Software Requirements

Requirements engineering

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

What is a requirement?

- Statements of a service or of a system constraint from
  - high-level (abstract)
  - Low-level (mathematical functional specifications)
- This is inevitable as requirements may serve a dual function
  - May be the basis for a bid for a contract
    - open to interpretation;
  - May be the basis for the contract itself
    - 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 organisation's needs. Once a contract has been awarded, the contractor must write a system definition for the client in more detail so that the client understands and can validate what the software will do. Both of these documents may be called the *requirements document* for the system."

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.
  - define what should be implemented
  - may be part of a contract between client and contractor.

Definitions and specifications

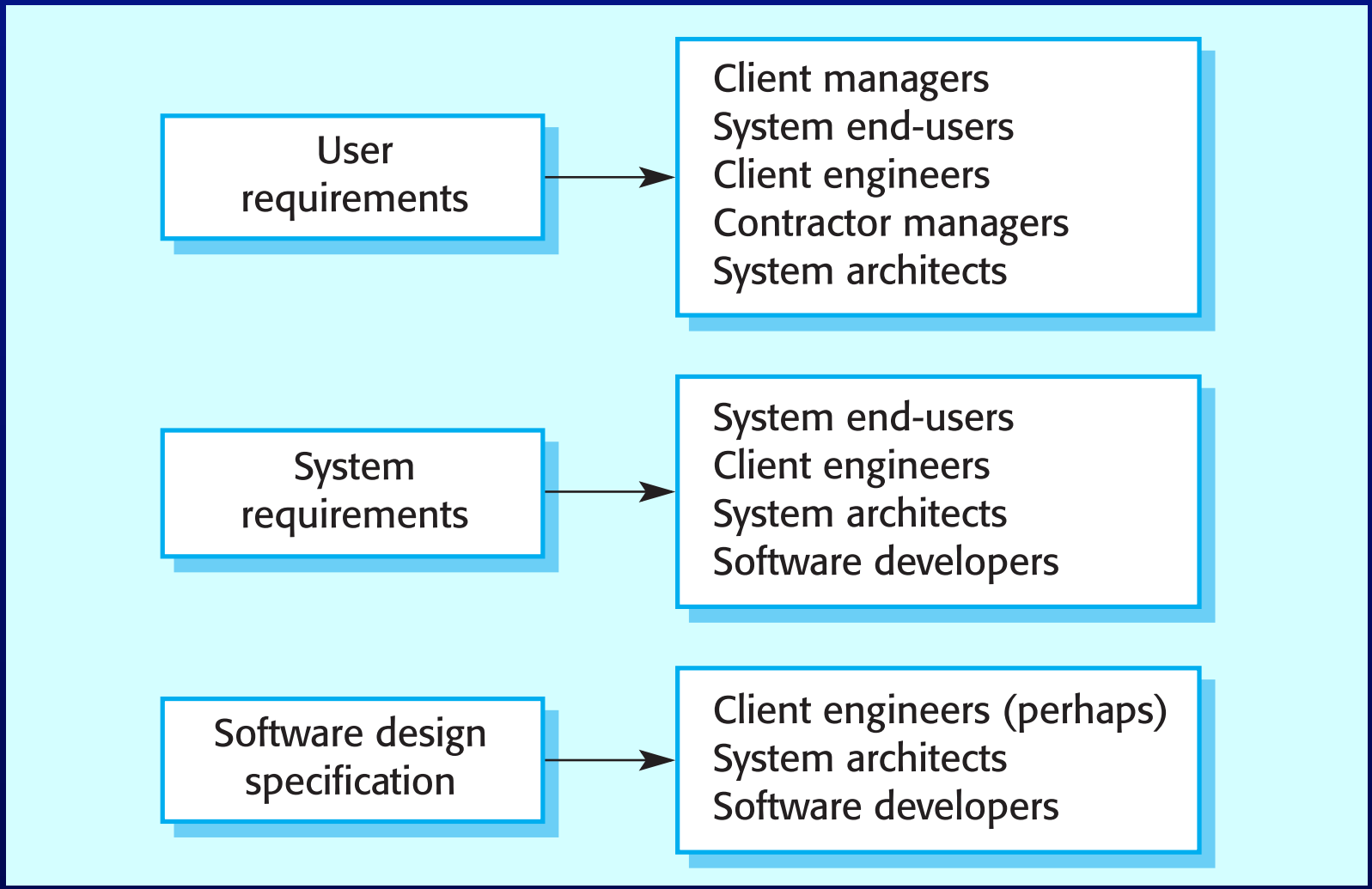
User requirement definition

1. The software must provide a means of representing and accessing external files created by other tools.

System requirements specification

1.1 The user should be provided with facilities to define the type of external files.
1.2 Each external file type may have an associated tool which may be applied to the file.
1.3 Each external file type may be represented as a specific icon on the user's display.
1.4 Facilities should be provided for the icon representing an external file type to be defined by the user.
1.5 When a user selects an icon representing an external file, the effect of that selection is to apply the tool associated with the type of the external file to the file represented by the selected icon.

Requirements readers



Functional and non-functional requirements

- Functional requirements
  - Describe functionality or system services.
  - What the system should provide
  - How the system should react to particular inputs and
  - how the system should behave in particular situations.
- Non-functional requirements
  - constraints on the services or functions
  - E.g. timing constraints, constraints on the development process, standards, etc.
- Domain requirements
  - come from the application domain
  - reflect characteristics of that domain.

Examples of functional requirements

- Some LIBSYS System requirements
  - The user shall be able to search either all of the initial set of databases or select a subset from it.
  - The system shall provide appropriate viewers for the user to read documents in the document store.
  - Every order shall be allocated a unique identifier (ORDER_ID) which the user shall be able to copy to the account's permanent storage area.

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 ‘appropriate viewers’
  - User intention - special purpose viewer for each different document type;
  - Developer interpretation - Provide a text viewer that shows the contents of the document.

Requirements completeness and consistency

- 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.

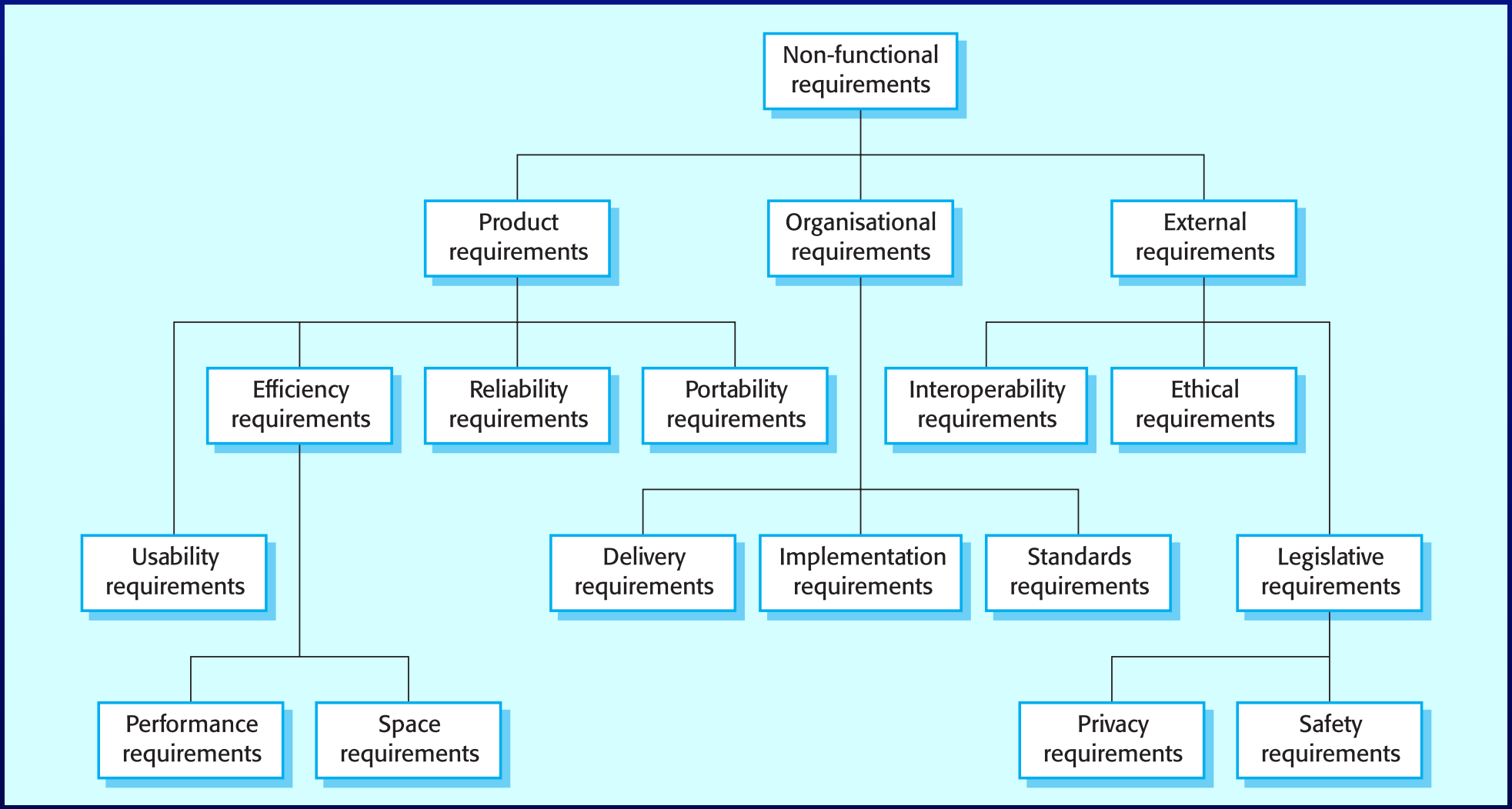
Non-functional requirements

- 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 CASE system, programming language or development method.
- Non-functional requirements may be more critical than functional requirements.
  - If these are not met, the system is useless.

Non-functional classifications

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

Non-functional requirement types



Non-functional requirements examples

- Product requirement
  - 8.1 The user interface for LIBSYS shall be implemented as simple HTML without frames or Java applets.
- Organisational requirement
  - 9.3.2 The system development process and deliverable documents shall conform to the process and deliverables defined in XYZCo-SP-STAN-95.
- External requirement
  - 7.6.5 The system shall not disclose any personal information about customers apart from their name and reference number to the operators of the system.

Goals and requirements

- Non-functional requirements
  - may be very difficult to state precisely
  - 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.

Examples

- **A system goal**
  - The system should be easy to use by experienced controllers and should be organised in such a way that user errors are minimised.
- **A verifiable non-functional requirement**
  - Experienced controllers shall be able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced users shall not exceed two per day.

Requirements measures

Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	M Bytes 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 interaction

- Conflicts between different non-functional requirements are common in complex systems.
- Spacecraft system
  - To minimise weight, the number of separate chips in the system should be minimised.
  - To minimise power consumption, lower power chips should be used.
  - However, using low power chips may mean that more chips have to be used. Which is the most critical requirement?

Domain requirements

- Derived from the application domain
- Describe system characteristics and features that reflect the domain.
- Domain requirements:
  - new functional requirements
    - constraints on existing requirements
    - define specific computations.
- If domain requirements are not satisfied, the system may be unworkable.

Train protection system

- The deceleration of the train shall be computed as:
  - $D_{train} = D_{control} + D_{gradient}$

  where $D_{gradient}$ is $9.81ms^2$ * compensated gradient/alpha and where the values of $9.81ms^2$ / alpha are known for different types of train.

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 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 design may be a domain requirement.

User requirements

- Functional and non-functional requirements
  - should be understandable by system users
    - who don't have detailed technical knowledge.
- User requirements are defined using:
  - natural language and tables and diagrams
  - as these can be understood by all users.

Problems with natural language (1)

- 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.

Problems with natural language (2)

- Ambiguity
  - The readers and writers of the requirement must interpret the same words in the same way.
  - NL is naturally ambiguous so this is very difficult.
- Over-flexibility
  - The same thing may be said in a number of different ways in the specification.
- Lack of modularisation
  - NL structures are inadequate to structure system requirements.

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.

System requirements

- More detailed specifications of system functions, services and constraints than user requirements.
- They are intended to be a basis for designing the system.
- They may be incorporated into the system contract.
- System requirements may be defined or illustrated using system models.

Structured language specifications

- The freedom of the requirements writer is limited by a predefined template for requirements.
- All requirements are written in a standard way.
- The terminology used in the description may be limited.
- The advantage: the most of the expressiveness is maintained
  - The limitation: a degree of uniformity is imposed on the specification.

Structured language specifications

- 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.
  - Indication of other entities required.
  - Pre and post conditions (if appropriate).
  - The side effects (if any) of the function.
- Tabular Specification
  - Used to supplement natural language.
  - Particularly useful when you have to define a number of possible alternative courses of action.
- Graphical models
  - show how state changes
  - describe a sequence of actions.

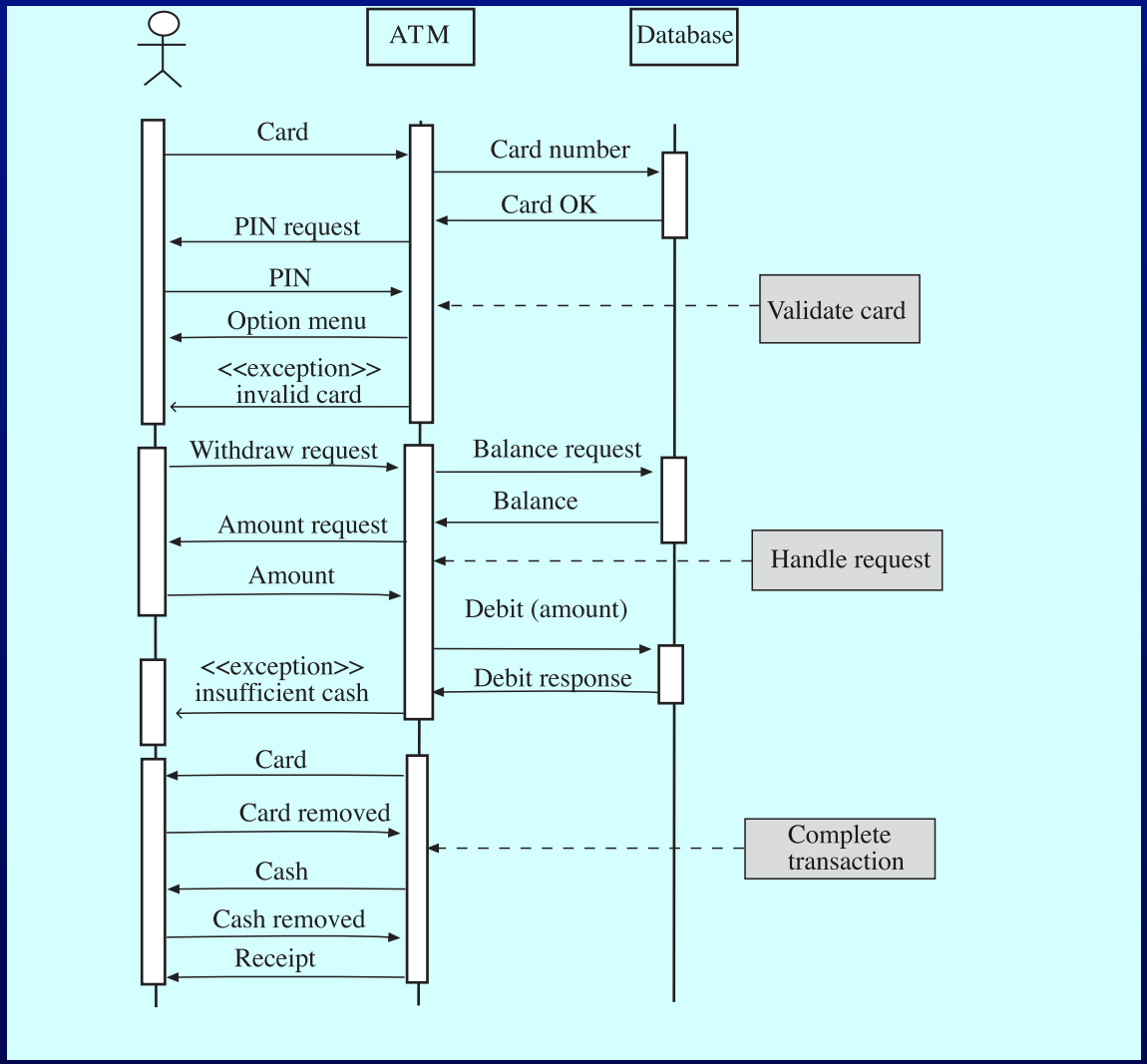
Form-based node specification

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
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.
Requires	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

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

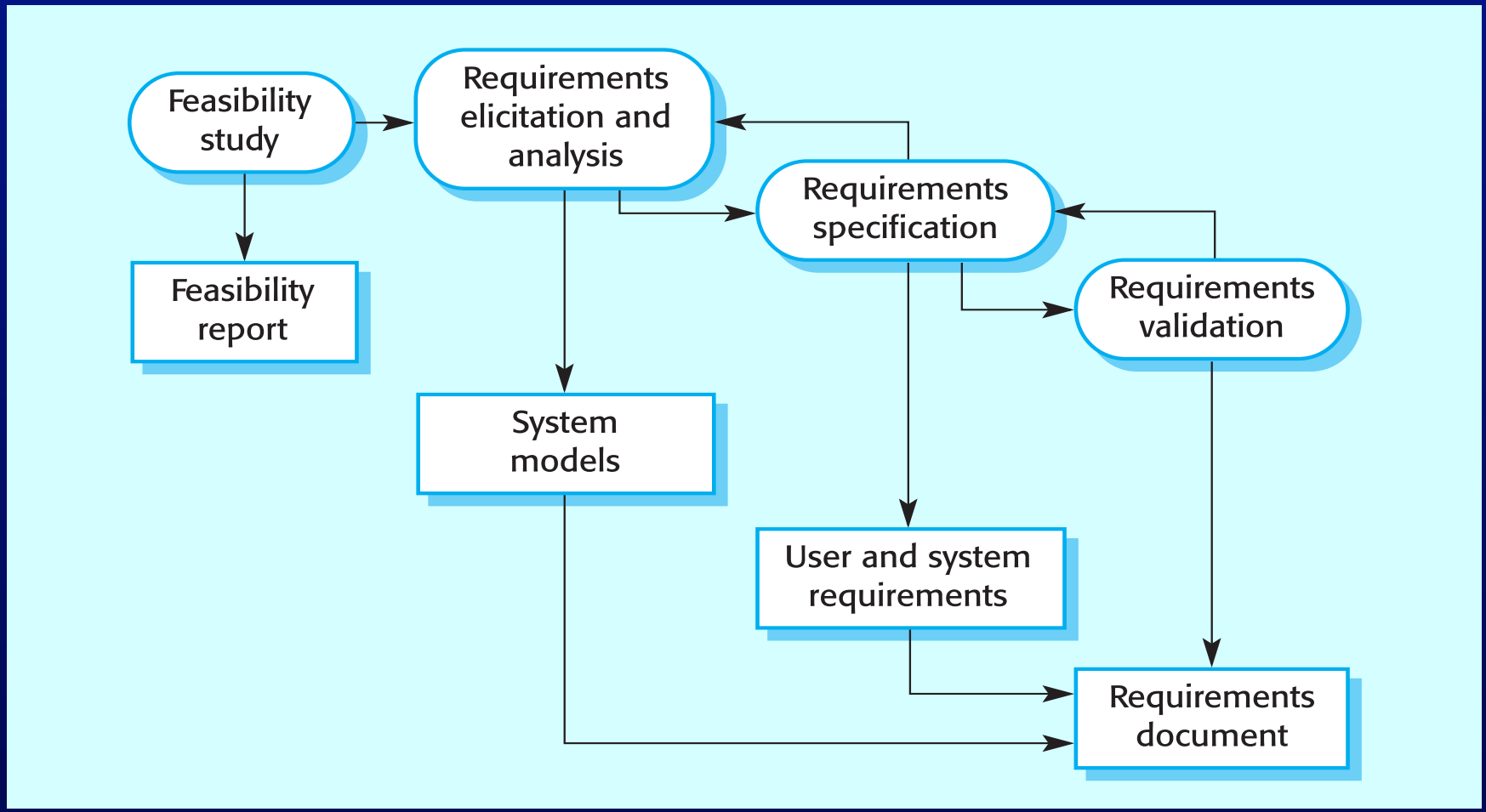
Sequence diagram of ATM withdrawal



Requirements engineering processes

- Vary widely depending on:
  - the application domain
  - the people involved and
  - the organisation developing the requirements.
- Generic activities common to all processes
  - Requirements elicitation;
  - Requirements analysis;
  - Requirements validation;
  - Requirements management.

The requirements engineering process



Feasibility studies

- A feasibility study decides whether or not the proposed system is worthwhile.
- A short focused study that checks
  - If the system contributes to organisational objectives;
  - If the system can be engineered using current technology and within budget;
  - If the system can be integrated with other systems that are used.

Feasibility study implementation

- Based on information assessment (what is required), information collection and report writing.
- Questions for people in the organisation
  - What if the system wasn't implemented?
  - What are current process problems?
  - How will the proposed system help?
  - What will be the integration problems?
  - Is new technology needed? What skills?
  - What facilities must be supported by the proposed system?

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.*

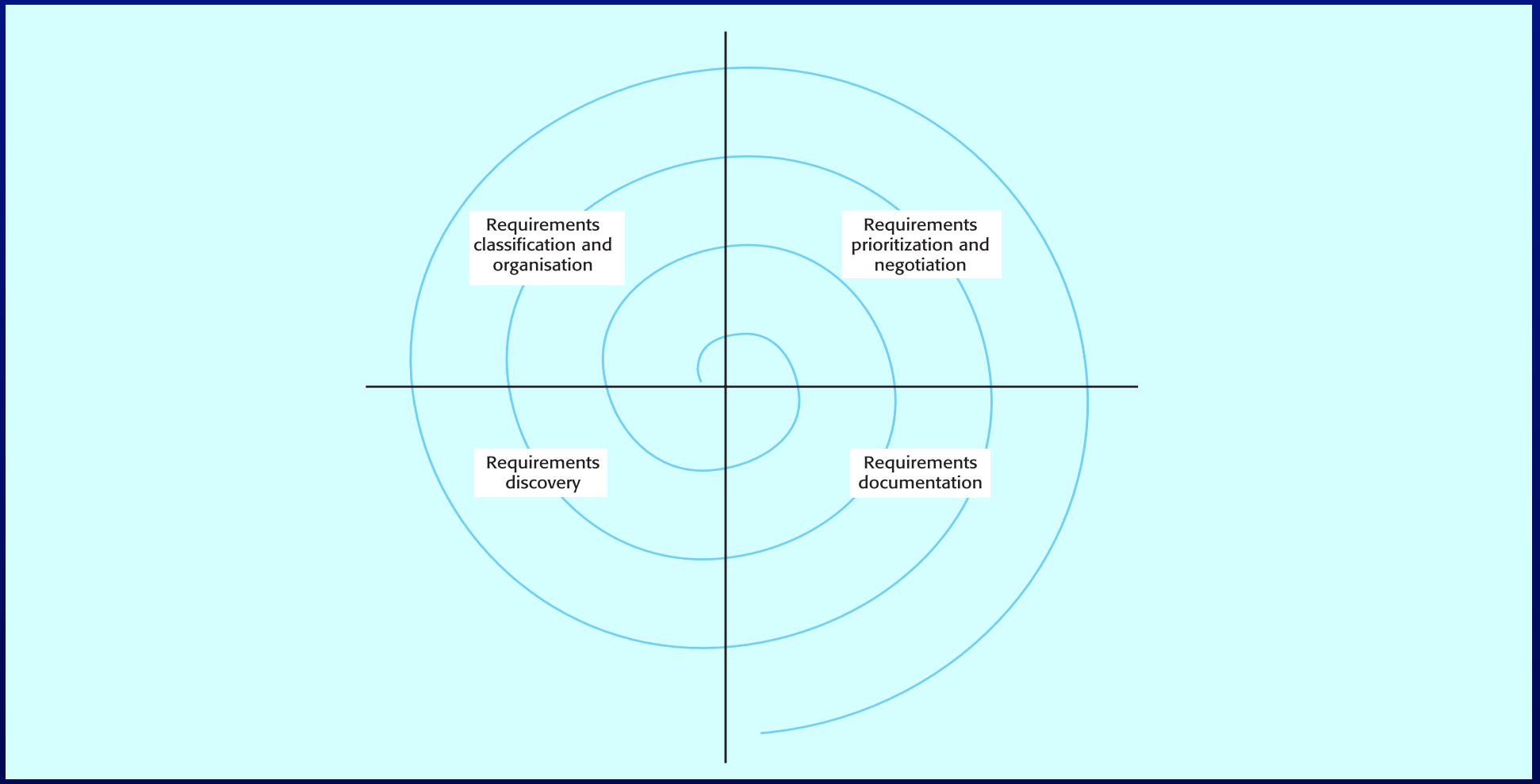
Problems of requirements analysis

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

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

The requirements spiral



Viewpoints

- Viewpoints are a way of structuring the requirements to represent the perspectives of different stakeholders.
  - Stakeholders may be classified under different viewpoints.
- This multi-perspective analysis is important
  - there is no single correct way to analyse system requirements.

Types of viewpoint

- Interactor viewpoints
  - People or other systems that interact directly with the system.
  - E.g. In an ATM, the customer's and the account database are interactor VPs.
- Indirect viewpoints
  - Stakeholders who do not use the system themselves but who influence the requirements.
  - E.g. In an ATM, management and security staff are indirect viewpoints.
- Domain viewpoints
  - Domain characteristics and constraints that influence the requirements.
  - E.g.In an ATM, an example would be standards for inter-bank communications.

Interviewing

- In formal or informal interviewing, the RE team puts questions to stakeholders about the system that they use and the system to be developed.
- There are two types of interview
  - Closed interviews where a pre-defined set of questions are answered.
  - Open interviews where there is no pre-defined agenda and a range of issues are explored with stakeholders.

Social and organisational factors

- Software systems are used in a social and organisational context. This can influence or even dominate the system requirements.
- Social and organisational factors are not a single viewpoint but are influences on all viewpoints.
- Good analysts must be sensitive to these factors but currently no systematic way to tackle their analysis.

Scope of ethnography

- Requirements that are derived from the way that people actually work rather than the way I which process definitions suggest that they ought to work.
- Requirements that are derived from cooperation and awareness of other people's activities.

Scenarios

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

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

System modelling

- System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers.
- Different models present the system from different perspectives
  - External perspective showing the system's context or environment;
  - Behavioural perspective showing the behaviour of the system;
  - Structural perspective showing the system or data architecture.

Model types

- Data processing model showing how the data is processed at different stages.
- Composition model showing how entities are composed of other entities.
- Architectural model showing principal sub-systems.
- Classification model showing how entities have common characteristics.
- Stimulus/response model showing the system's reaction to events.

Model Types

- Context models are used to illustrate the operational context of a system
  - they show what lies outside the system boundaries.
- Architectural models show the system and its relationship with other systems.
- Process models show the overall process and the processes that are supported by the system.
- Data flow models may be used to show the processes and the flow of information from one process to another.

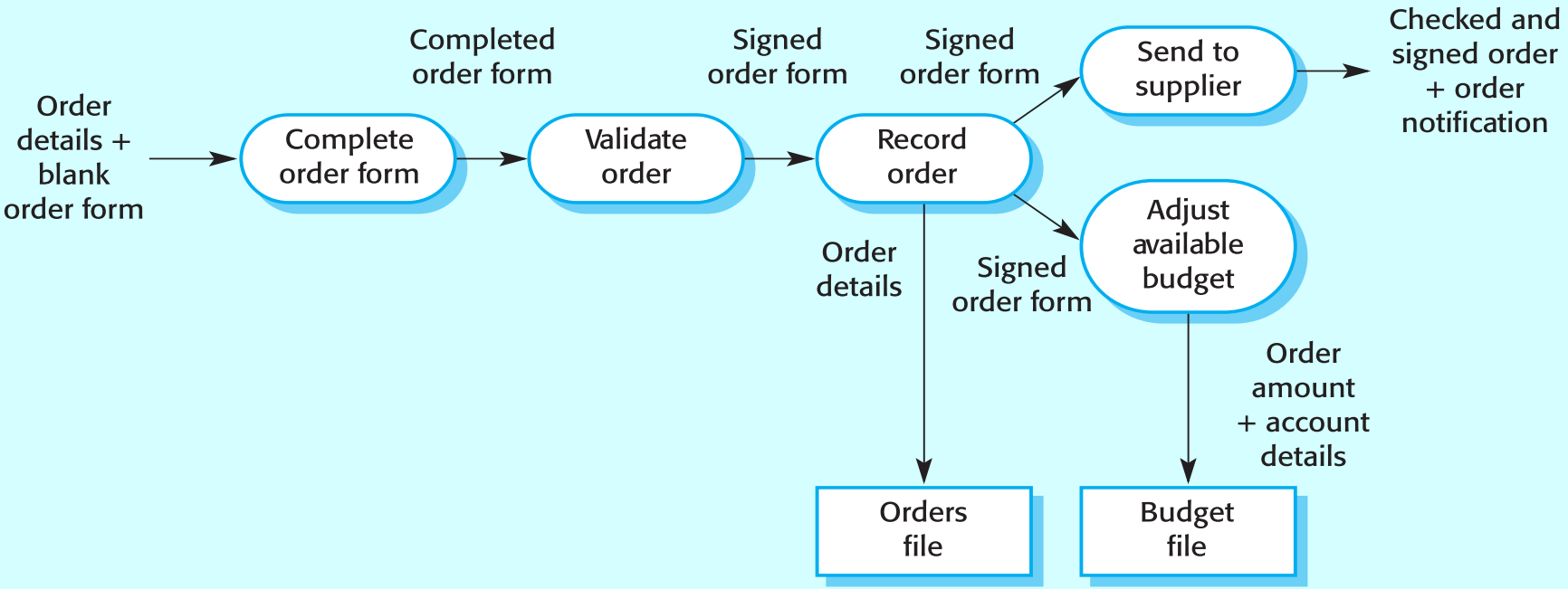
Behavioural models

- Behavioural models are used to describe the overall behaviour of a system.
- Two types of behavioural model are:
  - Data processing models that show how data is processed as it moves through the system;
  - State machine models that show the systems response to events.
- These models show different perspectives so both of them are required to describe the system's behaviour.

Data flow diagrams

- DFDs model the system from a functional perspective.
- Tracking and documenting how the data associated with a process is helpful to develop an overall understanding of the system.
- Data flow diagrams may also be used in showing the data exchange between a system and other systems in its environment.

Order processing DFD



State machine models

- These model the behaviour of the system in response to external and internal events.
- They show the system's responses to stimuli so are often used for modelling real-time systems.
- State machine models show system states as nodes and events as arcs between these nodes. When an event occurs, the system moves from one state to another.

Semantic data models

- Used to describe the logical structure of data processed by the system.
- An entity-relation-attribute model sets out the entities in the system, the relationships between these entities and the entity attributes
- Widely used in database design. Can readily be implemented using relational databases.
- No specific notation provided in the UML but objects and associations can be used.

Data dictionaries

- Data dictionaries are lists of all of the names used in the system models. Descriptions of the entities, relationships and attributes are also included.
- Advantages
  - Support name management and avoid duplication;
  - Store of organisational knowledge linking analysis, design and implementation;
- Many CASE workbenches support data dictionaries.

Data dictionary entries

Name	Description	Type	Date
Article	Details of the published article that may be ordered by people using LIBSYS.	Entity	30.12.2002
authors	The names of the authors of the article who may be due a share of the fee.	Attribute	30.12.2002
Buyer	The person or organisation that orders a co py of the article.	Entity	30.12.2002
fee-payable-to	A 1:1 relationship between Article and the Copyright Agency who should be paid the copyright fee.	Relation	29.12.2002
Address (Buyer)	The address of the buyer. This is used to any paper billing information that is required.	Attribute	31.12.2002

Object models

- Object models describe the system in terms of object classes and their associations.
- An object class is an abstraction over a set of objects with common attributes and the services (operations) provided by each object.
- Various object models may be produced
  - Inheritance models;
  - Aggregation models;
  - Interaction models.

Object models and the UML

- The UML is a standard representation devised by the developers of widely used object-oriented analysis and design methods.
- It has become an effective standard for object-oriented modelling.

Structured methods

- Structured methods incorporate system modelling as an inherent part of the method.
- Methods define a set of models, a process for deriving these models and rules and guidelines that should apply to the models.
- CASE tools support system modelling as part of a structured method.

Method weaknesses

- They do not model non-functional system requirements.
- They do not usually include information about whether a method is appropriate for a given problem.
- The may produce too much documentation.
- The system models are sometimes too detailed and difficult for users to understand.

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. Covered in Chapter 17.
- Test-case generation
  - Developing tests for requirements to check testability.

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 management

- Requirements management is the process of managing changing requirements during the requirements engineering process and system development.
- Requirements are inevitably incomplete and inconsistent
  - New requirements emerge during the process as business needs change and a better understanding of the system is developed;
  - Different viewpoints have different requirements and these are often contradictory.

Traceability

- Traceability is concerned with the relationships between requirements, their sources and the system design
- Source traceability
  - Links from requirements to stakeholders who proposed these requirements;
- Requirements traceability
  - Links between dependent requirements;
- Design traceability
  - Links from the requirements to the design;

A traceability matrix

Req. id	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2
1.1		D	R					
1.2			D			D		D
1.3	R			R				
2.1			R		D			D
2.2								D
2.3		R		D				
3.1								R
3.2							R	

CASE tool support

- Requirements storage
  - Requirements should be managed in a secure, managed data store.
- Change management
  - The process of change management is a workflow process whose stages can be defined and information flow between these stages partially automated.
- Traceability management
  - Automated retrieval of the links between requirements.

The requirements document

- The 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

IEEE requirements standard

- Defines a generic structure for a requirements document that must be instantiated for each specific system.
  - Introduction.
  - General description.
  - Specific requirements.
  - Appendices.
  - Index.

Requirements document structure

- Preface
- Introduction
- Glossary
- User requirements definition
- System architecture
- System requirements specification
- System models
- System evolution
- Appendices
- Index

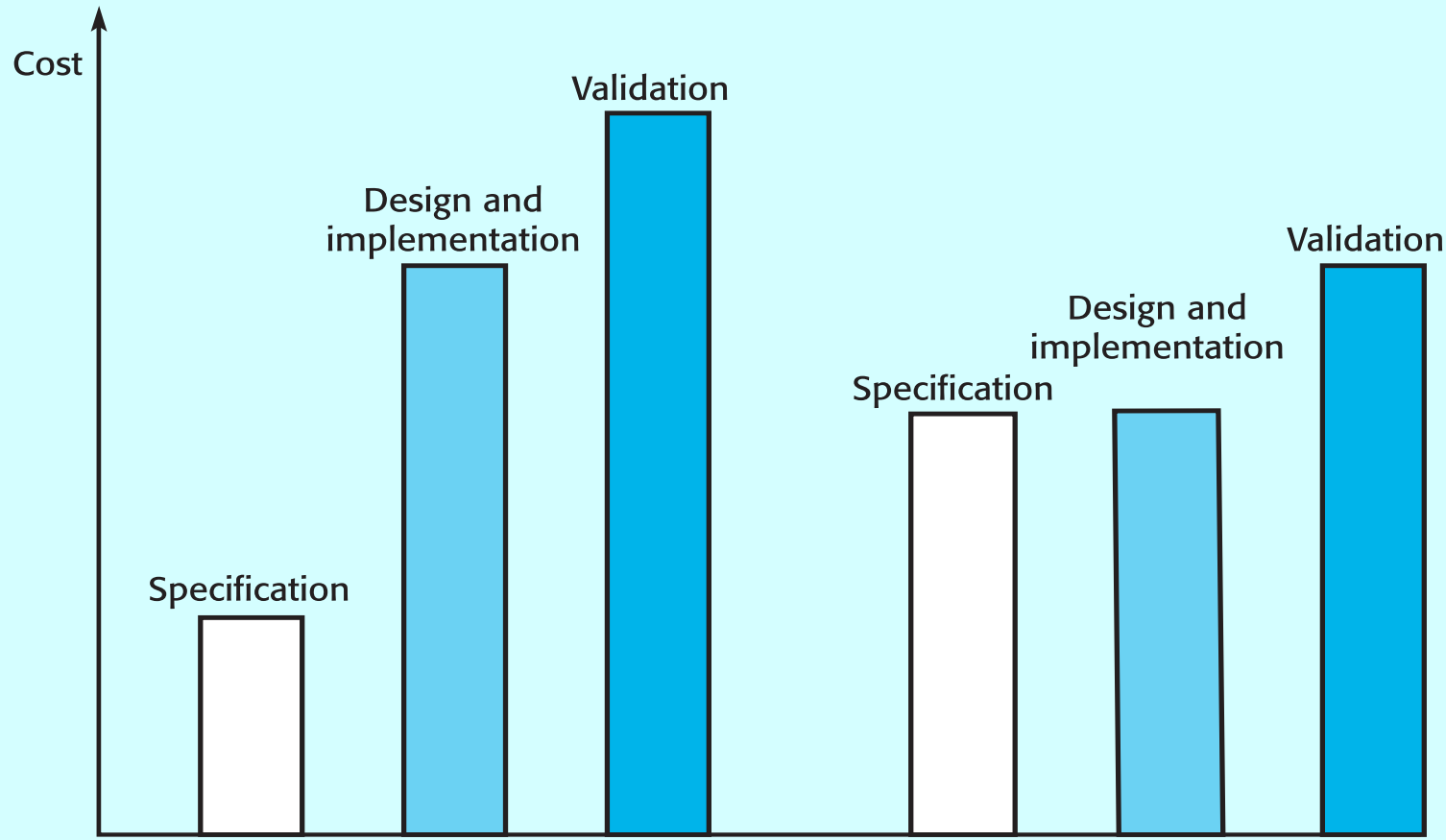
Formal methods

- Based on mathematical representation and analysis of software.
- Formal methods did not become largely used
  - Other software engineering techniques have been successful
  - Market changes (time-to-market vs software with a low error count)
  - Not well-suited to specifying and analysing user interfaces and user interaction;
  - Formal methods are still hard to scale up to large systems.

Use of formal methods

- The principal benefits of formal methods are in reducing the number of faults in systems.
- The main area of applicability is in critical systems engineering.
  - They are cost-effective because high system failure costs must be avoided.

Development costs with formal specification



Specification techniques

- Algebraic specification
  - The system is specified in terms of its operations and their relationships.
- Model-based specification
  - The system is specified in terms of a state model that is constructed using mathematical constructs such as sets and sequences. Operations are defined by modifications to the system's state.

List specification

LIST (Elem)

sort List
imports INTEGER

Defines a list where elements are added at the end and removed from the front. The operations are Create, which brings an empty list into existence, Cons, which creates a new list with an added member, Length, which evaluates the list size, Head, which evaluates the front element of the list, and Tail, which creates a list by removing the head from its input list. Undefined represents an undefined value of type Elem.

Create → List
Cons (List, Elem) → List
Head (List) → Elem
Length (List) → Integer
Tail (List) → List

Head (Create) = Undefined **exception** (empty list)
Head (Cons (L, v)) = **if** L = Create **then** v **else** Head (L)
Length (Create) = 0
Length (Cons (L, v)) = Length (L) + 1
Tail (Create) = Create
Tail (Cons (L, v)) = **if** L = Create **then** Create **else** Cons (Tail (L), v)