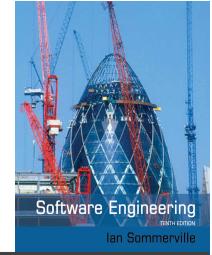




# Chapter 4 – Requirements Engineering

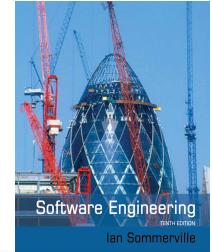
# Topics covered

---



- ✧ Functional and non-functional requirements
- ✧ Requirements engineering processes
- ✧ Requirements elicitation
- ✧ Requirements specification
- ✧ Requirements validation
- ✧ Requirements change

# Requirements engineering



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



# What is a requirement?

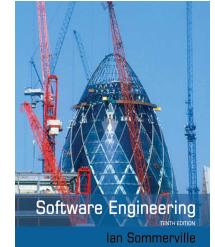
- ✧ It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.
- ✧ This is inevitable as requirements may serve a dual function
  - May be the basis for a bid for a contract - therefore must be open to interpretation;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

# 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 organization'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."

# User and system requirements



## User requirements definition

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

## System requirements specification

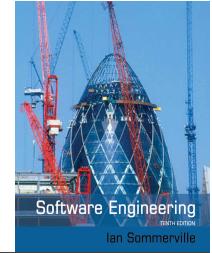
- 1.1** On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
- 1.2** The system shall generate the report for printing after 17.30 on the last working day of the month.
- 1.3** A report shall be created for each clinic and 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, 20mg, etc) separate reports shall be created for each dose unit.
- 1.5** Access to drug cost reports shall be restricted to authorized users as listed on a management access control list.

# Stakeholders



- ✧ Any person or organization who is affected by the system in some way and so who has a legitimate interest
- ✧ Stakeholder types
  - End users
  - System managers
  - System owners
  - External stakeholders

# Agile methods and requirements



- ✧ Many agile methods argue that producing detailed system requirements is a waste of time as requirements change so quickly.
- ✧ The requirements document is therefore always out of date.
- ✧ Agile methods usually use incremental requirements engineering and may express requirements as 'user stories'.
- ✧ This is practical for business systems but problematic for systems that require pre-delivery analysis (e.g. critical systems) or systems developed by several teams.



# Functional and non-functional requirements

# Functional and non-functional requirements



## ✧ Functional requirements

- Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
- May state what the system should not do.

## ✧ Non-functional requirements

- Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
- Often apply to the system as a whole rather than individual features or services.

## ✧ Domain requirements

- Constraints on the system from the domain of operation

# Functional requirements

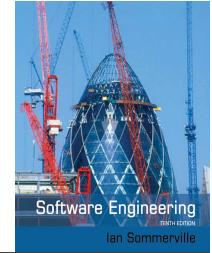


- ✧ Describe functionality or system services.
- ✧ Depend on the type of software, expected users and the type of system where the software is used.
- ✧ Functional user requirements may be high-level statements of what the system should do.
- ✧ Functional system requirements should describe the system services in detail.

# Mentcare system: functional requirements



- ✧ A user shall be able to search the appointments lists for all clinics.
- ✧ The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day.
- ✧ Each staff member using the system shall be uniquely identified by his or her 8-digit employee number.



# Requirements imprecision

- ✧ Problems arise when functional requirements are not precisely stated.
- ✧ Ambiguous requirements may be interpreted in different ways by developers and users.
- ✧ Consider the term 'search' in requirement 1
  - User intention – search for a patient name across all appointments in all clinics;
  - Developer interpretation – search for a patient name in an individual clinic. User chooses clinic then search.

# Requirements completeness and consistency



- ✧ In principle, requirements should be both complete and consistent.
- ✧ Complete
  - They should include descriptions of all facilities required.
- ✧ Consistent
  - There should be no conflicts or contradictions in the descriptions of the system facilities.

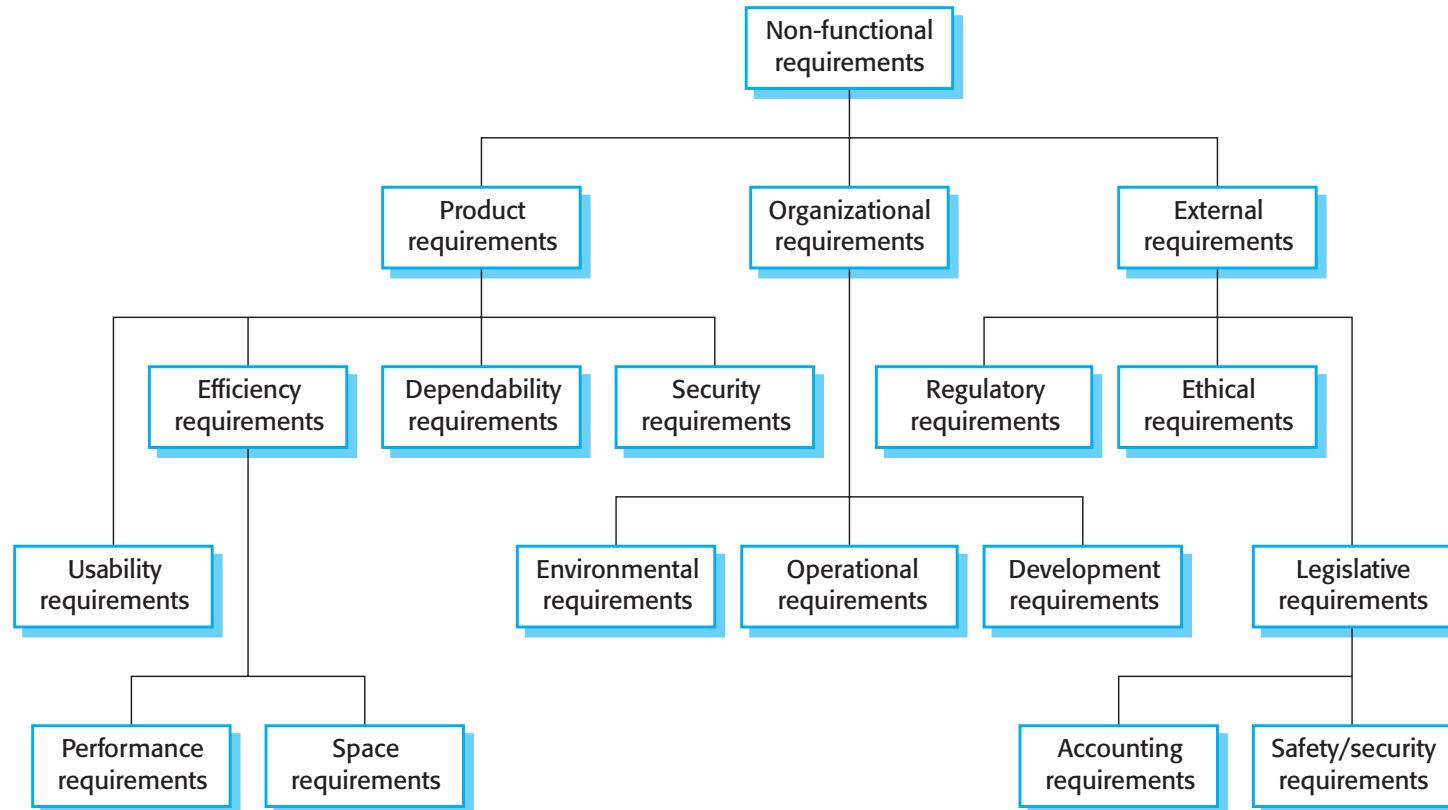
# Non-functional requirements



- ✧ These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, 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.



# Types of nonfunctional requirement



# Examples of nonfunctional requirements in the Mentcare system



## Product requirement

The Mentcare system 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.

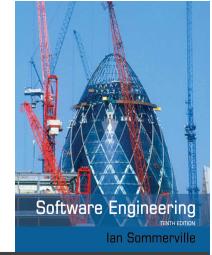
## Organizational requirement

Users of the Mentcare 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.

# Example for Usability requirements



- ✧ The system should be easy to use by medical staff and should be organized in such a way that user errors are minimized. (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)

# 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



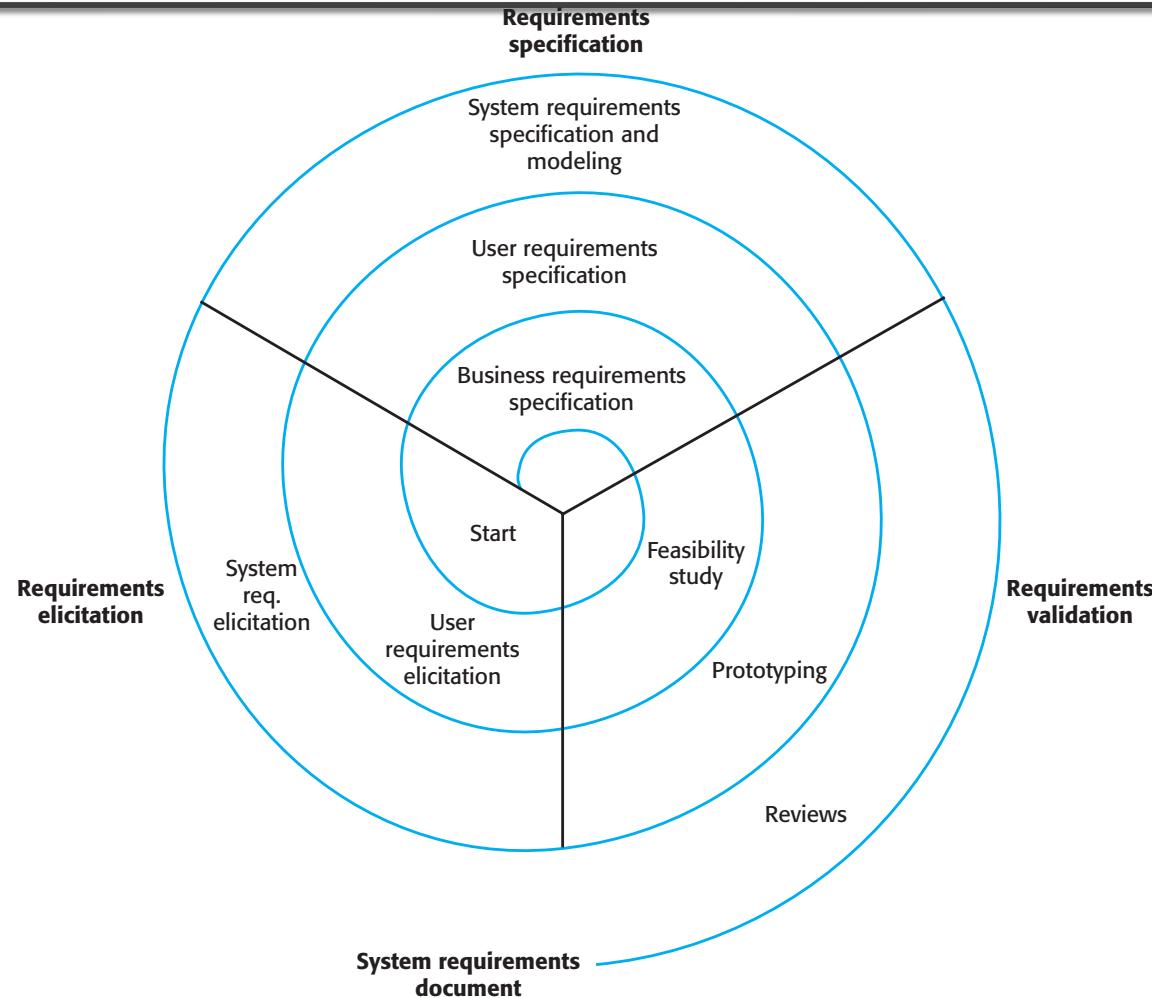
# Requirements engineering processes

# 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.

# A spiral view of the requirements engineering process





# Requirements elicitation

# Requirements elicitation



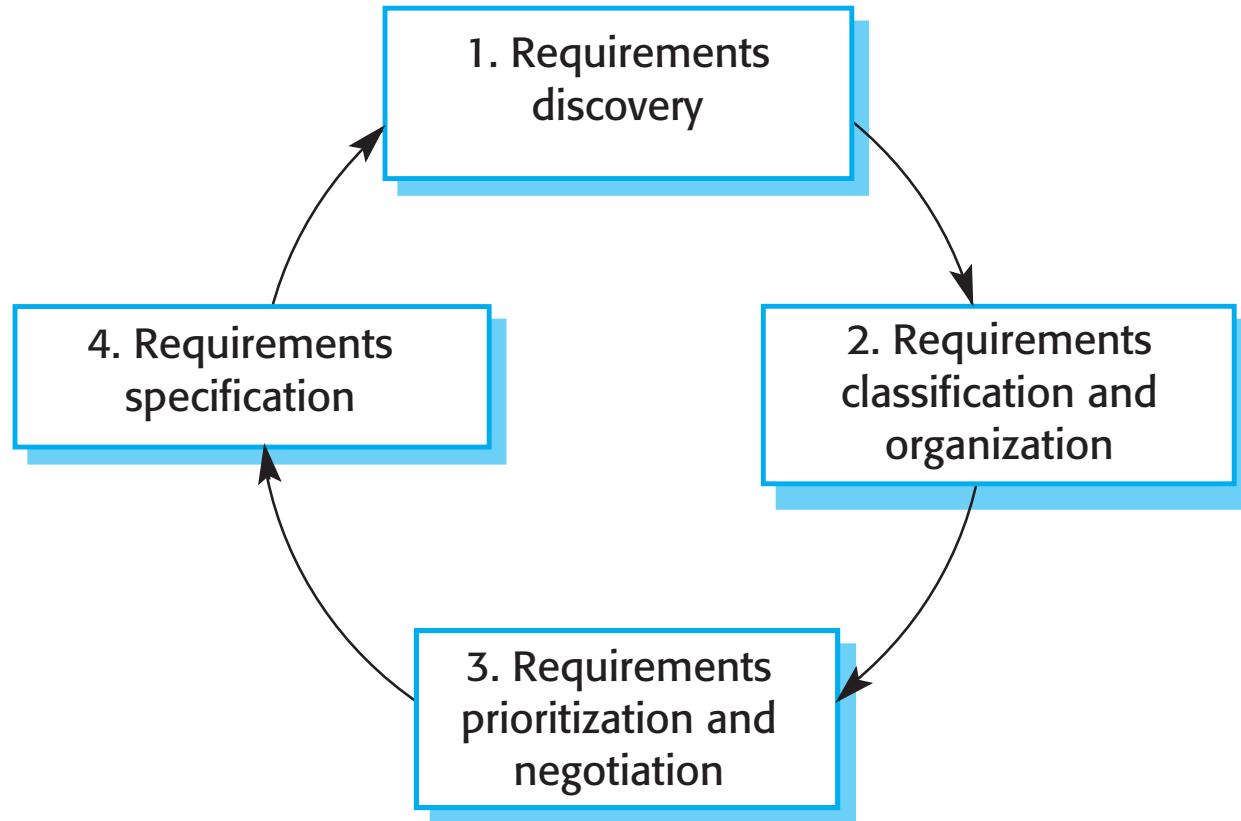
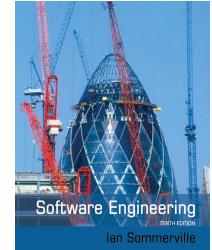
- ✧ Sometimes called requirements elicitation or requirements discovery.
- ✧ Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints.
- ✧ May involve end-users, managers, engineers involved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

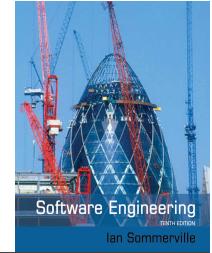
# Problems of requirements elicitation



- ✧ 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.

# The requirements elicitation 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

- Prioritising requirements and resolving requirements conflicts.

## ✧ Requirements specification

- Requirements are documented and input into the next round of the spiral.

# Methods

---

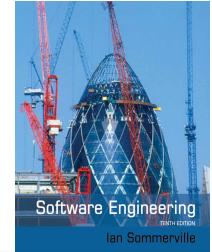


- ✧ Interviews
- ✧ Document analysis
- ✧ Stories and scenarios
- ✧ Statistical techniques
  - sampling



# Requirements specification

# Requirements specification



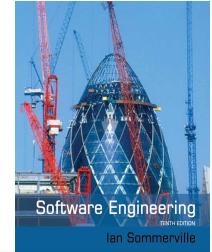
- ✧ The process of writing the user and system requirements in a requirements document.
- ✧ User requirements have to be understandable by end-users and customers who do not have a technical background.
- ✧ System requirements are more detailed requirements and may include more technical information.
- ✧ 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

# 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.

# 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



- ✧ 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.
- ✧ Avoid the use of computer jargon.
- ✧ Include an explanation (rationale) of why a requirement is necessary.

# Problems with natural language



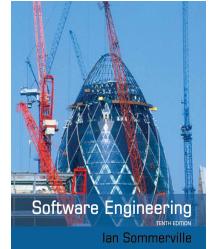
- ✧ Lack of clarity
  - Precision is difficult without making the document difficult to read.
- ✧ Requirements confusion
  - Functional and non-functional requirements tend to be mixed-up.
- ✧ Requirements amalgamation
  - Several different requirements may be expressed together.

# 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.*)

# 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.

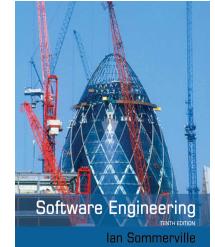
**Inputs** Current sugar reading ( $r_2$ ); the previous two readings ( $r_0$  and  $r_1$ ).

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

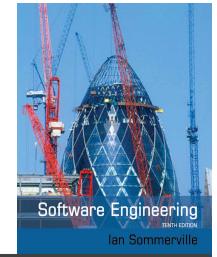
**Side effects**    None.

# Tabular specification of computation for an insulin pump



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

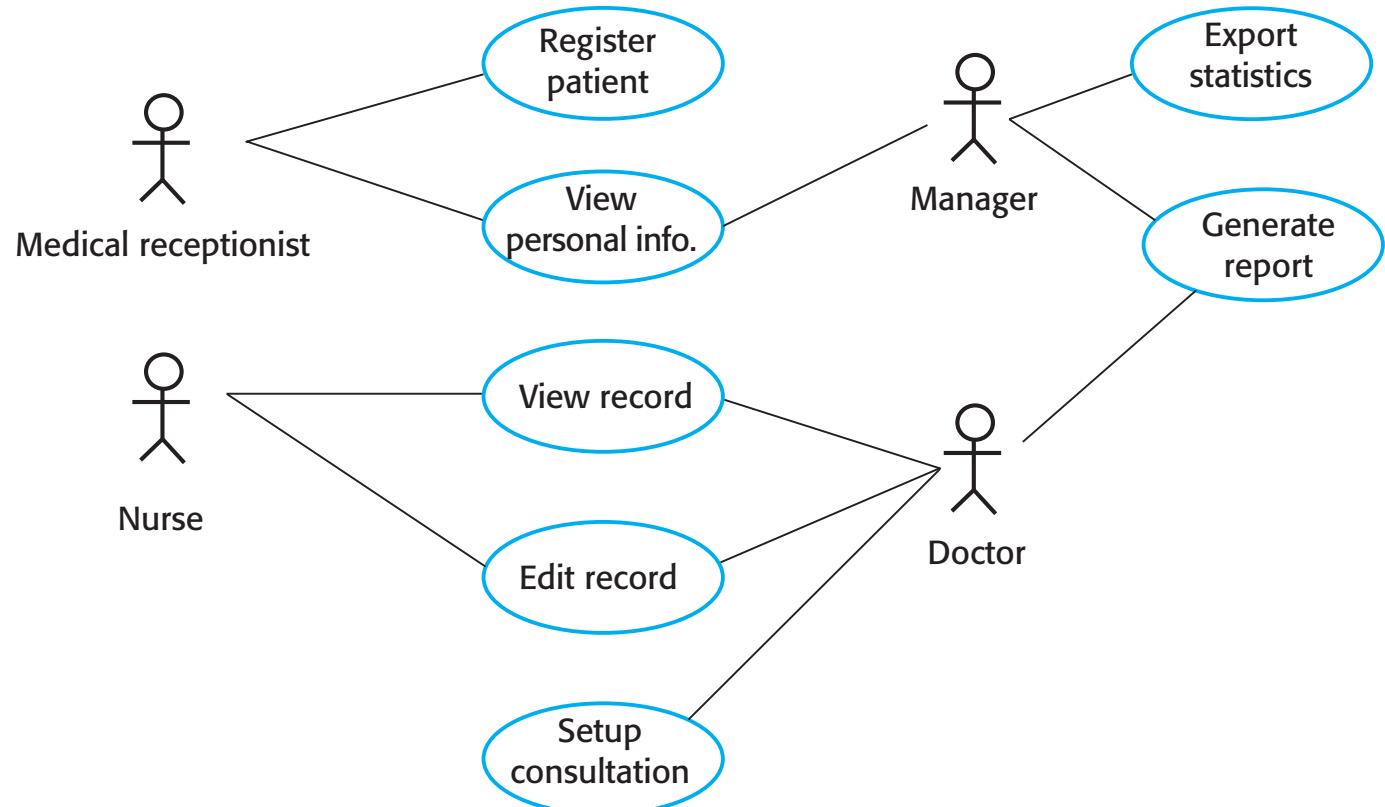
# Use cases



- ✧ Use-cases are a kind of scenario that are included in the UML.
- ✧ Use cases identify the actors in an interaction and which 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.
- ✧ UML sequence diagrams may be used to add detail to use-cases by showing the sequence of event processing in the system.



# Use cases for the Mentcare system



# The software requirements document



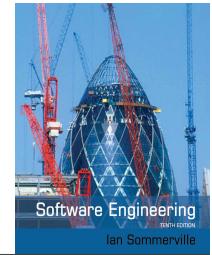
- ✧ The software requirements document is the official statement of what is required of the system developers.
- ✧ Should include both a definition of user requirements and a specification of the system requirements.
- ✧ It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it.

# 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.

# 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.



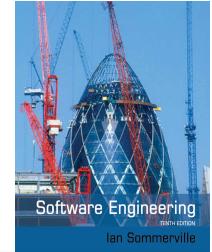
# Requirements validation

# Requirements validation



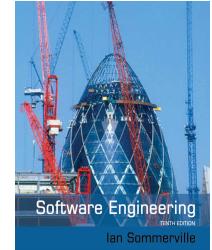
- ✧ Concerned with demonstrating that the requirements define the system that the customer really wants.
- ✧ Requirements error costs are high so validation is very important
  - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

# 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?

# 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.



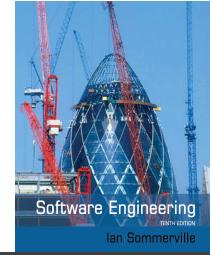
# Requirements change

# Changing requirements



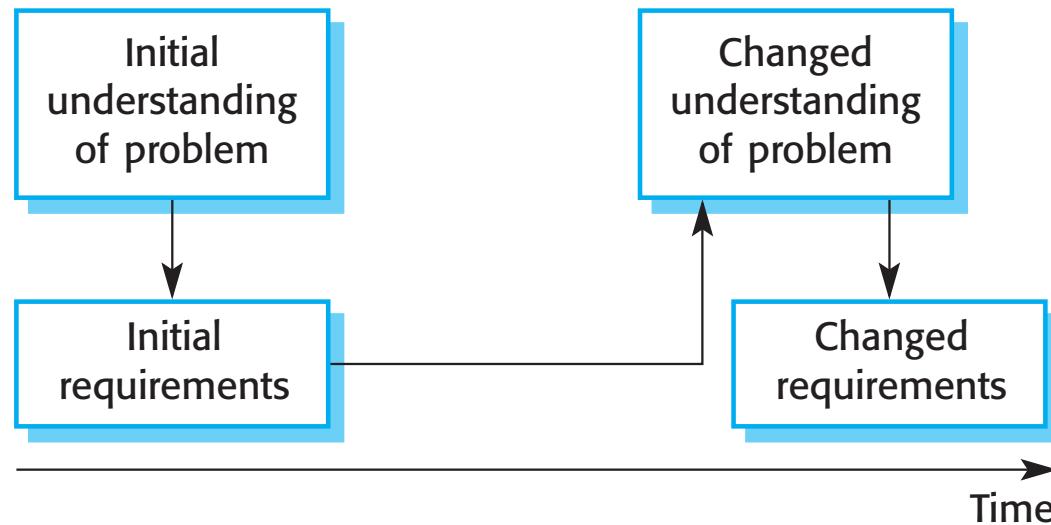
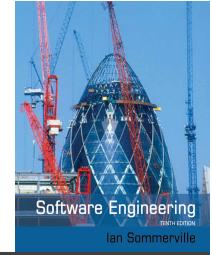
- ✧ The business and technical environment of the system always changes after installation.
  - New hardware may be introduced, it may be necessary to interface the system with other systems, business priorities may change (with consequent changes in the system support required), and new legislation 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 organizational 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.

# Changing requirements

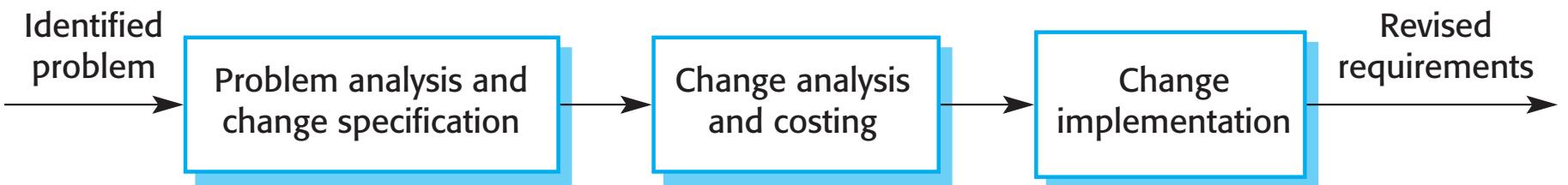


- ✧ Large systems usually have a diverse user community, with many users having different requirements and priorities that may be conflicting or contradictory.
  - The final system requirements are inevitably a compromise between them and, with experience, it is often discovered that the balance of support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

# Requirements evolution



# Requirements change management



# CMMI - Requirements Development process area (ML3)



- SG 1 Develop Customer Requirements
  - SP 1.1 Elicit Needs
  - SP 1.2 Transform Stakeholder Needs into Customer Requirements
- SG 2 Develop Product Requirements
  - SP 2.1 Establish Product and Product Component Requirements
  - SP 2.2 Allocate Product Component Requirements
  - SP 2.3 Identify Interface Requirements
- SG 3 Analyze and Validate Requirements
  - SP 3.1 Establish Operational Concepts and Scenarios
  - SP 3.2 Establish a Definition of Required Functionality and Quality Attributes
  - SP 3.3 Analyze Requirements
  - SP 3.4 Analyze Requirements to Achieve Balance
  - SP 3.5 Validate Requirements

# CMMI - Requirements Management process area (ML2)



## ■ SG 1 Manage Requirements

- SP 1.1 Understand Requirements
- SP 1.2 Obtain Commitment to Requirements
- SP 1.3 Manage Requirements Changes
- SP 1.4 Maintain Bidirectional Traceability of Requirements
- SP 1.5 Ensure Alignment Between Project Work and Requirements

# 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.

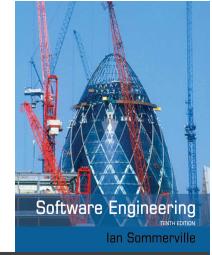
# Key points

---



- ✧ The requirements engineering process is an iterative process that includes requirements elicitation, specification and validation.
- ✧ Requirements elicitation is an iterative process that can be represented as a spiral of activities – requirements discovery, requirements classification and organization, requirements negotiation and requirements documentation.
- ✧ You can use a range of techniques for requirements elicitation including interviews and ethnography. User stories and scenarios may be used to facilitate discussions.

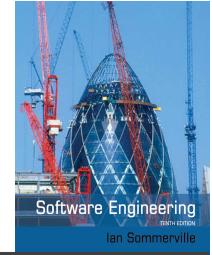
# Key points



- ✧ Requirements specification is the process of formally documenting the user and system requirements and creating a software requirements document.
- ✧ 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.

# Key points

---



- ✧ Requirements validation is the process of checking the requirements for validity, consistency, completeness, realism 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.