# Introduction to Critical Systems

- **Topic 1:**
  - **Processes and Techniques for developing Critical Systems –**
    - **An Introduction**


Supported by:
**Joint MSc curriculum in software engineering**
European Union TEMPUS Project CD_JEP-18035-2003

Version: Oct. 26, 2006

# What is a System?

- A collection of elements which are assembled to fulfil some defined purpose. Elements may be hardware or software components, organisational policies and procedures and operational processes.
- Systems have properties which are emergent i.e. they only come to light when the parts are put together, they have structure and mechanisms for communication and control.
  - Man vs. Collection of books
- In this Module, we are only interested in systems which include computers and where software plays a major part in the control of the systems.

# Socio-technical computer-based systems

***Systems in which some of the elements are software-controlled computers and which are used by people for some purpose (to support some human activity). They typically include:***

- Computer hardware
- Software
- Policies and procedures
- Operational processes

- **Computer-based system vs. system**

# Examples

- ***A payroll system***
- ***A navigation system***
- ***A system for testing blood samples***
- ***A mobile phone***
- ***A ticket reservation system***
- ***A chemical process control system***
- ***A pollution monitoring system***
- ***An air traffic control system***

# Emergent properties

- ***Properties which are properties of the system AS A WHOLE rather than of the collection of parts***
  - *Functioning bicycle (emergent: transportation device) vs. collection of its parts.*
- ***Not determined solely from the properties of the system parts but also from the system's structure and its interactions.***
- ***These properties are sometimes planned and predictable and sometime are not planned and not predictable.***
  - *Mobile phone: communication device (planned) / Interfering device (not predictable)*
  - *Unplanned – something is difficult to use*
- ***Examples***
  - The reliability of a computer depends on the reliability of the processor, memory, keyboard, monitor, disk, etc, but...

# Emergent system-wide properties

- ***Important emergent properties of a system are***
  - Performance
  - Reliability
  - Safety
  - Security
  - Usability
  - Maintainability

***These depend on the relationships between components as well as the components themselves.***

***For example you may end up with an unreliable system which was build from reliable components. On the other hand safety and security usually does not depend on individual compononents.***

- ***These are non-functional properties - they do not relate to any specific functionality of the system.***

- ***Some or all of these properties are usually more important than detailed system functionality (unreliable -> wrong results).***

# The role of software in systems

- ***Software in complex systems now has a number of different roles. For example*:**

  - ***Control and coordination.*** The operation of different parts of the system is coordinated by a controlling software system.
  - ***Information management.*** Large amounts of information that is required in many systems is managed and organised by software.
  - ***Input and output filtering.*** System inputs and outputs are pre and post processed by software to simplify their subsequent processing.
  - ***User interface.*** The user interface to many systems is now largely a software-based interface
  - ***System monitoring.*** The operation of the system is monitored by software and anomalies reported.

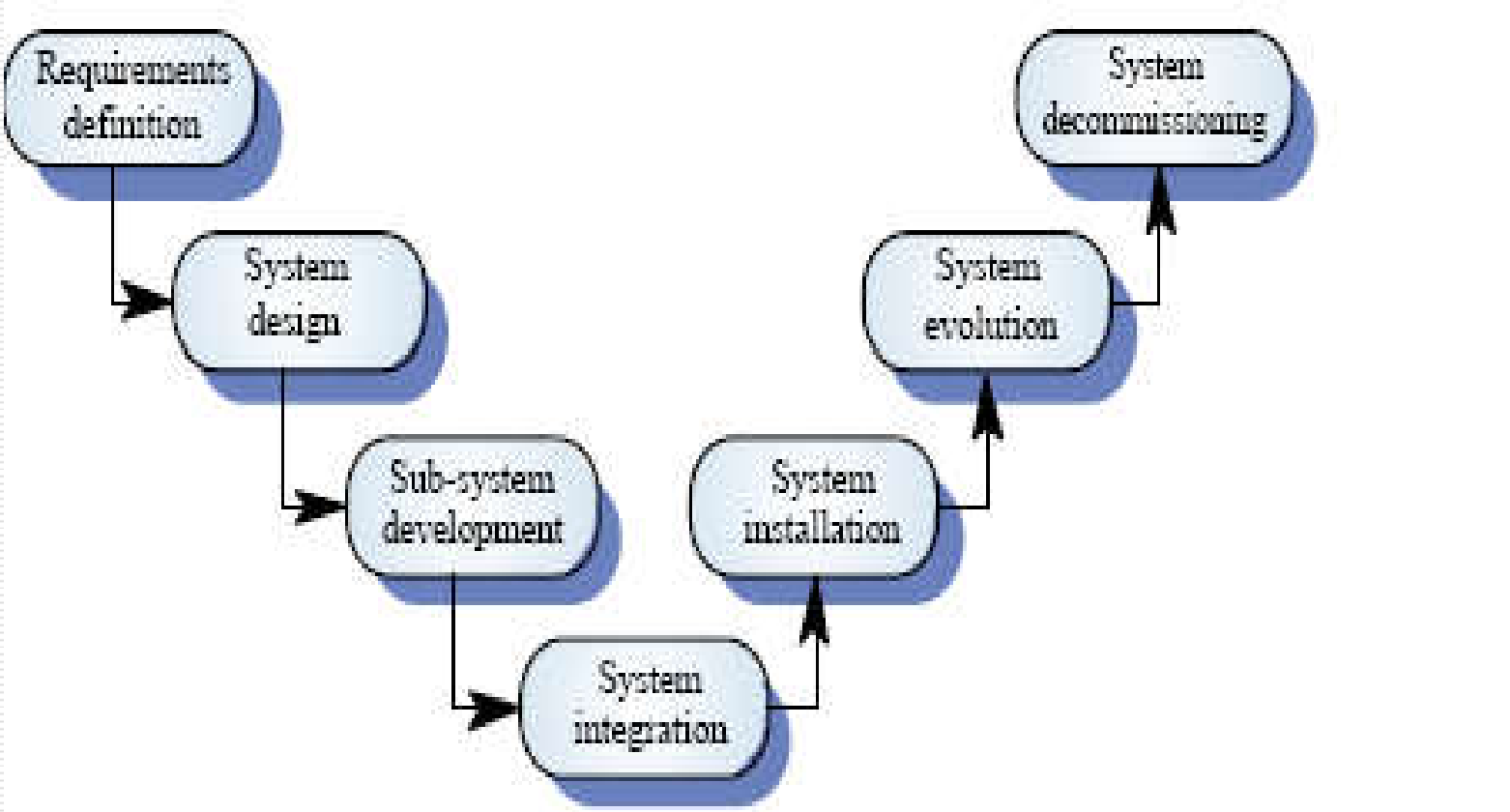
# Systems engineering

- ***The process of specifying, designing, implementing and installing computer-based systems to solve some identified problem***
  - *"To buy a bread" vs. to design a tunnel below Bosphorus.*

- ***Generally concerned with complex systems involving hardware, software and people.***

- ***Concerned with wider 'systems issues' rather than details of a system's functionality and implementation. Those will be designed as subsystems that will be engineerined in different processes.***

# The systems engineering process (idealized)



# System and software engineering

- ***Software is increasingly used in systems because it allows for more complex information processing AND it is malleable - changes can be made after the design is complete.***

- ***Many software system problems are a consequence of this malleability. Software changes become necessary because of problems elsewhere in the system or new requirements that emerge when the system is integrated.***

# Critical systems

- ***A critical system is any system whose 'failure' could threaten human life, the system's environment or the existence of the organisation which operates the system.***

- ***'Failure' in this context does NOT mean failure to conform to a specification but means any potentially threatening system behaviour.***
- ***Critical systems existed also before computer-based systems (airplane)***

# Critical system classes

- ***Safety-critical systems***
  - A system whose failure may result in the loss of human life, injury or major environment damage

- ***Mission-critical systems***
  - A system whose failure may result in the consequent failure of a goal-directed activity.

- ***Business-critical systems***
  - A system whose failure may result in the failure of the business that is using that system.

# Examples of critical systems

- ***Communication systems such as telephone switching systems, aircraft radio systems, etc.***
- ***Embedded control systems for process plants, medical devices (primary).***
- ***Command and control systems such as air-traffic control systems, disaster management systems.***
- ***Financial systems such as foreign exchange transaction systems, account management systems (secondary).***

# Critical systems usage

- ***Most critical systems are now computer-based systems.***
- ***Critical systems are becoming more widespread as society becomes more complex and more complex activities are automated.***
  - However, the procedures and practices which have evolved to integrate critical systems into society are based on much less complex systems. In many cases, we do not really understand what the overall impact of these critical systems will be (e.g. automated share trading systems).
- ***People and operational processes are very important elements of critical systems – they cannot simply be considered in terms of hardware and software (socio-technical systems).***

# Critical systems failure

- ***The cost of failure in a critical system is likely to exceed the cost of the system itself***

- ***As well as direct failure costs, there are indirect costs from a critical systems failure. These may be significantly greater than the direct costs.***
- ***Here are some:***

# Critical systems failure-cost

- **Direct costs of repairing the system -** can be high if expensive hardware is physically damaged**.**
- **Costs of investigating the cause of the problem -** again this, can be very high if there has been an accident with associated loss of life.
- **Loss of revenue while the system is out of service.**
- **Compensation costs for people/things damaged by the failure.**
- **Legal costs associated with compensation claims.**
- **Re-design and change costs for other systems which may be vulnerable to the same type of fail.**

# Critical systems failure-cost

- ***Society's views of critical systems are not static - they are modified by each high- profile system failure.***
  - People do not necessarily react logically to critical systems failure - they are much more concerned about train and plane crashes than car accidents although cars kill many more people annually

# Critical emergent properties

- ***Reliability***
  - *Concerned with failure to perform to specification.*
- ***Availability***
  - Concerned with failure to deliver required services
- ***Safety*** *(for safety-critical systems)*
  - Concerned with behaviour which directly or indirectly threatens human life
- ***Security***
  - Concerned with the ability of the system to protect itself from external attack

# Critical systems development

- ***Critical systems attributes are NOT independent - the systems development process must be organised so that all of them are satisfied at least to some minimum level.***
  - *security / maintainability*
  - *safety / availability*
  - *maintainability -> availability*
  - *security -> reliabilty*
- ***More rigorous (and expensive) development techniques have to be used for critical systems development because of the potential cost of failure***

# Development techniques

- ***Use of formal methods for system specification.***
- ***Use of formal verification to demonstrate that a program is consistent with its specification.***

- ***Separate teams for implementation and testing.***
- ***Incorporation of redundant code and self-checking in programs.***

- ***Redundant hardware units.***
- ***Measurement of test coverage***

# Classification

- **Classes of systems failures**
  - Class I: Physical failure
    - *System has failed fatally as a result of physical disturbance of its correct functioning*
  - Class II: Software errors
    - *Fatal errors were caused as a result of **design error** and/or not adequate testing of final product.*
  - Class III: Human computer interaction
    - *Fatal error were caused by human fallibility*
      - This is rather messier in research terms because there are many aspects of human interaction to be considered.

# Class I: Physical failure

- HMS Sheffield**'s defensive systems failure to intercept an attacking Exocet missile.**
- Cause: **electromagnetic interference. Interference from a transmission prevented her from picking up warning signals on the electronic support measurement equipment.**
- Casualty**: 20 persons died**

# Class II: Software errors

- Therac-25
- Cause: **Change from X-ray to electron mode therapy was performed while leaving the intensity at the current required for X-ray therapy.**
- Casualty: **2 patients died**
- Source of error: **Design error**

- Patriot air defence system
- Cause: **Round-off error in the 'range gate' algorithm and dynamic change management error.**
- Casualty: **30 people died**
- Source of error: **Design error**

# Class III: Human computer interaction

- Fly-by-wire Aircraft, the Airbus A320.
- Cause: **The 2-man crew have instructed the flight control system to descend at a rapid rate of 3,300 feet/minute rather than descending at an angle of 3.3 degrees. The two modes are represented by a similar 2-digit numbers. (The interface has now been changed and the vertical speed mode is represented by a 4-digit number).**
- Casualty: **2 death**

# Standardization efforts

- **UK: MoD DEF Stan 00-55 & 00-56**

**The standards identified two main aspects:**

*a) Safety Management*

- The establishment of techniques for Hazard analysis; safety risk assessment and safety integrity analysis (fault-tree)
- Certification
- Documentation

*b) Software Engineering Practices*

- "...The software Specification shall include a specification of the software using the specification notation of a **Formal Method**"
- Proof obligation – Validation and Verification

# Standardization efforts

**Coding of a CS is done in a programming language that has the following characteristics:**

a) a formally defined syntax;

b) a means of enforcing the use of any subset employed;

c) a well-understood semantics and a formal means of relating code to; a formal design;

d) block structure;

e) strongly typed.

- **Proof obligation is done on the Source Code by means of Path Analysis which include control flow, data use and information flow analysis.**

# Standardization efforts

- **European Space Agency (ESA):**

**These cover the whole software lifecycle, and, to some extent, deal with problems at the system level.**

- ESA BSSC 1A: ESA Software Engineering
- ESA BSSC: Software configuration management
- ESA BSSC 1B: ESA Software Configuration management standards

# Standardization efforts

- **USA:**

**In addition to the ANSI/IEEE we have the National Bureau of Standards (NBS)**

a) NBS SP500-93: Software validation, verification and testing techniques and tool reference

b) FIPS PUB 101: Guideline for lifecycle validation, verification and testing of computer software.

# Key points

- ***Computer-based systems are socio-technical systems which include hardware, software, operational processes and procedures and people.***
- ***An increasing number of socio-technical systems are critical systems.***
- ***Systems have emergent properties i.e. properties which are only apparent when all sub-systems are integrated.***
- ***Critical system attributes are reliability, availability, safety and security***

# Topic 2:

## Dependability-

- The extent to which a critical system is trusted by its users

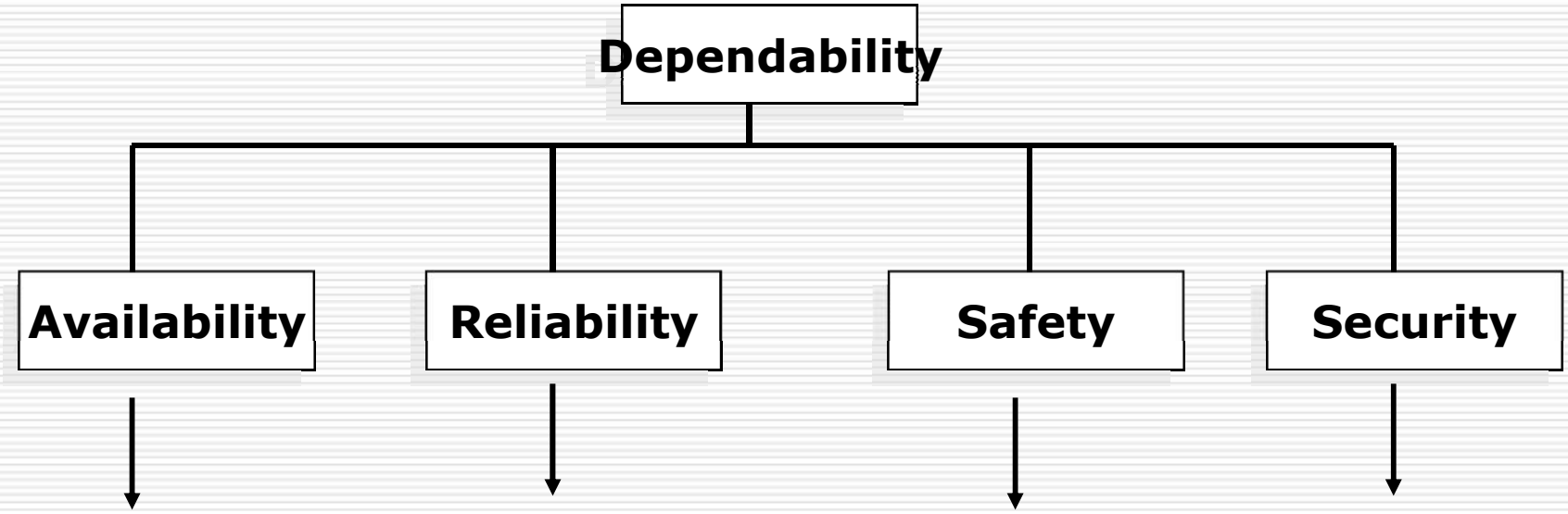

Supported by:
**Joint MSc curriculum in software engineering**
European Union TEMPUS Project CD_JEP-18035-2003

Version: Oct. 26, 2006

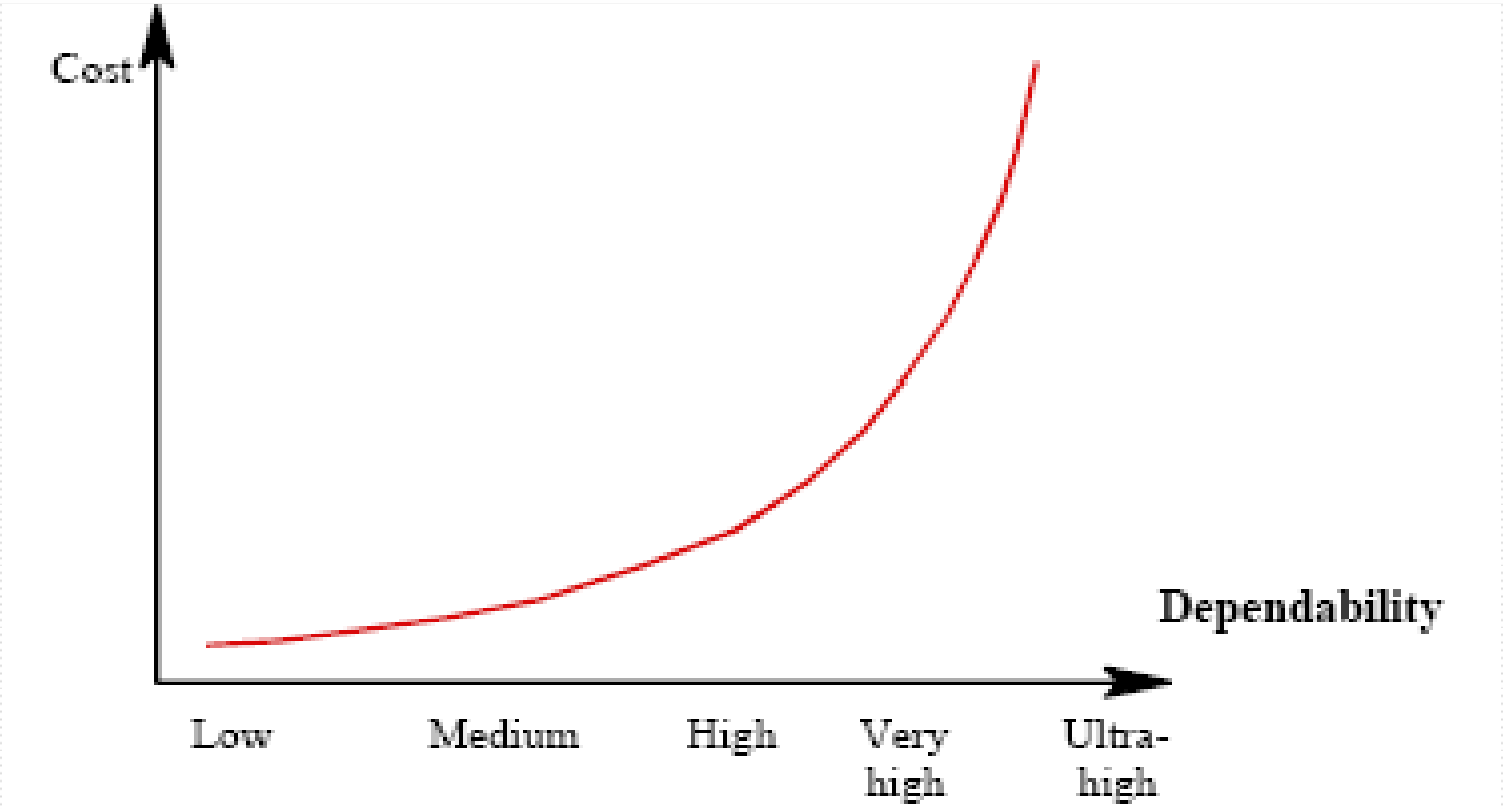
# The concept of dependability

- ***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……look at MS products!***

# Dimensions of dependability



# Costs of increasing dependability



# 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.

# 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 (e.g. deadlocks)***
- ***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***

# 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 | Erroneous system behaviour where the behaviour of the system does not conform to its specification. |
| System fault | An incorrect system state i.e. a system state that is unexpected by the designers of the system. |
| 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

# Topic 4:

## Dependable Software Development

### Programming techniques for building dependable software systems


Supported by:
**Joint MSc curriculum in software engineering**
European Union TEMPUS Project CD_JEP-18035-2003

Version: Oct. 26, 2006

# Dependability achievement

- ***Fault avoidance***
  - The software is developed in such a way that human error is avoided and thus system faults are minimised
  - The development process is organised so that faults in the software are detected and repaired before delivery to the customer
- ***Fault tolerance***
  - The software is designed so that faults in the delivered software do not result in system failure

# Exception management

- ***A program exception is an error or some unexpected event such as a power failure.***

- ***Exception handling constructs allow for such events to be handled without the need for continual status checking to detect exceptions.***

- ***Using normal control constructs to detect exceptions in a sequence of nested procedure calls needs many additional statements to be added to the program and adds a significant timing overhead.***

# Exceptions in Java

```
class SensorFailureException extends Exception {
        SensorFailureException (String msg) {
                super (msg) ;
                Alarm.activate (msg) ;
        }
} // SensorFailureException
class Sensor {
        int readVal () throws SensorFailureException {
        try {
          int theValue = DeviceIO.readInteger () ;
          if (theValue < 0)
             throw new SensorFailureException ("Sensor failure") ;
         return theValue ;
        }
       catch (deviceIOException e)
              { throw new SensorFailureException (" Sensor
                                                  read error ") ; }
       } // readVal
} // Sensor
```

# Damage assessment techniques

- ***Checksums are used for damage assessment in data transmission***
- ***Redundant pointers can be used to check the integrity of data structures***
- ***Watch dog timers can check for non-terminating processes. If no response after a certain time, a problem is assumed***

# Safe sort procedure

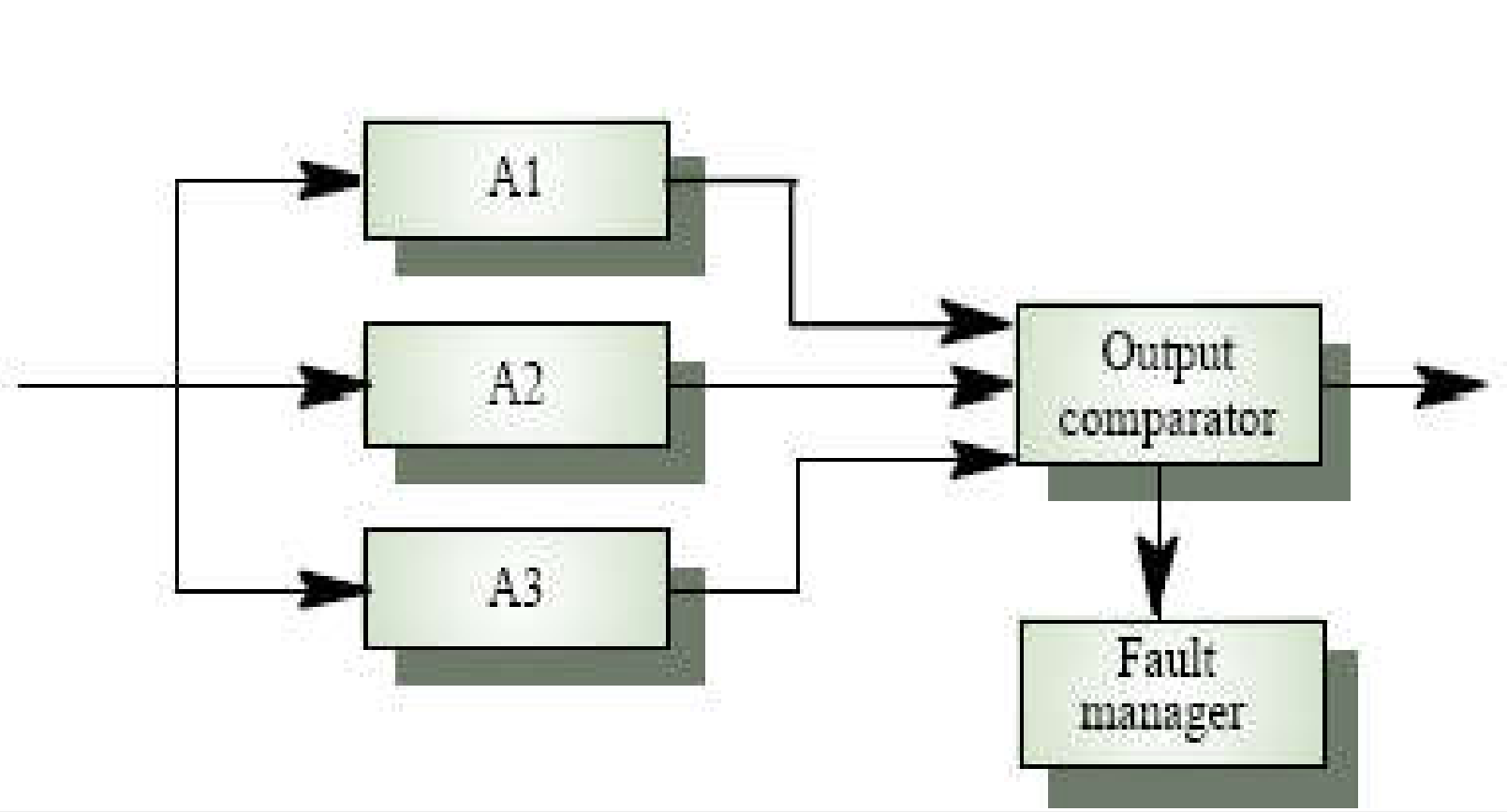
- ***Sort operation monitors its own execution and assesses if the sort has been correctly executed***
- ***Maintains a copy of its input so that if an error occurs, the input is not corrupted***
- ***Based on identifying and handling exceptions***
- ***Possible in this case as 'valid' sort is known. However, in many cases it is difficult to write validity checks***

# Key points

- ***Fault tolerant software can continue in execution in the presence of software faults***
- ***Fault tolerance requires failure detection, damage assessment, recovery and repair***

- ***Defensive programming is an approach to fault tolerance that relies on the inclusion of redundant checks in a program***
- ***Exception handling facilities simplify the process of defensive programming***

# Hardware reliability with TMR



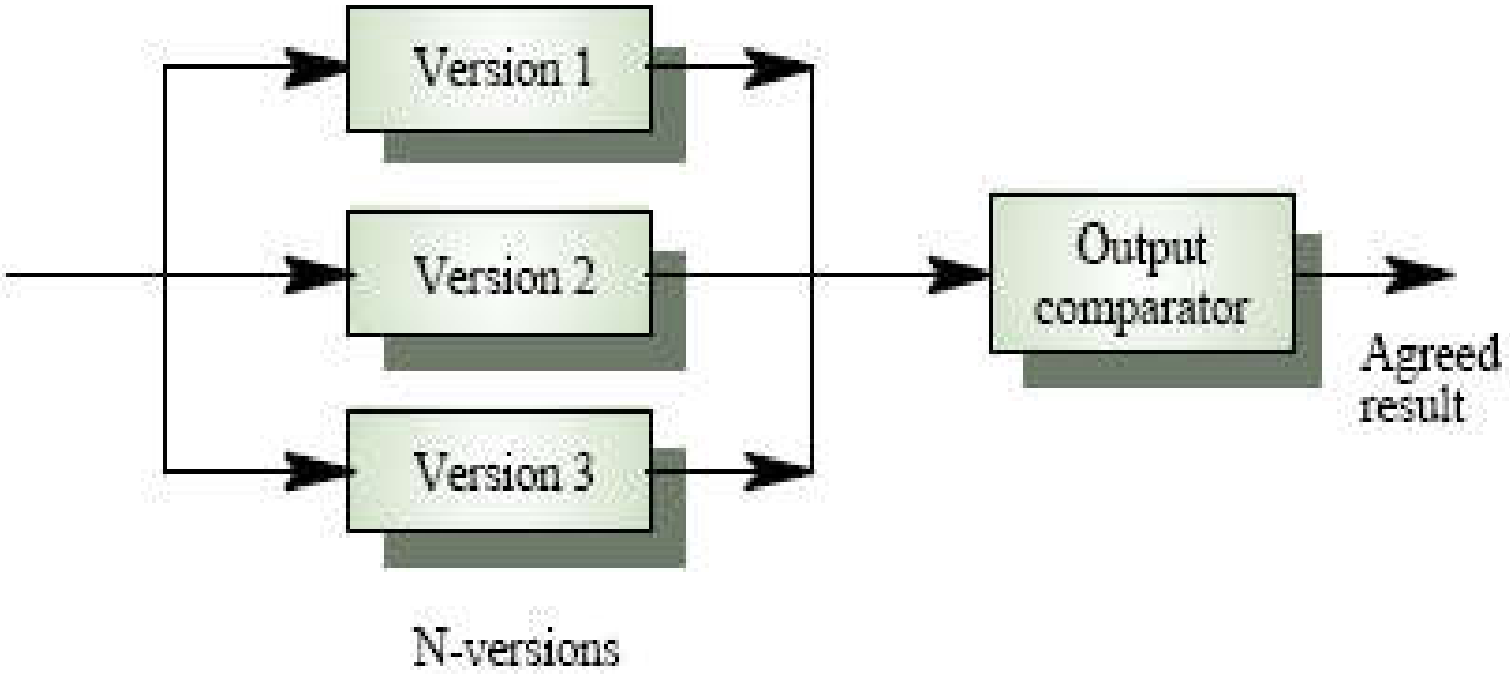
# Output selection

- ***The output comparator is a (relatively) simple hardware unit.***
- ***It compares its input signals and, if one is different from the others, it rejects it. Essentially, selection of the actual output depends on the majority vote.***
- ***The output comparator is connected to a fault management unit that can either try to repair the faulty unit or take it out of service.***

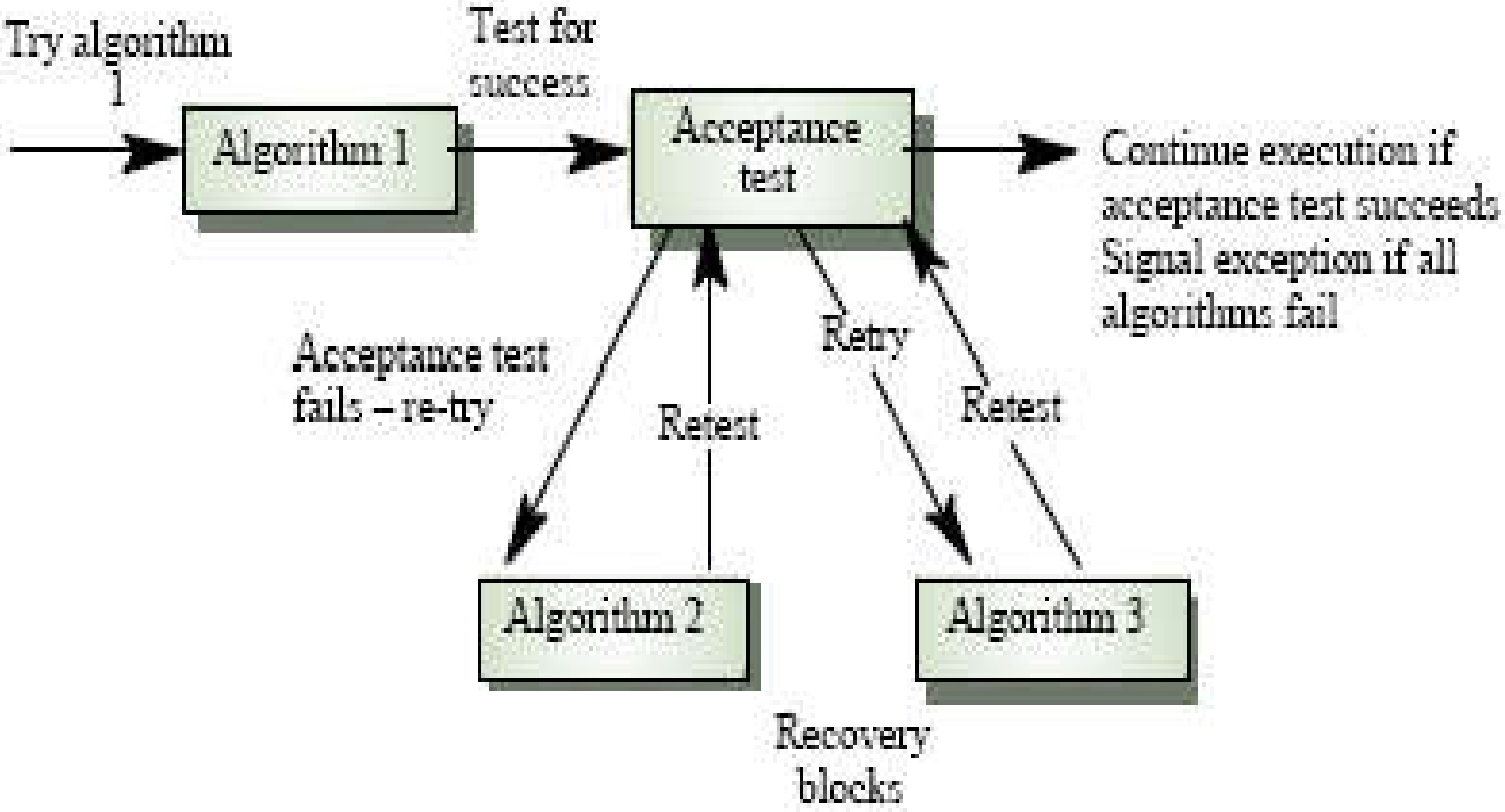
# Design diversity

- ***Different versions of the system are designed and implemented in different ways. They therefore ought to have different failure modes.***
- ***Different approaches to design (e.g object oriented and function oriented)***
  - Implementation in different programming languages
  - Use of different tools and development environments
  - Use of different algorithms in the

# N-version programming



# Recovery blocks



# Problems with design diversity

- ***Teams are not culturally diverse so they tend to tackle problems in the same way***
- ***Characteristic errors***
  - Different teams make the same mistakes. Some parts of an implementation are more difficult than others so all teams tend to make mistakes in the same place.
  - Specification errors
  - If there is an error in the specification then this is reflected in all implementations
  - This can be addressed to some extent by using multiple specification representations

# Specification dependency

- ***Both approaches to software redundancy are susceptible to specification errors. If the specification is incorrect, the system could fail***
- ***This is also a problem with hardware but software specifications are usually more complex than hardware specifications and harder to validate***
- ***This has been addressed in some cases by developing separate***

# Is software redundancy needed?

- ***Unlike hardware, software faults are not an inevitable consequence of the physical world***
- ***Some people therefore believe that a higher level of reliability and availability can be attained by investing effort in reducing software complexity.***
- ***Redundant software is much more complex so there is scope for a range of additional errors that affect the system reliability but are caused by the existence of the fault-tolerance controllers.***

# Structured programming

- ***First discussed in the 1970's***
- ***Programming without gotos***
- ***While loops and if statements as the only control statements.***
- ***Top-down design.***
- ***Important because it promoted thought and discussion about programming***
- ***Leads to programs that are easier to read and understand***

# Error-prone constructs

- ***Floating-point numbers***
  - Inherently imprecise. The imprecision may lead to invalid comparisons
- ***Pointers***
  - Pointers referring to the wrong memory areas can corrupt data. Aliasing can make programs difficult to understand and change
- ***Dynamic memory allocation***
- ***Parallelism***
  - Can result in subtle timing errors because of unforeseen interaction between parallel processes

# Error-prone constructs

- ***Recursion***
  - Errors in recursion can cause memory overflow
- ***Interrupts***
  - Interrupts can cause a critical operation to be terminated and make a program difficult to understand. they are comparable to goto statements.
- ***Inheritance***
  - Code is not localised. This can result in unexpected behaviour when changes are made and problems of understanding
- ***These constructs don't have to be avoided but they must be used with great care.***

# Information hiding

- ***Information should only be exposed to those parts of the program which need to access it. This involves the creation of objects or abstract data types which maintain state and operations on that state***

- ***This avoids faults for three reasons:***
  - the probability of accidental corruption of information
  - the information is surrounded by 'firewalls' so that problems are less likely to spread to other parts of the program
  - as all information is localised, the programmer is less likely to make errors and reviewers are more likely to find errors

# Reliable software processes

- ***To ensure a minimal number of software faults, it is important to have a well-defined, repeatable software process***
- ***A well-defined repeatable process is one that does not depend entirely on individual skills; rather can be enacted by different people***
- ***For fault minimisation, it is clear that the process activities should include significant verification and validation***

# Key points

- ***Dependability in a system can be achieved through fault avoidance and fault tolerance***
- ***Some programming language constructs such as gotos, recursion and pointers are inherently error-prone***
- ***Data typing allows many potential faults to be trapped at compile time.***