribing medication



Task 1: Change dose of prescribed drug

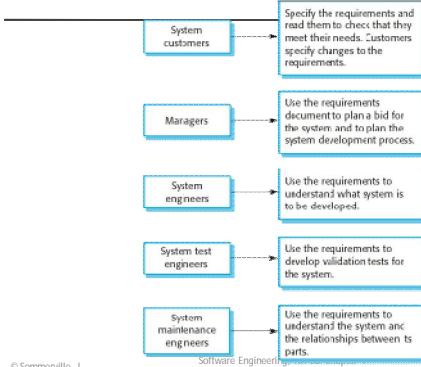
Task 2: Formulary selection

Task 3: Dose checking

Dose checking is a safety precaution to check that the doctor has not prescribed a dangerously small or large dose. Using the formulary id for the generic drug name, lookup the formulary and retrieve the recommended maximum and minimum dose. Check the prescribed dose against the minimum and maximum. If outside the range, issue an error message saying that the dose is too high or too low. If within the range, enable the 'Confirm' button.

34

Users of a requirements document



© Sommerville, I.

Software Engineering, 9th ed.
Requirements Engineering (extract)

35

Requirements document variability



- ◊ Information in requirements document depends on type of system and the approach to development used.
- ◊ Systems developed **incrementally** will, typically, have **less detail** in the requirements document.
- ◊ Requirements documents standards have been designed e.g. IEEE standard.
 - These are mostly applicable to the requirements for large systems engineering projects.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

36

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 organisation 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 organisation 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.

© Sommerville, I.
Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

38

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.
- ◊ **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.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

39

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

© Sommerville, I.
Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

40

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.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

41

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

© Sommerville, I.
Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

42

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.

© Sommerville, I. Software Engineering, 9th ed. Chapter 4 Requirements Engineering (extract) 43

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.*)

© Sommerville, I. Software Engineering, 9th ed. Chapter 4 Requirements Engineering (extract) 44

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 a control system
- ◊ but is sometimes too rigid for writing business system requirements.

© Sommerville, I. Software Engineering, 9th ed. Chapter 4 Requirements Engineering (extract) 45

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.

© Sommerville, I. Software Engineering, 9th ed. Chapter 4 Requirements Engineering (extract) 46

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.

© Sommerville, I. Software Engineering, 9th ed. Chapter 4 Requirements Engineering (extract) 47

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 r0 is replaced by r1 then r1 is replaced by r2.

Side effects None.

© Sommerville, I. Software Engineering, 9th ed. Chapter 4 Requirements Engineering (extract) 48

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.

© Sommerville, I. Software Engineering, 9th ed. Chapter 4 Requirements Engineering (extract) 49

Tabular specification of computation for an insulin pump



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 $((r2 - r1) \geq (r1 - r0))$	$CompDose = \text{round}((r2 - r1)/4)$ If rounded result = 0 then $CompDose = \text{MinimumDose}$

© Sommerville, I. Software Engineering, 9th ed. Chapter 4 Requirements Engineering (extract) 50

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

- Requirements elicitation;
- Requirements analysis;
- Requirements validation;
- Requirements management.

In practice, RE is an iterative activity in which these processes are interleaved.

© Sommerville, I. Software Engineering, 9th ed. Chapter 4 Requirements Engineering (extract) 51

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**.

© Sommerville, I. Software Engineering, 9th ed. Chapter 4 Requirements Engineering (extract) 52

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 organisation,
- Requirements prioritisation and negotiation,
- Requirements specification.

© Sommerville, I. Software Engineering, 9th ed. Chapter 4 Requirements Engineering (extract) 53

The requirements elicitation and analysis process



```

graph TD
    A[1. Requirements discovery] --> B[2. Requirements classification and organization]
    B --> C[3. Requirements prioritization and negotiation]
    C --> D[4. Requirements specification]
    D -- feedback --> A
    
```

© Sommerville, I. Software Engineering, 9th ed. Chapter 4 Requirements Engineering (extract) 54

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
 - Prioritising requirements and resolving requirements conflicts.
- ◊ Requirements specification
 - Requirements are documented and input into the next round of the spiral.

© Sommerville, I. Software Engineering, 9th ed. Chapter 4 Requirements Engineering (extract) 55

Typical Requirements Gathering Situation...



The comic strip illustrates a common interaction between a software developer and a user during requirements gathering. The developer asks for specific requirements before starting, while the user wants the developer to design the software. The user's responses are increasingly vague and abstract, such as 'whatever I design it to do' and 'tell you my requirements'. The developer becomes increasingly frustrated.

© Scott Adams, Inc./Dist. by UFS, Inc.

Problems of requirements elicitation and 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.

© Sommerville, I. Software Engineering, 9th ed. Chapter 4 Requirements Engineering (extract) 57

Problems of requirements elicitation and analysis ...



- ◊ Users won't commit to a set of written requirements
- ◊ Users insist on new requirements after the cost and schedule have been fixed.
- ◊ Communication with users is slow
- ◊ Users often do not participate in reviews or are incapable of doing so.
- ◊ Users are technically unsophisticated
- ◊ Users don't understand the software development process
- ◊ Programmers usually lack the people skills and real world knowledge needed to understand a business process properly

Software Engineering, 9th ed. Chapter 4 Requirements Engineering (extract) 58

Key points



- ◊ The software requirements document is an agreed statement of the system requirements.
 - It should be organised 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 organisation**,
 - requirements **negotiation** and
 - requirements **documentation**.

© Sommerville, I. Software Engineering, 9th ed. Chapter 4 Requirements Engineering (extract) 59

Requirements discovery



- ◊ The process of **gathering information** about the required and existing systems and distilling the **user and system requirements** from this information.
- ◊ Interaction is with system stakeholders from managers to external regulators.
- ◊ Systems normally have a range of stakeholders.

© Sommerville, I. Software Engineering, 9th ed. Chapter 4 Requirements Engineering (extract) 60

Stakeholders in the MHC-PMS



- ◊ **Patients** whose information is recorded in the system.
- ◊ **Doctors** who are responsible for assessing and treating patients.
- ◊ **Nurses** who coordinate the consultations with doctors and administer some treatments.
- ◊ **Medical receptionists** who manage patients' appointments.
- ◊ **IT staff** who are responsible for installing and maintaining the system.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

61

Stakeholders in the MHC-PMS



- ◊ A **medical ethics manager** who must ensure that the system meets current ethical guidelines for patient care.
- ◊ **Health care managers** who obtain management information from the system.
- ◊ **Medical records staff** who are responsible for ensuring that system information can be maintained and preserved, and that record keeping procedures have been properly implemented.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

62

Interviewing



- ◊ Formal or informal interviews with stakeholders are part of most RE processes.
- ◊ Types of interview
 - Closed interviews based on pre-determined list of questions
 - Open interviews where various issues are explored with stakeholders.
- ◊ Effective interviewing
 - Be open-minded, avoid pre-conceived ideas about the requirements and are willing to listen to stakeholders.
 - Prompt the interviewee to get discussions going using a springboard question, a requirements proposal, or by working together on a prototype system.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

63

Interviews in practice



- ◊ Normally a mix of closed and open-ended interviewing.
- ◊ Interviews are good for getting an overall understanding of what stakeholders do and how they might interact with the system.
- ◊ Interviews are not good for understanding domain requirements
 - Requirements engineers cannot understand specific domain terminology;
 - Some domain knowledge is so familiar that people find it hard to articulate or think that it isn't worth articulating.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

64

Scenarios



- ◊ Scenarios are real-life examples of how a system can be used.
- ◊ They should include
 - A description of the starting situation;
 - A description of the normal flow of events;
 - A description of what can go wrong;
 - Information about other concurrent activities;
 - A description of the state when the scenario finishes.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

65

Scenario for collecting medical history in MHC-PMS



Initial assumption: The patient has seen a medical receptionist who has created a record in the system and collected the patient's personal information (name, address, age, etc.). A nurse is logged on to the system and is collecting medical history.

Normal: The nurse searches for the patient by family name. If there is more than one patient with the same surname, the given name (first name in English) and date of birth are used to identify the patient.

The nurse chooses the menu option to add medical history.

The nurse then follows a series of prompts from the system to enter information about consultations elsewhere on mental health problems (free text input), existing medical conditions (nurse selects conditions from menu), medication currently taken (selected from menu), allergies (free text), and home life (form).

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

66

Scenario for collecting medical history in MHC-PMS

What can go wrong: The patient's record does not exist or cannot be found. The nurse should create a new record and record personal information.

Patient conditions or medication are not entered in the menu. The nurse should choose the 'other' option and enter free text describing the condition/medication.

Patient cannot/will not provide information on medical history. The nurse should enter free text recording the patient's inability/unwillingness to provide information. The system should print the standard exclusion form stating that the lack of information may mean that treatment will be limited or delayed. This should be signed and handed to the patient.

Other activities: Record may be consulted but not edited by other staff while information is being entered.

System state on completion: User is logged on. The patient record including medical history is entered in the database, a record is added to the system log showing the start and end time of the session and the nurse involved.

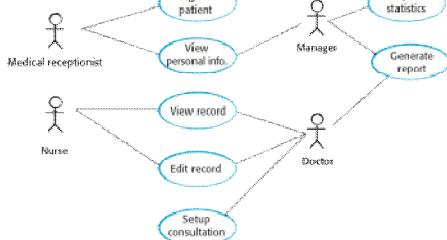
© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

67



Use cases for the MHC-PMS



© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

69

Use cases

❖ **Use-cases** are a scenario based technique in the [UML](#) which

- identify the actors in an interaction and
- describe the interaction itself.

❖ A set of use cases should [describe all possible interactions with the system](#).

❖ High-level graphical model supplemented by more detailed tabular description.

❖ **Sequence diagrams** may be used to add detail to use-cases by showing the sequence of event processing in the system.

© Sommerville, I.
Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

68



Ethnography

❖ A social scientist spends a considerable time observing and analysing how people actually work.

❖ People do not have to explain or articulate their work.

❖ Social and organisational factors of importance may be observed.

❖ Ethnographic studies have shown that work is usually richer and more complex than suggested by simple system models.

© Sommerville, I.
Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

70



Scope of ethnography

- ❖ Requirements that are derived from the way that people actually work rather than the way in which process definitions suggest that they ought to work.
- ❖ Requirements that are derived from cooperation and awareness of other people's activities.
 - Awareness of what other people are doing leads to changes in the ways in which we do things.
- ❖ Ethnography is effective for understanding existing processes but cannot identify new features that should be added to a system.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

71



Focused ethnography

❖ Developed in a project studying the air traffic control process

❖ **Combines ethnography with prototyping**

- Prototype development results in unanswered questions which focus the ethnographic analysis.

❖ The problem with ethnography is that it studies existing practices which may have some historical basis which is no longer relevant.

© Sommerville, I.
Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

72



Requirements validation



- ✧ Concerned with demonstrating that the requirements define the system that 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.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

73

Requirements checking



- ✧ **Validity.** Does the system provide the functions which best support the customer's needs?
- ✧ **Consistency.** Are there any requirements conflicts?
- ✧ **Completeness.** Are all functions required by the customer included?
- ✧ **Realism.** Can the requirements be implemented given available budget and technology
- ✧ **Verifiability.** Can the requirements be checked?

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

74

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.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

75

Requirements reviews



- ✧ Regular reviews should be held while the requirements definition is being formulated.
- ✧ Both client and contractor staff should be involved in reviews.
- ✧ Reviews may be formal (with completed documents) or informal.
 - Good communications between developers, customers and users can resolve problems at an early stage.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

76

Review checks



- ✧ **Verifiability**
 - Is the requirement realistically testable?
- ✧ **Comprehensibility**
 - Is the requirement properly understood?
- ✧ **Traceability**
 - Is the origin of the requirement clearly stated?
- ✧ **Adaptability**
 - Can the requirement be changed without a large impact on other requirements?

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

77

Requirements management



- ✧ Requirements management is the process of **managing changing requirements** during the requirements engineering process and system development.
- ✧ New requirements emerge as a system is being developed and after it has gone into use.
- ✧ You need to keep track of individual requirements and maintain links between **dependent requirements** so that you can assess the impact of requirements changes.
 - You need to establish a formal process for making change proposals and linking these to system requirements.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

78

Changing requirements



- ✧ The business and technical environment of the system always changes after installation.
 - New hardware may be introduced, it may be necessary to interface the system with other systems, business priorities may change (with consequent changes in the system support required), and new legislation and regulations may be introduced that the system must necessarily abide by.
- ✧ The people who pay for a system and the users of that system are rarely the same people.
 - **System customers** impose requirements because of organisational and budgetary constraints. These may conflict with **end-user requirements** and, after delivery, new features may have to be added for user support if the system is to meet its goals.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

79

Changing requirements



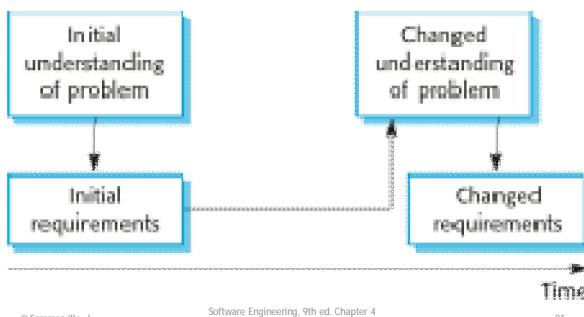
- ✧ Large systems usually have a diverse user community, with many users having different requirements and priorities that may be conflicting or contradictory.
 - The final system requirements are inevitably a compromise between them and,
 - with experience, it is often discovered that the balance of support given to different users has to be changed.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

80

Requirements evolution



© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

81

Requirements management planning



- ✧ Establishes the level of requirements management detail that is required.
- ✧ Requirements management decisions:
 - **Requirements identification** Each requirement must be uniquely identified so that it can be cross-referenced with other requirements.
 - **A change management process** This is the set of activities that assess the impact and cost of changes.
 - **Traceability policies** These policies define the relationships between each requirement and between the requirements and the system design that should be recorded.
 - **Tool support** Tools that may be used range from specialist requirements management systems to spreadsheets and simple database systems.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

82

Requirements change management



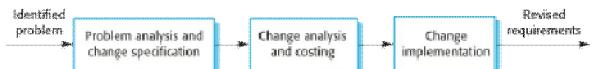
- ✧ Deciding if a requirements change should be accepted
 - **Problem analysis and change specification**
 - During this stage, the problem or the change proposal is analysed to check that it is valid. This analysis is fed back to the change requestor who may respond with a more specific requirements change proposal, or decide to withdraw the request.
 - **Change analysis and costing**
 - The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. Once this analysis is completed, a decision is made whether or not to proceed with the requirements change.
 - **Change implementation**
 - The requirements document and, where necessary, the system design and implementation, are modified. Ideally, the document should be organised so that changes can be easily implemented.

© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

83

Requirements change management



© Sommerville, I.

Software Engineering, 9th ed. Chapter 4
Requirements Engineering (extract)

84

Key points

- ✧ You can use a range of techniques for requirements elicitation including interviews, scenarios, use-cases and ethnography.
- ✧ Requirements validation is the process of checking the requirements for validity, consistency, completeness, realism and verifiability.
- ✧ Business, organisational 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.