An Introduction to Software Engineering

Objectives

- To introduce software engineering and to explain its importance
- To set out the answers to key questions about software engineering
- To introduce ethical and professional issues and to explain why they are of concern to software engineers

Topics covered

- FAQs about software engineering
- Professional and ethical responsibility

Software engineering

- The economies of ALL developed nations are dependent on software.
- More and more systems are software controlled
- Software engineering is concerned with theories, methods and tools for professional software development.
- Expenditure on software represents a significant fraction of GNP in all developed countries.

Software costs

- Software costs often dominate computer system costs. The costs of software on a PC are often greater than the hardware cost.
- Software costs more to maintain than it does to develop. For systems with a long life, maintenance costs may be several times development costs.
- Software engineering is concerned with cost-effective software development.

FAQs about software engineering

- What is software?
- What is software engineering?
- What is the difference between software engineering and computer science?
- What is the difference between software engineering and system engineering?
- What is a software process?
- What is a software process model?

FAQs about software engineering

- What are the costs of software engineering?
- What are software engineering methods?
- What is CASE (Computer-Aided Software Engineering)
- What are the attributes of good software?
- What are the key challenges facing software engineering?

What is software?

- Computer programs and associated documentation such as requirements, design models and user manuals.
- Software products may be developed for a particular customer or may be developed for a general market.
- Software products may be
  - Generic - developed to be sold to a range of different customers e.g. PC software such as Excel or Word.
  - Bespoke (custom) - developed for a single customer according to their specification.
- New software can be created by developing new programs, configuring generic software systems or reusing existing software.

What is software engineering?

- Software engineering is an engineering discipline that is concerned with all aspects of software production.
- Software engineers should adopt a systematic and organised approach to their work and use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available.

What is the difference between software engineering and computer science?

- Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
- Computer science theories are still insufficient to act as a complete underpinning for software engineering (unlike e.g. physics and electrical engineering).

What is the difference between software engineering and system engineering?

- System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this process concerned with developing the software infrastructure, control, applications and databases in the system.
- System engineers are involved in system specification, architectural design, integration and deployment.

What is a software process?

- A set of activities whose goal is the development or evolution of software.
- Generic activities in all software processes are:
  - Specification - what the system should do and its development constraints
  - Development - production of the software system
  - Validation - checking that the software is what the customer wants
  - Evolution - changing the software in response to changing demands.

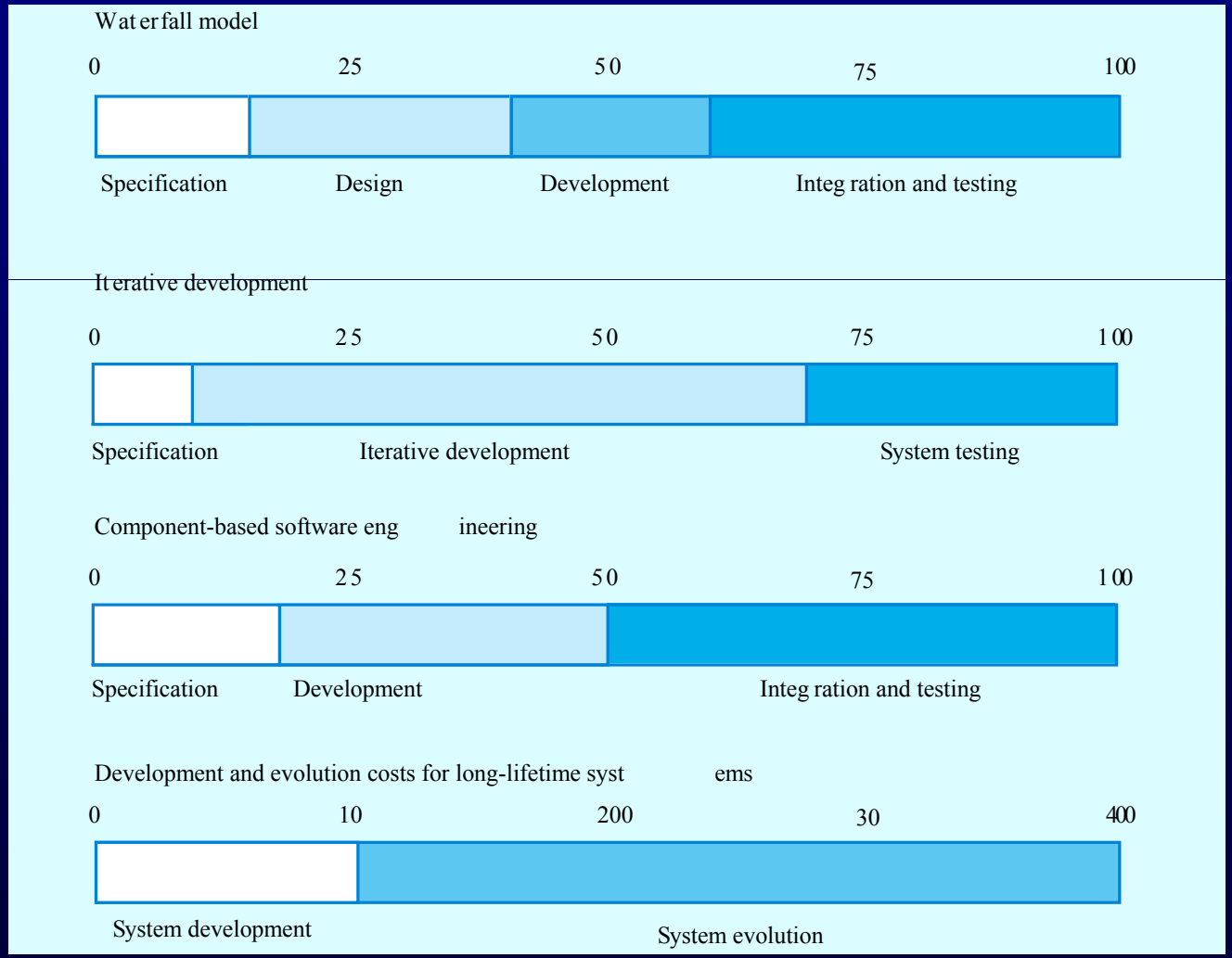
What is a software process model?

- A simplified representation of a software process, presented from a specific perspective.
- Examples of process perspectives are
  - Workflow perspective - sequence of activities;
  - Data-flow perspective - information flow;
  - Role/action perspective - who does what.
- Generic process models
  - Waterfall;
  - Iterative development;
  - Component-based software engineering.

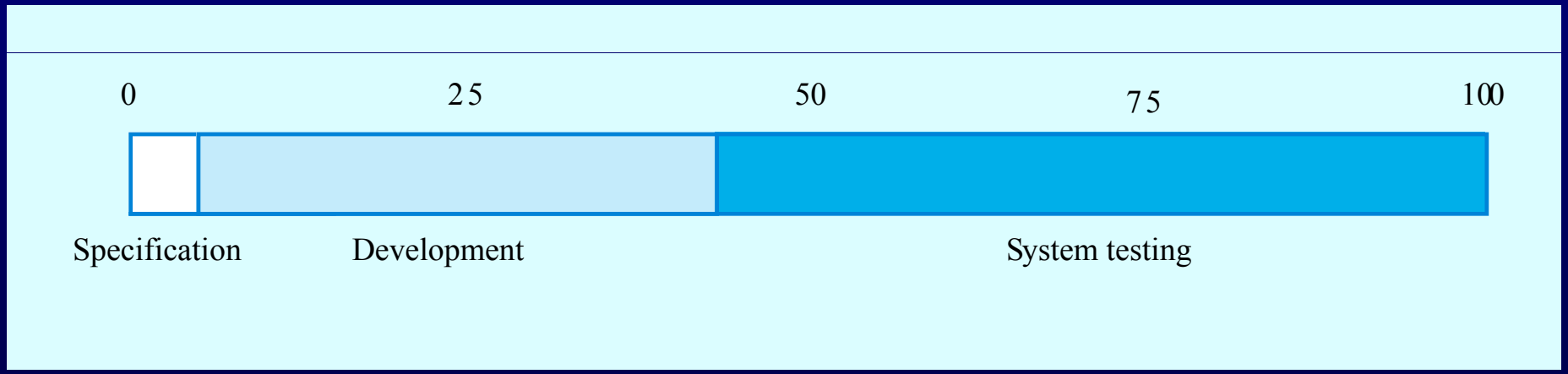
What are the costs of software engineering?

- Roughly 60% of costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.
- Costs vary depending on the type of system being developed and the requirements of system attributes such as performance and system reliability.
- Distribution of costs depends on the development model that is used.

Activity cost distribution



Product development costs



What are software engineering methods?

- Structured approaches to software development which include system models, notations, rules, design advice and process guidance.
- Model descriptions
  - Descriptions of graphical models which should be produced;
- Rules
  - Constraints applied to system models;
- Recommendations
  - Advice on good design practice;
- Process guidance
  - What activities to follow.

What is CASE (Computer-Aided Software Engineering)

- Software systems that are intended to provide automated support for software process activities.
- CASE systems are often used for method support.
- Upper-CASE
  - Tools to support the early process activities of requirements and design;
- Lower-CASE
  - Tools to support later activities such as programming, debugging and testing.

What are the attributes of good software?

- The software should deliver the required functionality and performance to the user and should be maintainable, dependable and acceptable.
- Maintainability
  - Software must evolve to meet changing needs;
- Dependability
  - Software must be trustworthy;
- Efficiency
  - Software should not make wasteful use of system resources;
- Acceptability
  - Software must accepted by the users for which it was designed. This means it must be understandable, usable and compatible with other systems.

What are the key challenges facing software engineering?

- Heterogeneity, delivery and trust.
- Heterogeneity
  - Developing techniques for building software that can cope with heterogeneous platforms and execution environments;
- Delivery
  - Developing techniques that lead to faster delivery of software;
- Trust
  - Developing techniques that demonstrate that software can be trusted by its users.

Professional and ethical responsibility

- Software engineering involves wider responsibilities than simply the application of technical skills.
- Software engineers must behave in an honest and ethically responsible way if they are to be respected as professionals.
- Ethical behaviour is more than simply upholding the law.

Issues of professional responsibility

- Confidentiality
  - Engineers should normally respect the confidentiality of their employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.
- Competence
  - Engineers should not misrepresent their level of competence. They should not knowingly accept work which is outwith their competence.

Issues of professional responsibility

- Intellectual property rights
  - Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc. They should be careful to ensure that the intellectual property of employers and clients is protected.
- Computer misuse
  - Software engineers should not use their technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious (dissemination of viruses).

ACM/IEEE Code of Ethics

- The professional societies in the US have cooperated to produce a code of ethical practice.
- Members of these organisations sign up to the code of practice when they join.
- The Code contains eight Principles related to the behaviour of and decisions made by professional software engineers, including practitioners, educators, managers, supervisors and policy makers, as well as trainees and students of the profession.

Code of ethics - preamble

- Preamble
  - The short version of the code summarizes aspirations at a high level of the abstraction; the clauses that are included in the full version give examples and details of how these aspirations change the way we act as software engineering professionals. Without the aspirations, the details can become legalistic and tedious; without the details, the aspirations can become high sounding but empty; together, the aspirations and the details form a cohesive code.
  - Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

Code of ethics - principles

- PUBLIC
  - Software engineers shall act consistently with the public interest.
- CLIENT AND EMPLOYER
  - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
- PRODUCT
  - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.

Code of ethics - principles

- JUDGMENT
  - Software engineers shall maintain integrity and independence in their professional judgment.
- MANAGEMENT
  - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
- PROFESSION
  - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.

Code of ethics - principles

- COLLEAGUES
  - Software engineers shall be fair to and supportive of their colleagues.
- SELF
  - Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

Ethical dilemmas

- Disagreement in principle with the policies of senior management.
- Your employer acts in an unethical way and releases a safety-critical system without finishing the testing of the system.
- Participation in the development of military weapons systems or nuclear systems.

Key points

- Software engineering is an engineering discipline that is concerned with all aspects of software production.
- Software products consist of developed programs and associated documentation. Essential product attributes are maintainability, dependability, efficiency and usability.
- The software process consists of activities that are involved in developing software products. Basic activities are software specification, development, validation and evolution.
- Methods are organised ways of producing software. They include suggestions for the process to be followed, the notations to be used, rules governing the system descriptions which are produced and design guidelines.

Key points

- CASE tools are software systems which are designed to support routine activities in the software process such as editing design diagrams, checking diagram consistency and keeping track of program tests which have been run.
- Software engineers have responsibilities to the engineering profession and society. They should not simply be concerned with technical issues.
- Professional societies publish codes of conduct which set out the standards of behaviour expected of their members.

Socio-technical Systems

Objectives

- To explain what a socio-technical system is and the distinction between this and a computer-based system
- To introduce the concept of emergent system properties such as reliability and security
- To explain system engineering and system procurement processes
- To explain why the organisational context of a system affects its design and use
- To discuss legacy systems and why these are critical to many businesses

Topics covered

- Emergent system properties
- Systems engineering
- Organizations, people and computer systems
- Legacy systems

What is a system?

- A purposeful collection of inter-related components working together to achieve some common objective.
- A system may include software, mechanical, electrical and electronic hardware and be operated by people.
- System components are dependent on other system components
- The properties and behaviour of system components are inextricably inter-mingled

System categories

- Technical computer-based systems
  - Systems that include hardware and software but where the operators and operational processes are not normally considered to be part of the system. The system is not self-aware.
- Socio-technical systems
  - Systems that include technical systems but also operational processes and people who use and interact with the technical system. Socio-technical systems are governed by organisational policies and rules.

Socio-technical system characteristics

- Emergent properties
  - Properties of the system of a whole that depend on the system components and their relationships.
- Non-deterministic
  - They do not always produce the same output when presented with the same input because the systems's behaviour is partially dependent on human operators.
- Complex relationships with organisational objectives
  - The extent to which the system supports organisational objectives does not just depend on the system itself.

Emergent properties

- Properties of the system as a whole rather than properties that can be derived from the properties of components of a system
- Emergent properties are a consequence of the relationships between system components
- They can therefore only be assessed and measured once the components have been integrated into a system

Examples of emergent properties

Property	Description
Volume	The volume of a system (the total space occupied) varies depending on how the component assemblies are arranged and connected.
Reliability	System reliability depends on component reliability but unexpected interactions can cause new types of failure and therefore affect the reliability of the system.
Security	The security of the system (its ability to resist attack) is a complex property that cannot be easily measured. Attacks may be devised that were not anticipated by the system designers and so may defeat built-in safeguards.
Repairability	This property reflects how easy it is to fix a problem with the system once it has been discovered. It depends on being able to diagnose the problem, access the components that are faulty and modify or replace these components.
Usability	This property reflects how easy it is to use the system. It depends on the technical system components, its operators and its operating environment.

Types of emergent property

- Functional properties
  - These appear when all the parts of a system work together to achieve some objective. For example, a bicycle has the functional property of being a transportation device once it has been assembled from its components.
- Non-functional emergent properties
  - Examples are reliability, performance, safety, and security. These relate to the behaviour of the system in its operational environment. They are often critical for computer-based systems as failure to achieve some minimal defined level in these properties may make the system unusable.

System reliability engineering

- Because of component inter-dependencies, faults can be propagated through the system.
- System failures often occur because of unforeseen inter-relationships between components.
- It is probably impossible to anticipate all possible component relationships.
- Software reliability measures may give a false picture of the system reliability.

Influences on reliability

- *Hardware reliability*
  - What is the probability of a hardware component failing and how long does it take to repair that component?
- *Software reliability*
  - How likely is it that a software component will produce an incorrect output. Software failure is usually distinct from hardware failure in that software does not wear out.
- *Operator reliability*
  - How likely is it that the operator of a system will make an error?

Reliability relationships

- Hardware failure can generate spurious signals that are outside the range of inputs expected by the software.
- Software errors can cause alarms to be activated which cause operator stress and lead to operator errors.
- The environment in which a system is installed can affect its reliability.

The ‘shall-not’ properties

- Properties such as performance and reliability can be measured.
- However, some properties are properties that the system should not exhibit
  - Safety - the system should not behave in an unsafe way;
  - Security - the system should not permit unauthorised use.
- Measuring or assessing these properties is very hard.

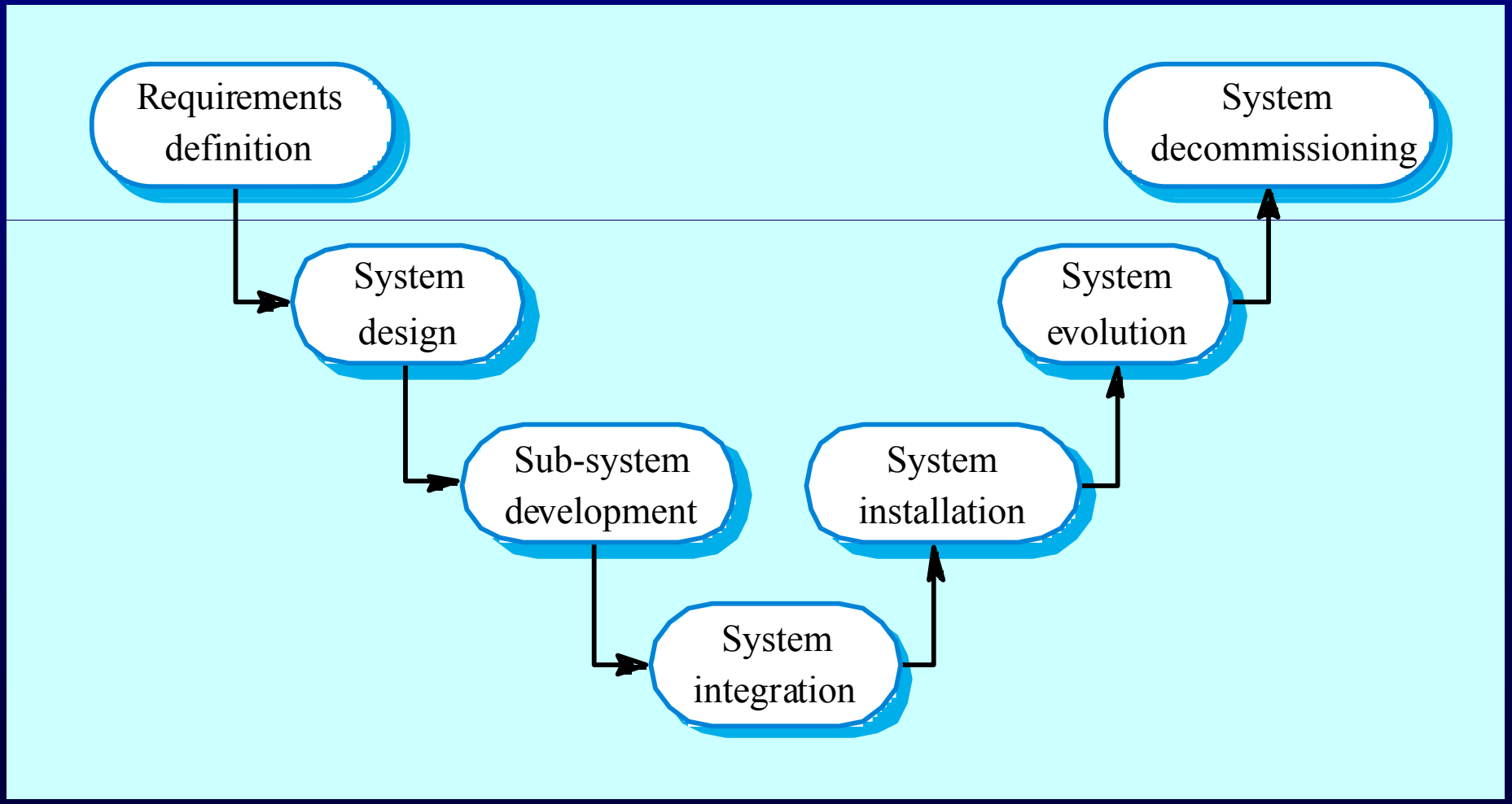
Systems engineering

- Specifying, designing, implementing, validating, deploying and maintaining socio-technical systems.
- Concerned with the services provided by the system, constraints on its construction and operation and the ways in which it is used.

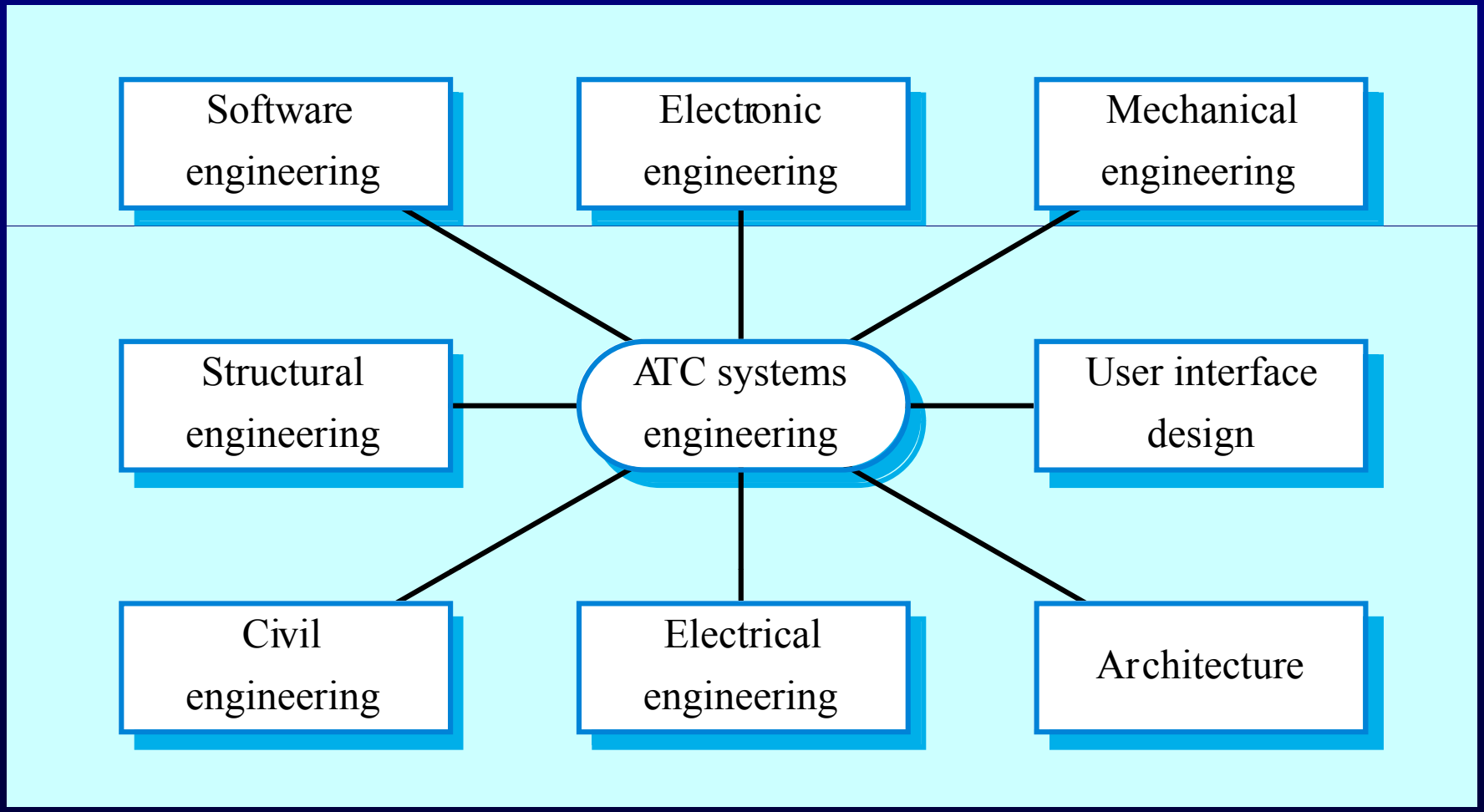
The system engineering process

- Usually follows a ‘waterfall’ model because of the need for parallel development of different parts of the system
  - Little scope for iteration between phases because hardware changes are very expensive. Software may have to compensate for hardware problems.
- Inevitably involves engineers from different disciplines who must work together
  - Much scope for misunderstanding here. Different disciplines use a different vocabulary and much negotiation is required. Engineers may have personal agendas to fulfil.

The systems engineering process



Inter-disciplinary involvement



System requirements definition

- Three types of requirement defined at this stage
  - Abstract functional requirements. System functions are defined in an abstract way;
  - System properties. Non-functional requirements for the system in general are defined;
  - Undesirable characteristics. Unacceptable system behaviour is specified.
- Should also define overall organisational objectives for the system.

System objectives

- Should define why a system is being procured for a particular environment.
- Functional objectives
  - To provide a fire and intruder alarm system for the building which will provide internal and external warning of fire or unauthorized intrusion.
- Organisational objectives
  - To ensure that the normal functioning of work carried out in the building is not seriously disrupted by events such as fire and unauthorized intrusion.

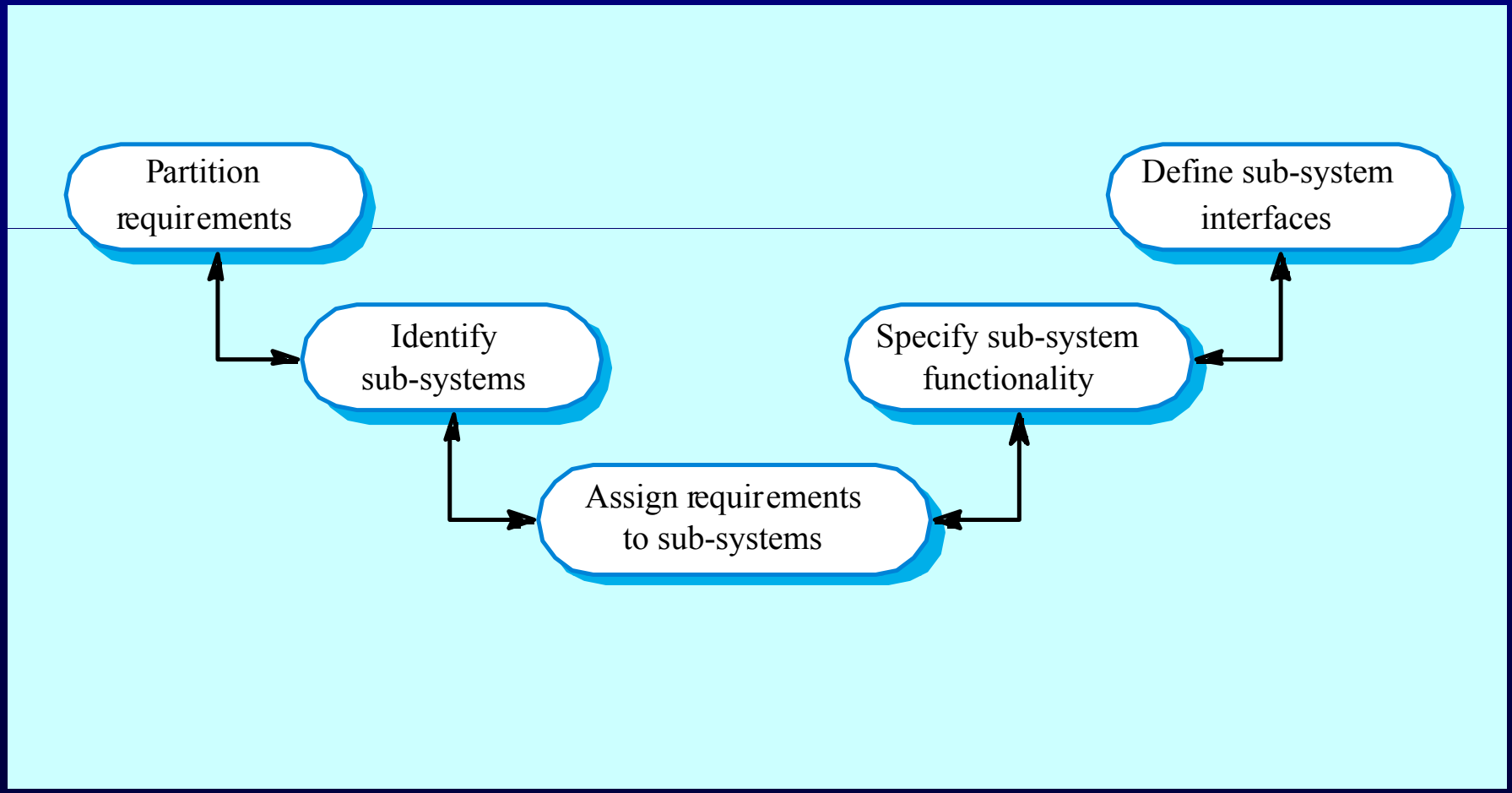
System requirements problems

- Complex systems are usually developed to address wicked problems
  - Problems that are not fully understood;
  - Changing as the system is being specified.
- Must anticipate hardware/communications developments over the lifetime of the system.
- Hard to define non-functional requirements (particularly) without knowing the component structure of the system.

The system design process

- Partition requirements
  - Organise requirements into related groups.
- Identify sub-systems
  - Identify a set of sub-systems which collectively can meet the system requirements.
- Assign requirements to sub-systems
  - Causes particular problems when COTS are integrated.
- Specify sub-system functionality.
- Define sub-system interfaces
  - Critical activity for parallel sub-system development.

The system design process



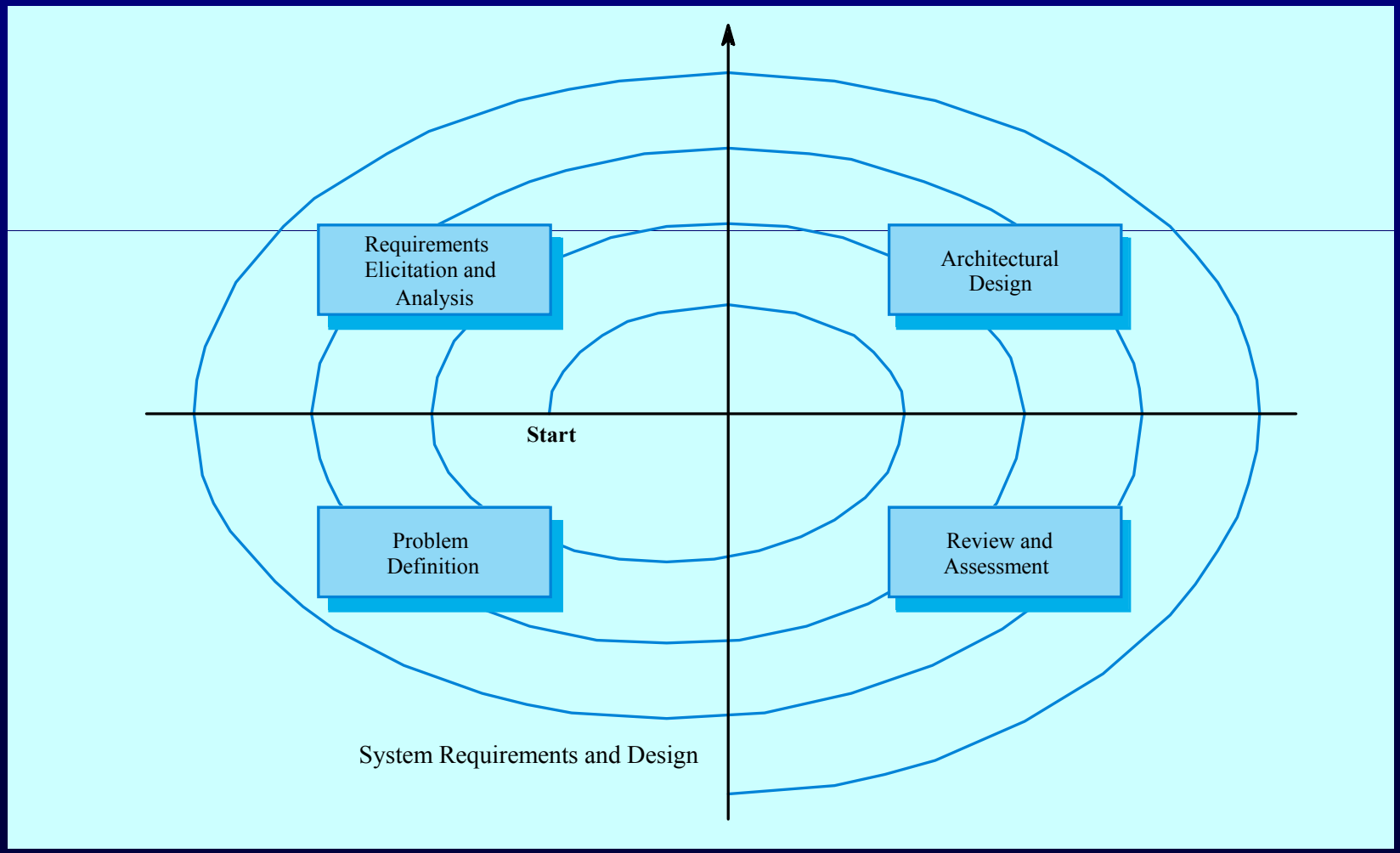
System design problems

- Requirements partitioning to hardware, software and human components may involve a lot of negotiation.
- Difficult design problems are often assumed to be readily solved using software.
- Hardware platforms may be inappropriate for software requirements so software must compensate for this.

Requirements and design

- Requirements engineering and system design are inextricably linked.
- Constraints posed by the system's environment and other systems limit design choices so the actual design to be used may be a requirement.
- Initial design may be necessary to structure the requirements.
- As you do design, you learn more about the requirements.

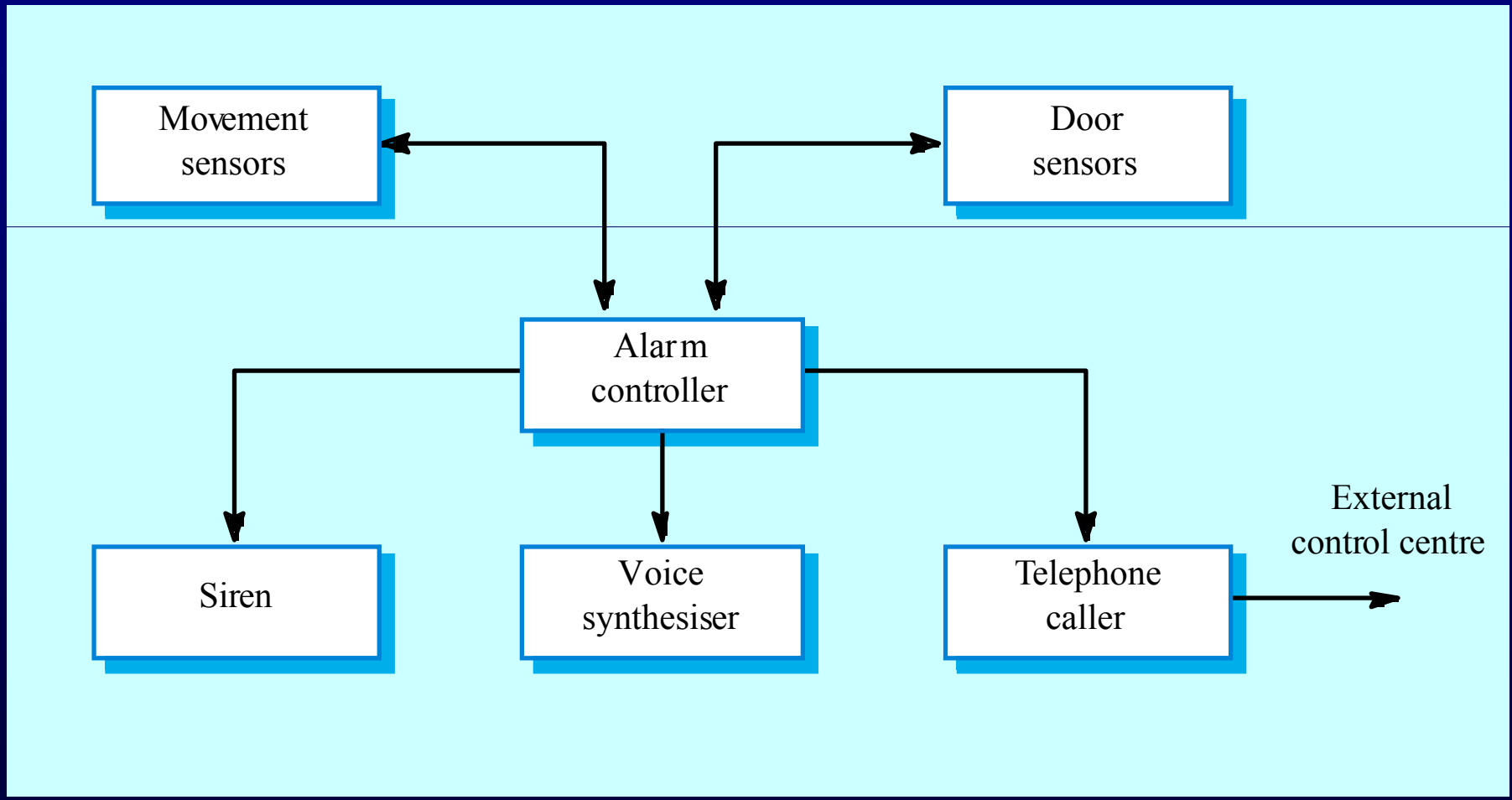
Spiral model of requirements/design



System modelling

- An architectural model presents an abstract view of the sub-systems making up a system
- May include major information flows between sub-systems
- Usually presented as a block diagram
- May identify different types of functional component in the model

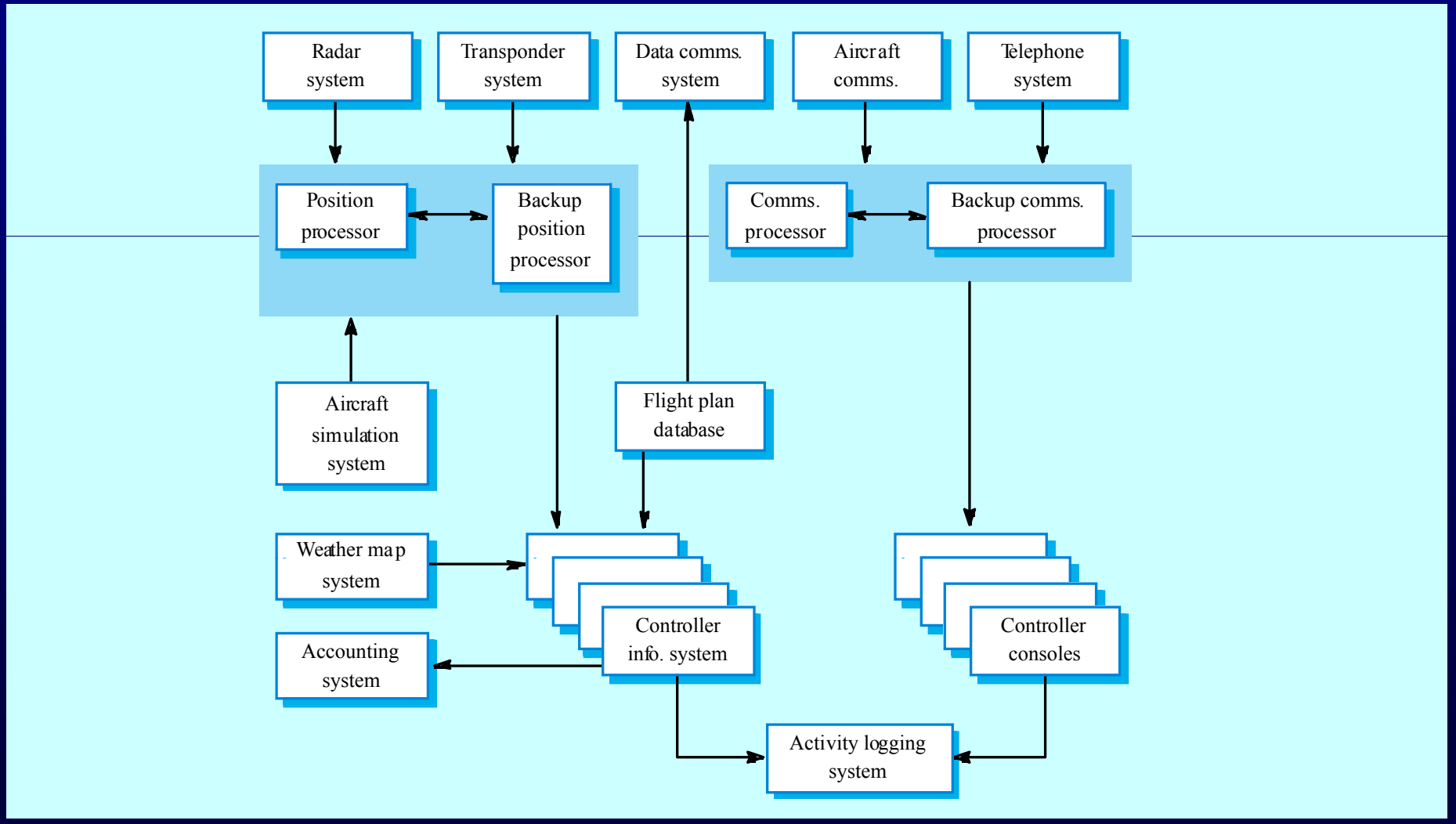
Burglar alarm system



Sub-system description

Sub-system	Description
Movement sensors	Detects movement in the rooms monitored by the system
Door sensors	Detects door opening in the external doors of the building
Alarm controller	Controls the operation of the system
Siren	Emits an audible warning when an intruder is suspected
Voice synthesizer	Synthesizes a voice message giving the location of the suspected intruder
Telephone caller	Makes external calls to notify security, the police, etc.

ATC system architecture



Sub-system development

- Typically parallel projects developing the hardware, software and communications.
- May involve some COTS (Commercial Off-the-Shelf) systems procurement.
- Lack of communication across implementation teams.
- Bureaucratic and slow mechanism for proposing system changes means that the development schedule may be extended because of the need for rework.

System integration

- The process of putting hardware, software and people together to make a system.
- Should be tackled incrementally so that sub-systems are integrated one at a time.
- Interface problems between sub-systems are usually found at this stage.
- May be problems with uncoordinated deliveries of system components.

System installation

- After completion, the system has to be installed in the customer's environment
  - Environmental assumptions may be incorrect;
  - May be human resistance to the introduction of a new system;
  - System may have to coexist with alternative systems for some time;
  - May be physical installation problems (e.g. cabling problems);
  - Operator training has to be identified.

System evolution

- Large systems have a long lifetime. They must evolve to meet changing requirements.
- Evolution is inherently costly
  - Changes must be analysed from a technical and business perspective;
  - Sub-systems interact so unanticipated problems can arise;
  - There is rarely a rationale for original design decisions;
  - System structure is corrupted as changes are made to it.
- Existing systems which must be maintained are sometimes called legacy systems.

System decommissioning

- Taking the system out of service after its useful lifetime.
- May require removal of materials (e.g. dangerous chemicals) which pollute the environment
  - Should be planned for in the system design by encapsulation.
- May require data to be restructured and converted to be used in some other system.

Organisations/people/systems

- Socio-technical systems are organisational systems intended to help deliver some organisational or business goal.
- If you do not understand the organisational environment where a system is used, the system is less likely to meet the real needs of the business and its users.

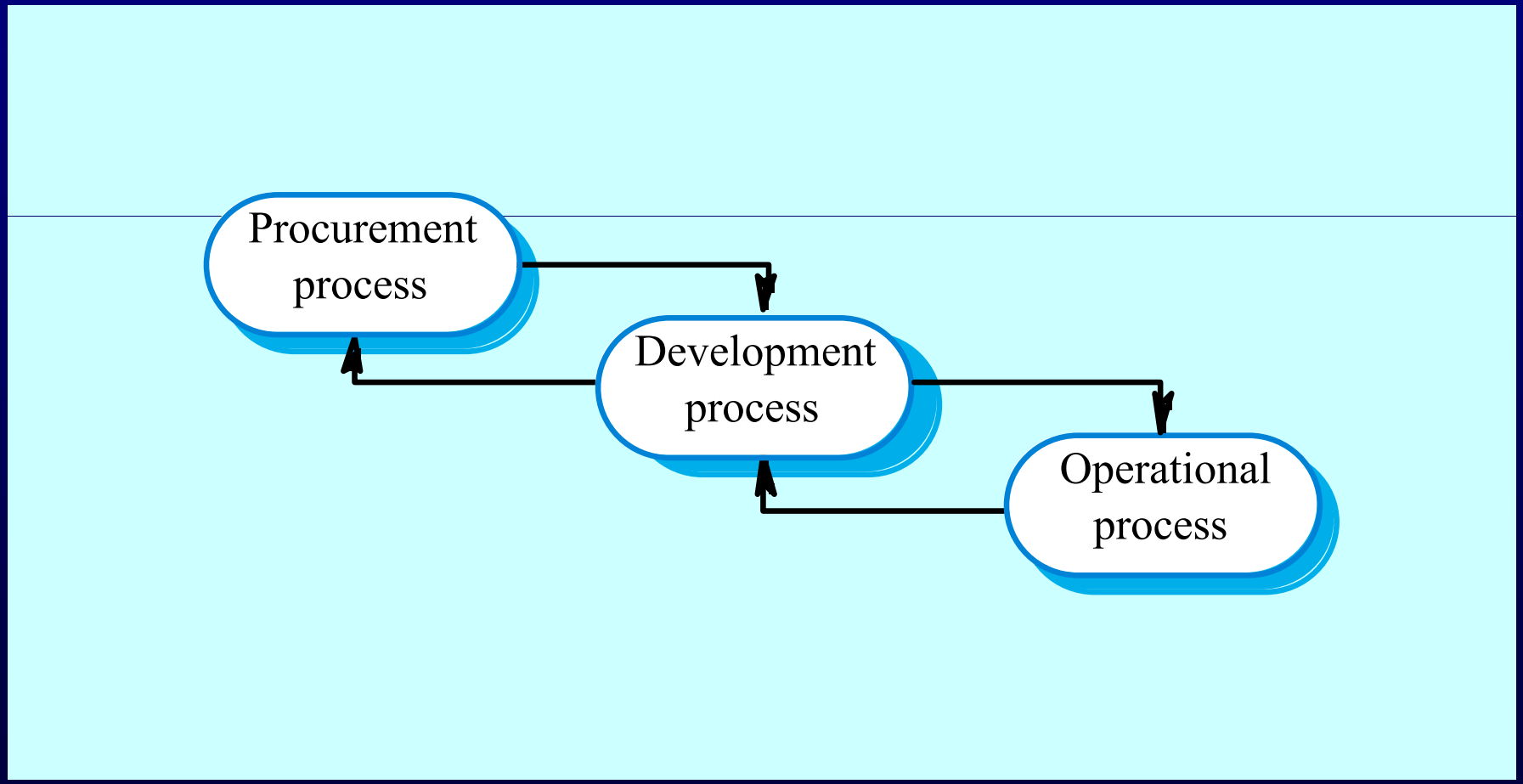
Human and organisational factors

- *Process changes*
  - Does the system require changes to the work processes in the environment?
- *Job changes*
  - Does the system de-skill the users in an environment or cause them to change the way they work?
- *Organisational changes*
  - Does the system change the political power structure in an organisation?

Organisational processes

- The processes of systems engineering overlap and interact with organisational procurement processes.
- Operational processes are the processes involved in using the system for its intended purpose. For new systems, these have to be defined as part of the system design.
- Operational processes should be designed to be flexible and should not force operations to be done in a particular way. It is important that human operators can use their initiative if problems arise.

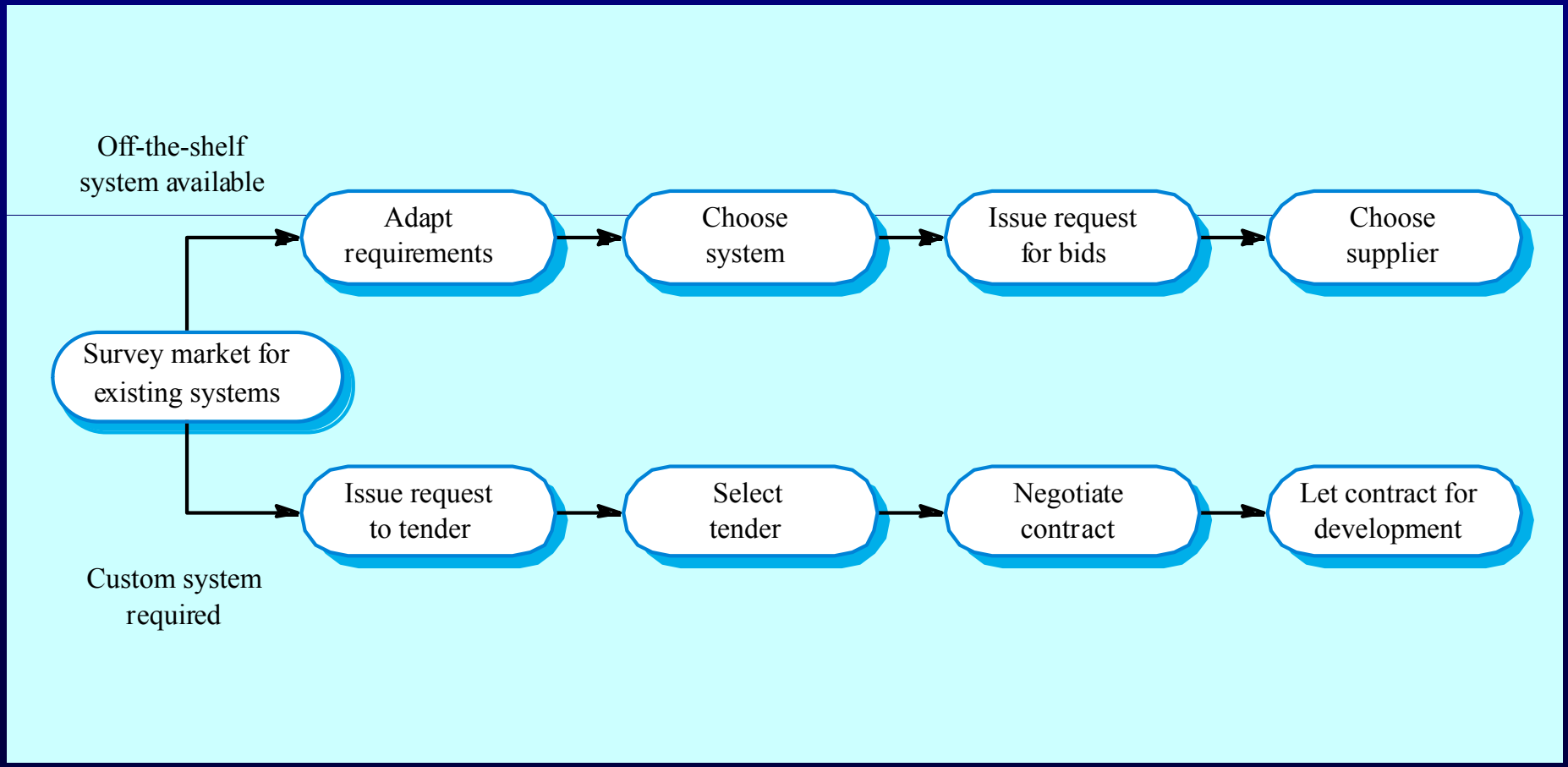
Procurement/development processes



System procurement

- Acquiring a system for an organization to meet some need
- Some system specification and architectural design is usually necessary before procurement
  - You need a specification to let a contract for system development
  - The specification may allow you to buy a commercial off-the-shelf (COTS) system. Almost always cheaper than developing a system from scratch
- Large complex systems usually consist of a mix of off the shelf and specially designed components. The procurement processes for these different types of component are usually different.

The system procurement process



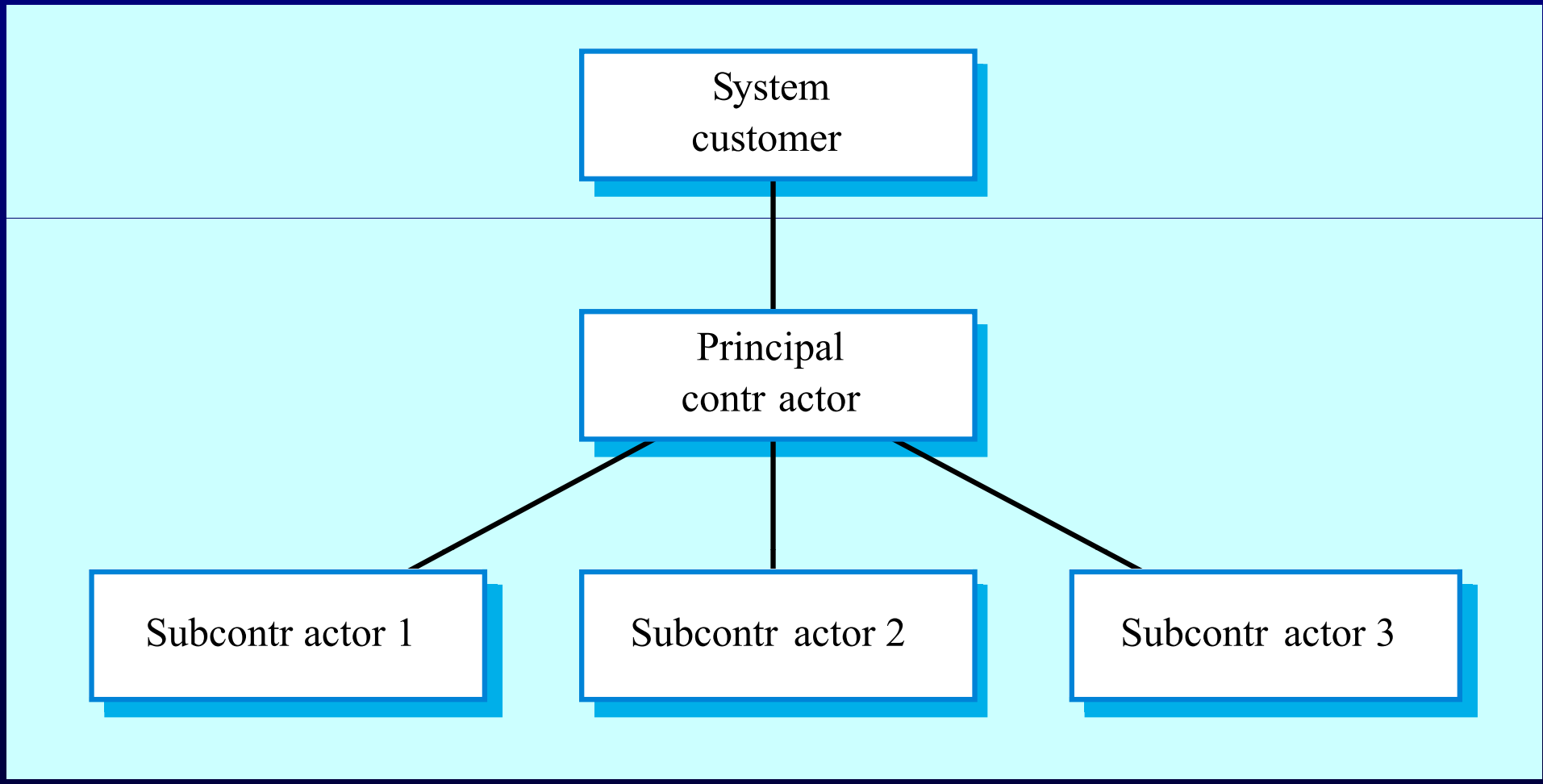
Procurement issues

- Requirements may have to be modified to match the capabilities of off-the-shelf components.
- The requirements specification may be part of the contract for the development of the system.
- There is usually a contract negotiation period to agree changes after the contractor to build a system has been selected.

Contractors and sub-contractors

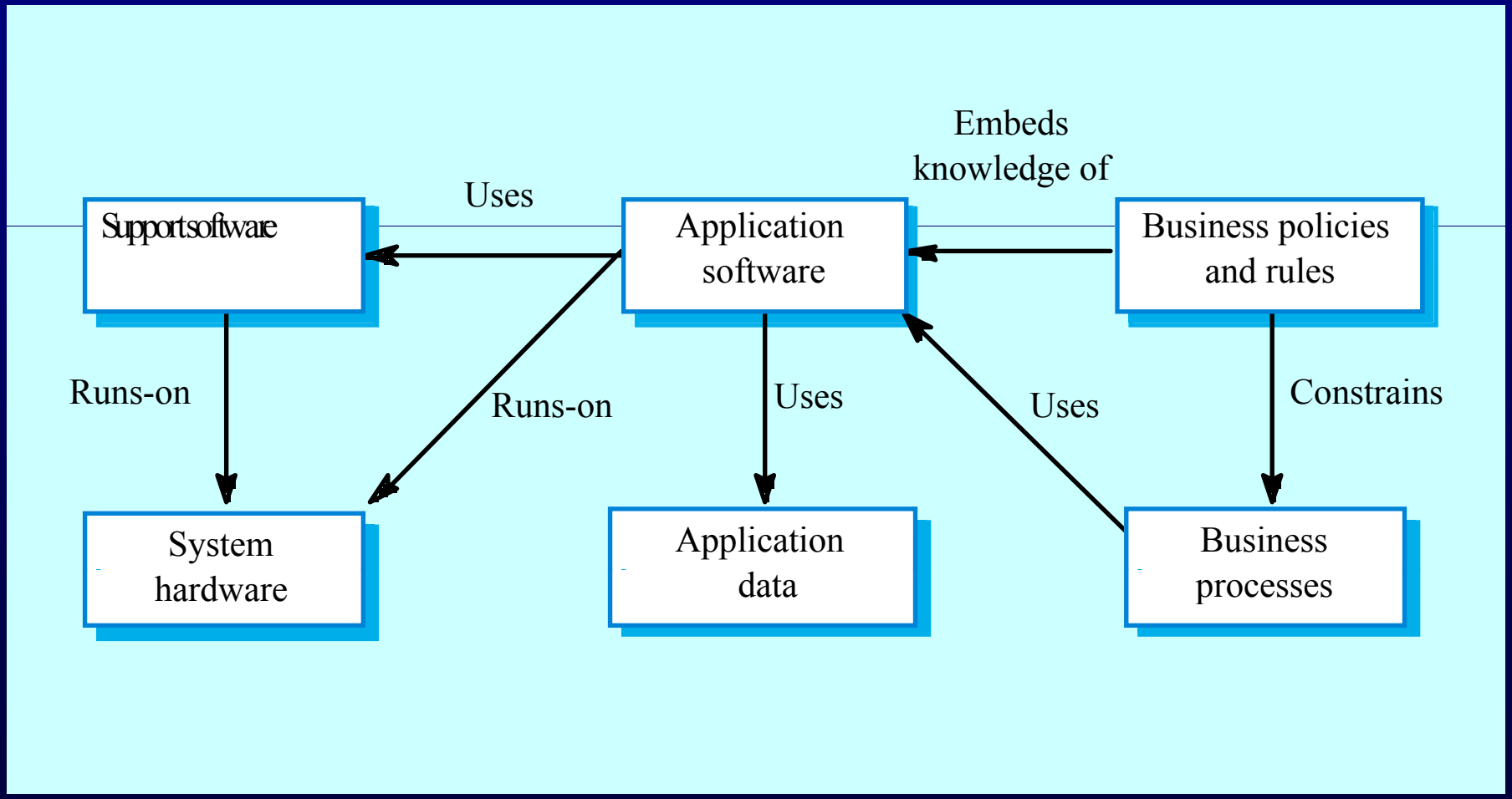
- The procurement of large hardware/software systems is usually based around some principal contractor.
- Sub-contracts are issued to other suppliers to supply parts of the system.
- Customer liases with the principal contractor and does not deal directly with sub-contractors.

Contractor/Sub-contractor model



Legacy systems

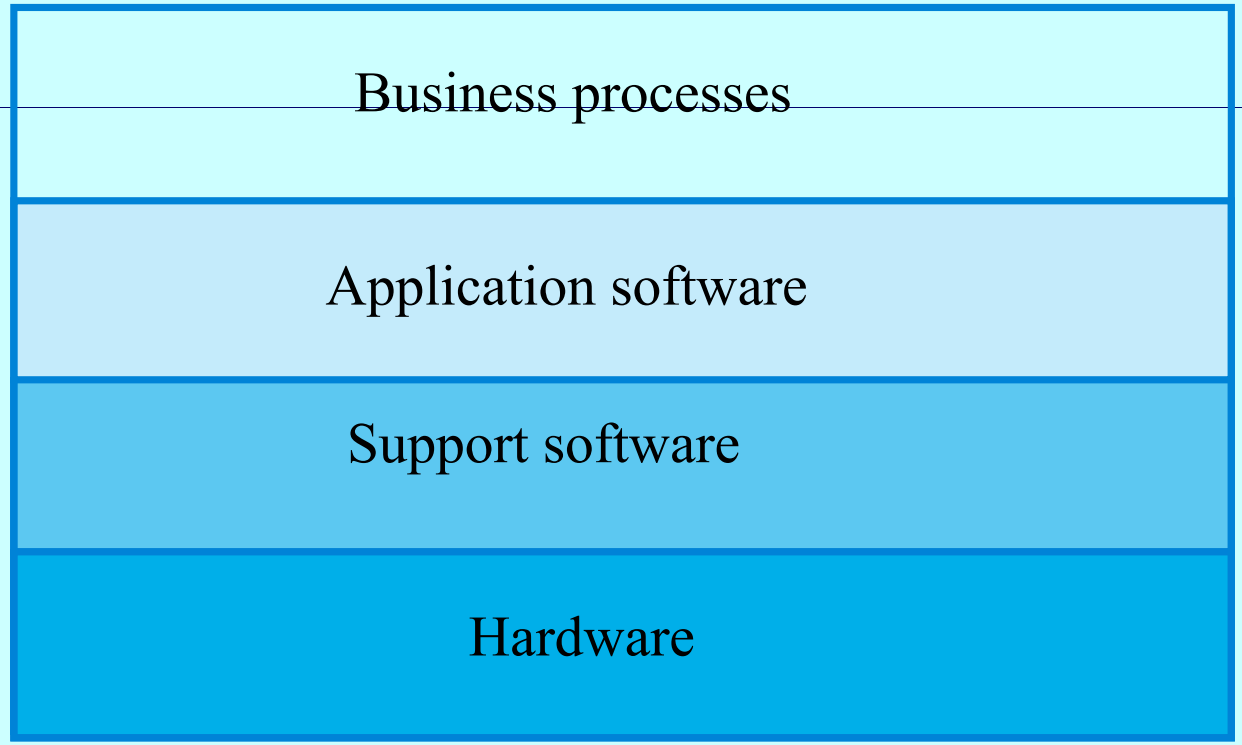
- Socio-technical systems that have been developed using old or obsolete technology.
- Crucial to the operation of a business and it is often too risky to discard these systems
  - Bank customer accounting system;
  - Aircraft maintenance system.
- Legacy systems constrain new business processes and consume a high proportion of company budgets.


Embeds
knowledge of
Uses
Support software
Application
software
Business policies
and rules
Runs-on
Runs-on
Uses
Uses
Constrains
System
hardware
Application
data
Business
processes

Legacy system components

- Hardware - may be obsolete mainframe hardware.
- Support software - may rely on support software from suppliers who are no longer in business.
- Application software - may be written in obsolete programming languages.
- Application data - often incomplete and inconsistent.
- Business processes - may be constrained by software structure and functionality.
- Business policies and rules - may be implicit and embedded in the system software.

Socio-technical system



Key points

- Socio-technical systems include computer hardware, software and people and are designed to meet some business goal.
- Emergent properties are properties that are characteristic of the system as a whole and not its component parts.
- The systems engineering process includes specification, design, development, integration and testing. System integration is particularly critical.

Key points

- Human and organisational factors have a significant effect on the operation of socio-technical systems.
- There are complex interactions between the processes of system procurement, development and operation.
- A legacy system is an old system that continues to provide essential services.
- Legacy systems include business processes, application software, support software and system hardware.

Critical Systems

Objectives

- To explain what is meant by a critical system where system failure can have severe human or economic consequence.
- To explain four dimensions of dependability - availability, reliability, safety and security.
- To explain that, to achieve dependability, you need to avoid mistakes, detect and remove errors and limit damage caused by failure.

Topics covered

- A simple safety-critical system
- System dependability
- Availability and reliability
- Safety
- Security

Critical Systems

- Safety-critical systems
  - Failure results in loss of life, injury or damage to the environment;
  - Chemical plant protection system;
- Mission-critical systems
  - Failure results in failure of some goal-directed activity;
  - Spacecraft navigation system;
- Business-critical systems
  - Failure results in high economic losses;
  - Customer accounting system in a bank;

System dependability

- For critical systems, it is usually the case that the most important system property is the dependability of the system.
- The dependability of a system reflects the user's degree of trust in that system. It reflects the extent of the user's confidence that it will operate as users expect and that it will not 'fail' in normal use.
- Usefulness and trustworthiness are not the same thing. A system does not have to be trusted to be useful.

Importance of dependability

- Systems that are not dependable and are unreliable, unsafe or insecure may be rejected by their users.
- The costs of system failure may be very high.
- Undependable systems may cause information loss with a high consequent recovery cost.

Development methods for critical systems

- The costs of critical system failure are so high that development methods may be used that are not cost-effective for other types of system.
- Examples of development methods
  - Formal methods of software development
  - Static analysis
  - External quality assurance

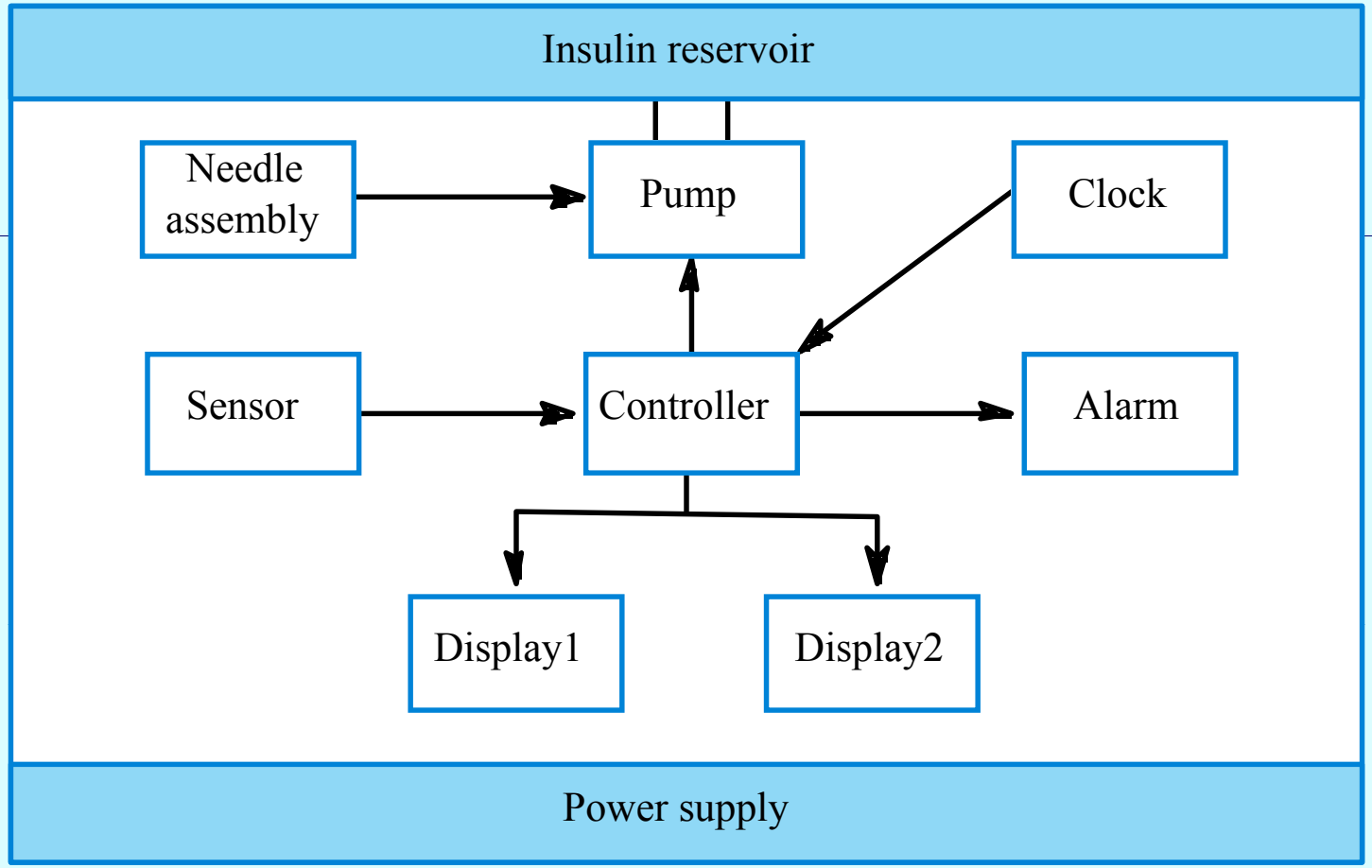
Socio-technical critical systems

- Hardware failure
  - Hardware fails because of design and manufacturing errors or because components have reached the end of their natural life.
- Software failure
  - Software fails due to errors in its specification, design or implementation.
- Operational failure
  - Human operators make mistakes. Now perhaps the largest single cause of system failures.

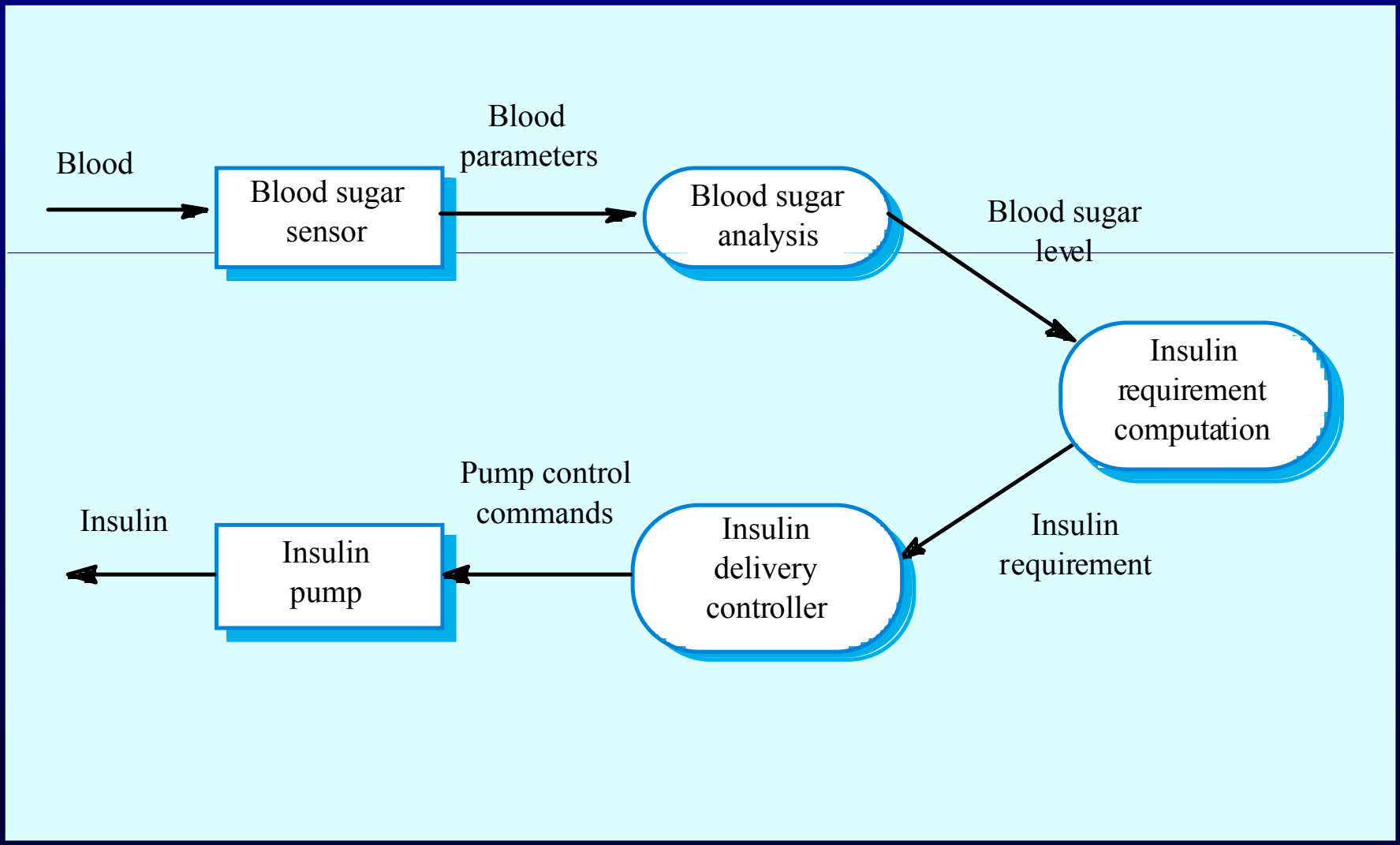
A software-controlled insulin pump

- Used by diabetics to simulate the function of the pancreas which manufactures insulin, an essential hormone that metabolises blood glucose.
- Measures blood glucose (sugar) using a micro-sensor and computes the insulin dose required to metabolise the glucose.

Insulin pump organisation



Insulin pump data-flow



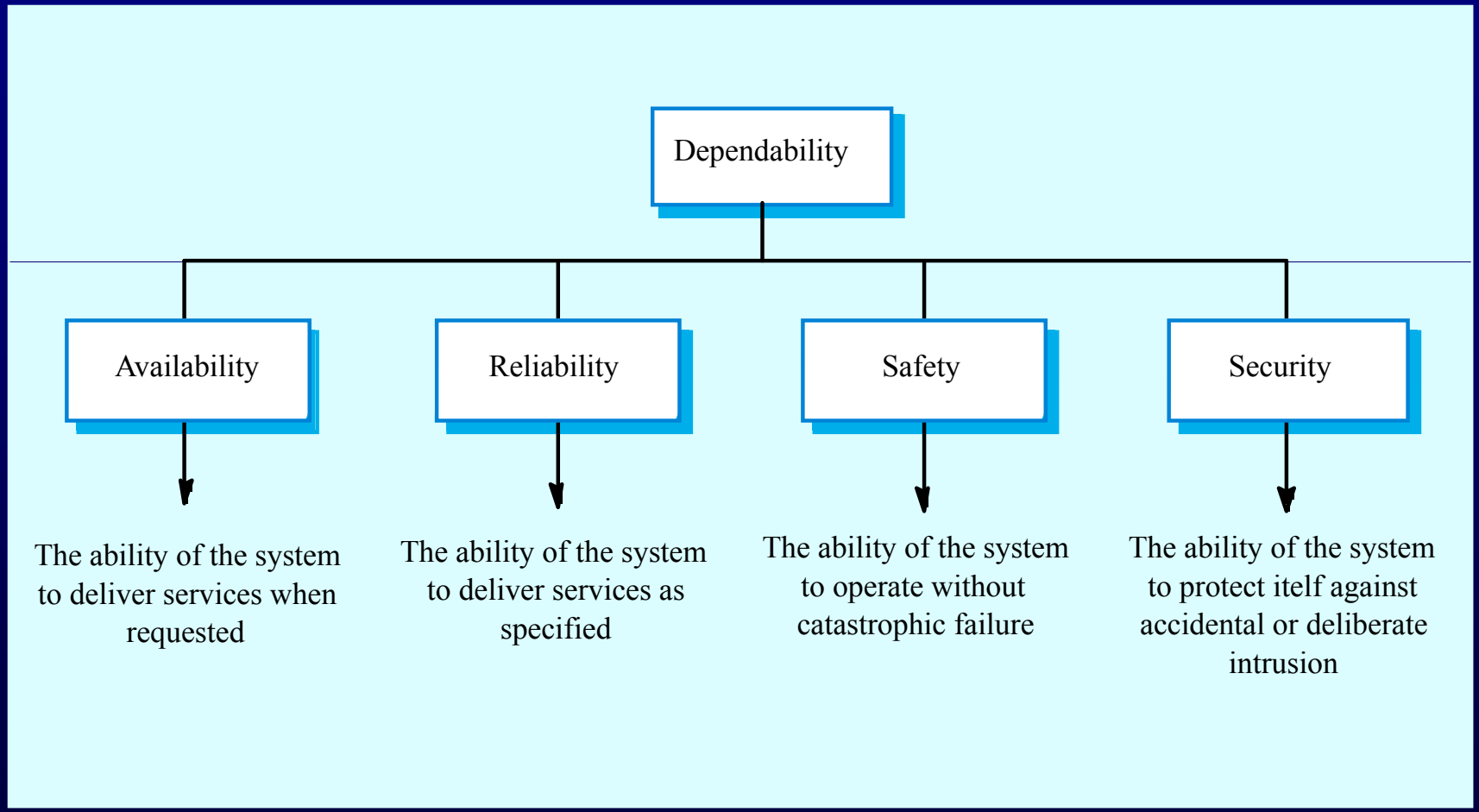
Dependability requirements

- The system shall be available to deliver insulin when required to do so.
- The system shall perform reliability and deliver the correct amount of insulin to counteract the current level of blood sugar.
- The essential safety requirement is that excessive doses of insulin should never be delivered as this is potentially life threatening.

Dependability

- The dependability of a system equates to its trustworthiness.
- A dependable system is a system that is trusted by its users.
- Principal dimensions of dependability are:
  - Availability;
  - Reliability;
  - Safety;
  - Security

Dimensions of dependability



Other dependability properties

- Repairability
  - Reflects the extent to which the system can be repaired in the event of a failure
- Maintainability
  - Reflects the extent to which the system can be adapted to new requirements;
- Survivability
  - Reflects the extent to which the system can deliver services whilst under hostile attack;
- Error tolerance
  - Reflects the extent to which user input errors can be avoided and tolerated.

Maintainability

- A system attribute that is concerned with the ease of repairing the system after a failure has been discovered or changing the system to include new features
- Very important for critical systems as faults are often introduced into a system because of maintenance problems
- Maintainability is distinct from other dimensions of dependability because it is a static and not a dynamic system attribute. I do not cover it in this course.

Survivability

- The ability of a system to continue to deliver its services to users in the face of deliberate or accidental attack
- This is an increasingly important attribute for distributed systems whose security can be compromised
- Survivability subsumes the notion of resilience - the ability of a system to continue in operation in spite of component failures

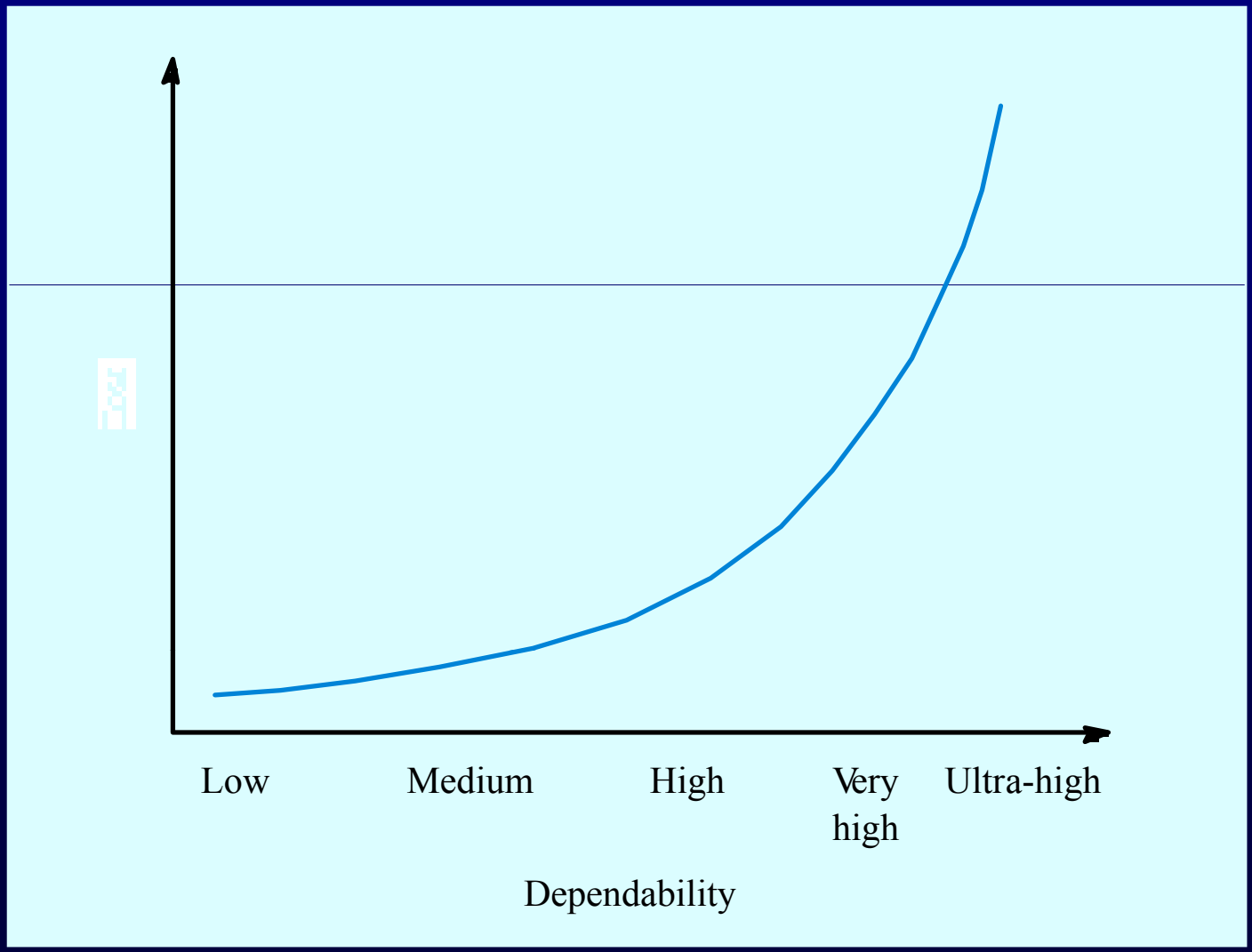
Dependability vs performance

- Untrustworthy systems may be rejected by their users
- System failure costs may be very high
- It is very difficult to tune systems to make them more dependable
- It may be possible to compensate for poor performance
- Untrustworthy systems may cause loss of valuable information

Dependability costs

- Dependability costs tend to increase exponentially as increasing levels of dependability are required
- There are two reasons for this
  - The use of more expensive development techniques and hardware that are required to achieve the higher levels of dependability
  - The increased testing and system validation that is required to convince the system client that the required levels of dependability have been achieved

Costs of increasing dependability



Dependability economics

- Because of very high costs of dependability achievement, it may be more cost effective to accept untrustworthy systems and pay for failure costs
- However, this depends on social and political factors. A reputation for products that can't be trusted may lose future business
- Depends on system type - for business systems in particular, modest levels of dependability may be adequate

Availability and reliability

- Reliability
  - The probability of failure-free system operation over a specified time in a given environment for a given purpose
- Availability
  - The probability that a system, at a point in time, will be operational and able to deliver the requested services
- Both of these attributes can be expressed quantitatively

Availability and reliability

- It is sometimes possible to subsume system availability under system reliability
  - Obviously if a system is unavailable it is not delivering the specified system services
- However, it is possible to have systems with low reliability that must be available. So long as system failures can be repaired quickly and do not damage data, low reliability may not be a problem
- Availability takes repair time into account

Reliability terminology

Term	Description
System failure	An event that occurs at some point in time when the system does not deliver a service as expected by its users
System error	An erroneous system state that can lead to system behaviour that is unexpected by system users.
System fault	A characteristic of a software system that can lead to a system error. For example, failure to initialise a variable could lead to that variable having the wrong value when it is used.
Human error or mistake	Human behaviour that results in the introduction of faults into a system.

Faults and failures

- Failures are a usually a result of system errors that are derived from faults in the system
- However, faults do not necessarily result in system errors
  - The faulty system state may be transient and ‘corrected’ before an error arises
- Errors do not necessarily lead to system failures
  - The error can be corrected by built-in error detection and recovery
  - The failure can be protected against by built-in protection facilities. These may, for example, protect system resources from system errors

Perceptions of reliability

- The formal definition of reliability does not always reflect the user's perception of a system's reliability
  - The assumptions that are made about the environment where a system will be used may be incorrect
    - Usage of a system in an office environment is likely to be quite different from usage of the same system in a university environment
  - The consequences of system failures affects the perception of reliability
    - Unreliable windscreen wipers in a car may be irrelevant in a dry climate
    - Failures that have serious consequences (such as an engine breakdown in a car) are given greater weight by users than failures that are inconvenient

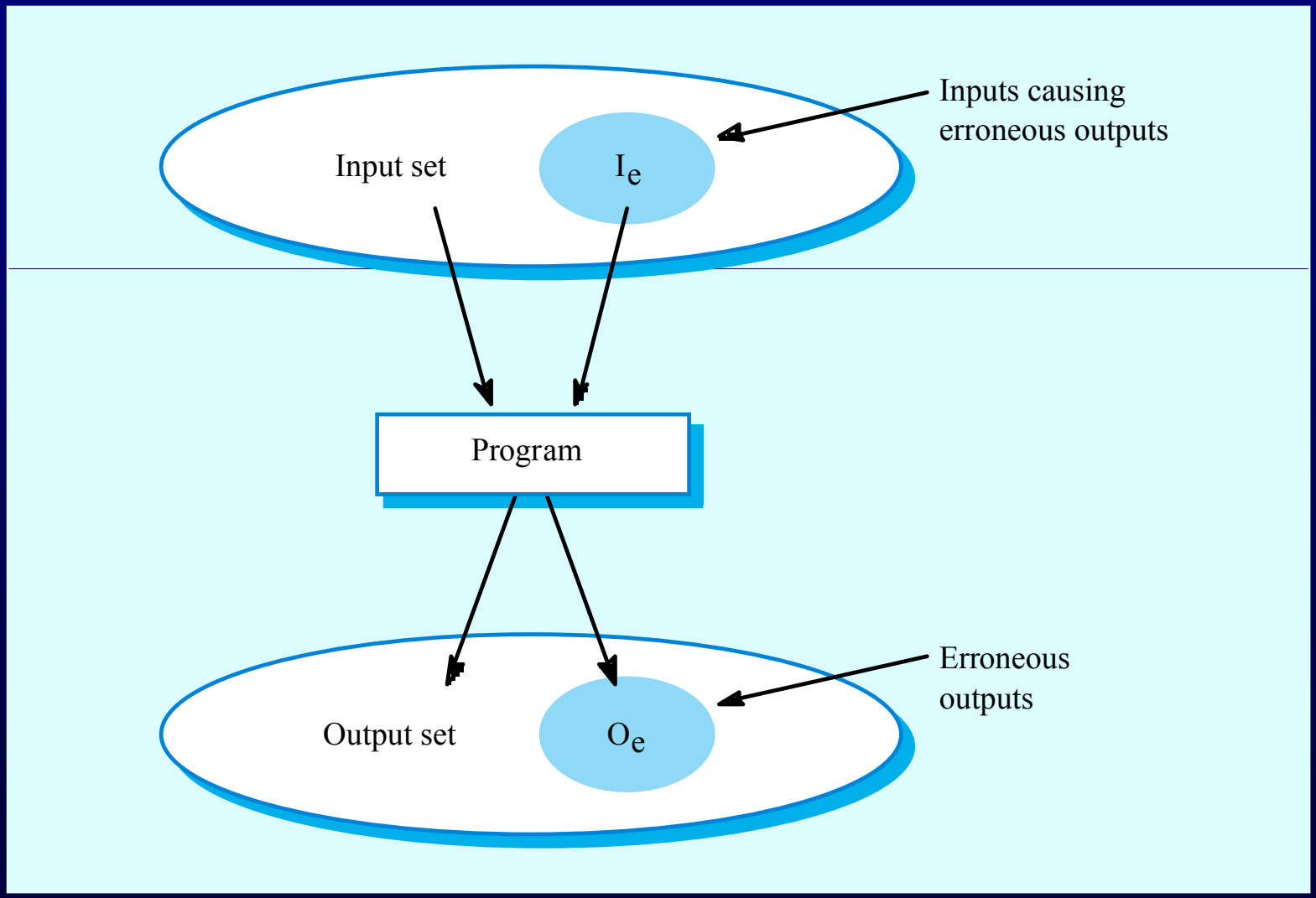
Reliability achievement

- Fault avoidance
  - Development technique are used that either minimise the possibility of mistakes or trap mistakes before they result in the introduction of system faults
- Fault detection and removal
  - Verification and validation techniques that increase the probability of detecting and correcting errors before the system goes into service are used
- Fault tolerance
  - Run-time techniques are used to ensure that system faults do not result in system errors and/or that system errors do not lead to system failures

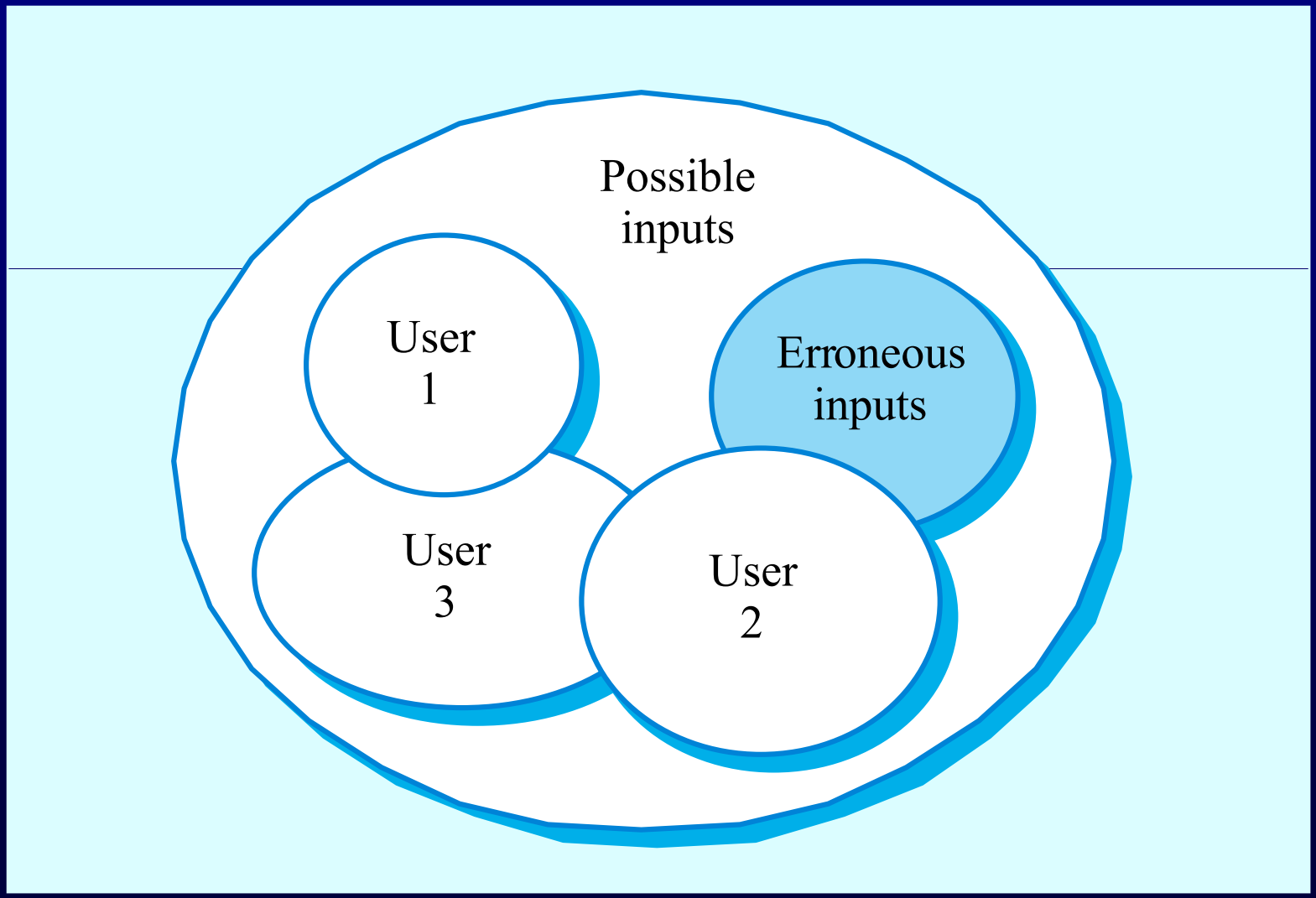
Reliability modelling

- You can model a system as an input-output mapping where some inputs will result in erroneous outputs
- The reliability of the system is the probability that a particular input will lie in the set of inputs that cause erroneous outputs
- Different people will use the system in different ways so this probability is not a static system attribute but depends on the system's environment

Input/output mapping



Reliability perception



Reliability improvement

- Removing X% of the faults in a system will not necessarily improve the reliability by X%. A study at IBM showed that removing 60% of product defects resulted in a 3% improvement in reliability
- Program defects may be in rarely executed sections of the code so may never be encountered by users. Removing these does not affect the perceived reliability
- A program with known faults may therefore still be seen as reliable by its users

Safety

- Safety is a property of a system that reflects the system's ability to operate, normally or abnormally, without danger of causing human injury or death and without damage to the system's environment
- It is increasingly important to consider software safety as more and more devices incorporate software-based control systems
- Safety requirements are exclusive requirements i.e. they exclude undesirable situations rather than specify required system services

Safety criticality

- Primary safety-critical systems
  - Embedded software systems whose failure can cause the associated hardware to fail and directly threaten people.
- Secondary safety-critical systems
  - Systems whose failure results in faults in other systems which can threaten people
- Discussion here focuses on primary safety-critical systems
  - Secondary safety-critical systems can only be considered on a one-off basis

Safety and reliability

- Safety and reliability are related but distinct
  - In general, reliability and availability are necessary but not sufficient conditions for system safety
- Reliability is concerned with conformance to a given specification and delivery of service
- Safety is concerned with ensuring system cannot cause damage irrespective of whether
  or not it conforms to its specification

Unsafe reliable systems

- Specification errors
  - If the system specification is incorrect then the system can behave as specified but still cause an accident
- Hardware failures generating spurious inputs
  - Hard to anticipate in the specification
- Context-sensitive commands i.e. issuing the right command at the wrong time
  - Often the result of operator error

Safety terminology

Term	Definition
Accident (or mishap)	An unplanned event or sequence of events which results in human death or injury, damage to property or to the environment. A computer-controlled machine injuring its operator is an example of an accident.
Hazard	A condition with the potential for causing or contributing to an accident. A failure of the sensor that detects an obstacle in front of a machine is an example of a hazard.
Damage	A measure of the loss resulting from a mishap. Damage can range from many people killed as a result of an accident to minor injury or property damage.
Hazard severity	An assessment of the worst possible damage that could result from a particular hazard. Hazard severity can range from catastrophic where many people are killed to minor where only minor damage results.
Hazard probability	The probability of the events occurring which create a hazard. Probability values tend to be arbitrary but range from *probable* (say 1/100 chance of a hazard occurring) to implausible (no conceivable situations are likely where the hazard could occur).
Risk	This is a measure of the probability that the system will cause an accident. The risk is assessed by considering the hazard probability, the hazard severity and the probability that a hazard will result in an accident.

Safety achievement

- Hazard avoidance
  - The system is designed so that some classes of hazard simply cannot arise.
- Hazard detection and removal
  - The system is designed so that hazards are detected and removed before they result in an accident
- Damage limitation
  - The system includes protection features that minimise the damage that may result from an accident

Normal accidents

- Accidents in complex systems rarely have a single cause as these systems are designed to be resilient to a single point of failure
  - Designing systems so that a single point of failure does not cause an accident is a fundamental principle of safe systems design
- Almost all accidents are a result of combinations of malfunctions
- It is probably the case that anticipating all problem combinations, especially, in software controlled systems is impossible so achieving complete safety is impossible

Security

- The security of a system is a system property that reflects the system's ability to protect itself from accidental or deliberate external attack
- Security is becoming increasingly important as systems are networked so that external access to the system through the Internet is possible
- Security is an essential pre-requisite for availability, reliability and safety

Fundamental security

- If a system is a networked system and is insecure then statements about its reliability and its safety are unreliable
- These statements depend on the executing system and the developed system being the same. However, intrusion can change the executing system and/or its data
- Therefore, the reliability and safety assurance is no longer valid

Security terminology

Term	Definition
Exposure	Possible loss or harm in a computing system. This can be loss or damage to data or can be a loss of time and effort if recovery is necessary after a security breach.
Vulnerability	A weakness in a computer-based system that may be exploited to cause loss or harm.
Attack	An exploitation of a system vulnerability. Generally, this is from outside the system and is a deliberate attempt to cause some damage.
Threats	Circumstances that have potential to cause loss or harm. You can think of these as a system vulnerability that is subjected to an attack.
Control	A protective measure that reduces a system vulnerability. Encryption would be an example of a control that reduced a vulnerability of a weak access control system.

Damage from insecurity

- Denial of service
  - The system is forced into a state where normal services are unavailable or where service provision is significantly degraded
- Corruption of programs or data
  - The programs or data in the system may be modified in an unauthorised way
- Disclosure of confidential information
  - Information that is managed by the system may be exposed to people who are not authorised to read or use that information

Security assurance

- Vulnerability avoidance
  - The system is designed so that vulnerabilities do not occur. For example, if there is no external network connection then external attack is impossible
- Attack detection and elimination
  - The system is designed so that attacks on vulnerabilities are detected and neutralised before they result in an exposure. For example, virus checkers find and remove viruses before they infect a system
- Exposure limitation
  - The system is designed so that the adverse consequences of a successful attack are minimised. For example, a backup policy allows damaged information to be restored

Key points

- A critical system is a system where failure can lead to high economic loss, physical damage or threats to life.
- The dependability in a system reflects the user's trust in that system
- The availability of a system is the probability that it will be available to deliver services when requested
- The reliability of a system is the probability that system services will be delivered as specified
- Reliability and availability are generally seen as necessary but not sufficient conditions for safety and security

Key points

- Reliability is related to the probability of an error occurring in operational use. A system with known faults may be reliable
- Safety is a system attribute that reflects the system's ability to operate without threatening people or the environment
- Security is a system attribute that reflects the system's ability to protect itself from external attack
- Dependability improvement requires a socio-technical approach to design where you consider the humans as well as the hardware and software

Software Processes

Objectives

- To introduce software process models
- To describe three generic process models and when they may be used
- To describe outline process models for requirements engineering, software development, testing and evolution
- To explain the Rational Unified Process model
- To introduce CASE technology to support software process activities

Topics covered

- Software process models
- Process iteration
- Process activities
- The Rational Unified Process
- Computer-aided software engineering

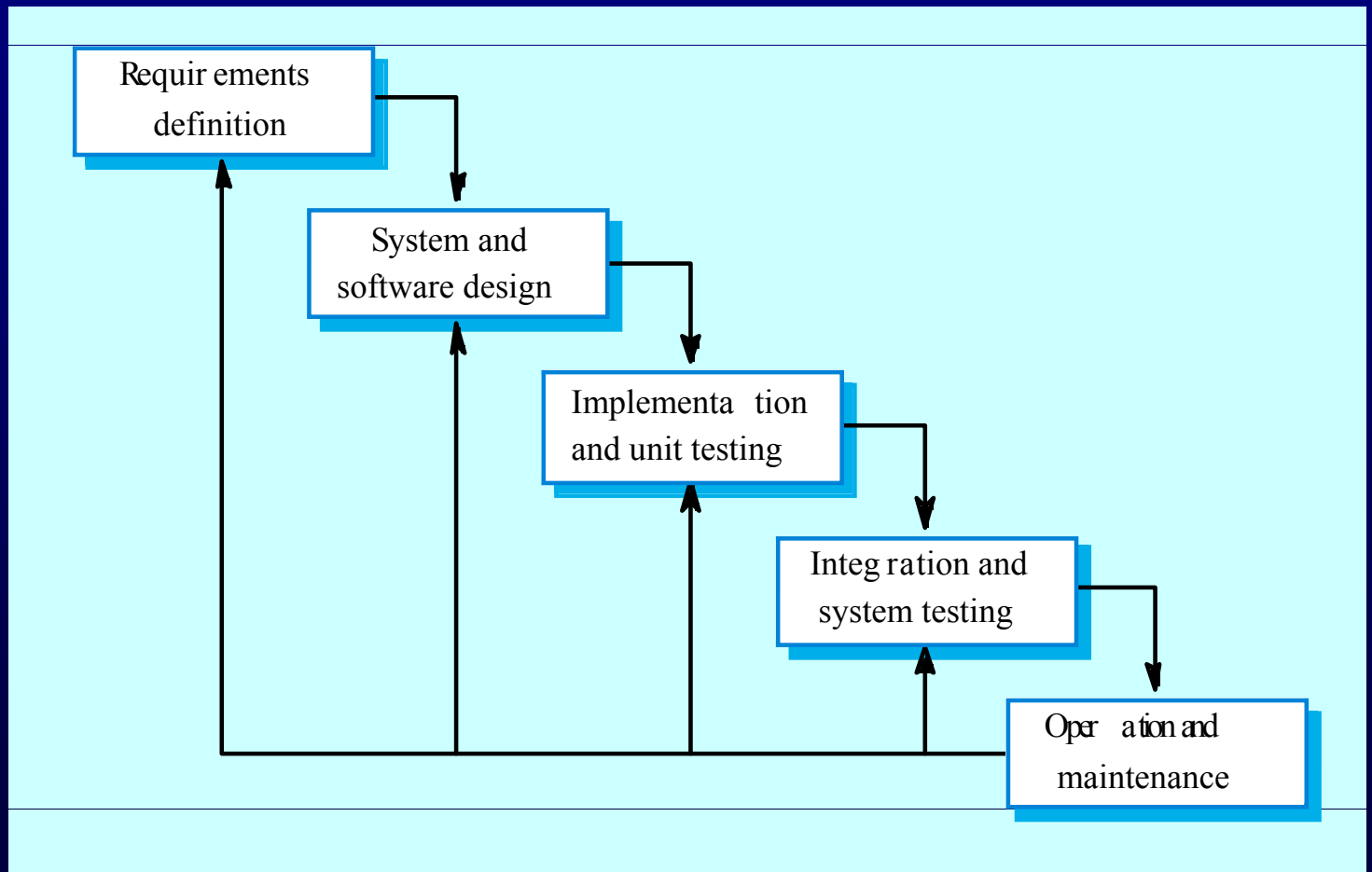
The software process

- A structured set of activities required to develop a software system
  - Specification;
  - Design;
  - Validation;
  - Evolution.
- A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

Generic software process models

- The waterfall model
  - Separate and distinct phases of specification and development.
- Evolutionary development
  - Specification, development and validation are interleaved.
- Component-based software engineering
  - The system is assembled from existing components.
- There are many variants of these models e.g. formal development where a waterfall-like process is used but the specification is a formal specification that is refined through several stages to an implementable design.

Waterfall model



Waterfall model phases

- Requirements analysis and definition
- System and software design
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance
- The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. One phase has to be complete before moving onto the next phase.

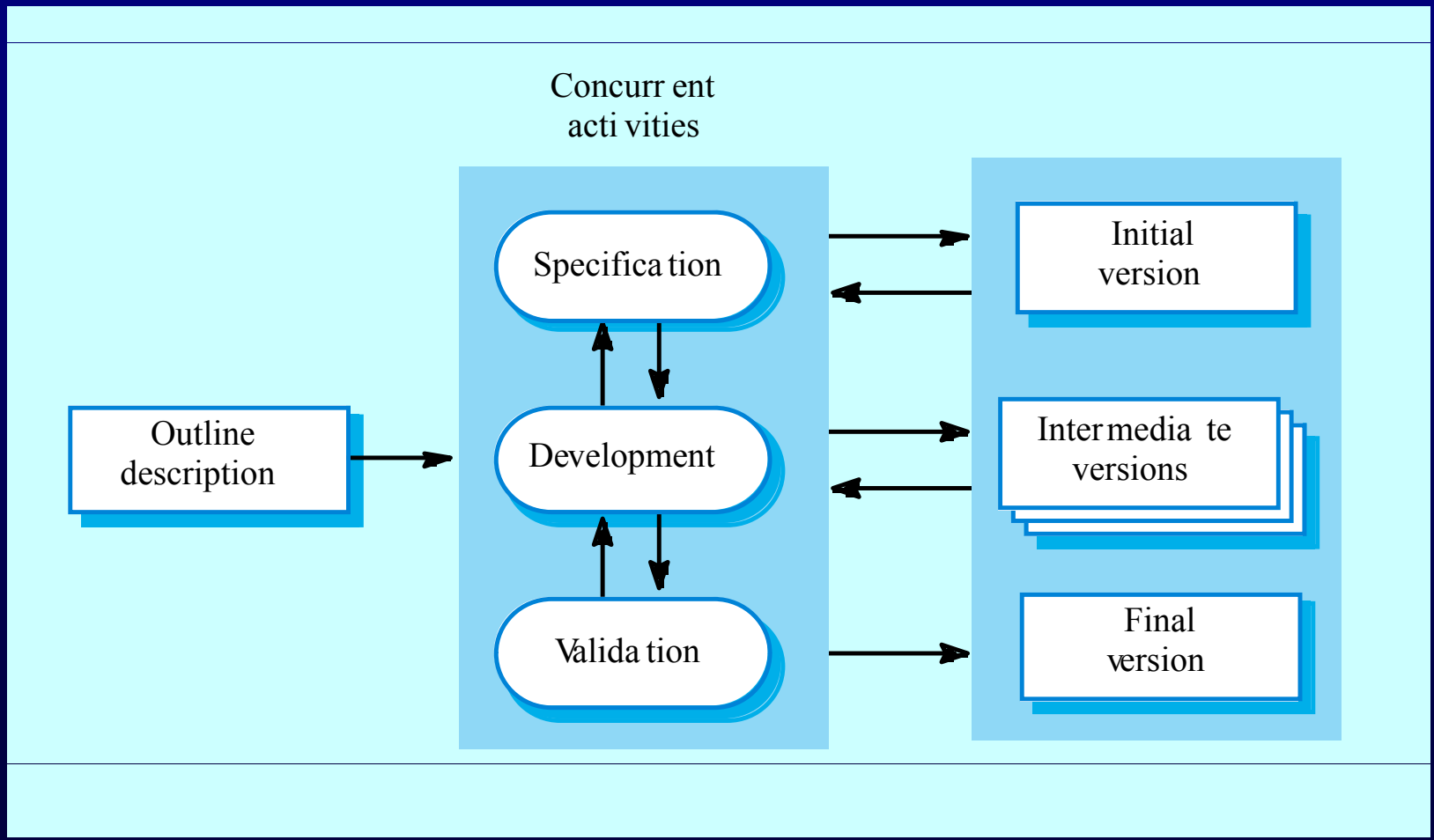
Waterfall model problems

- 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.

Evolutionary development

- Exploratory development
  - Objective is to work with customers and to evolve a final system from an initial outline specification. Should start with well-understood requirements and add new features as proposed by the customer.
- Throw-away prototyping
  - Objective is to understand the system requirements. Should start with poorly understood requirements to clarify what is really needed.

Evolutionary development



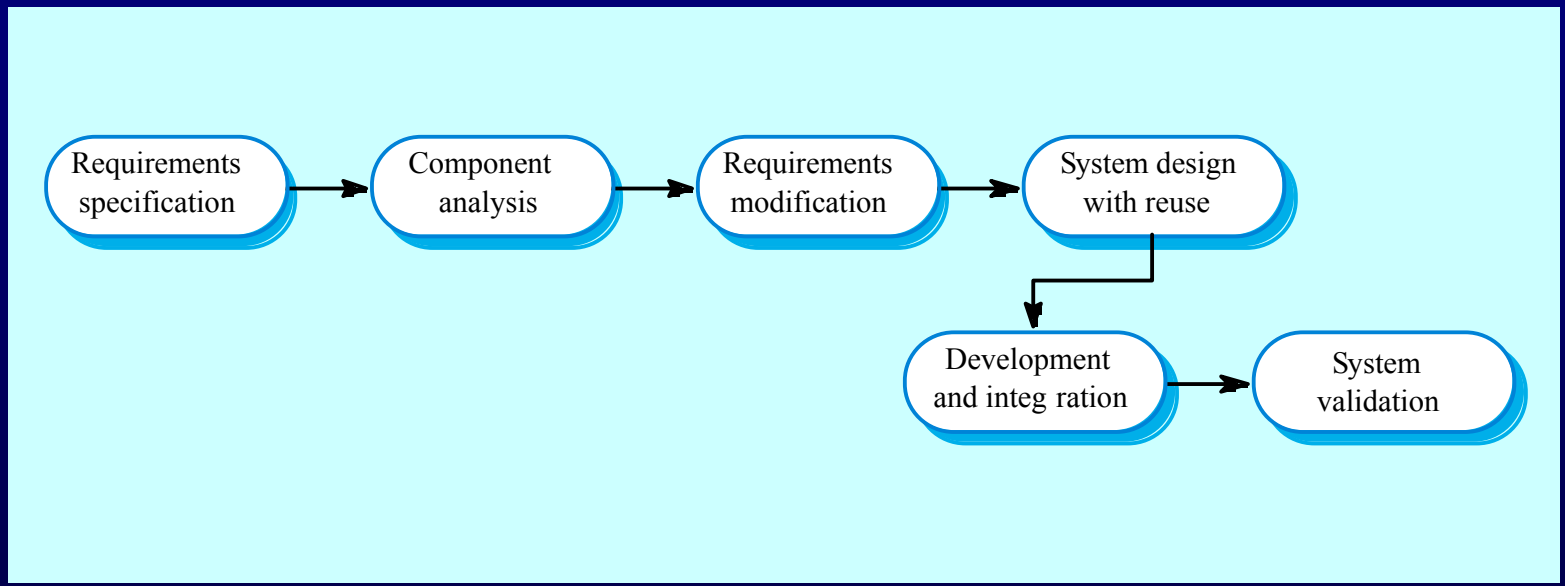
Evolutionary development

- Problems
  - Lack of process visibility;
  - Systems are often poorly structured;
  - Special skills (e.g. in languages for rapid prototyping) may be required.
- Applicability
  - For small or medium-size interactive systems;
  - For parts of large systems (e.g. the user interface);
  - For short-lifetime systems.

Component-based software engineering

- Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.
- Process stages
  - Component analysis;
  - Requirements modification;
  - System design with reuse;
  - Development and integration.
- This approach is becoming increasingly used as component standards have emerged.

Reuse-oriented development



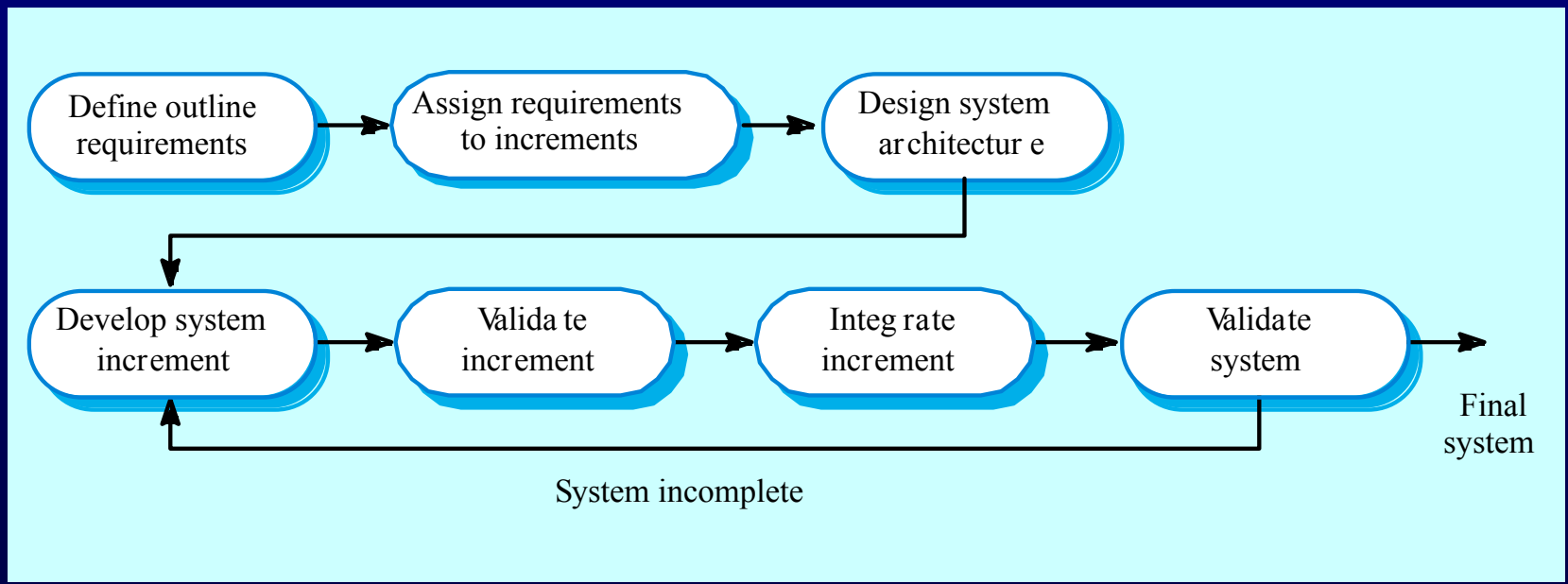
Process iteration

- System requirements ALWAYS evolve in the course of a project so process iteration where earlier stages are reworked is always part of the process for large systems.
- Iteration can be applied to any of the generic process models.
- Two (related) approaches
  - Incremental delivery;
  - Spiral development.

Incremental delivery

- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.
- User requirements are prioritised and the highest priority requirements are included in early increments.
- Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.

Incremental development



Incremental development advantages

- Customer value can be delivered with each increment so system functionality is available earlier.
- Early increments act as a prototype to help elicit requirements for later increments.
- Lower risk of overall project failure.
- The highest priority system services tend to receive the most testing.

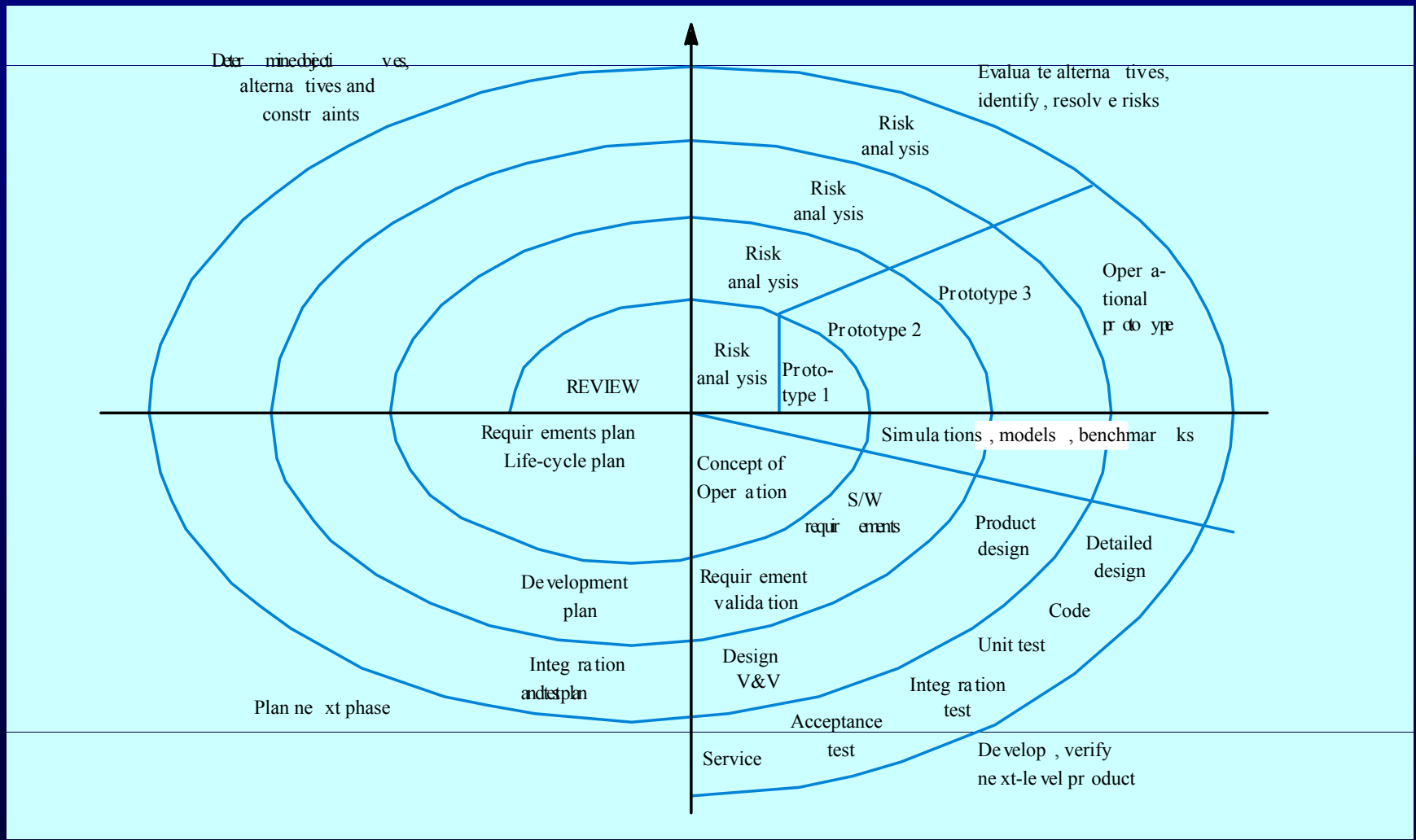
Extreme programming

- An approach to development based on the development and delivery of very small increments of functionality.
- Relies on constant code improvement, user involvement in the development team and pairwise programming.
- Covered in Chapter 17

Spiral development

- Process is represented as a spiral rather than as a sequence of activities with backtracking.
- Each loop in the spiral represents a phase in the process.
- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required.
- Risks are explicitly assessed and resolved throughout the process.

Spiral model of the software process



Spiral model sectors

- Objective setting
  - Specific objectives for the phase are identified.
- Risk assessment and reduction
  - Risks are assessed and activities put in place to reduce the key risks.
- Development and validation
  - A development model for the system is chosen which can be any of the generic models.
- Planning
  - The project is reviewed and the next phase of the spiral is planned.

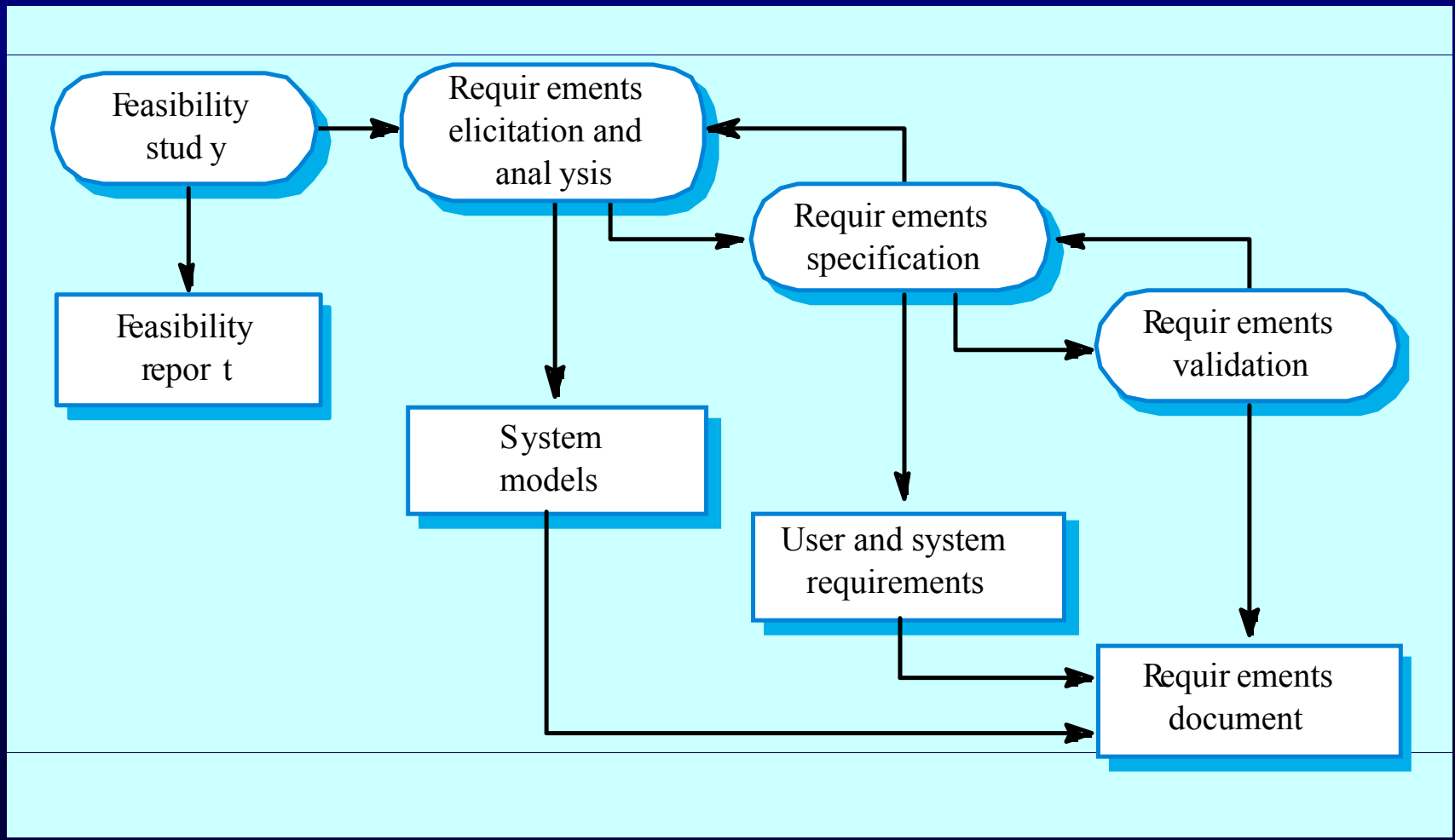
Process activities

- Software specification
- Software design and implementation
- Software validation
- Software evolution

Software specification

- The process of establishing what services are required and the constraints on the system's operation and development.
- Requirements engineering process
  - Feasibility study;
  - Requirements elicitation and analysis;
  - Requirements specification;
  - Requirements validation.

The requirements engineering process



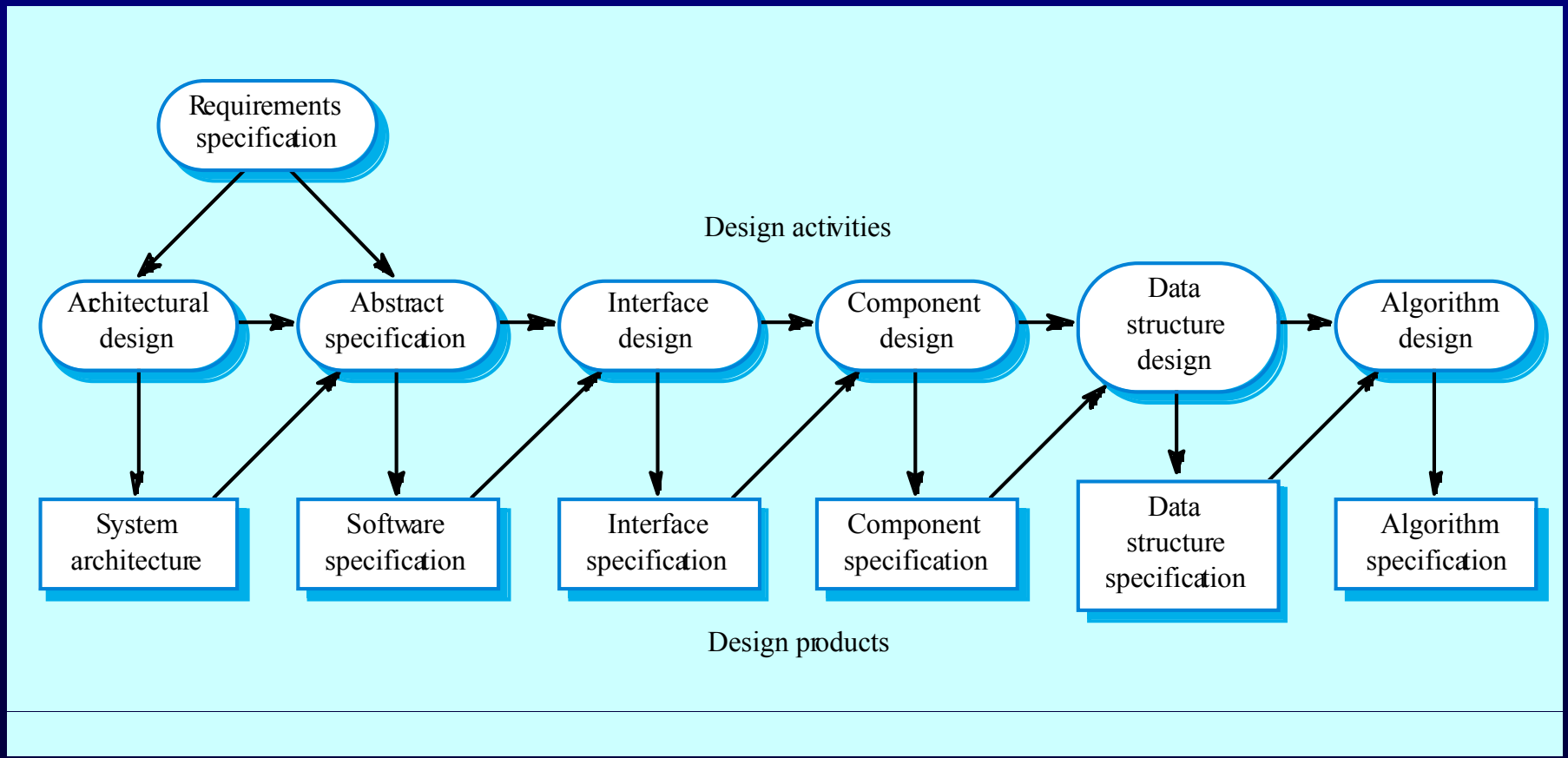
Software design and implementation

- 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.

Design process activities

- Architectural design
- Abstract specification
- Interface design
- Component design
- Data structure design
- Algorithm design

The software design process



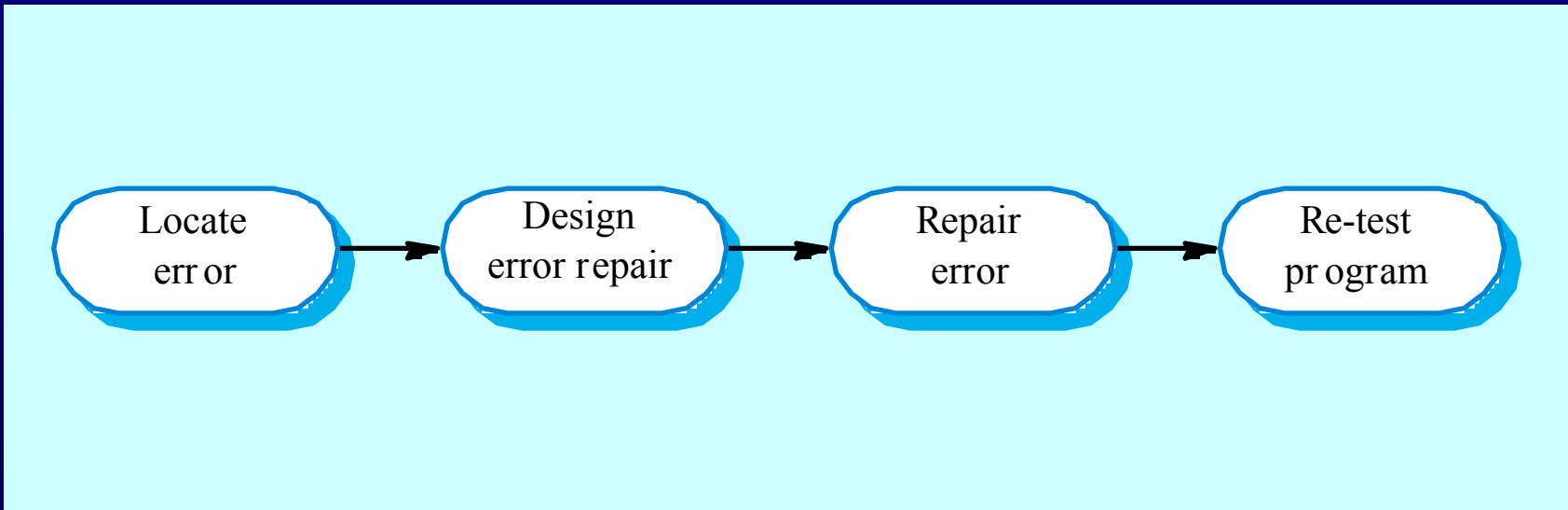
Structured methods

- Systematic approaches to developing a software design.
- The design is usually documented as a set of graphical models.
- Possible models
  - Object model;
  - Sequence model;
  - State transition model;
  - Structural model;
  - Data-flow model.

Programming and debugging

- Translating a design into a program and removing errors from that program.
- Programming is a personal activity - there is no generic programming process.
- Programmers carry out some program testing to discover faults in the program and remove these faults in the debugging process.

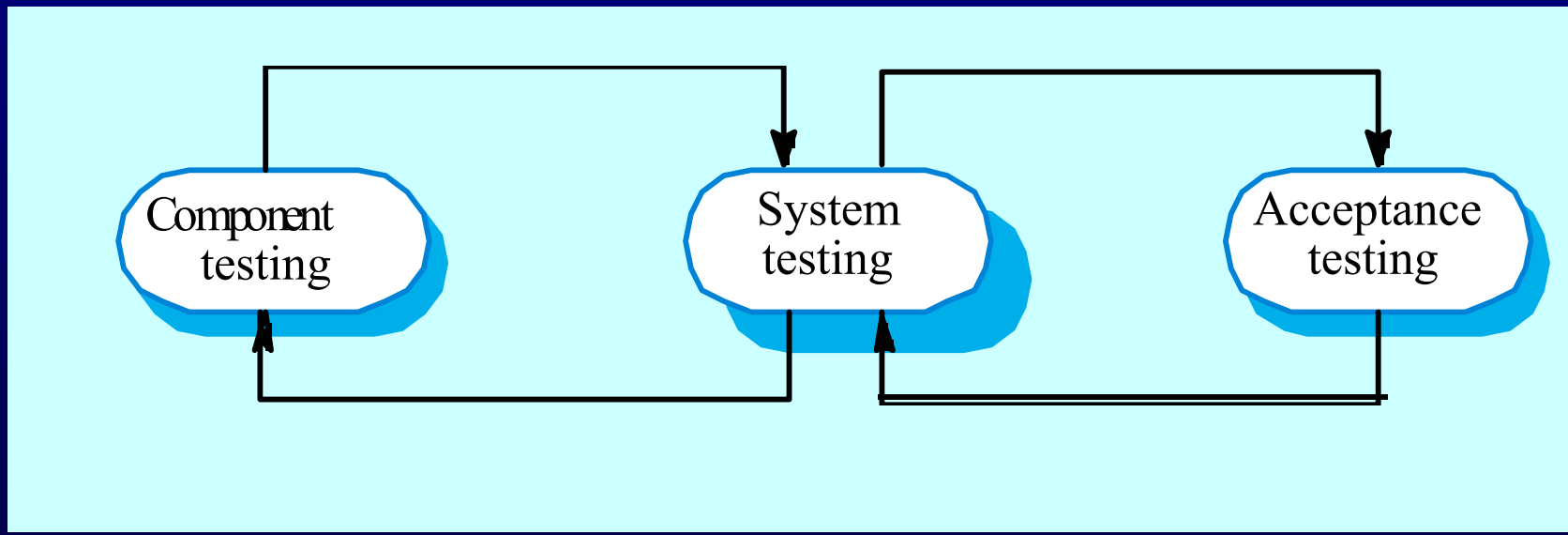
The debugging process



Software validation

- Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer.
- Involves checking and 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.

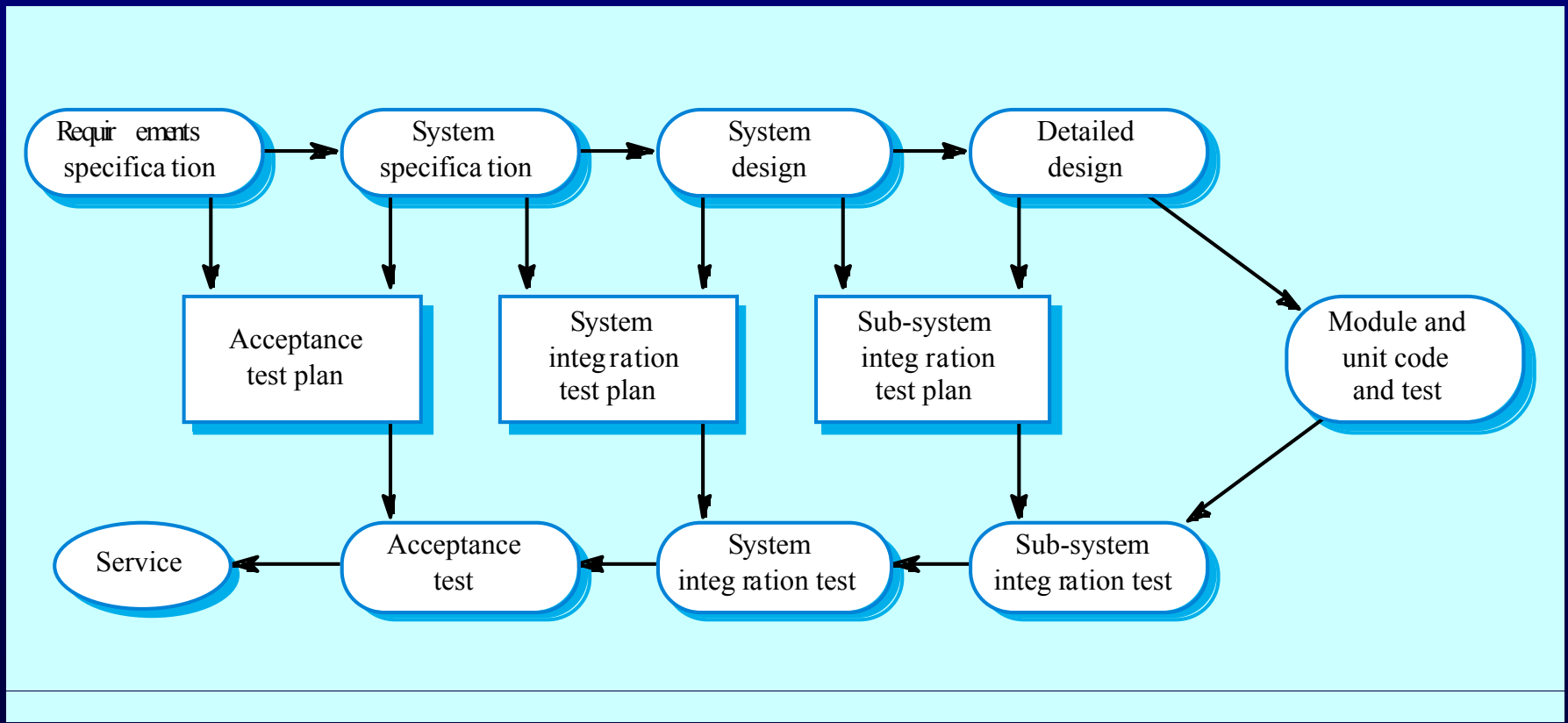
The testing process



Testing stages

- Component or unit 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 testing
  - Testing with customer data to check that the system meets the customer's needs.

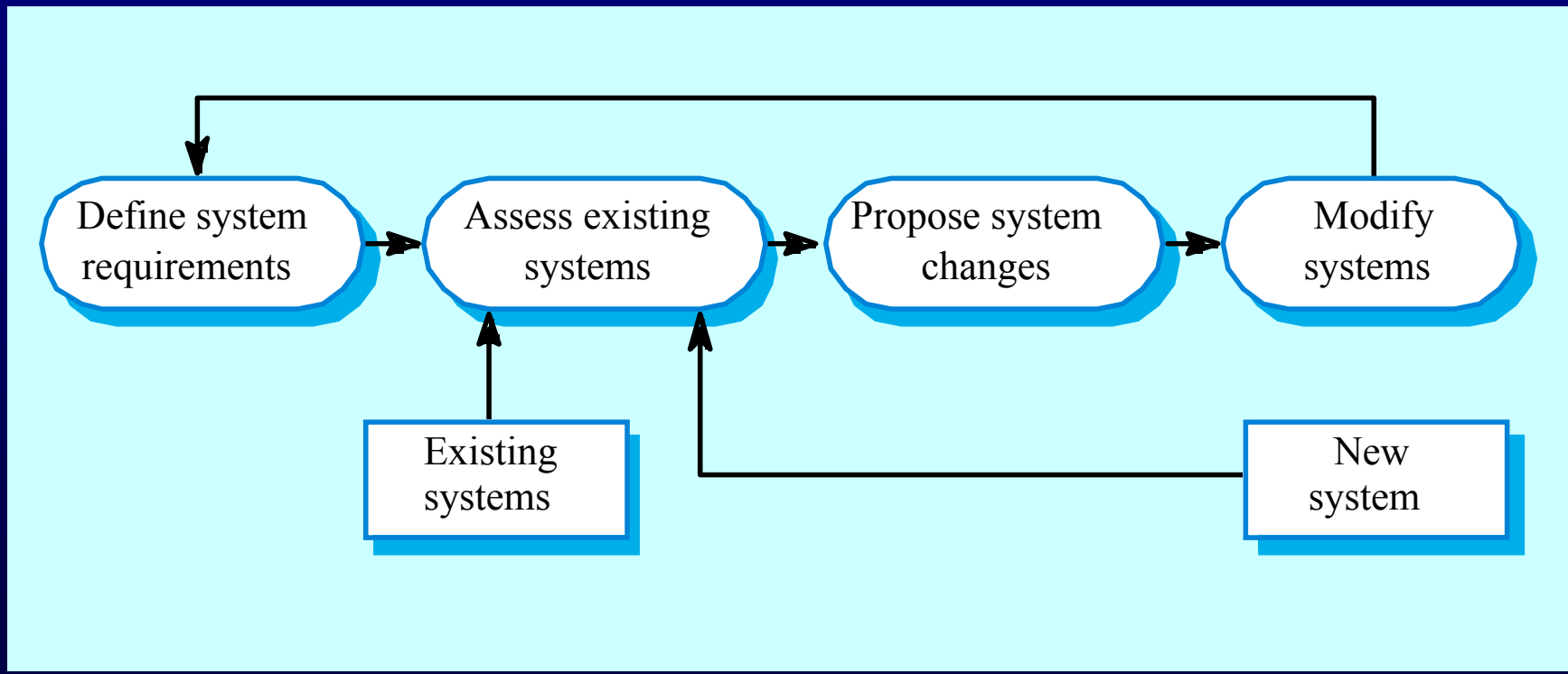
Testing phases



Software 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.

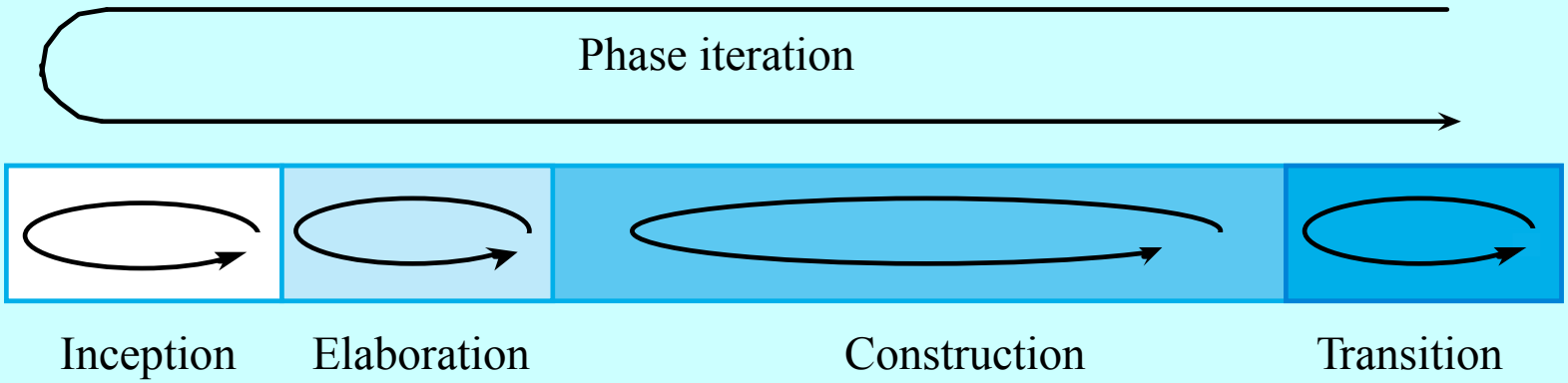
System evolution



The Rational Unified Process

- A modern process model derived from the work on the UML and associated process.
- Normally described from 3 perspectives
  - A dynamic perspective that shows phases over time;
  - A static perspective that shows process activities;
  - A practive perspective that suggests good practice.

RUP phase model



RUP phases

- Inception
  - Establish the business case for the system.
- Elaboration
  - Develop an understanding of the problem domain and the system architecture.
- Construction
  - System design, programming and testing.
- Transition
  - Deploy the system in its operating environment.

RUP good practice

- Develop software iteratively
- Manage requirements
- Use component-based architectures
- Visually model software
- Verify software quality
- Control changes to software

Static workflows

Workflow	Description
Business modelling	The business processes are modelled using business use cases.
Requirements	Actors who interact with the system are identified and use cases are developed to model the system requirements.
Analysis and design	A design model is created and documented using architectural models, component models, object models and sequence models.
Implementation	The components in the system are implemented and structured into implementation sub-systems. Automatic code generation from design models helps accelerate this process.
Test	Testing is an iterative process that is carried out in conjunction with implementation. System testing follows the completion of the implementation.
Deployment	A product release is created, distributed to users and installed in their workplace.
Configuration and change management	This supporting workflow managed changes to the system (see Chapter 29).
Project management	This supporting workflow manages the system development (see Chapter 5).
Environment	This workflow is concerned with making appropriate software tools available to the software development team.

Computer-aided software engineering

- Computer-aided software engineering (CASE) is software to support software development and evolution processes.
- Activity automation
  - Graphical editors for system model development;
  - Data dictionary to manage design entities;
  - Graphical UI builder for user interface construction;
  - Debuggers to support program fault finding;
  - Automated translators to generate new versions of a program.

Case technology

- Case technology has led to significant improvements in the software process. However, these are not the order of magnitude improvements that were once predicted
  - Software engineering requires creative thought - this is not readily automated;
  - Software engineering is a team activity and, for large projects, much time is spent in team interactions. CASE technology does not really support these.

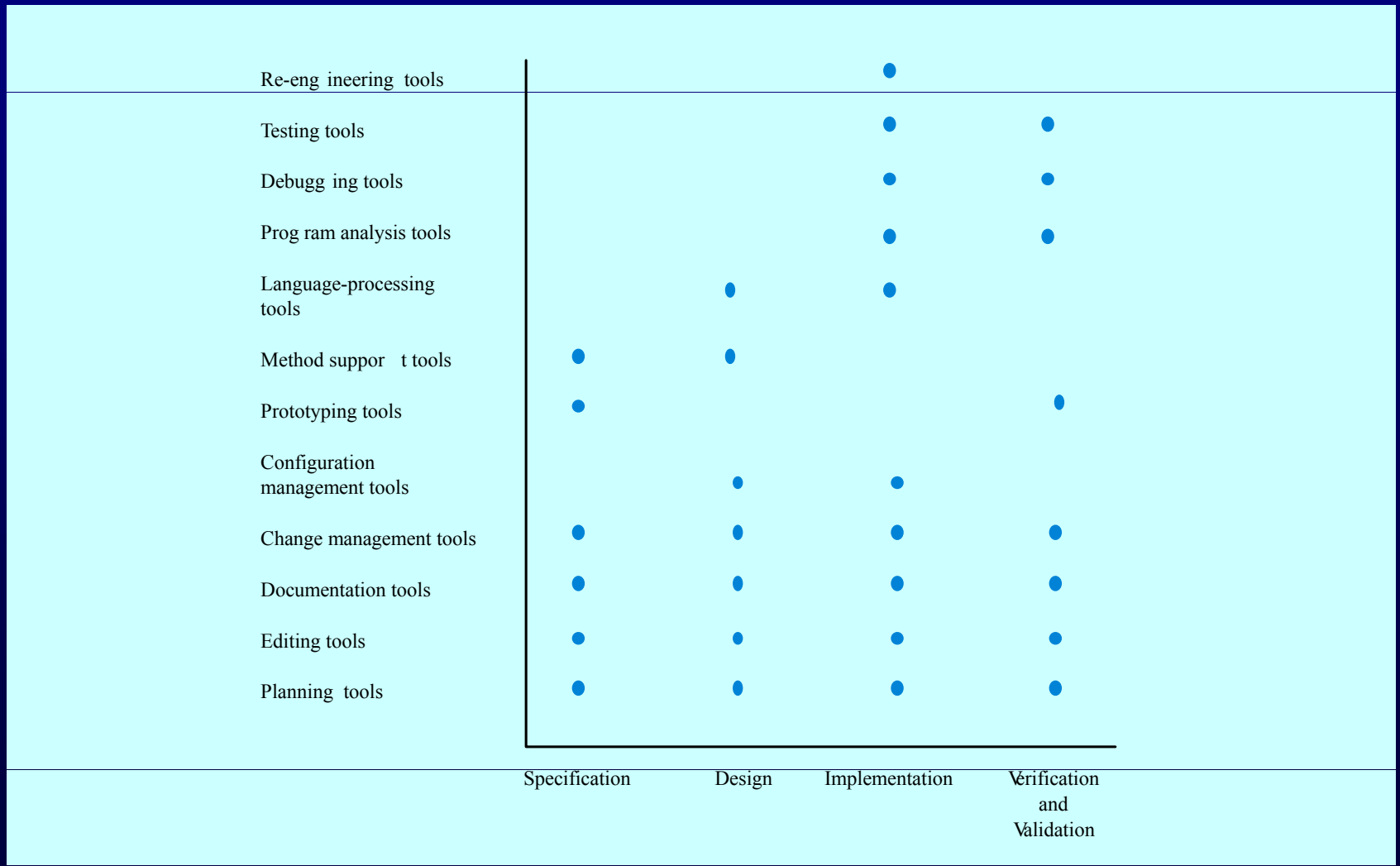
CASE classification

- Classification helps us understand the different types of CASE tools and their support for process activities.
- Functional perspective
  - Tools are classified according to their specific function.
- Process perspective
  - Tools are classified according to process activities that are supported.
- Integration perspective
  - Tools are classified according to their organisation into integrated units.

Functional tool classification

Tool type	Examples
Planning tools	PERT tools, estimation tools, spreadsheets
Editing tools	Text editors, diagram editors, word processors
Change management tools	Requirements traceability tools, change control systems
Configuration management tools	Version management systems, system building tools
Prototyping tools	Very high-level languages, user interface generators
Method-support tools	Design editors, data dictionaries, code generators
Language-processing tools	Compilers, interpreters
Program analysis tools	Cross reference generators, static analysers, dynamic analysers
Testing tools	Test data generators, file comparators
Debugging tools	Interactive debugging systems
Documentation tools	Page layout programs, image editors
Re-engineering tools	Cross-reference systems, program re-structuring systems

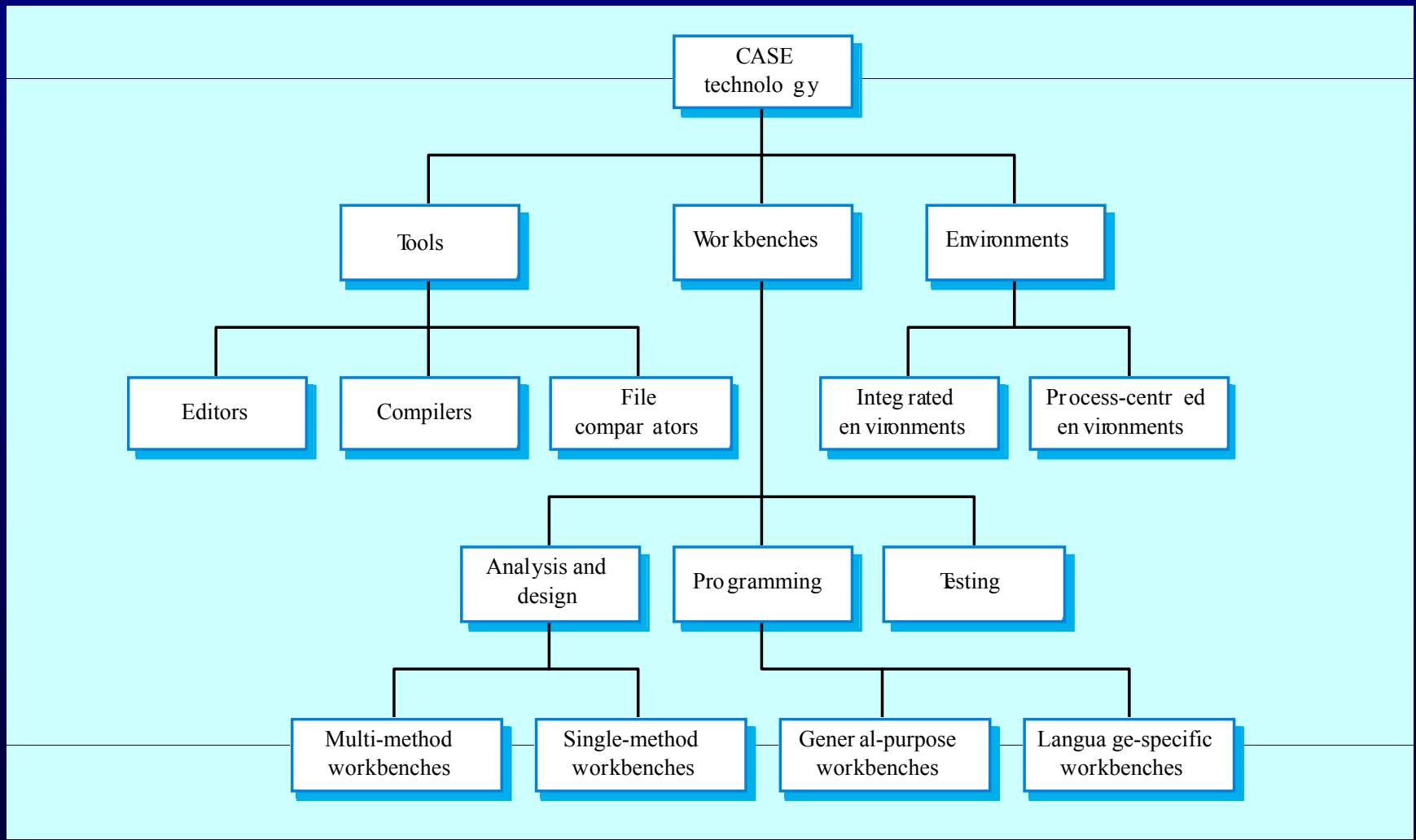
Activity-based tool classification



CASE integration

- Tools
  - Support individual process tasks such as design consistency checking, text editing, etc.
- Workbenches
  - Support a process phase such as specification or design, Normally include a number of integrated tools.
- Environments
  - Support all or a substantial part of an entire software process. Normally include several integrated workbenches.

Tools, workbenches, environments



Key points

- Software processes are the activities involved in producing and evolving a software system.
- Software process models are abstract representations of these processes.
- General activities are specification, design and implementation, validation and evolution.
- Generic process models describe the organisation of software processes. Examples include the waterfall model, evolutionary development and component-based software engineering.
- Iterative process models describe the software process as a cycle of activities.

Key points

- Requirements engineering is the process of developing a software specification.
- Design and implementation processes transform the specification to an executable program.
- Validation involves checking that the system meets to its specification and user needs.
- Evolution is concerned with modifying the system after it is in use.
- The Rational Unified Process is a generic process model that separates activities from phases.
- CASE technology supports software process activities.

Project management

Objectives

- To explain the main tasks undertaken by project managers
- To introduce software project management and to describe its distinctive characteristics
- To discuss project planning and the planning process
- To show how graphical schedule representations are used by project management
- To discuss the notion of risks and the risk management process

Topics covered

- Management activities
- Project planning
- Project scheduling
- Risk management

Software project management

- Concerned with activities involved in ensuring that software is delivered on time and on schedule and in accordance with the requirements of the organisations developing and procuring the software.
- Project management is needed because software development is always subject to budget and schedule constraints that are set by the organisation developing the software.

Software management distinctions

- The product is intangible.
- The product is uniquely flexible.
- Software engineering is not recognized as an engineering discipline with the sane status as mechanical, electrical engineering, etc.
- The software development process is not standardised.
- Many software projects are 'one-off' projects.

Management activities

- Proposal writing.
- Project planning and scheduling.
- Project costing.
- Project monitoring and reviews.
- Personnel selection and evaluation.
- Report writing and presentations.

Management commonalities

- These activities are not peculiar to software management.
- Many techniques of engineering project management are equally applicable to software project management.
- Technically complex engineering systems tend to suffer from the same problems as software systems.

Project staffing

- May not be possible to appoint the ideal people to work on a project
  - Project budget may not allow for the use of highly-paid staff;
  - Staff with the appropriate experience may not be available;
  - An organisation may wish to develop employee skills on a software project.
- Managers have to work within these constraints especially when there are shortages of trained staff.

Project planning

- Probably the most time-consuming project management activity.
- Continuous activity from initial concept through to system delivery. Plans must be regularly revised as new information becomes available.
- Various different types of plan may be developed to support the main software project plan that is concerned with schedule and budget.

Types of project plan

Plan	Description
Quality plan	Describes the quality procedures and standards that will be used in a project. See Chapter 27.
Validation plan	Describes the approach, resources and schedule used for system validation. See Chapter 22.
Configuration management plan	Describes the configuration management procedures and structures to be used. See Chapter 29.
Maintenance plan	Predicts the maintenance requirements of the system, maintenance costs and effort required. See Chapter 21.
Staff development plan.	Describes how the skills and experience of the project team members will be developed. See Chapter 25.

Project planning process

```
Establish the project constraints
Make initial assessments of the project parameters
Define project milestones and deliverables
while project has not been completed or cancelled loop
        Draw up project schedule
        Initiate activities according to schedule
        Wait ( for a while )
        Review project progress
        Revise estimates of project parameters
        Update the project schedule
        Re-negotiate project constraints and deliverables
        if ( problems arise ) then
                Initiate technical review and possible revision
        end if
end loop
```

The project plan

- The project plan sets out:
  - The resources available to the project;
  - The work breakdown;
  - A schedule for the work.

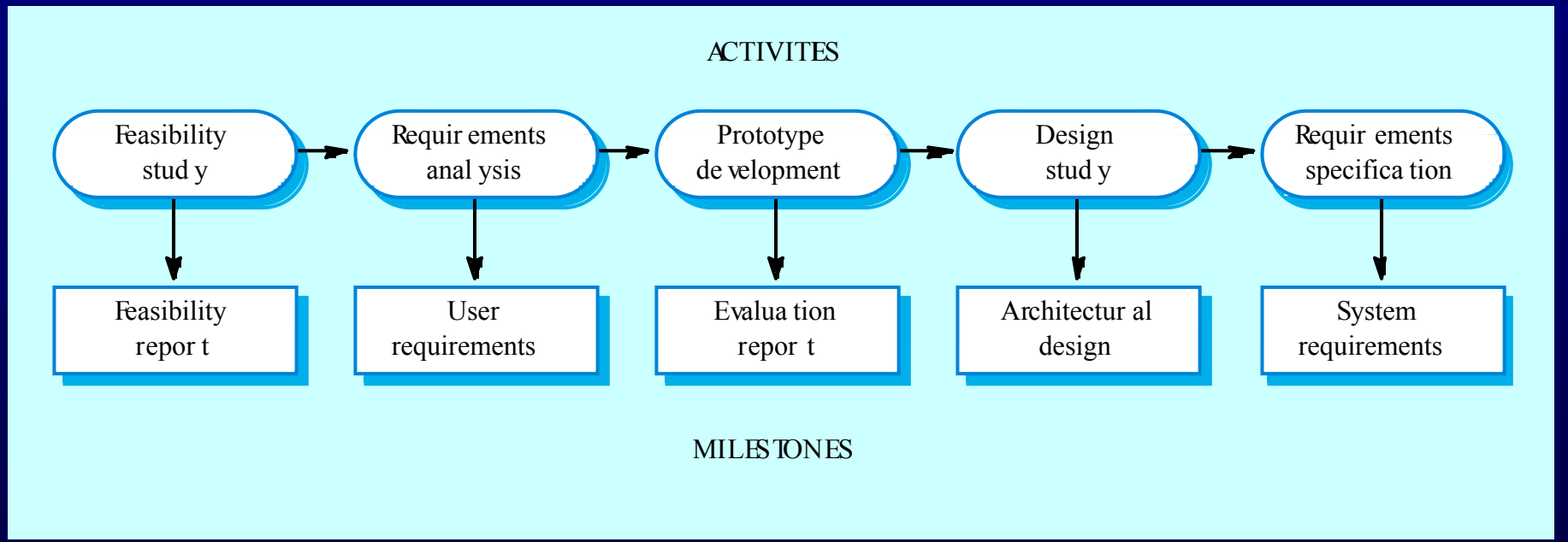
Project plan structure

- Introduction.
- Project organisation.
- Risk analysis.
- Hardware and software resource requirements.
- Work breakdown.
- Project schedule.
- Monitoring and reporting mechanisms.

Activity organization

- Activities in a project should be organised to produce tangible outputs for management to judge progress.
- *Milestones* are the end-point of a process activity.
- *Deliverables* are project results delivered to customers.
- The waterfall process allows for the straightforward definition of progress milestones.

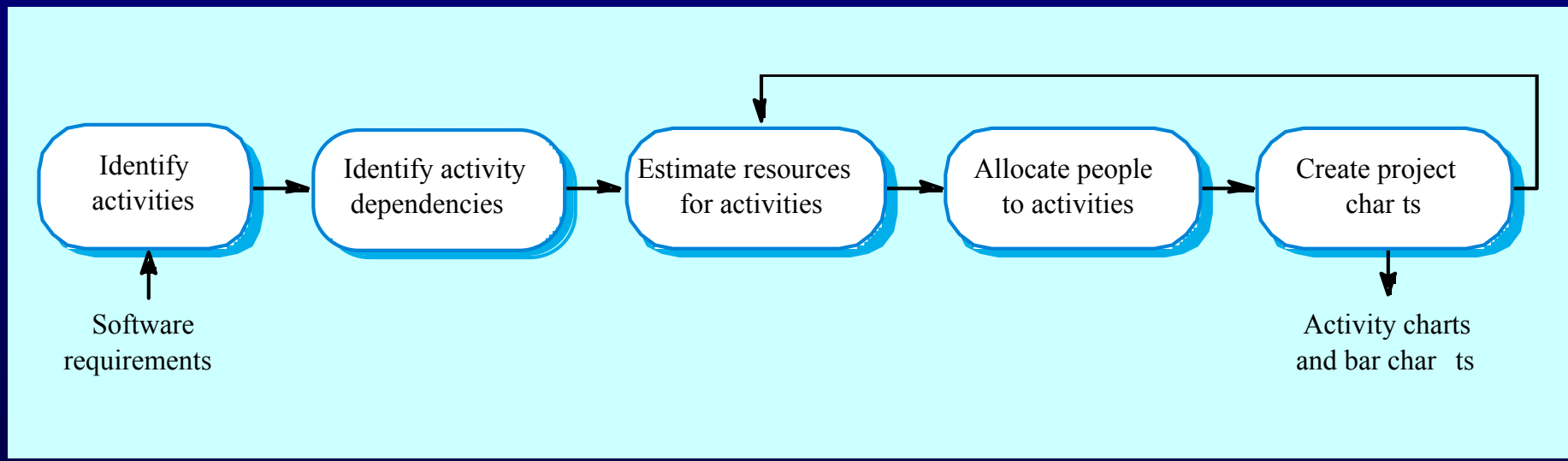
Milestones in the RE process



Project scheduling

- Split project into tasks and estimate time and resources required to complete each task.
- Organize tasks concurrently to make optimal use of workforce.
- Minimize task dependencies to avoid delays caused by one task waiting for another to complete.
- Dependent on project managers intuition and experience.

The project scheduling process



Scheduling problems

- Estimating the difficulty of problems and hence the cost of developing a solution is hard.
- Productivity is not proportional to the number of people working on a task.
- Adding people to a late project makes it later because of communication overheads.
- The unexpected always happens. Always allow contingency in planning.

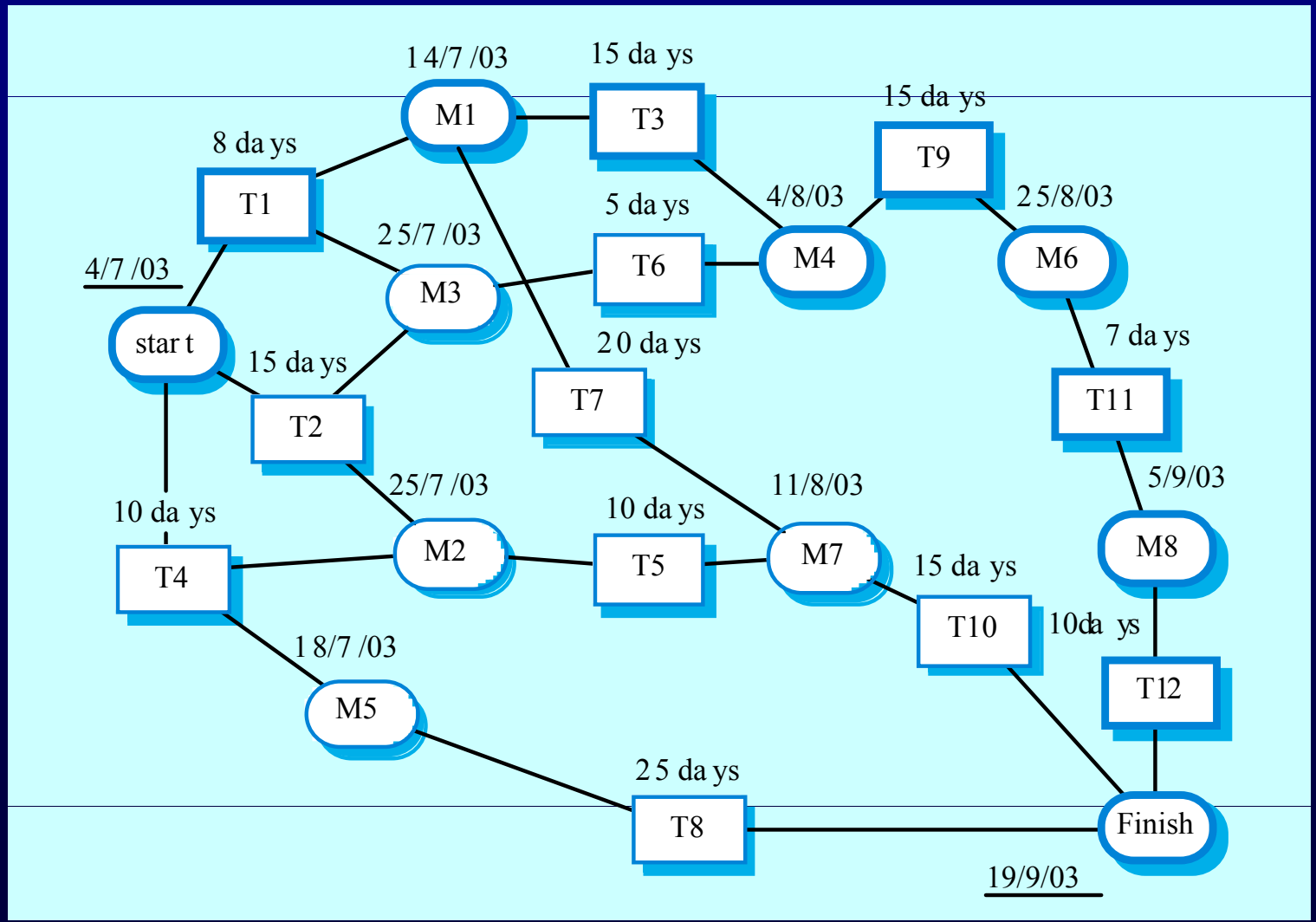
Bar charts and activity networks

- Graphical notations used to illustrate the project schedule.
- Show project breakdown into tasks. Tasks should not be too small. They should take about a week or two.
- Activity charts show task dependencies and the the critical path.
- Bar charts show schedule against calendar time.

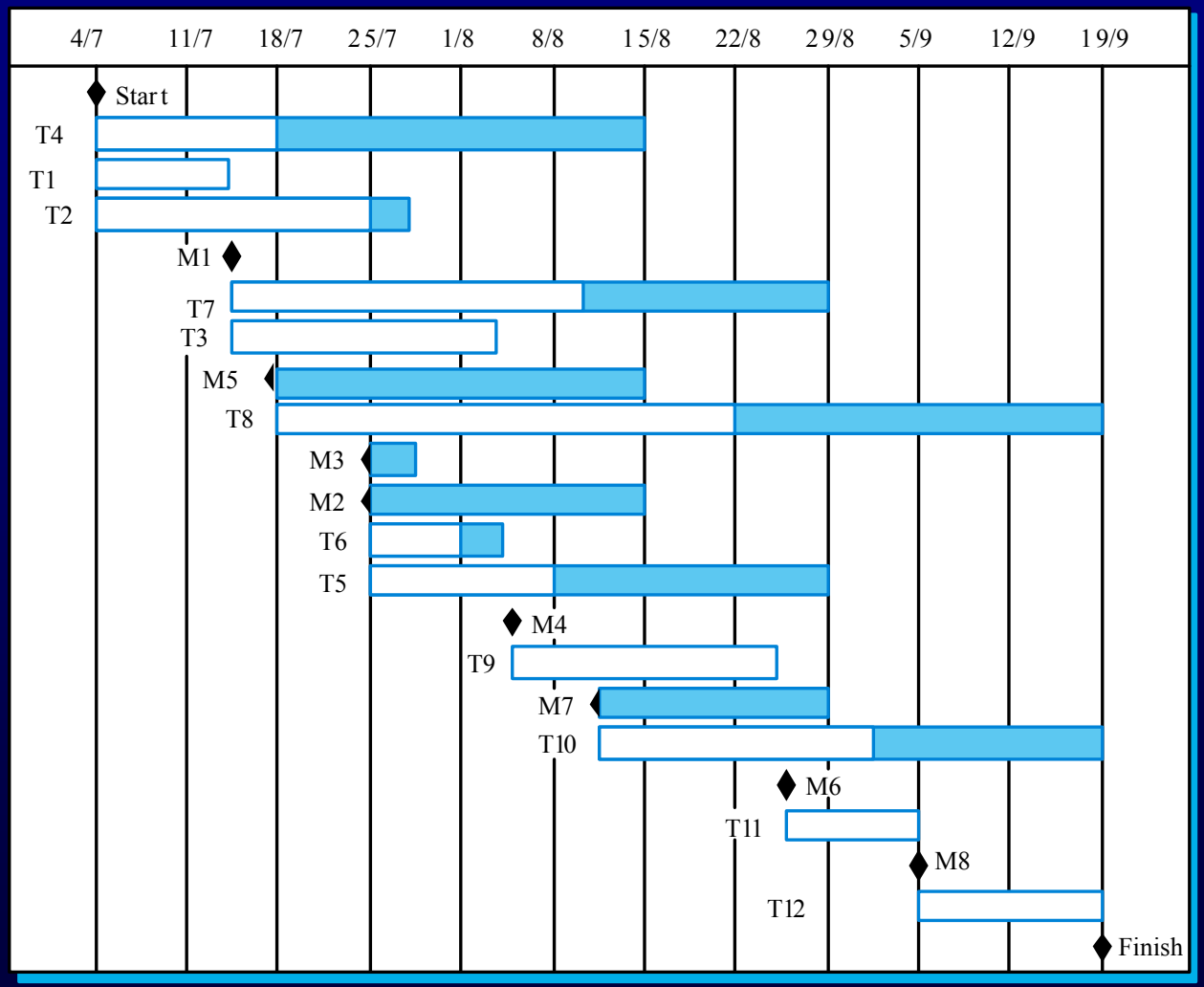
Task durations and dependencies

Activity	Duration (days)	Dependencies
T1	8	
T2	15	
T3	15	T1 (M1)
T4	10	
T5	10	T2, T4 (M2)
T6	5	T1, T2 (M3)
T7	20	T1 (M1)
T8	25	T4 (M5)
T9	15	T3, T6 (M4)
T10	15	T5, T7 (M7)
T11	7	T9 (M6)
T12	10	T11 (M8)

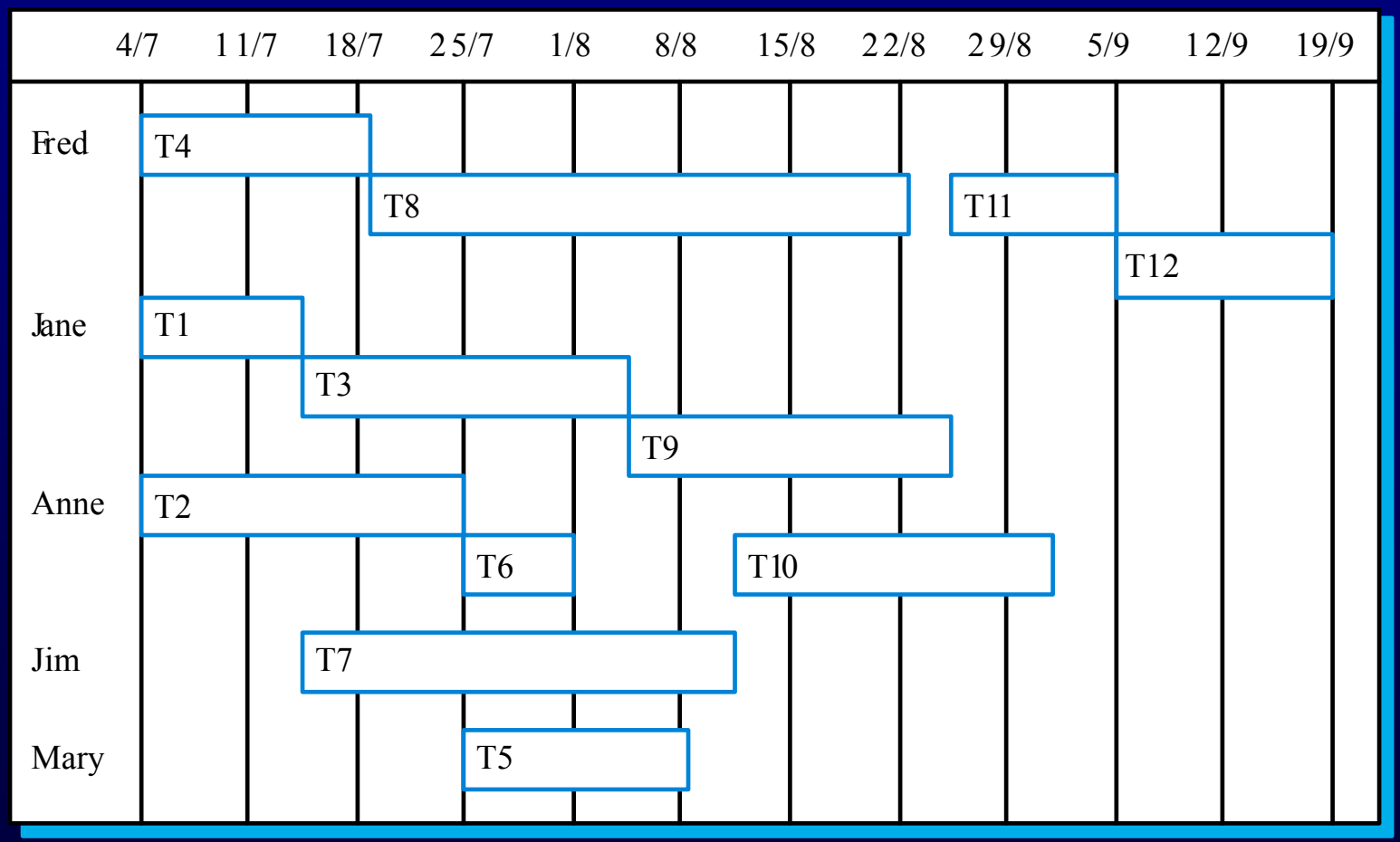
Activity network



Activity timeline



Staff allocation



Risk management

- Risk management is concerned with identifying risks and drawing up plans to minimise their effect on a project.
- A risk is a probability that some adverse circumstance will occur
  - Project risks affect schedule or resources;
  - Product risks affect the quality or performance of the software being developed;
  - Business risks affect the organisation developing or procuring the software.

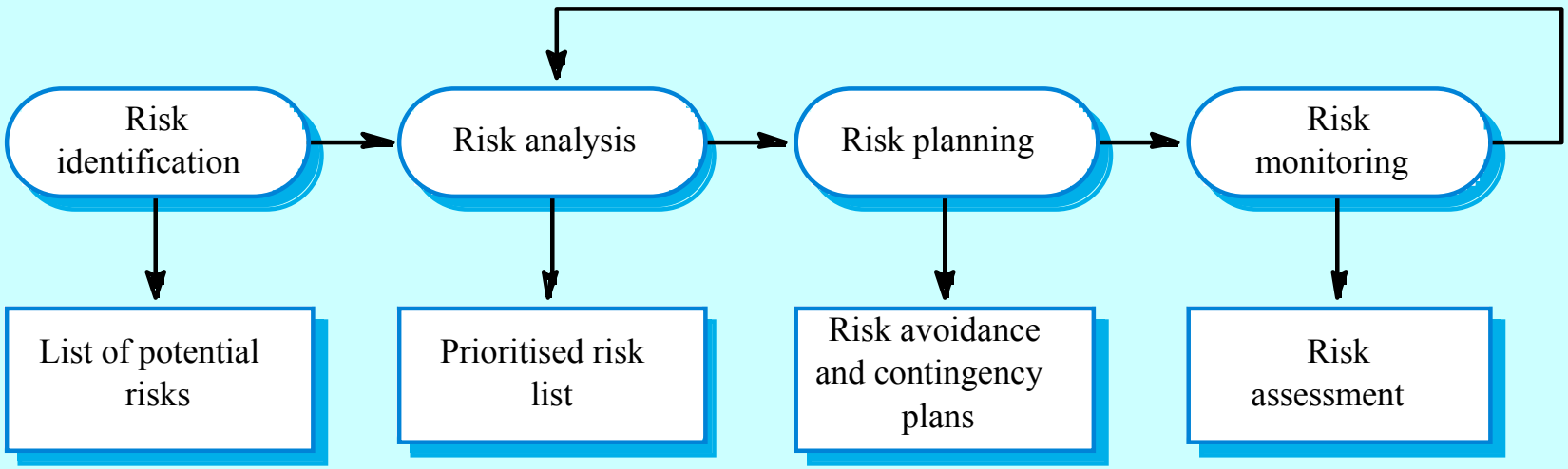
Software risks

Risk	Affects	Description
Staff turnover	Project	Experienced staff will leave the project before it is finished.
Management change	Project	There will be a change of organisational management with different priorities.
Hardware unavailability	Project	Hardware that is essential for the project will not be delivered on schedule.
Requirements change	Project and product	There will be a larger number of changes to the requirements than anticipated.
Specification delays	Project and product	Specifications of essential interfaces are not available on schedule
Size underestimate	Project and product	The size of the system has been underestimated.
CASE tool under-performance	Product	CASE tools which support the project do not perform as anticipated
Technology change	Business	The underlying technology on which the system is built is superseded by new technology.
Product competition	Business	A competitive product is marketed before the system is completed.

The risk management process

- Risk identification
  - Identify project, product and business risks;
- Risk analysis
  - Assess the likelihood and consequences of these risks;
- Risk planning
  - Draw up plans to avoid or minimise the effects of the risk;
- Risk monitoring
  - Monitor the risks throughout the project;

The risk management process



Risk identification

- Technology risks.
- People risks.
- Organisational risks.
- Requirements risks.
- Estimation risks.

Risks and risk types

Risk type	Possible risks
Technology	The database used in the system cannot process as many transactions per second as expected. Software components that should be reused contain defects that limit their functionality.
People	It is impossible to recruit staff with the skills required. Key staff are ill and unava ilable at critical times. Required training for staff is not available.
Organisational	The organisation is restructured so that different management are responsible for the project. Organisational financial problems force reductions in the project budget.
Tools	The code generated by CASE tools is inefficient. CASE tools cannot be integrated.
Requirements	Changes to requirements that require major design rework are proposed. Customers fail to understand the impact of requirements changes.
Estimation	The time required to develop the software is underestimated. The rate of defect repair is underestimated. The size of the software is underestimated.

Risk analysis

- Assess probability and seriousness of each risk.
- Probability may be very low, low, moderate, high or very high.
- Risk effects might be catastrophic, serious, tolerable or insignificant.

Risk analysis (i)

Risk	Probability	Effects
Organisational financial problems force reductions in the project budget.	Low	Catastrophic
It is impossible to recruit staff with the skills required for the project.	High	Catastrophic
Key staff are ill at critical times in the project.	Moderate	Serious
Software components that should be reused contain defects which limit their functionality.	Moderate	Serious
Changes to requirements that require major design rework are proposed.	Moderate	Serious
The organisation is restructured so that different management are responsible for the project.	High	Serious

Risk analysis (ii)

Risk	Probability	Effects
The database used in the system cannot process as many transactions per second as expec ted.	Moderate	Serious
The time required to develop the software is underestimated.	High	Serious
CASE tools cannot be integrated.	High	Tolerable
Customers fail to understand the impact of requirements changes.	Moderate	Tolerable
Required training for staff is not available.	Moderate	Tolerable
The rate of defect repair is underestimated.	Moderate	Tolerable
The size of the software is underestimated.	High	Tolerable
The code generated by CASE tools is inefficient.	Moderate	Insignificant

Risk planning

- Consider each risk and develop a strategy to manage that risk.
- Avoidance strategies
  - The probability that the risk will arise is reduced;
- Minimisation strategies
  - The impact of the risk on the project or product will be reduced;
- Contingency plans
  - If the risk arises, contingency plans are plans to deal with that risk;

Risk management strategies (i)

Risk	Strategy
Organisational financial problems	Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business.
Recruitment problems	Alert customer of potential difficulties and the possibility of delays, investigate buying-in components.
Staff illness	Reorganise team so that there is more overlap of work and people therefore understand each other's jobs.
Defective components	Replace potentially defective components with bought-in components of known reliability.

Risk management strategies (ii)

Risk	Strategy
Requirements changes	Derive traceability information to assess requirements change impact, maximise information hiding in the design.
Organisational restructuring	Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business.
Database performance	Investigate the possibility of buying a higher-performance database.
Underestimated development time	Investigate buying in components, investigate use of a program generator

Risk monitoring

- Assess each identified risks regularly to decide whether or not it is becoming less or more probable.
- Also assess whether the effects of the risk have changed.
- Each key risk should be discussed at management progress meetings.

Risk indicators

Risk type	Potential indicators
Technology	Late delivery of hardware or support software, many reported technology problems
People	Poor staff morale, poor relationships amongst team member, job availability
Organisational	Organisational gossip, lack of action by senior management
Tools	Reluctance by team members to use tools, complaints about CASE tools, demands for higher-powered workstations
Requirements	Many requirements change requests, customer complaints
Estimation	Failure to meet agreed schedule, failure to clear reported defects

Key points

- Good project management is essential for project success.
- The intangible nature of software causes problems for management.
- Managers have diverse roles but their most significant activities are planning, estimating and scheduling.
- Planning and estimating are iterative processes which continue throughout the course of a project.

Key points

- A project milestone is a predictable state where a formal report of progress is presented to management.
- Project scheduling involves preparing various graphical representations showing project activities, their durations and staffing.
- Risk management is concerned with identifying risks which may affect the project and planning to ensure that these risks do not develop into major threats.

Software Requirements

Objectives

- To introduce the concepts of user and system requirements
- To describe functional and non-functional requirements
- To explain how software requirements may be organised in a requirements document

Topics covered

- Functional and non-functional requirements
- User requirements
- System requirements
- Interface specification
- The software requirements document

Requirements engineering

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

What is a requirement?

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

Requirements abstraction (Davis)

"If a company wishes to let a contract for a large software development project, it must define its needs in a sufficiently abstract way that a solution is not pre-defined. The requirements must be written so that several contractors can bid for the contract, offering, perhaps, different ways of meeting the client 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. Defines what should be implemented so may be part of a contract between client and contractor.

Definitions and specifications

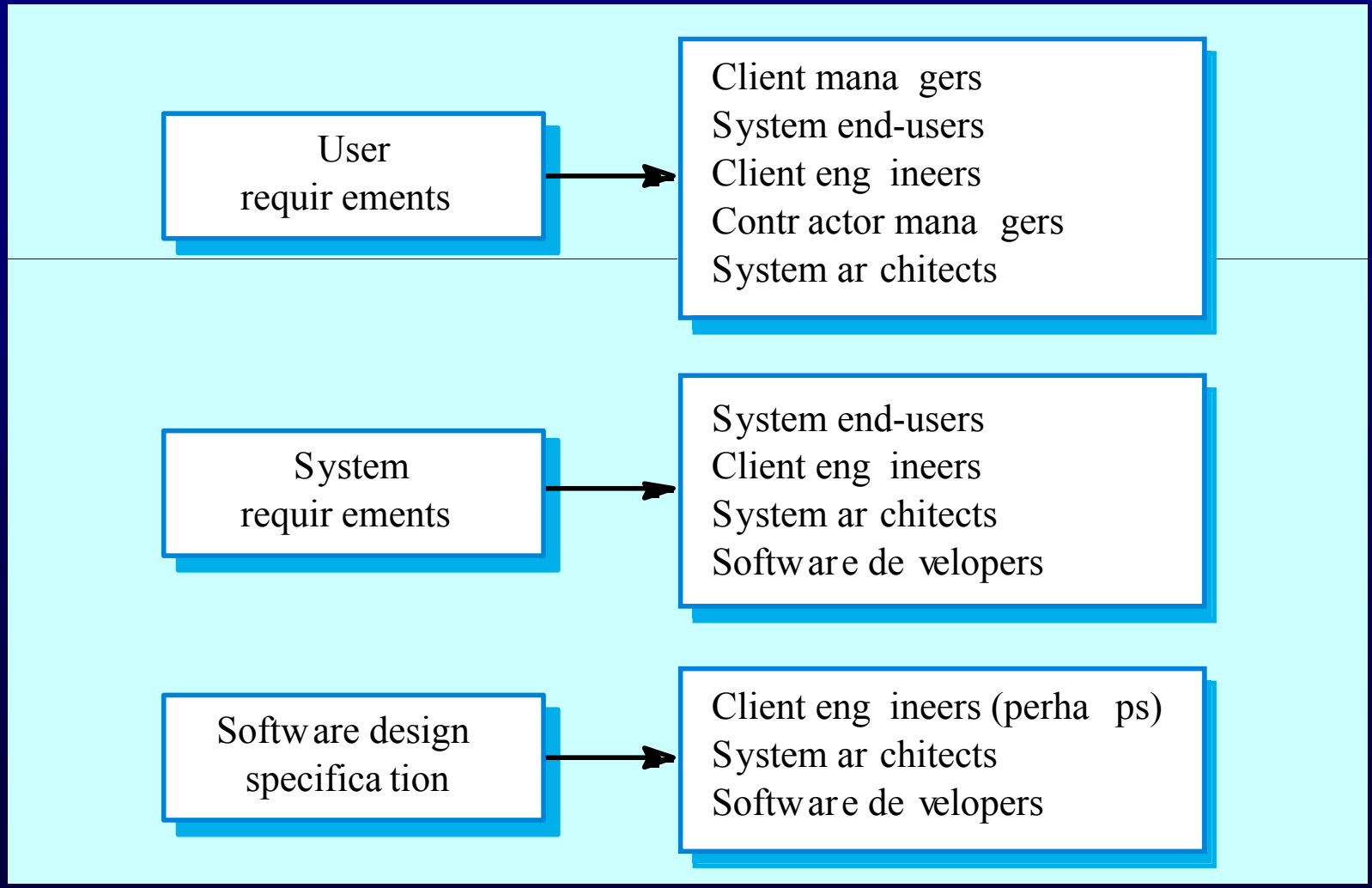
User requirement definition

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
  - Statements of services 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 offered by the system such as timing constraints, constraints on the development process, standards, etc.
- Domain requirements
  - Requirements that come from the application domain of the system and that reflect characteristics of that domain.

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 but functional system requirements should describe the system services in detail.

The LIBSYS system

- A library system that provides a single interface to a number of databases of articles in different libraries.
- Users can search for, download and print these articles for personal study.

Examples of functional 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

- 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.

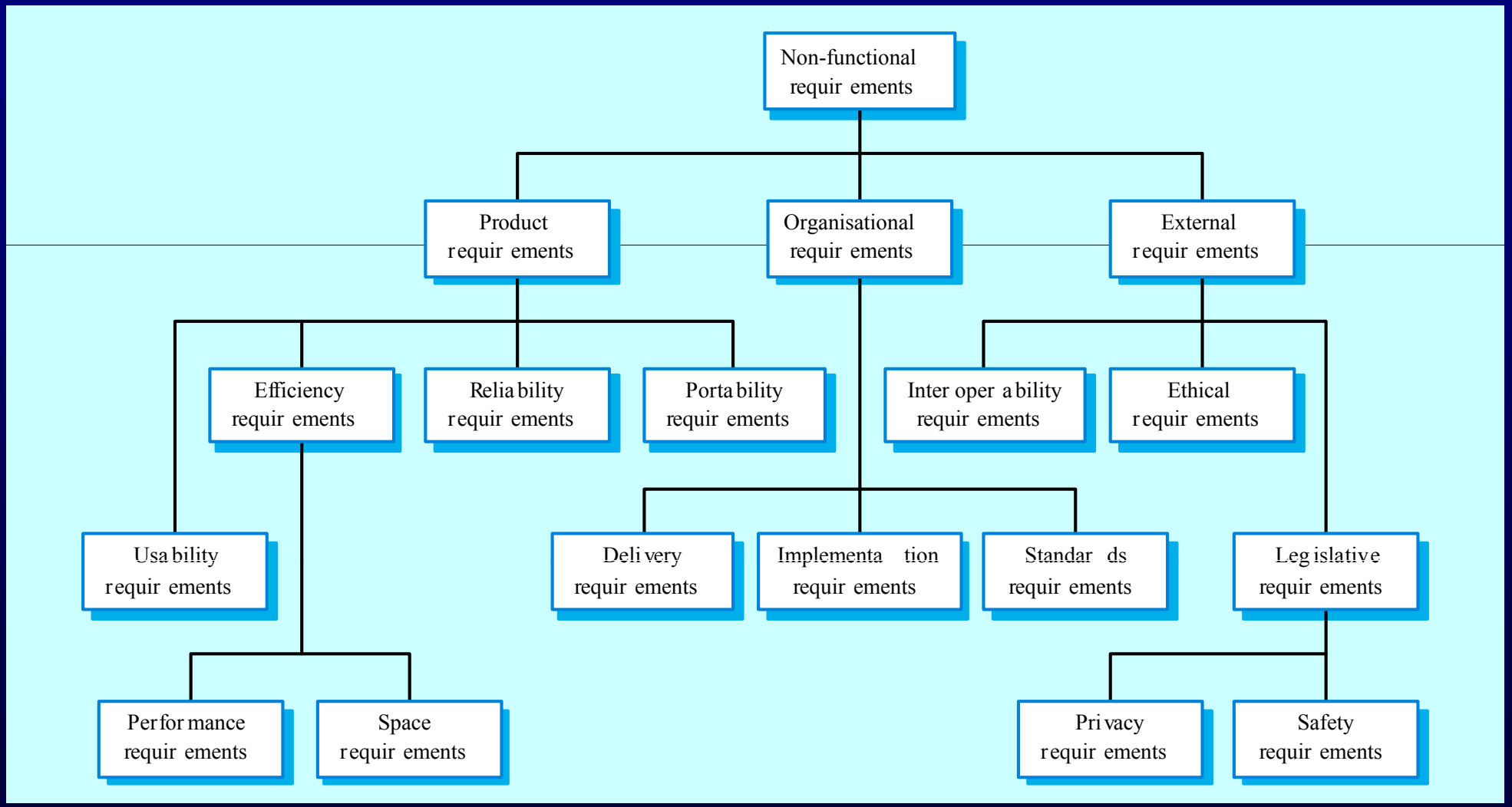
Non-functional requirements

- 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 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
  - 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.

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 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.

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

Library system domain requirements

- There shall be a standard user interface to all databases which shall be based on the Z39.50 standard.
- Because of copyright restrictions, some documents must be deleted immediately on arrival. Depending on the user's requirements, these documents will either be printed locally on the system server for manually forwarding to the user or routed to a network printer.

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.

User requirements

- Should describe functional and non-functional requirements in such a way that they are understandable by system users who don't have detailed technical knowledge.
- User requirements are defined using natural language, tables and diagrams as these can be understood by all users.

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.

LIBSYS requirement

4..5 LIBSYS shall provide a financial accounting system that maintains records of all payments made by users of the system. System managers may configure this system so that regular users may receive discounted rates.

Editor grid requirement

2.6 Grid facilities To assist in the positioning of entities on a diagram the user may turn on a grid in either centimetres or inches, via an option on the control panel. Initially, the grid is off. The grid may be turned on and off at any time during an editing session and can be toggled between inches and centimetres at any time. A grid option will be provided on the reduce-to-fit view but the number of grid lines shown will be reduced to avoid filling the smaller diagram with grid lines.

Requirement problems

- Database requirements includes both conceptual and detailed information
  - Describes the concept of a financial accounting system that is to be included in LIBSYS;
  - However, it also includes the detail that managers can configure this system - this is unnecessary at this level.
- Grid requirement mixes three different kinds of requirement
  - Conceptual functional requirement (the need for a grid);
  - Non-functional requirement (grid units);
  - Non-functional UI requirement (grid switching).

Structured presentation

2.6.1 Grid facilities

The editor shall provide a grid facility where a matrix of horizontal and vertical lines provide a background to the editor window. This grid shall be a passive grid where the alignment of entities is the user's responsibility.

Rationale: A grid helps the user to create a tidy diagram with well-spaced entities. Although an active grid, where entities 'snap-to' grid lines can be useful, the positioning is imprecise. The user is the best person to decide where entities should be positioned.

Specification: ECLIPSE/WS/Tools/DE/FS Section 5.6
Source: Ray Wilson, Glasgow Office

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 discussed in Chapter 8.

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.

Problems with NL specification

- 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.

Alternatives to NL specification

Notation	Description
Structured natural language	This approach depends on defining standard forms or templates to express the requirements specification.
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 not now widely used although it can be useful for interface specifications.
Graphical notations	A graphical language, supplemented by text annotations is used to define the functional requirements for the system. An early example of such a graphical language was SADT. Now, use-case descriptions and sequence diagrams are commonly used .
Mathematical specifications	These are notations based on mathematical concepts such as finite-state machines or sets. These unambiguous specifications reduce the arguments between customer and contractor about system functionality. However, most customers don't understand formal specifications and are reluctant to accept it as a system contract.

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 is that the most of the expressiveness of natural language is maintained but a degree of uniformity is imposed on the specification.

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.

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 Ğthe 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

- Used to supplement natural language.
- Particularly useful when you have to define a number of possible alternative courses of action.

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

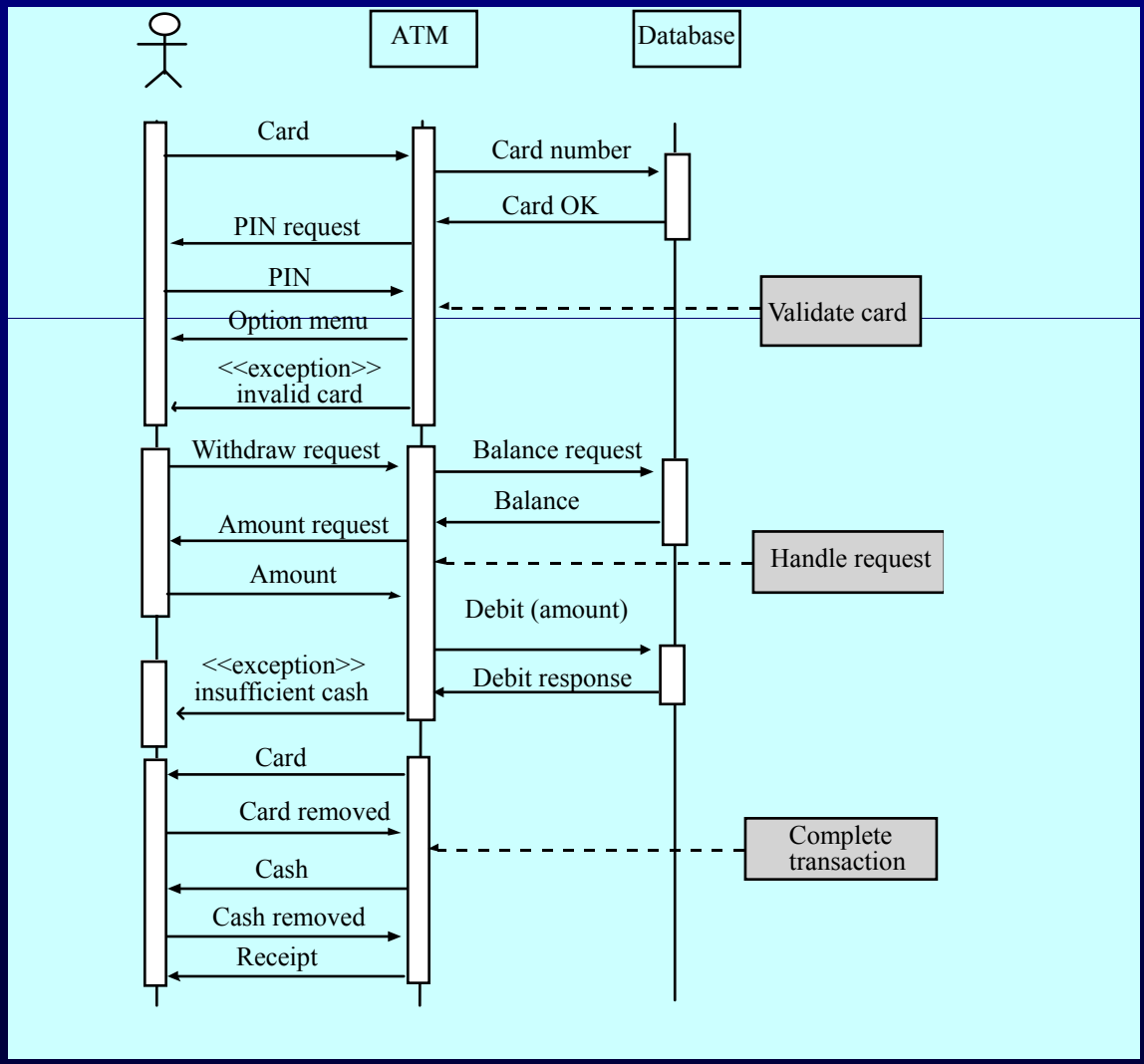
Graphical models

- Graphical models are most useful when you need to show how state changes or where you need to describe a sequence of actions.
- Different graphical models are explained in Chapter 8.

Sequence diagrams

- These show the sequence of events that take place during some user interaction with a system.
- You read them from top to bottom to see the order of the actions that take place.
- Cash withdrawal from an ATM
  - Validate card;
  - Handle request;
  - Complete transaction.

Sequence diagram of ATM withdrawal



Interface specification

- Most systems must operate with other systems and the operating interfaces must be specified as part of the requirements.
- Three types of interface may have to be defined
  - Procedural interfaces;
  - Data structures that are exchanged;
  - Data representations.
- Formal notations are an effective technique for interface specification.

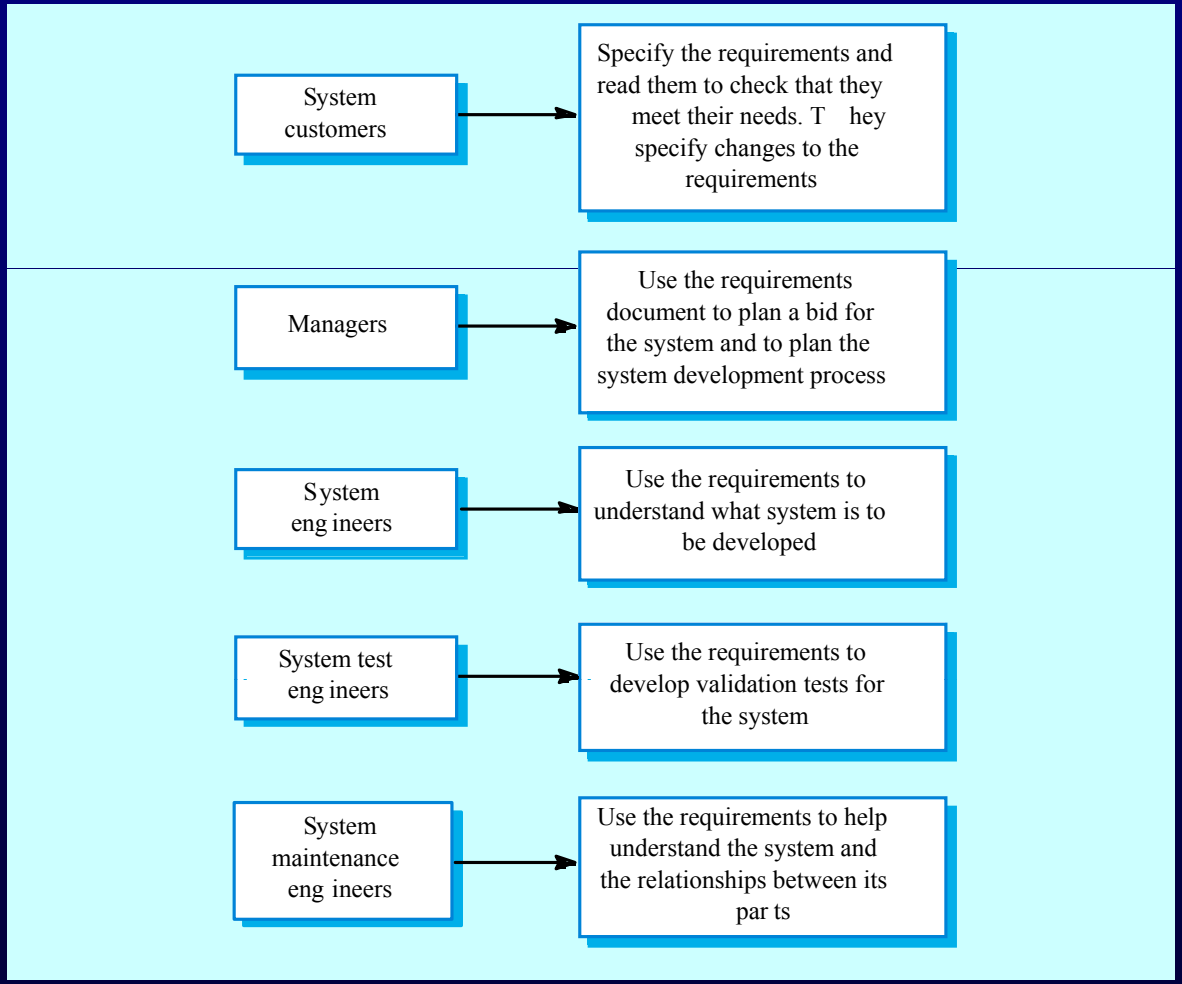
PDL interface description

```
interface PrintServer {

// defines an abstract printer server
// requires:          interface Printer, interface PrintDoc
// provides: initialize, print, displayPrintQueue, cancelPrintJob, switchPrinter

        void initialize ( Printer p ) ;
        void print ( Printer p, PrintDoc d ) ;
        void displayPrintQueue ( Printer p ) ;
        void cancelPrintJob (Printer p, PrintDoc d) ;
        void switchPrinter (Printer p1, Printer p2, PrintDoc d) ;
} //PrintServer
```

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

Users of a requirements document



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

Key points

- Requirements set out what the system should do and define constraints on its operation and implementation.
- Functional requirements set out services the system should provide.
- Non-functional requirements constrain the system being developed or the development process.
- User requirements are high-level statements of what the system should do. User requirements should be written using natural language, tables and diagrams.

Key points

- System requirements are intended to communicate the functions that the system should provide.
- A software requirements document is an agreed statement of the system requirements.
- The IEEE standard is a useful starting point for defining more detailed specific requirements standards.