Module 4: Requirements Engineering

Software Engineering

Faculty of Computing
Universiti Teknologi Malaysia

Outline

- Types of Requirements
- Functional and Non-Functional Requirements
- Requirements Engineering Process
  - Requirements Elicitation
  - Requirements Specification
  - Requirement Validation
  - Requirement Change

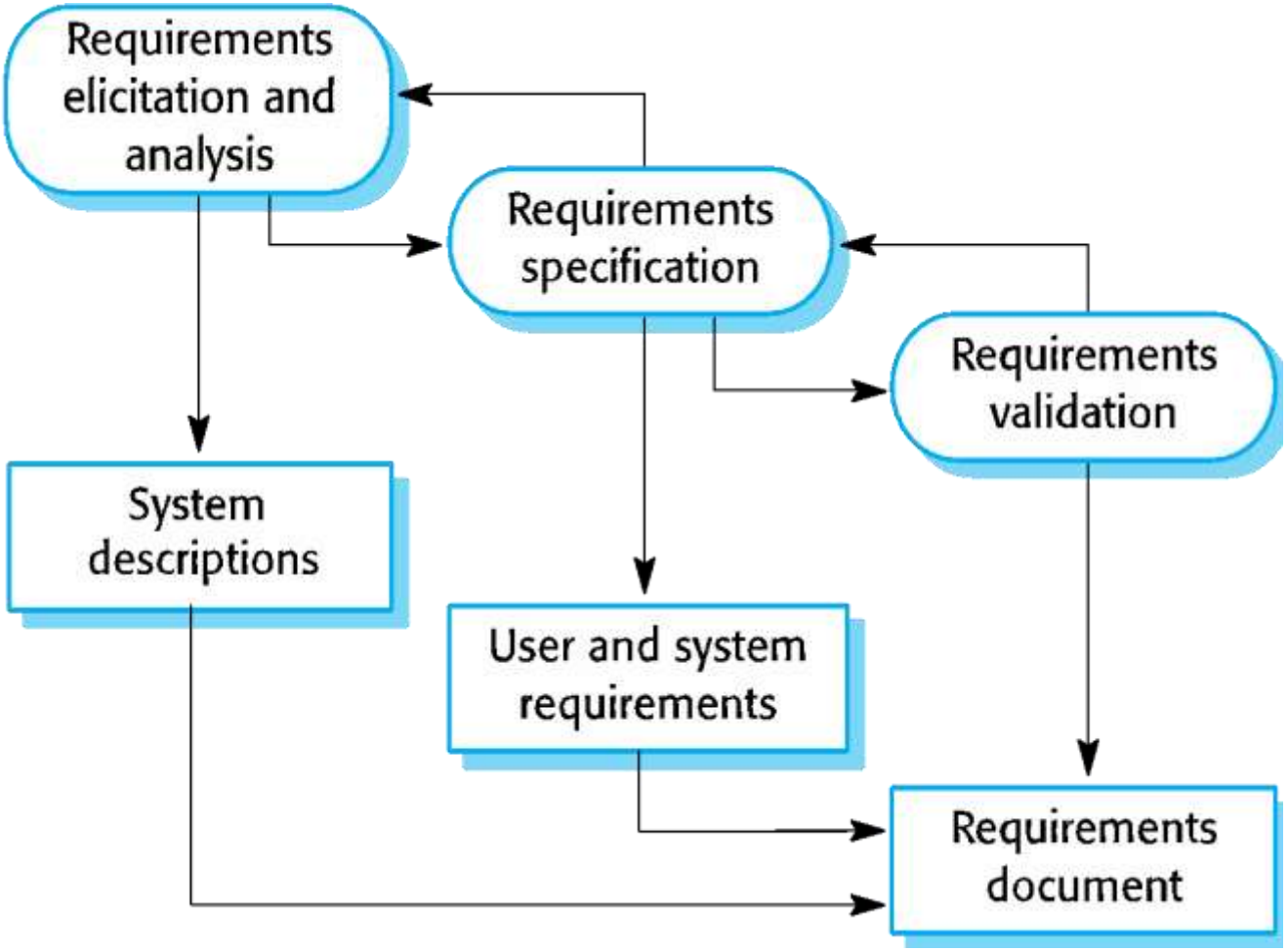
Note: The overall contents of the slide are based on the main reference that is Sommerville (2016) with other references specified directly in respective slides (if any)

Objectives

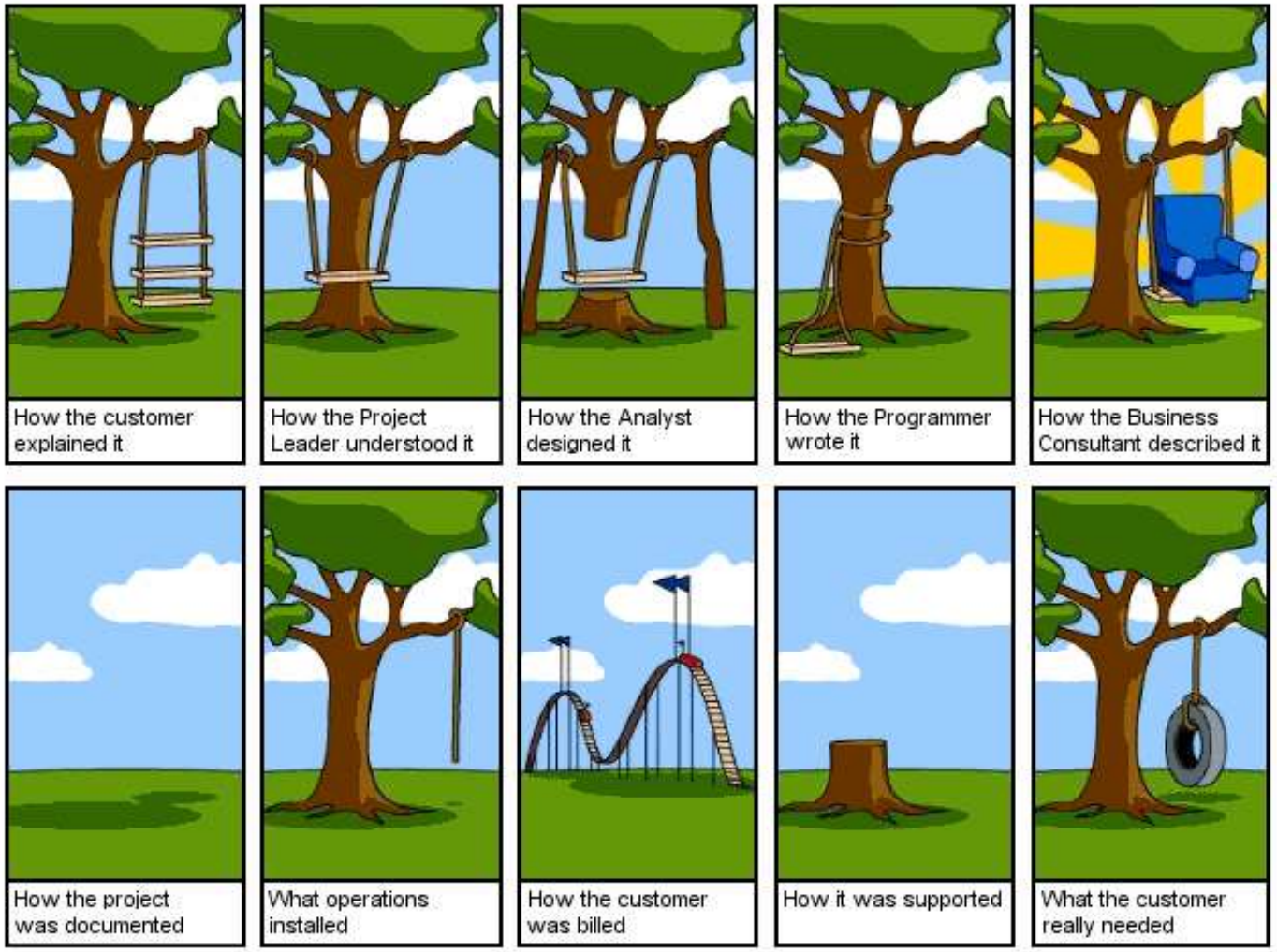
The objectives of this module are:

- To differentiate users and system requirements
- To understand functional requirements (FR) and non-functional requirements (NFR)
- To understand the main requirements engineering activities that are elicitation, analysis and validation
- To know diverse techniques to write requirements specification
- To understand the important of requirements management and how it supports requirement change.

Recap: Requirements Engineering Process in Software Process Model



Requirements in Different Perspectives



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 **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

TYPE OF REQUIREMENTS

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**

User and System Requirements

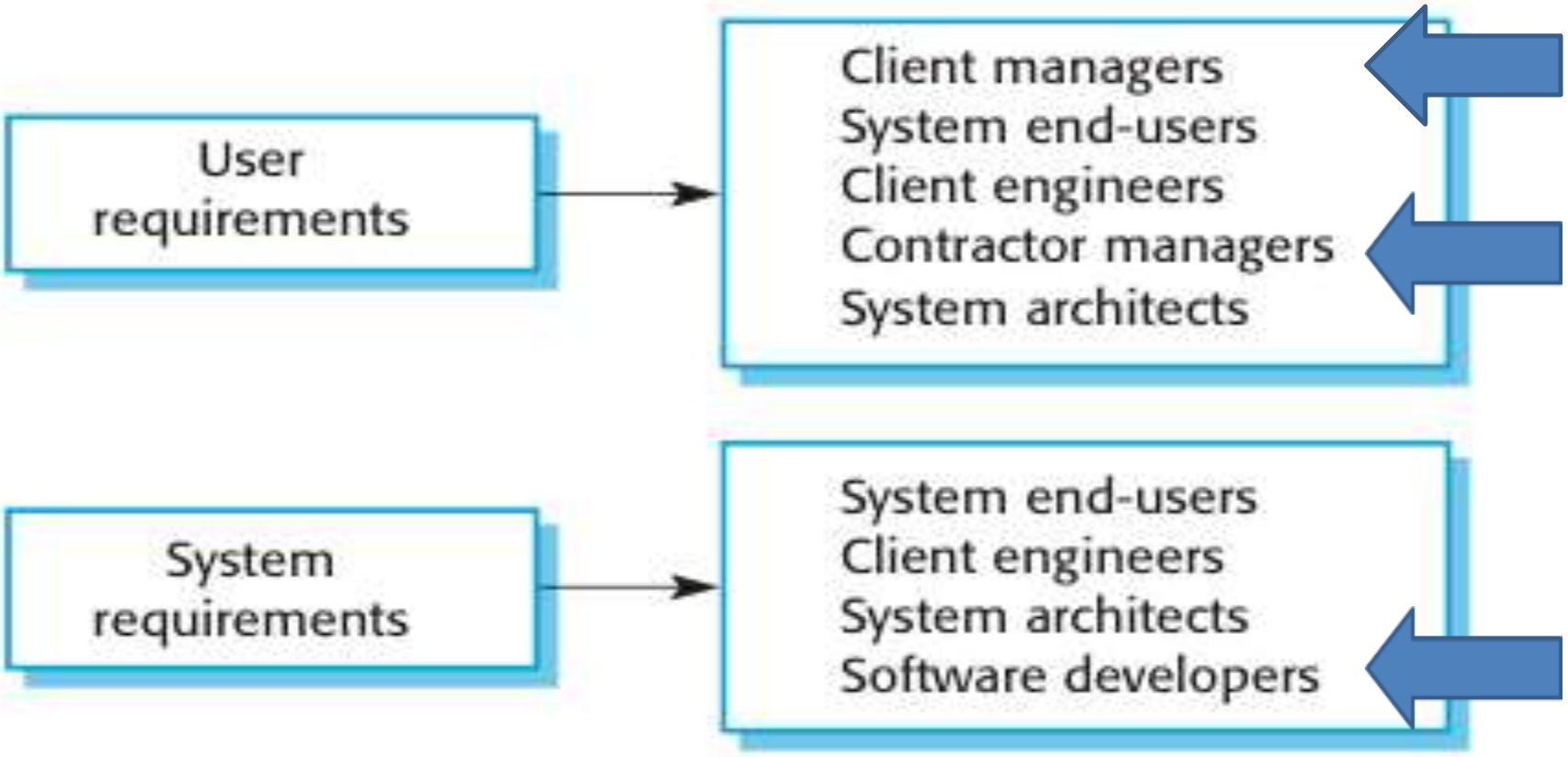
User requirements definition

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.

Readers of Different Types of Requirements Specification



FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

Functional and Non-Functional Requirements

- **Functional requirements (FR)**
  - **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 (NFR)**
  - **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
- Besides FR and NFR, there are **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

Example: Functional Requirements in Mentcare System

- 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 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.

> A user shall be able to search the appointments lists for all clinics

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
- In practice, it is impossible to produce a complete and consistent requirements document

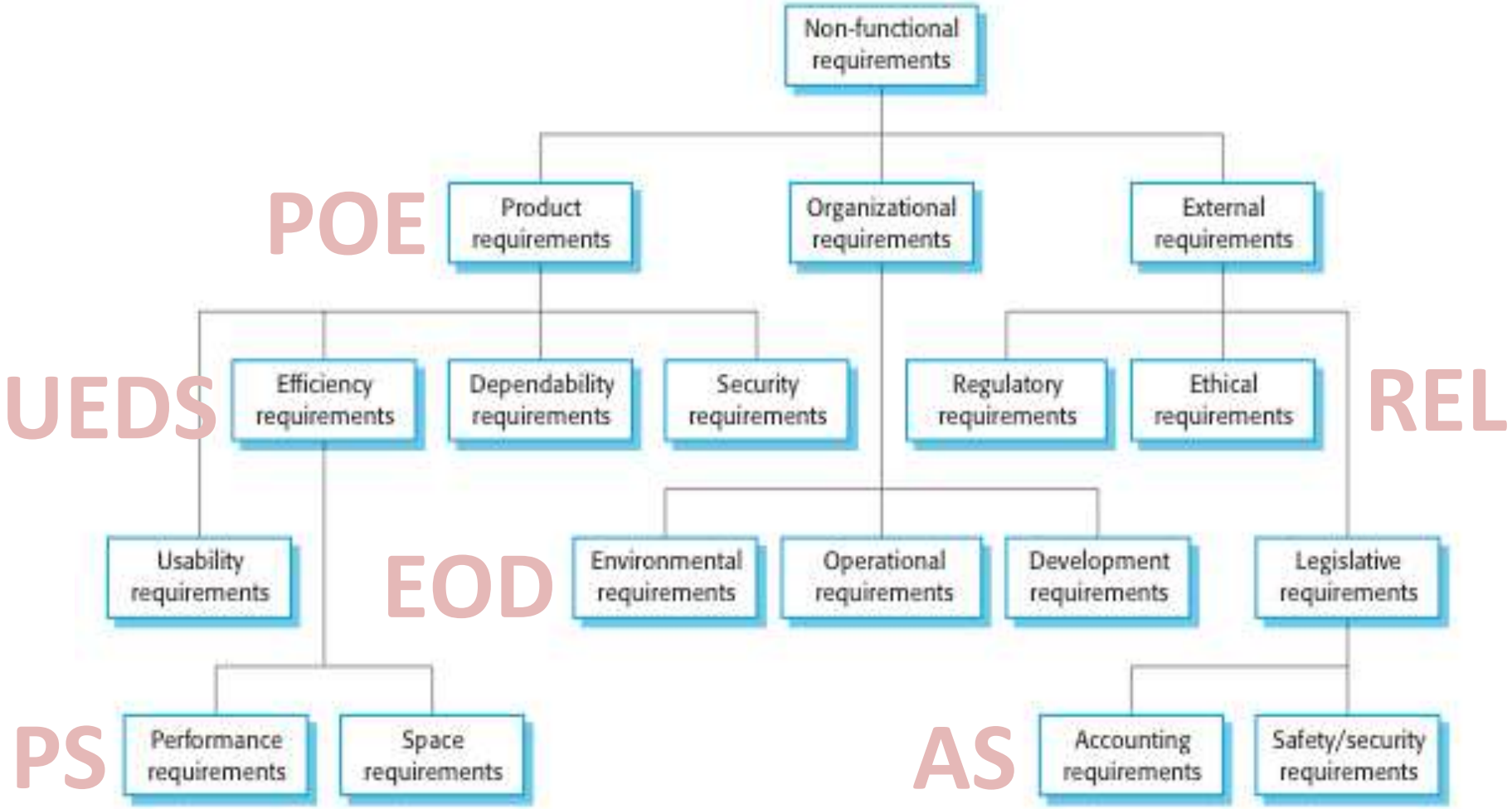
Non-Functional Requirements

- 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 Integrated Development Environment (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

Non-Functional Classifications

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

Types of Non-Functional Requirements



Non-Functional Requirements Implementation

- Non-functional requirements **may affect the overall architecture** of a system rather than the individual components
  - E.g. to ensure that performance requirements are met, developers may have to organize the system to minimize communications between components
- A single non-functional requirement e.g. 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

Examples of Non-Functional 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.

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 measures that can be objectively tested
- Goals are helpful to developers as they convey the intentions of the system users

Example: 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 Measuring Non-Functional Requirements

Metrics	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

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

Example: Train Protection System

- Example of a domain requirement for a train protection system: The deceleration of the train shall be computed as:

  Dtrain = Dcontrol + Dgradient

  where Dgradient is 9.81ms2 * compensated gradient/alpha and where the values of 9.81ms2 /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

Domain Requirements Problems

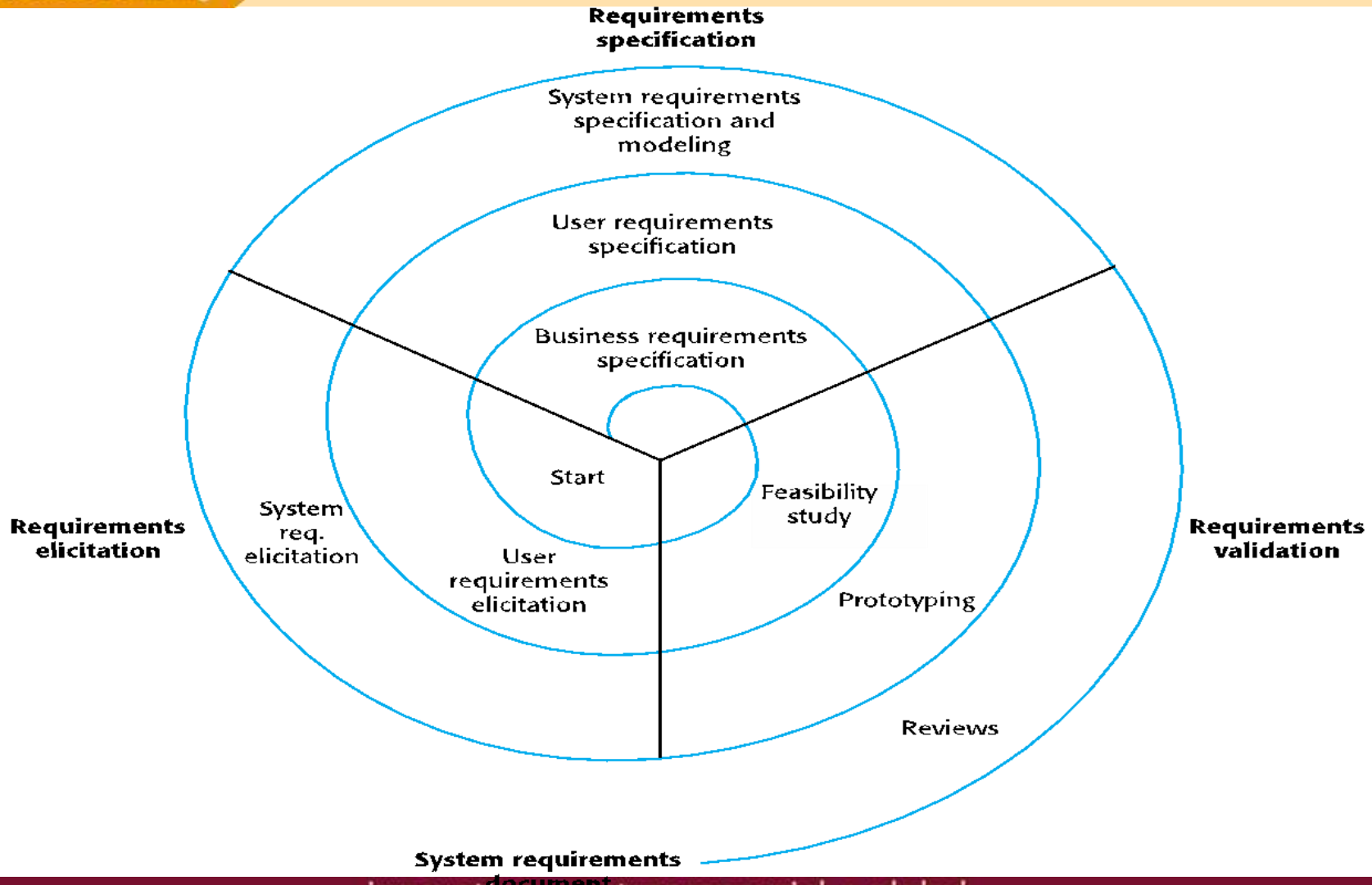
- Understandability:
  - Requirements are expressed in the language of the application domain
  - This is often not understood by software engineers developing the system
- Implicitness:
  - Domain specialists understand the area so well that they do not think of making the domain requirements explicit

REQUIREMENTS ENGINEERING PROCESSES

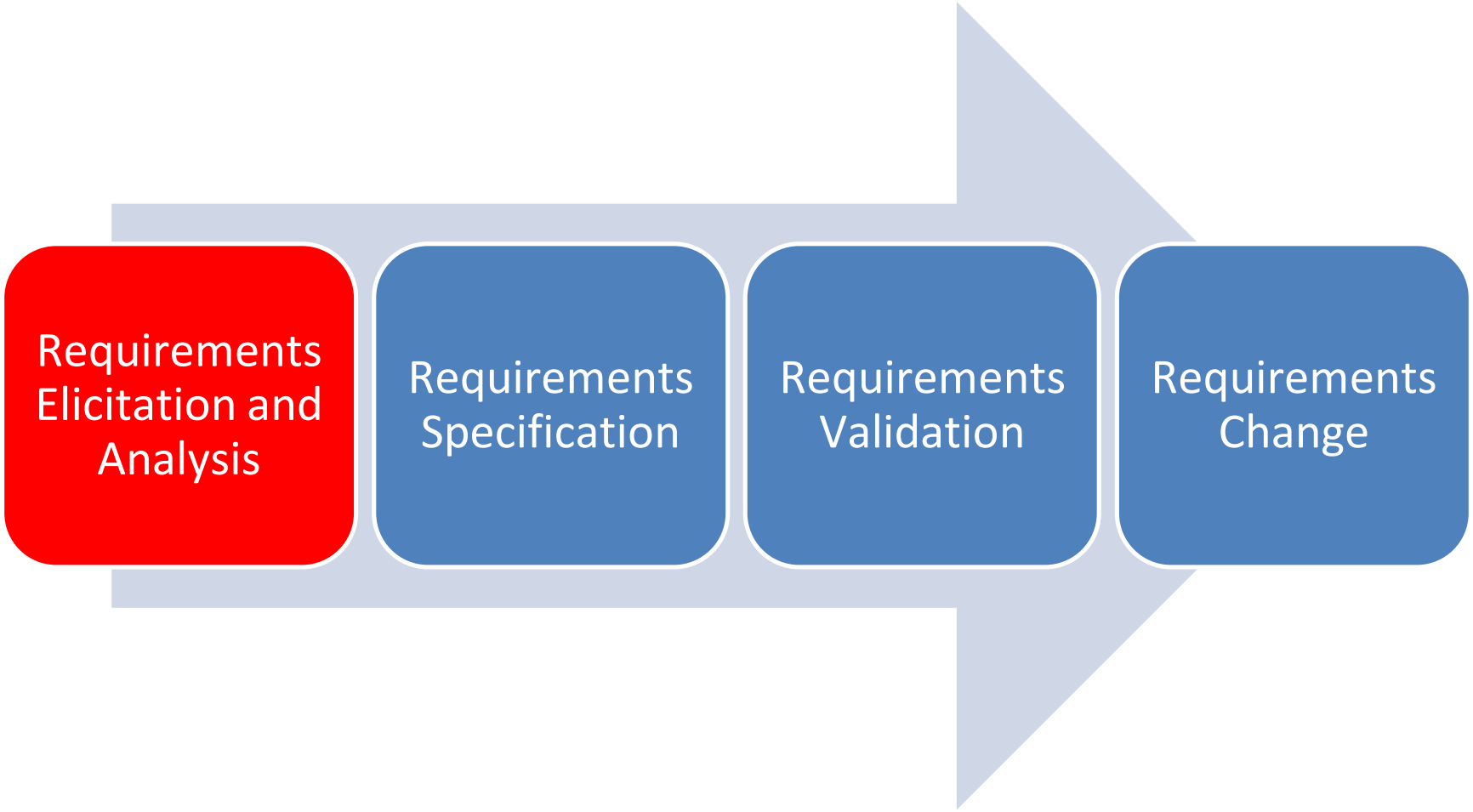
Requirements Engineering Processes

- 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 and analysis
  - Requirements specification
  - Requirements validation
  - Requirements change
- In practice, RE is an iterative activity in which these processes are interleaved

A Spiral View: Requirements Engineering Processes



Requirements Engineering Processes

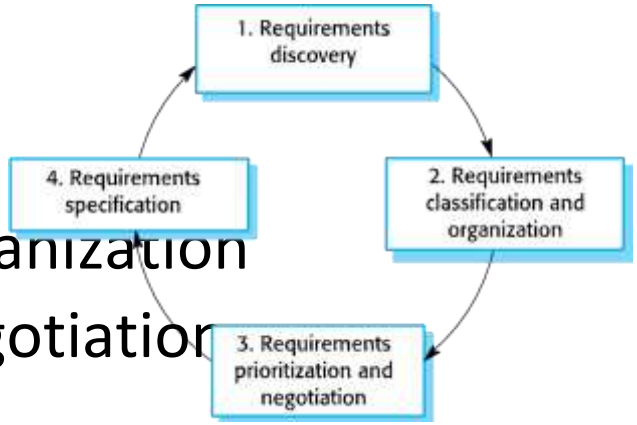


Requirements Elicitation and Analysis

- 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



Problems of Requirements Elicitation

- Stakeholders do not 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 change
- Therefore, there are range of techniques

Requirements Elicitation and Analysis: 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

Requirements Discovery

- The process of gathering information about the required and existing systems and distilling the **user and system requirements** from this information
- Sources of information during requirements discovery phase:
  - Documentation
  - System stakeholders
  - Specification from similar systems
- Systems normally have a range of **stakeholders.**

System stakeholders

- Any person or organization who is affected by the system in some way and so who has a legitimate interest.

- **Stakeholders types**
  - End users
  - System managers
  - System owners
  - External stakeholders

Example: Stakeholders in the Mentcare System

- 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.

Example: Stakeholders in the Mentcare System

- 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

User Characteristics and Personas

- RE-related tasks involve end-users in the process
- The more users involvements, the greater the likelihood that the requirements specifications will satisfy the users
- User Characteristics and Personas are versatile tools and can be used in all RE-related tasks

User Characteristics

- Attributes of the *types* of users who will interact with the system.
  - skill level (novice vs expert)
  - frequency of use (occasional vs heavy user)
  - environment (mobile, desktop, in-office, field)
  - accessibility requirements (vision/hearing/physical limitations)
  - educational background, language proficiency, etc.
- more high-level, abstract compare to Persona

Personas

- A more detailed *fictional* representation of a user type — a profile that includes name, background, motivations, goals, behaviours, pain-points, etc.
- Example :
  - "Maria, 45-year school teacher, uses the system twice a day, low comfort with new software, needs fast onboarding.."

Techniques to Elicit Requirements

- **Interviewing**: for formal or informal interview with stakeholders
- **Ethnography**: to observe operational process in the organization
- **Stories and Scenarios**: representing real life examples

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

Interviews in Practice

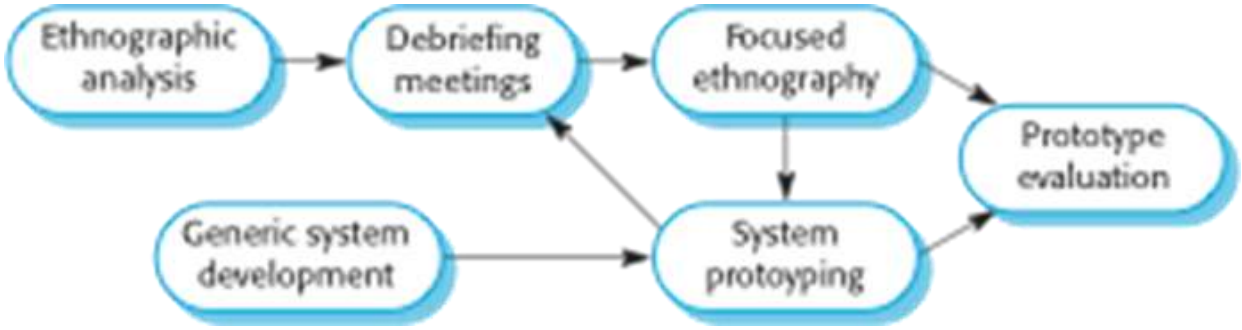
- Normally a mix of closed and open-ended interviewing.
- Interviews are good for getting an overall understanding of what stakeholders do and how they might interact with the system
- Interviewers need to be open-minded without pre-conceived ideas of what the system should do.

Problems with Interviews

- Application specialists may use language to describe their work that isn't easy for the requirements engineer to understand.
- 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 is not worth articulating

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



Scope of Ethnography

- Requirements that are derived from the way that people actually work rather than the way 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

Stories and Scenarios

- Scenarios and user stories are real-life examples of how a system can be used.
- Scenarios is a structured form of user story.
- Scenarios 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

Scenario for Collecting Medical History in Mentcare System

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).

Scenario for Collecting Medical History in Mentcare System

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.

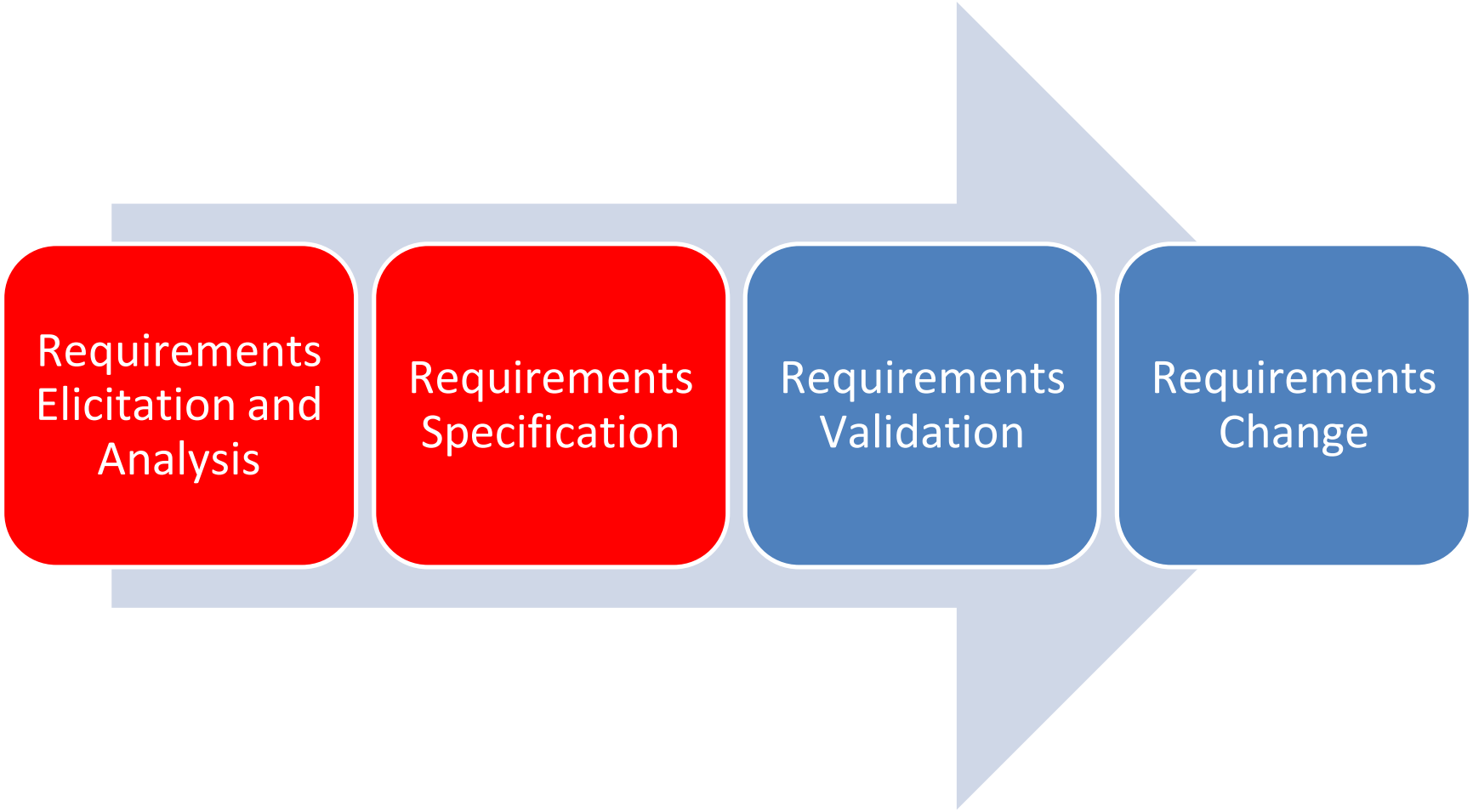
Patient conditions or medication are not entered in the menu. The nurse should choose the 'other' option and enter free text describing the condition/medication.

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.

Requirements Engineering Processes



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, thus it is important that these are as complete as possible

Software Requirements Document

- The software requirements document is the official statement of what is required of the system developers - sometimes called the Software Requirement Specification (SRS).
- 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

Agile Methods and Requirements

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

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 language	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
  - This may be the consequence of a regulatory 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
- Use text highlighting to identify key parts of the requirement
- Avoid the use of computer jargon
- Include an explanation (rationale) of why a requirement is necessary

Problems With Natural Language

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

Example of 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

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	r0 is replaced by r1 then r1 is replaced by r2.
Side effects	None.

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

Template for Writing Functional Requirements

Template: Requirements Statement Example

- The <**user**> shall be able to <**capability**>
- The **librarian** shall be able to **register library user**

- There shall be a <**system function**>
- There shall be a **library registration information facility**

- The <**system function**> shall able to <**action**> <**entity**>
- The **library registration information facility** shall be able to **store information**

Template for Writing Non-Functional Requirements

Quality Requirements: **Template Requirements Statement Example**

- The <**user**> shall be able to <**capability**> at a minimum rate of <**quantity**> times per <**time unit**>
- The **borrower** shall be able to **renew books** at a minimum rate of **3** times per **borrowing period**
- The <**user**> shall be able to <**capability**> within <**quantity**> <**time unit**>s from <**event**>
- The **borrower** shall be able to **renew books** within **10 seconds** from **renewal process**
- The <**system function**> shall be able to <**action**> <**entity**> at least <**quantity**> times per <**time unit**>
- The **library registration facility** shall be able to **add new user information** at least **15** times per **minute**
- The <**system function**> shall not allow <**user**> to <**action**>
- The **library system** shall not allow **borrower** to **edit any sensitive data in the system**

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

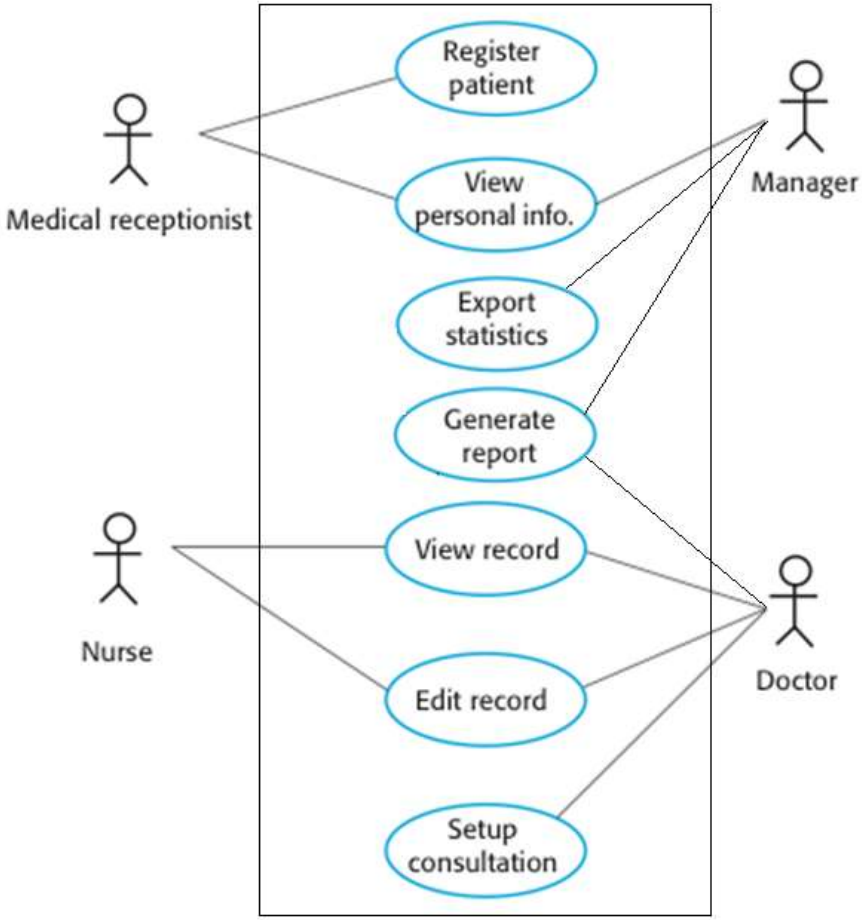
Tabular Specification of Computation for an Insulin Pump

Condition	Action
Sugar level falling (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) ≥ (r1 – r0))	CompDose = round ((r2 – r1)/4) If rounded result = 0 then CompDose = MinimumDose

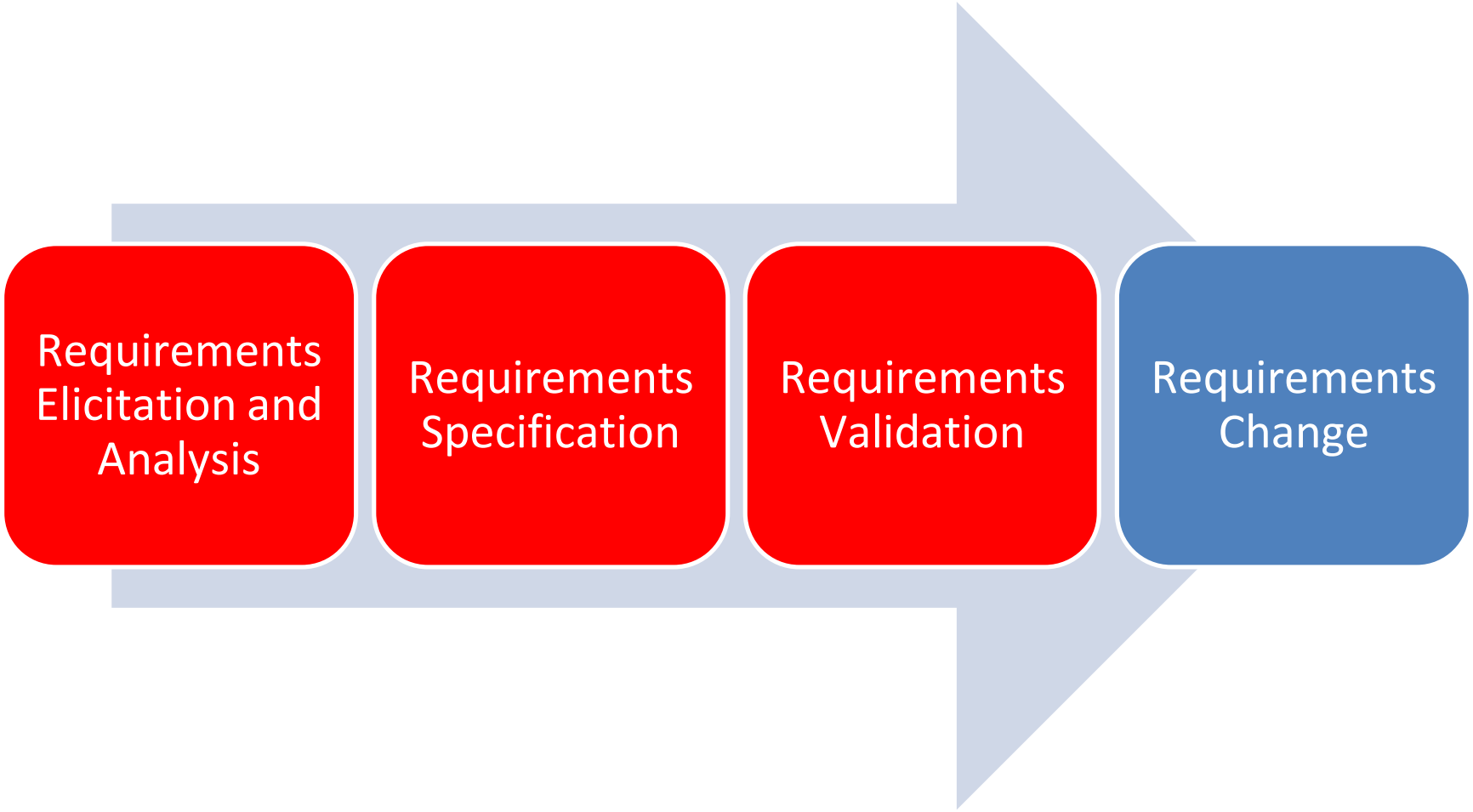
Use Cases

- Use-cases are a **scenario based technique** in the UML which 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
- **Sequence diagrams** may be used to **add detail to use-cases** by showing the sequence of event processing in the system

Use Case Diagram for the Mentcare System



Requirements Engineering Processes



Requirements Validation

- Concerned with demonstrating that the requirements define the system as what 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

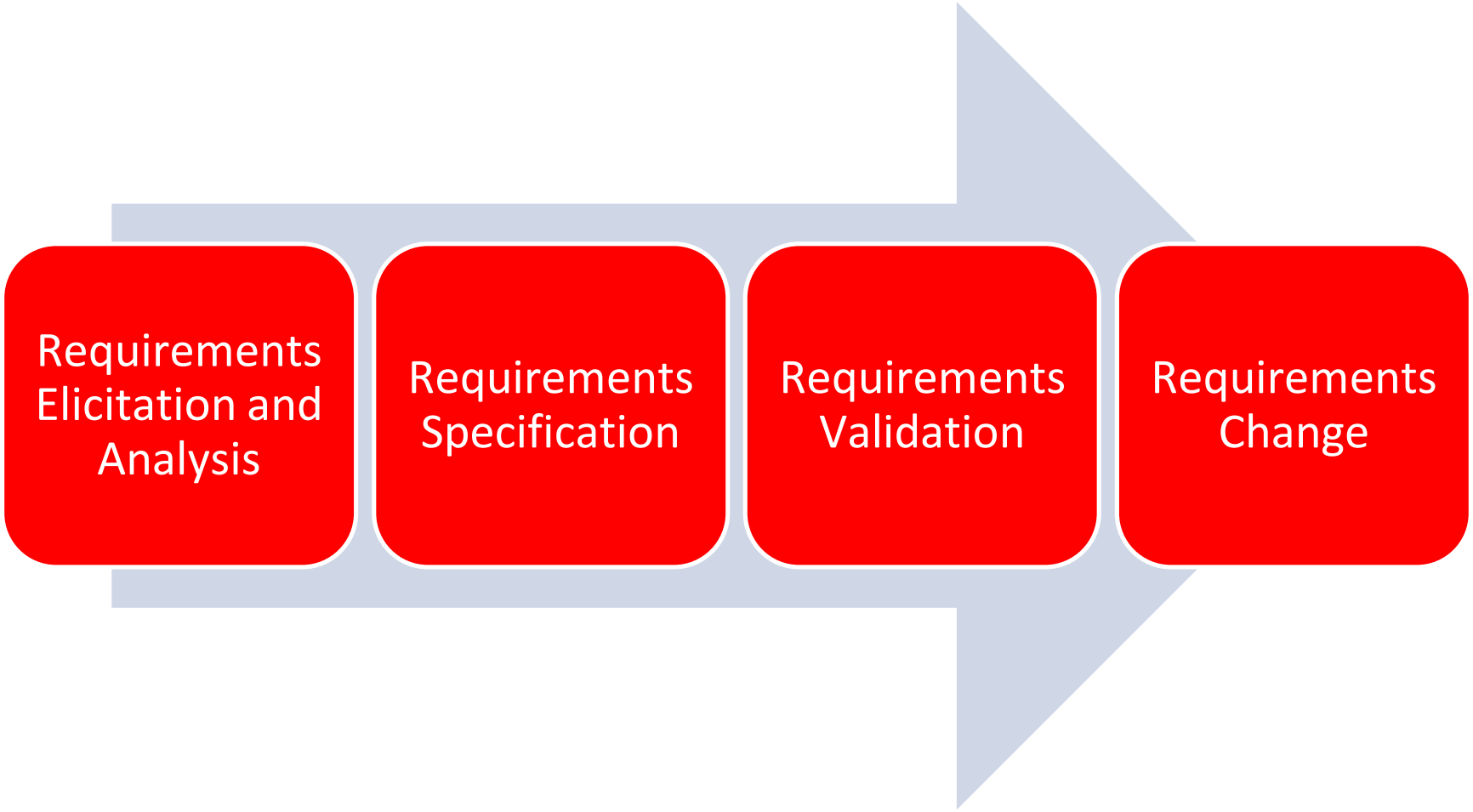
Requirements Reviews

- 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

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?

Requirements Engineering Processes

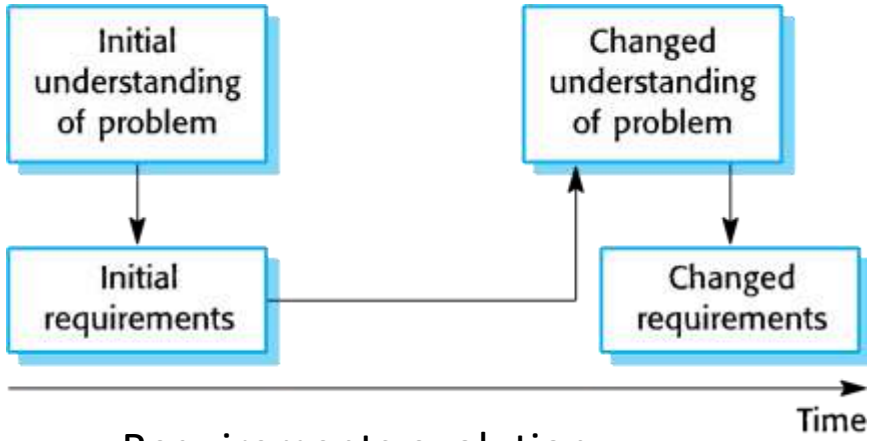


Changing Requirements

- 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 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
- Need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the **impact of requirements changes**
- Need to establish a **formal process for making change proposals** and linking these to system 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 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.

Key Points

- 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, 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.