COMP1023 Software Engineering

Spring Semester 2019-2020

Dr. Radu Muschevici

School of Computer Science, University of Nottingham, Malaysia

Lecture 2

2020-02-12

Topics covered

A. Software Processes
   1. Process models (plan-driven & incremental)
   2. Process activities

B. Requirements Engineering (RE)
   - User & system requirements
   - Functional & non-functional requirements
   - The RE process:
     1. elicitation
     2. specification
     3. validation
     4. change

A. Software Processes

The software process

- ✧ A **structured set of activities** required to develop a software system.
- ✧ Many different software processes but **all involve**:
  - **Specification** – defining what the system should do;
  - **Design and implementation** – defining the organization of the system and implementing the system;
  - **Validation** – checking that it does what the customer wants;
  - **Evolution** – changing the system in response to changing customer needs.
- ✧ Software process **model** = abstract representation of a process.

Plan-driven and Agile processes

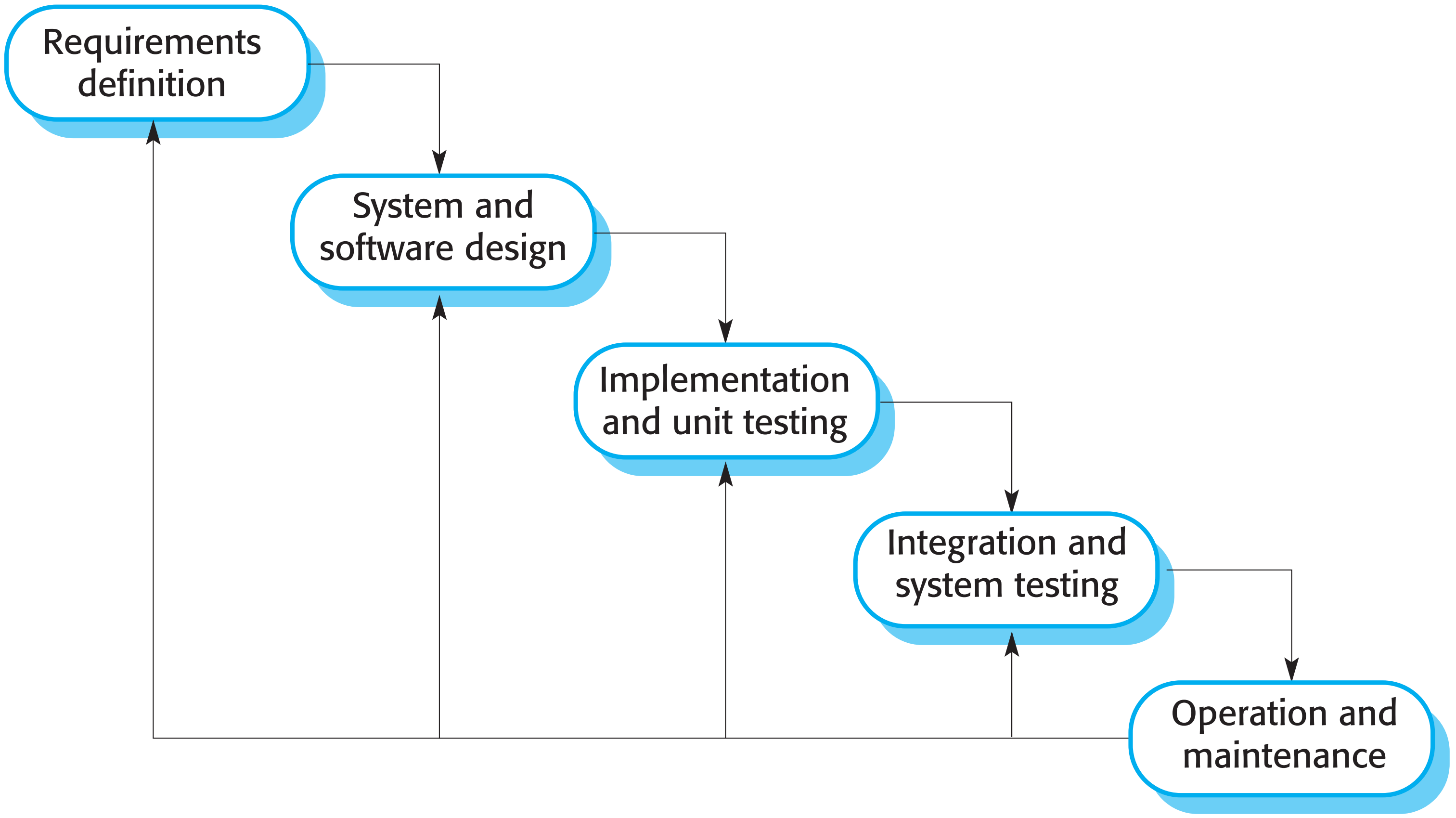
- ✧ **Plan-driven** processes are processes where all of the process activities are planned in advance and progress is measured against this plan.
- ✧ In **Agile** processes, planning is incremental and it is easier to change the process to reflect changing customer requirements.
- ✧ In practice, most practical processes include elements of **both** plan-driven and agile approaches.
- ✧ There are **no right or wrong** software processes.

1. Software process models

Software process models

- ✧ The **Waterfall** model
  - ▪ Plan-driven model. Separate and distinct phases of specification and development.
- ✧ **Incremental development**
  - ▪ Specification, development and validation are interleaved.
- ✧ In practice, most large systems are developed using a process that incorporates elements from all of these models.

The Waterfall model



Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

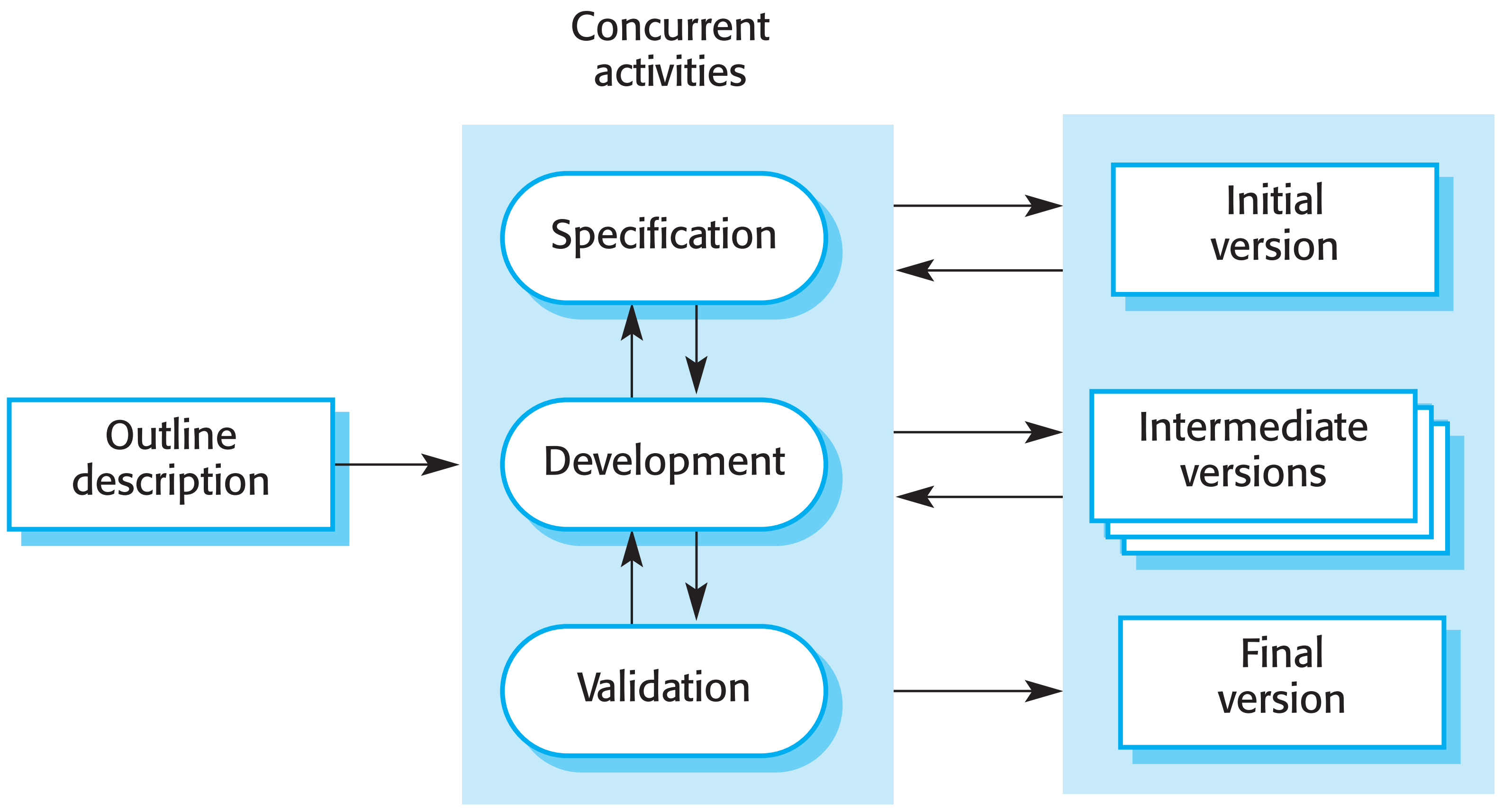
Waterfall model phases

- There are separate identified phases in the waterfall model:
  - **Requirements** analysis and definition
  - **System** and software design
  - **Implementation** and unit testing
  - **Integration** and system testing
  - **Operation** and maintenance
- Main **drawback** of the waterfall model: difficulty of accommodating change after the process is underway. *In principle,* a phase has to be complete before moving onto the next phase.

Waterfall model problems

- ✧ Inflexible partitioning of the project into distinct stages makes it **difficult** to respond to **changing customer requirements**.
  - Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
  - Few business systems have stable requirements.
- ✧ The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.
  - In those circumstances, the plan-driven nature of the waterfall model helps **coordinate** the work.

Incremental development



Incremental development benefits

- ✧ The **cost** of accommodating changing customer requirements is **reduced**.
  - ▪ The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
- ✧ It is easier to get **customer feedback** on the development work that has been done.
  - ▪ Customers can comment on demonstrations of the software and see how much has been implemented.
- ✧ More **rapid delivery and deployment** of useful software to the customer is possible.
  - ▪ Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

Incremental development problems

- ✧ The process is **not visible**.
  - ▪ Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.
- ✧ System structure tends to **degrade** as new increments are added.
  - ▪ Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.

2. Software process activities

Process activities

- ✧ Real software processes are **sequences of** technical, collaborative and managerial **activities.**
- ✧ **Goal:** specifying, designing, implementing and testing a software system.
- ✧ Four basic process activities:
  1. Specification
  2. Design & Implementation
  3. Validation
  4. Evolution

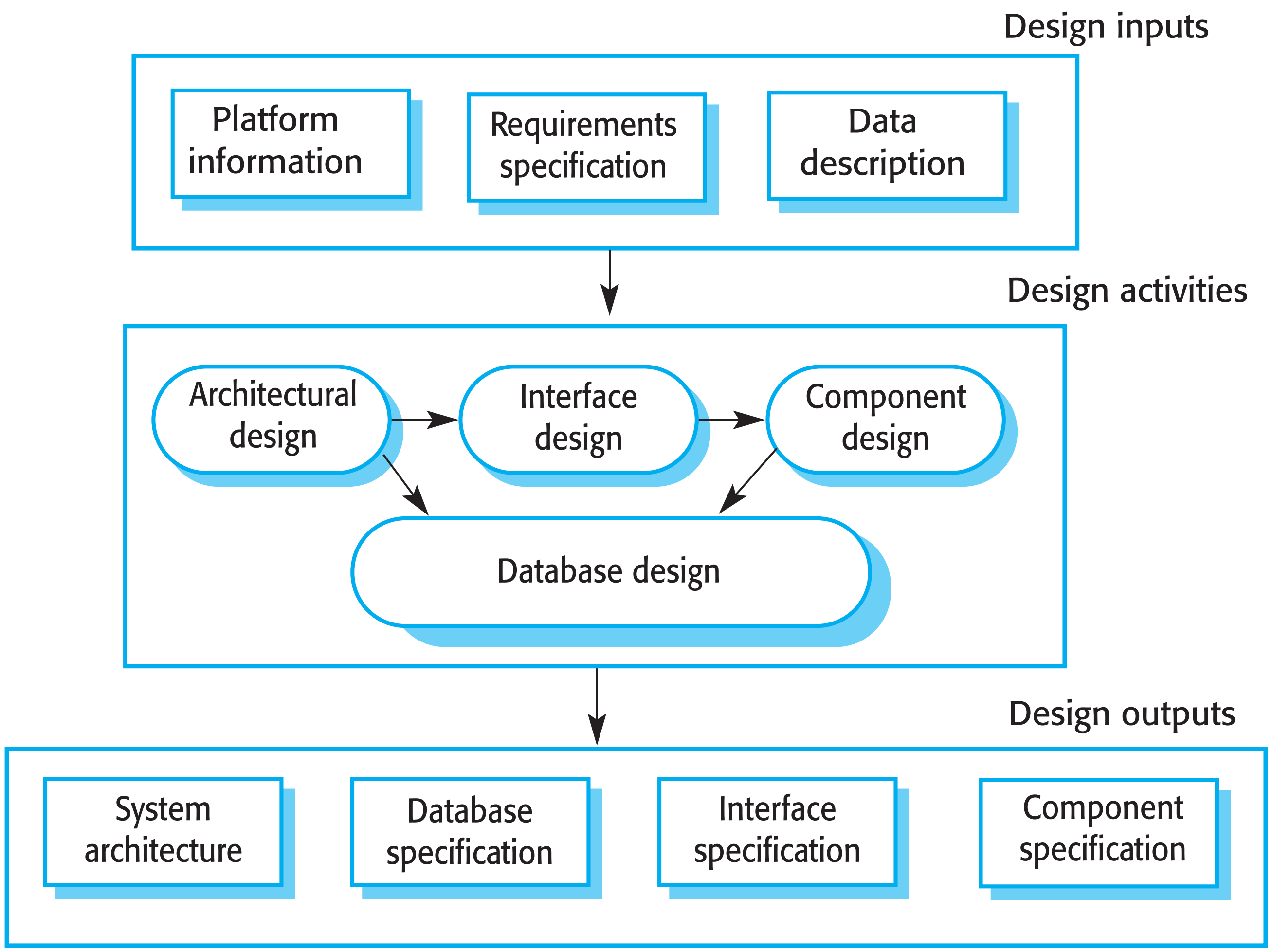
Specification

- ✧ The process of establishing what services are required and the constraints on the system’s operation and development.
- ✧ Requirements engineering process ➡ Specification
  - ▪ Requirements elicitation and analysis
    - What do the system stakeholders require or expect from the system?
  - ▪ Requirements specification
    - Defining the requirements in detail
  - ▪ Requirements validation
    - Checking the validity of the requirements

Design and implementation

- ✧ The process of converting the system specification into an executable system.
- ✧ Software design
  - Design a software structure that realises the specification;
- ✧ Implementation
  - Translate this structure into an executable program;
- ✧ The activities of design and implementation are closely related and may be inter-leaved.

A general model of the design process



Design activities

- ✧ *Architectural design,* where you identify the overall structure of the system, the principal components (subsystems or modules), their relationships and how they are distributed.
- ✧ *Database design,* where you design the system data structures and how these are to be represented in a database.
- ✧ *Interface design,* where you define the interfaces between system components.
- ✧ *Component selection and design,* where you search for reusable components. If unavailable, you design how it will operate.

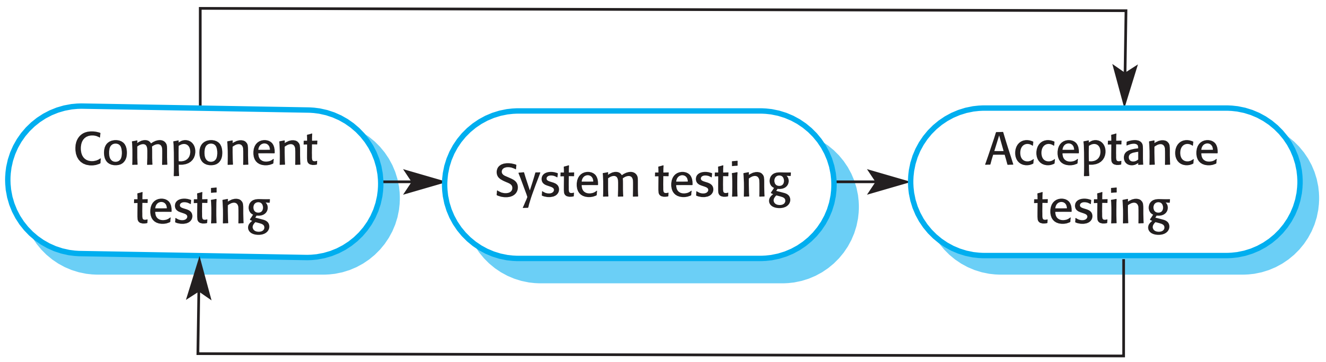
Implementation

- ✧ The software is implemented by developing a **program**.
- ✧ Design and implementation are interleaved activities for most types of software system.
- ✧ **Programming** is an individual activity with no standard process.
- ✧ **Debugging** is the activity of finding program faults and correcting these faults.

Validation

- ✧ Verification and validation (V&V) is intended to show that a software system conforms to its specification and meets the requirements of the system customer.
- ✧ Involves **review** processes and system testing.
- ✧ System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.
- ✧ **Testing** is the most commonly used V&V activity.

Stages of testing



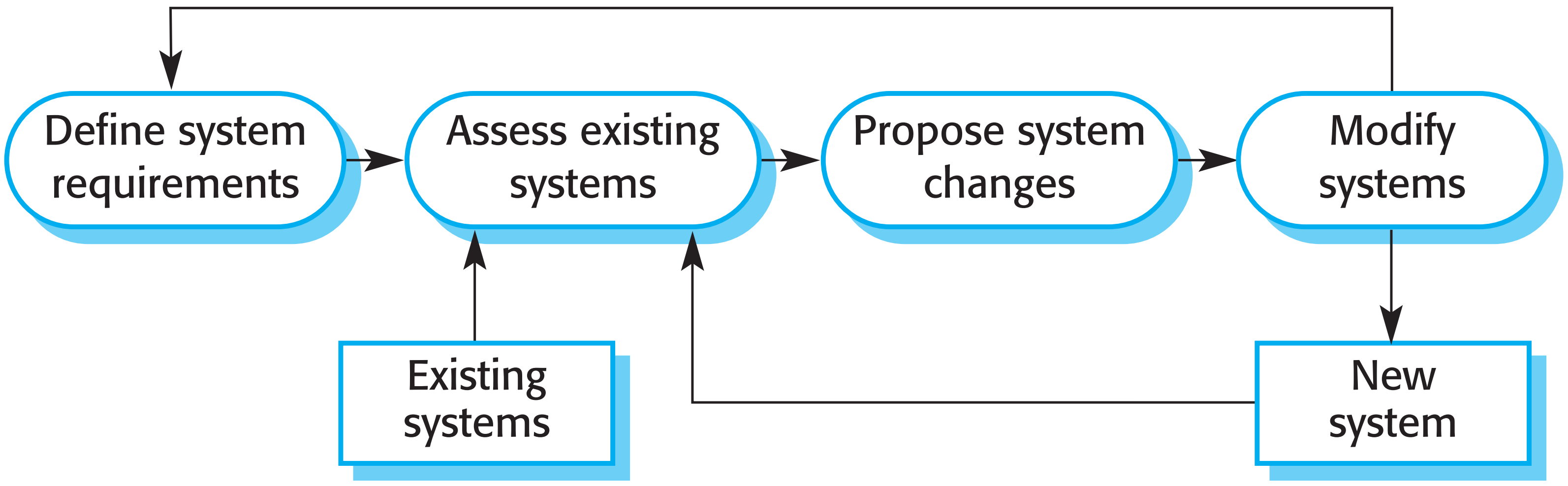
Testing stages

- ✧ Component testing
  - Individual components are tested independently;
  - Components may be functions or objects or coherent groupings of these entities.
- ✧ System testing
  - Testing of the system as a whole. Testing of emergent properties is particularly important.
- ✧ Acceptance (customer) testing
  - Testing with customer data to check that the system meets the customer's needs.

Evolution

- ✧ Software is inherently **flexible** and can **change**.
- ✧ As requirements change through changing business circumstances, the software that supports the business must also evolve and change.
- ✧ Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new.

System evolution



Software processes: key points

- Software processes are the **activities** involved in producing a software system. Software process models are abstract representations of these processes.
- General process models describe the organization of software processes.
  - Examples of these general models include the ‘waterfall’ model and incremental development.

Software processes: key points

- ✧ **Design and implementation** processes are concerned with transforming a requirements specification into an executable software system.
- ✧ Software **validation** is the process of checking that the system conforms to its specification and that it meets the real needs of the users of the system.
- ✧ Software **evolution** takes place when you change existing software systems to meet new requirements. The software must evolve to remain useful.

Short break

B. Requirements Engineering (RE)

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.

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

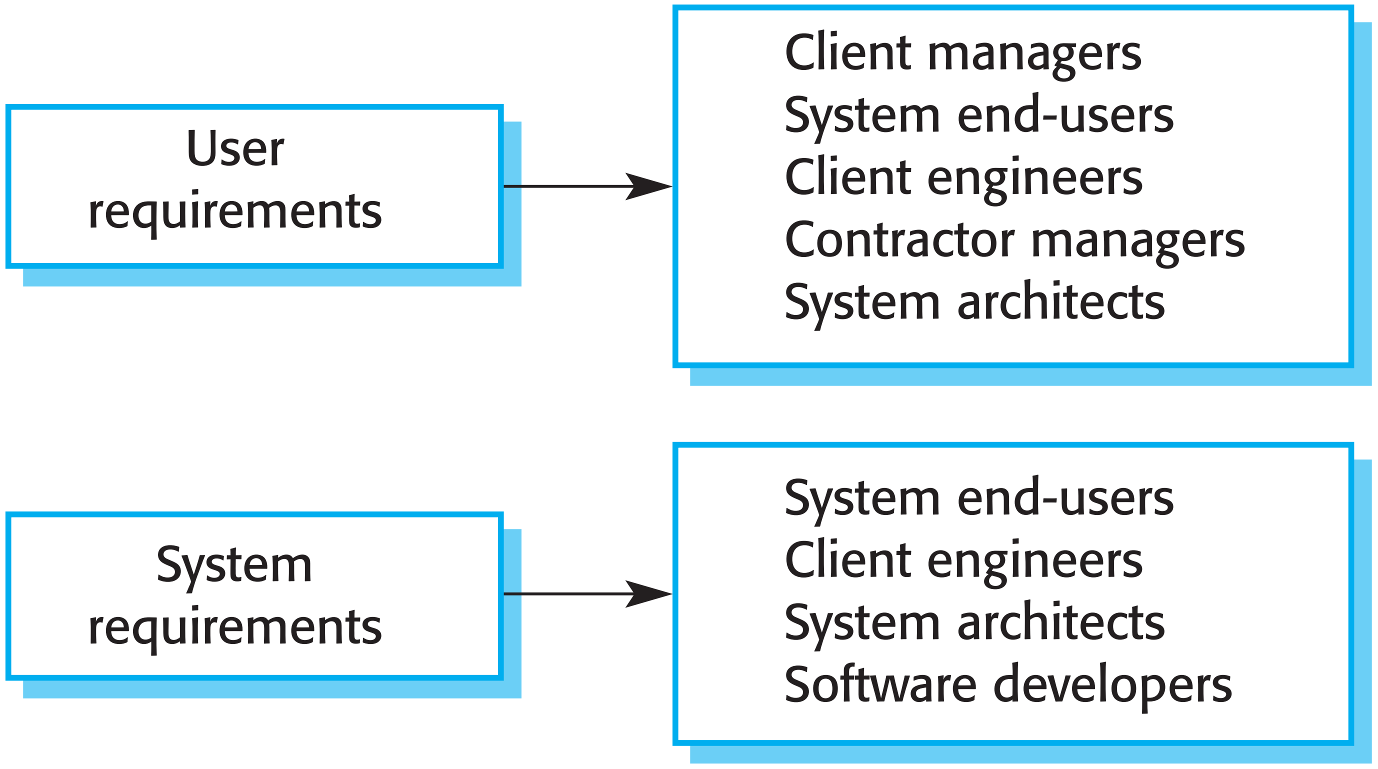
User requirements definition

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



System 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

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.

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.

Functional requirements examples (Mentcare system)

1. A user shall be able to search the appointments lists for all clinics.
2. The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day.
3. 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.

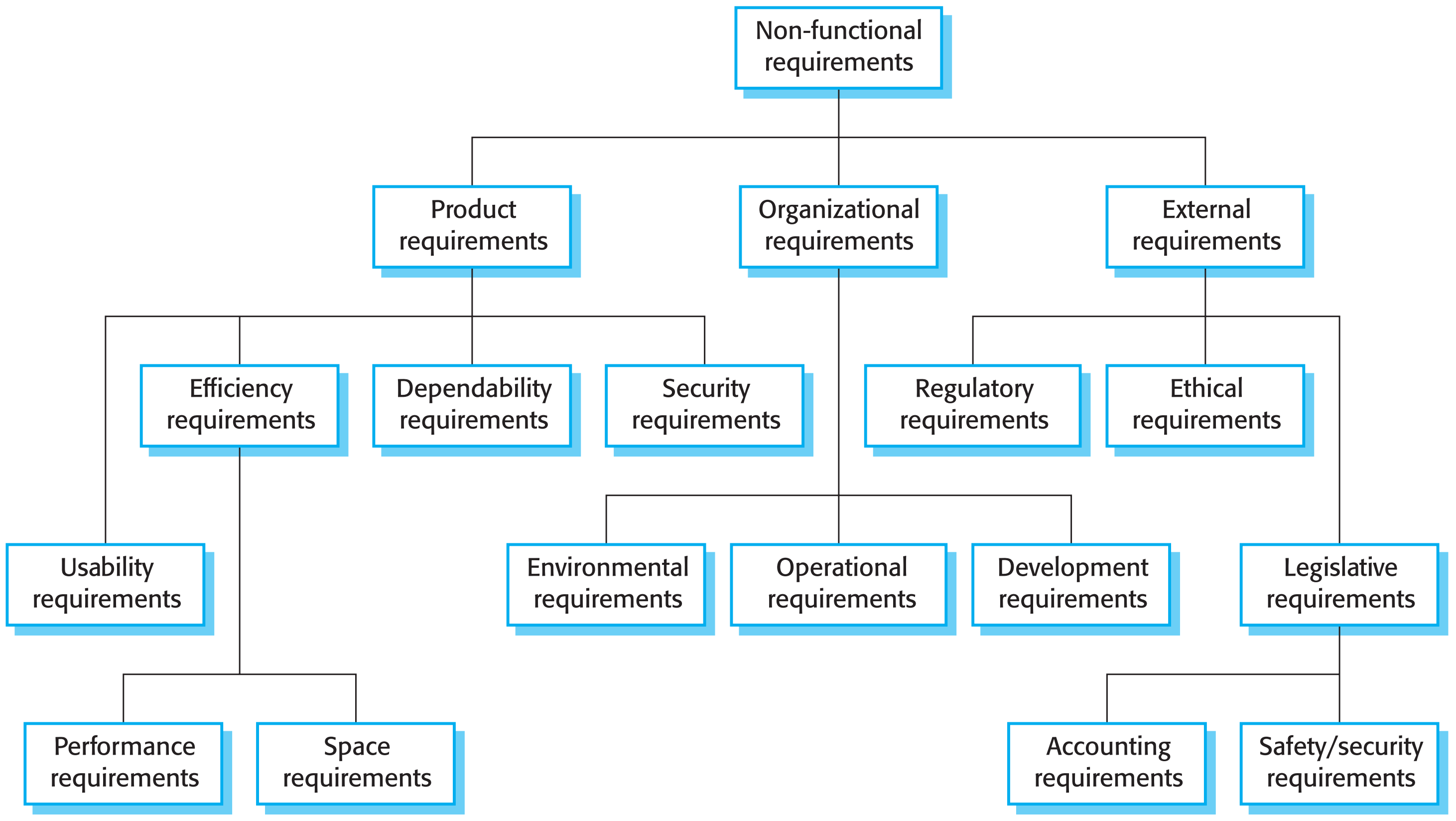
Requirements completeness and consistency

- ✧ 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, because of system and environmental complexity, 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 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



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 organize the system to minimize 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.

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.

Examples of nonfunctional requirements (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 measure 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 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

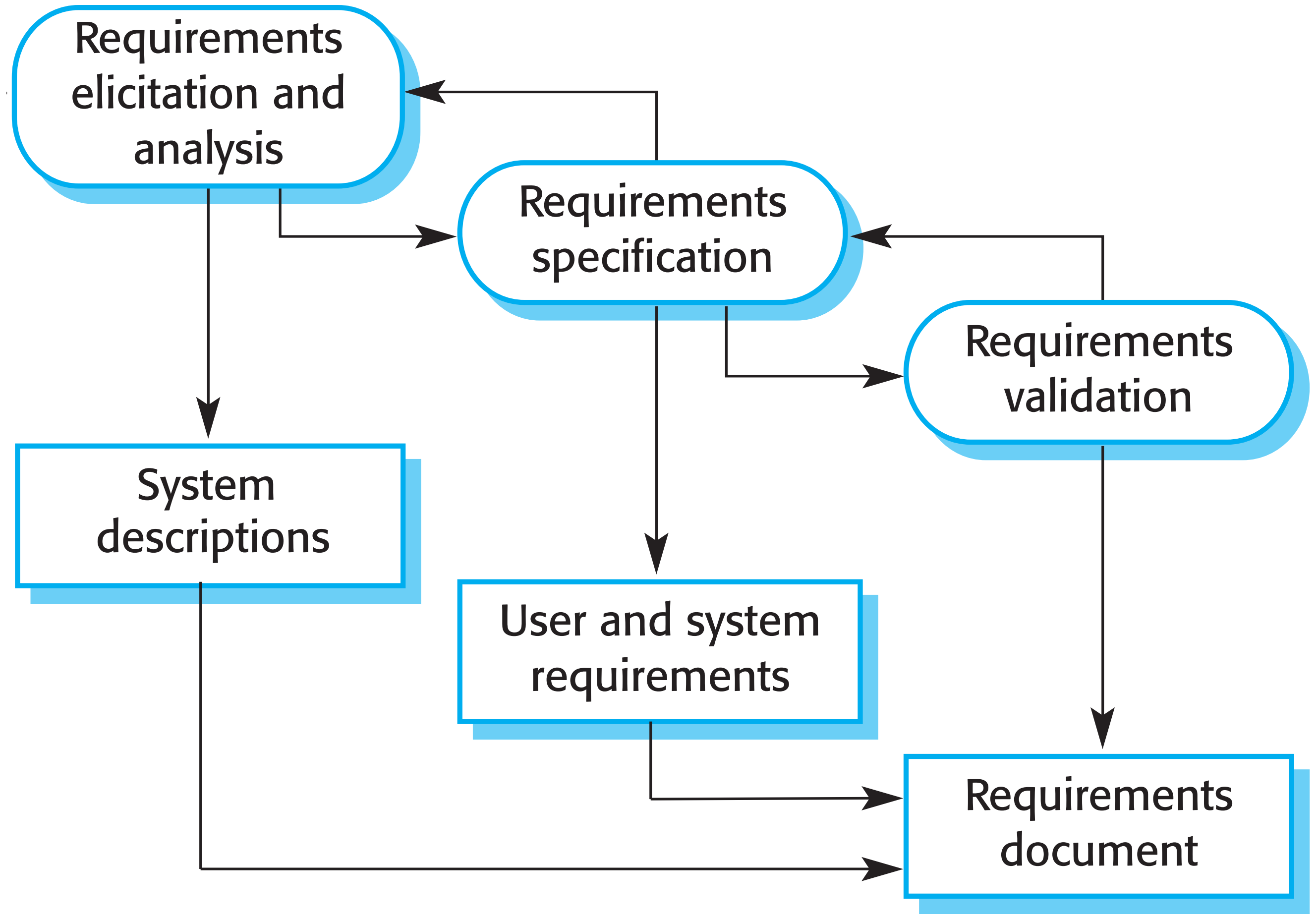
✧ The processes used for RE vary widely depending on the application domain, the people involved and the organisation developing the requirements.

✧ However, there are a number of generic activities common to all processes:

1. Requirements elicitation
2. Requirements analysis
3. Requirements validation
4. Requirements change management

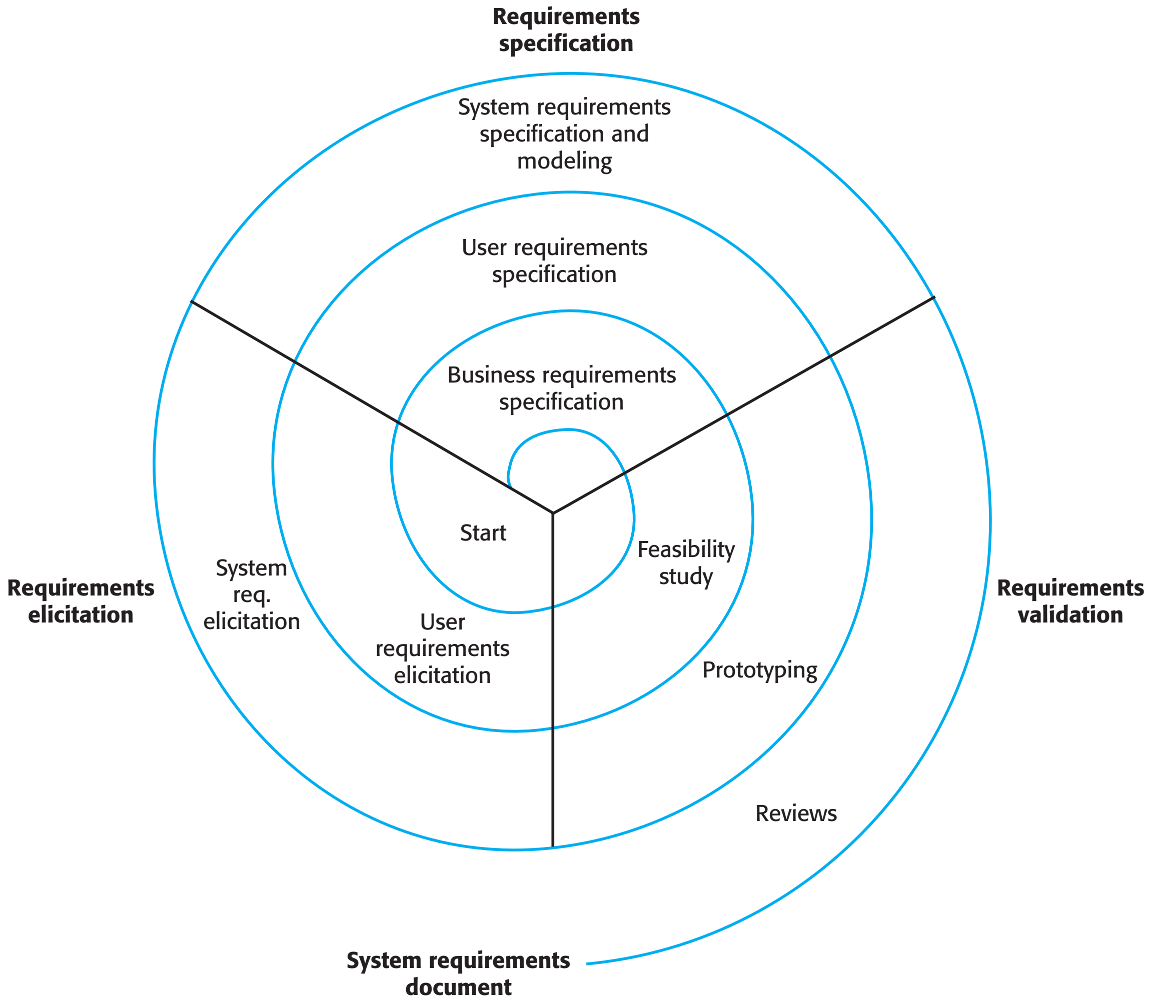
✧ In practice, RE is an iterative activity in which these processes are interleaved.

The requirements engineering process



Requirements elicitation and analysis
Requirements specification
Requirements validation
System descriptions
User and system requirements
Requirements document

A spiral view of the requirements engineering process


Requirements specification
System requirements specification and modeling
User requirements specification
Business requirements specification
Start
Feasibility study
Requirements elicitation
System req. elicitation
User requirements elicitation
Prototyping
Requirements validation
Reviews
System requirements document

1. Requirements elicitation

Requirements elicitation and analysis

- ✧ Sometimes called 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.*

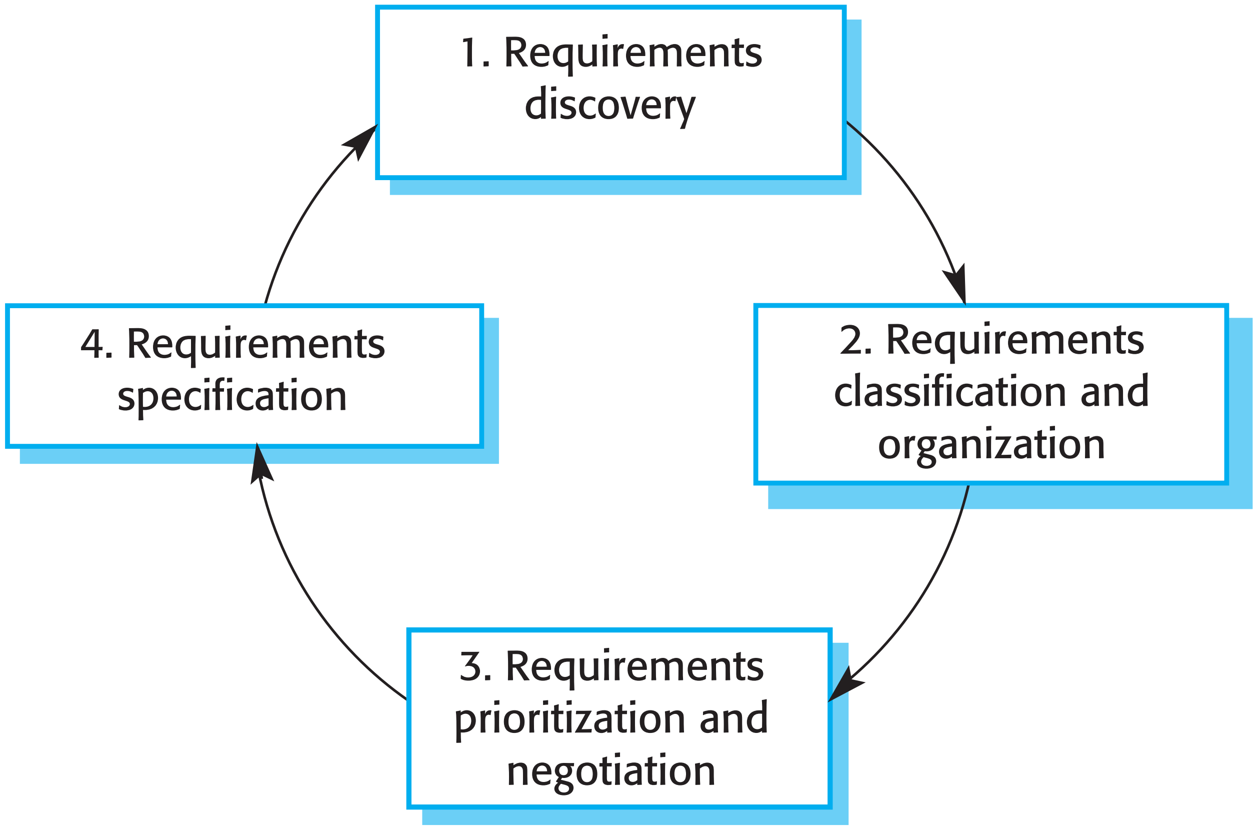
Requirements elicitation

- ✧ 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 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 and analysis process


1. Requirements discovery
2. Requirements classification and organization
3. Requirements prioritization and negotiation
4. Requirements specification

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.
- ✧ Interaction is with system stakeholders from managers to external regulators.
- ✧ Systems normally have a range of stakeholders.

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.

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 isn't worth articulating.

Stories and scenarios

- ✧ Scenarios and user stories are real-life examples of how a system can be used.
- ✧ Stories and scenarios are a description of how a system may be used for a particular task.
- ✧ Because they are based on a practical situation, stakeholders can relate to them and can comment on their situation with respect to the story.

Example: Photo sharing in the classroom (iLearn)

- Jack is a primary school teacher in Ullapool (a village in northern Scotland). He has decided that a class project should be focused around the fishing industry in the area, looking at the history, development and economic impact of fishing. As part of this, pupils are asked to gather and share reminiscences from relatives, use newspaper archives and collect old photographs related to fishing and fishing communities in the area. Pupils use an iLearn wiki to gather together fishing stories and SCRAN (a history resources site) to access newspaper archives and photographs. However, Jack also needs a photo sharing site as he wants pupils to take and comment on each others' photos and to upload scans of old photographs that they may have in their families.

  Jack sends an email to a primary school teachers group, which he is a member of to see if anyone can recommend an appropriate system. Two teachers reply and both suggest that he uses KidsTakePics, a photo sharing site that allows teachers to check and moderate content. As KidsTakePics is not integrated with the iLearn authentication service, he sets up a teacher and a class account. He uses the iLearn setup service to add KidsTakePics to the services seen by the pupils in his class so that when they log in, they can immediately use the system to upload photos from their mobile devices and class computers.

Scenarios

- ✧ 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.

2. Requirements specification

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.

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

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

- ✧ Choose 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
  - Several different requirements may be expressed together.

Example requirements for the insulin pump software system

3.2 The system shall measure the blood sugar and deliver insulin, if required, every 10 minutes. *(Changes in blood sugar are relatively slow so more frequent measurement is unnecessary; less frequent measurement could lead to unnecessarily high sugar levels.)*

3.6 The system shall run a self-test routine every minute with the conditions to be tested and the associated actions defined in Table 1. *(A self-test routine can discover hardware and software problems and alert the user to the fact the normal operation may be impossible.)*

Structured specifications

- ✧ An approach to writing requirements where the freedom of the requirements writer is limited by a given structure (template).
- ✧ 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.

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.

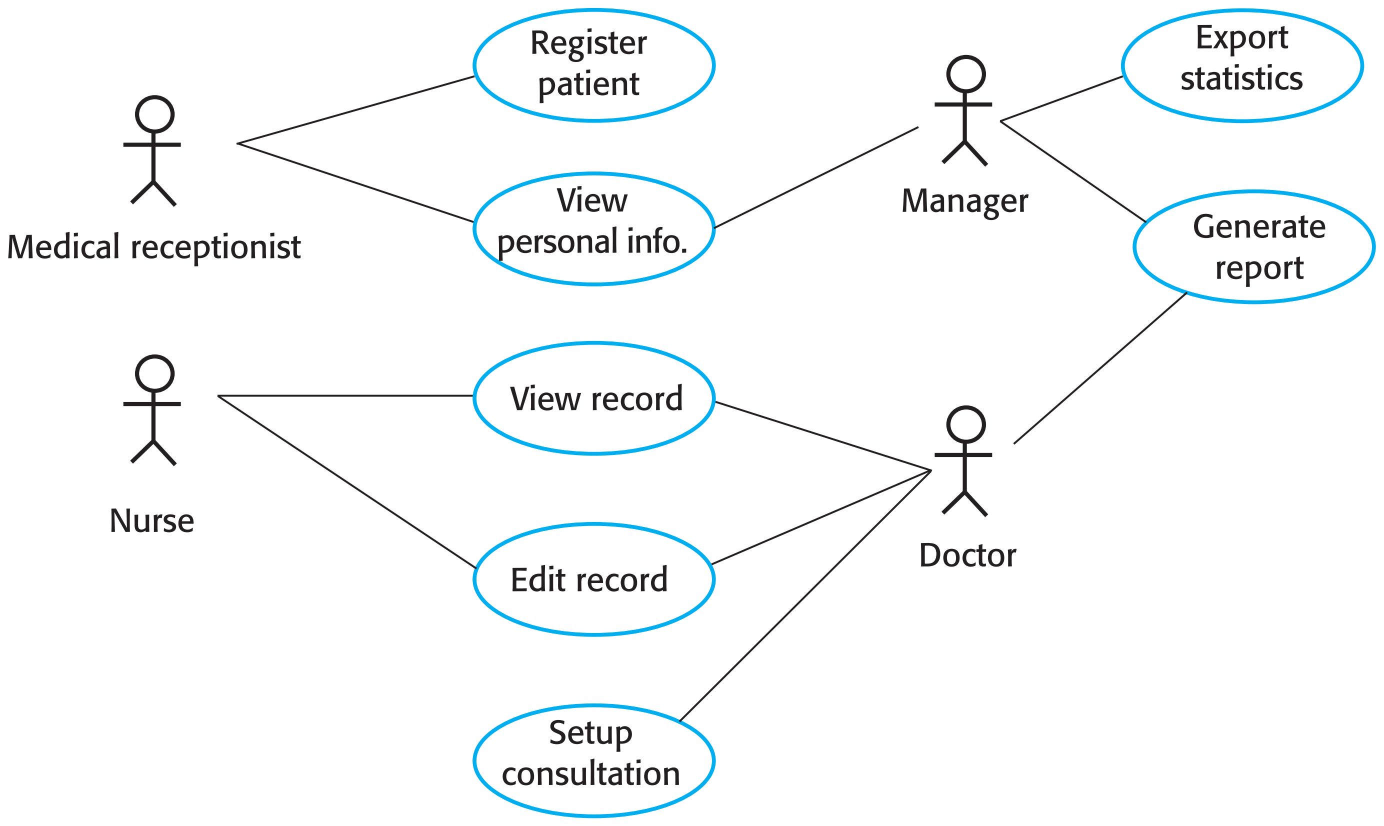
Tabular specification of computation for an insulin pump

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 $((r2 – r1) \geq (r1 – r0))$	CompDose = round $((r2 – r1)/4)$ If rounded result = 0 then CompDose = MinimumDose

Use cases

- ✧ Use cases are a kind of **scenario** that are included in the Unified Modelling Language (UML).
- ✧ High-level graphical model (diagrams).
- ✧ 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.
- ✧ UML sequence diagrams may be used to add detail to use cases by showing the sequence of event processing in the system.

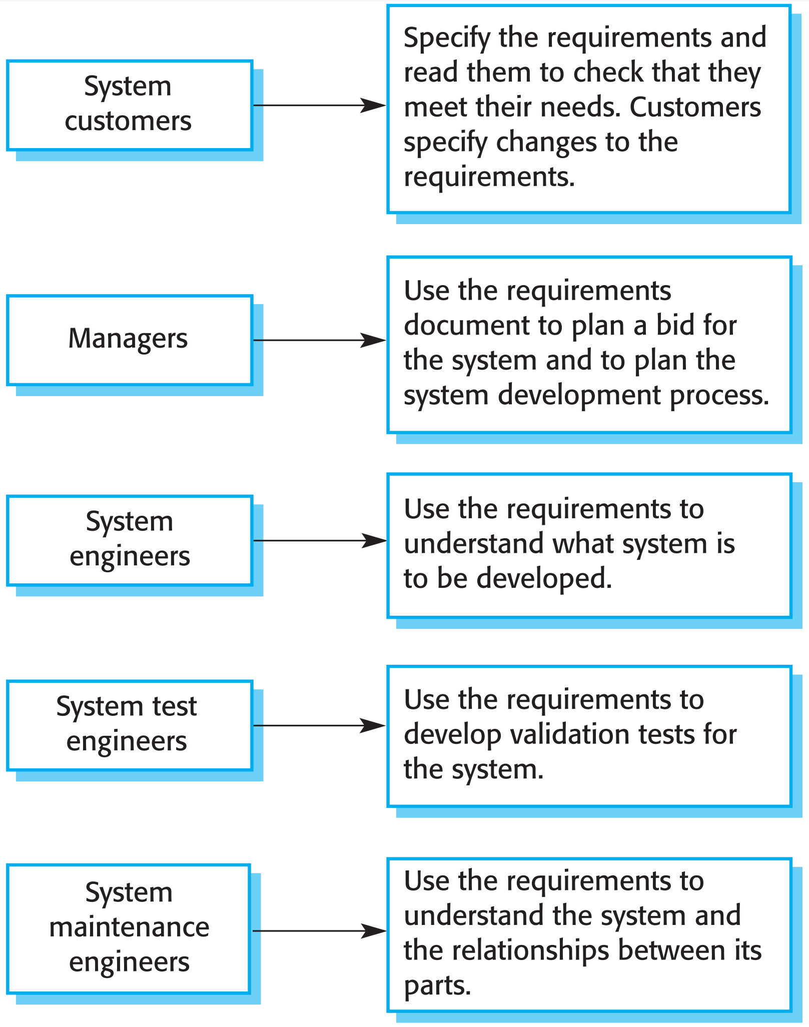
Use cases for the Mentcare system



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.

Users of a requirements document


System customers
Specify the requirements and read them to check that they meet their needs. Customers specify changes to the requirements.
Managers
Use the requirements document to plan a bid for the system and to plan the system development process.
System engineers
Use the requirements to understand what system is to be developed.
System test engineers
Use the requirements to develop validation tests for the system.
System maintenance engineers
Use the requirements to understand the system and the relationships between its parts.

Requirements document variability

- ✧ Information in requirements document depends on type of system and the development **process** used.
- ✧ Systems developed incrementally will, typically, have less detail in the requirements document.

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.

3. 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. Introduced in the next lecture.
- ✧ 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.

Requirements 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?

2. Requirements change

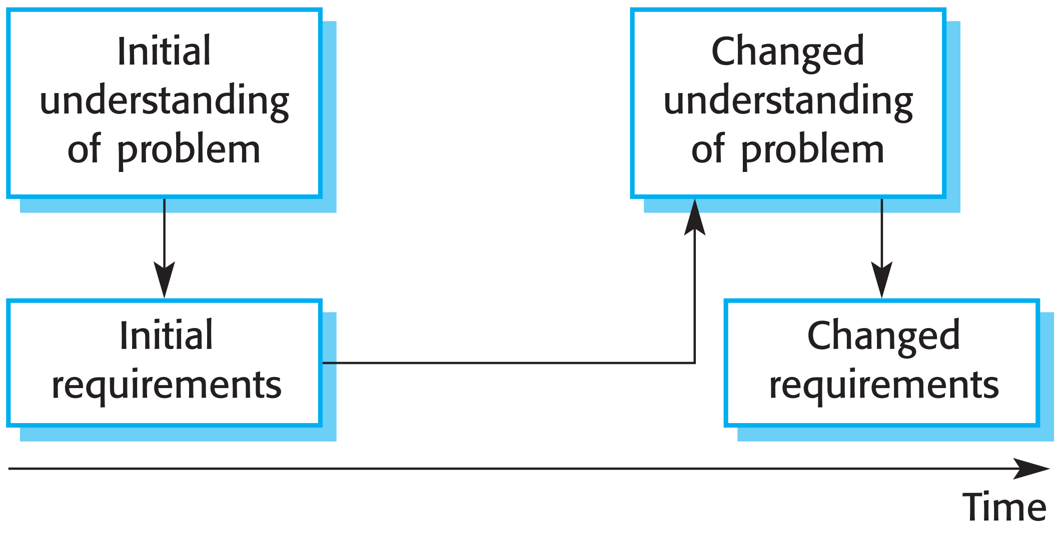
Changing requirements

- ✧ The business and technical environment of the system will eventually change.
  - 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


Initial understanding of problem
Changed understanding of problem
Initial requirements
Changed requirements
Time

Requirements Engineering: 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.

Requirements Engineering: 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, user stories and scenarios.

Requirements Engineering: 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.

Requirements Engineering: 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.

Watch:

- An introduction to Requirements Engineering
  https://www.youtube.com/watch?v=Ec0s0z5uXQ8