

# Lecture 9

## Dependability; safety-critical systems

Kari Systä

17.3.2014

Week	Lecture	Exercise
10.3	Quality in general; Quality management systems	Patterns
17.3	Dependable and safety-critical systems	ISO9001
24.3	Work planning; effort estimation	Code inspections
31.3	Version and configuration management	Effort estimation
7.4	Role of software architecture; product families; software evolution	?
14.4	Specifics of some domains, e.g. web system and/or embedded and real time systems	Break?
21.4	Easter	Break?
28.4	Software business, software start-ups	?
5.5	Last lecture; summary; recap for exam	?

# Safety-critical and dependable systems

## Learning goals

- Understand role of software in critical systems
- Basic understanding of issues and methods
- Sommerville chapters 11-13

# Introduction: Therac-25 incident

- This is very famous case. See for instance:  
[http://courses.cs.vt.edu/professionalism/Therac\\_25/Therac\\_1.html](http://courses.cs.vt.edu/professionalism/Therac_25/Therac_1.html)
- Therac was a radiation therapy device with two modes:
  1. High energy to be used through HW filters
  2. Lower energy without these devices
- Earlier models had HW protection to prevent using of high energy without HW protection
- In the new model, the protection was done in Software

# A safety analysis was made during design

The assumptions:

- (1) Programming errors have been reduced by extensive testing on a hardware simulator and under field conditions on teletherapy units. Any residual software errors are not included in the analysis.
- (2) Program software does not degrade due to wear, fatigue, or reproduction process.
- (3) Computer execution errors are caused by faulty hardware components and by "soft" (random) errors induced by alpha particles and electromagnetic noise.

# But

- The system gave overdoses because high energy was sent without filters
- There were several problems, also in the user interface and programming errors like:
  - a one-byte counter in a testing routine frequently overflowed; if an operator provided manual input to the machine at the precise moment that this counter overflowed, the interlock would fail

# Analysis said later

Basic software-engineering principles that apparently were violated with the Therac-25 include:

- Documentation should not be an afterthought.
- Software quality assurance practices and standards should be established.
- Designs should be kept simple.
- Ways to get information about errors -- for example, software audit trails -- should be designed into the software from the beginning.
- The software should be subjected to extensive testing and formal analysis at the module and software level; system testing alone is not adequate.

# Now think of all SW you depend on!

Digital fly-by-wire technologies of airplanes.

Digital phone network

- You should always be able to call 112



ABS break systems of cars



Pacemakers (luckily I do not need)



# Some terms

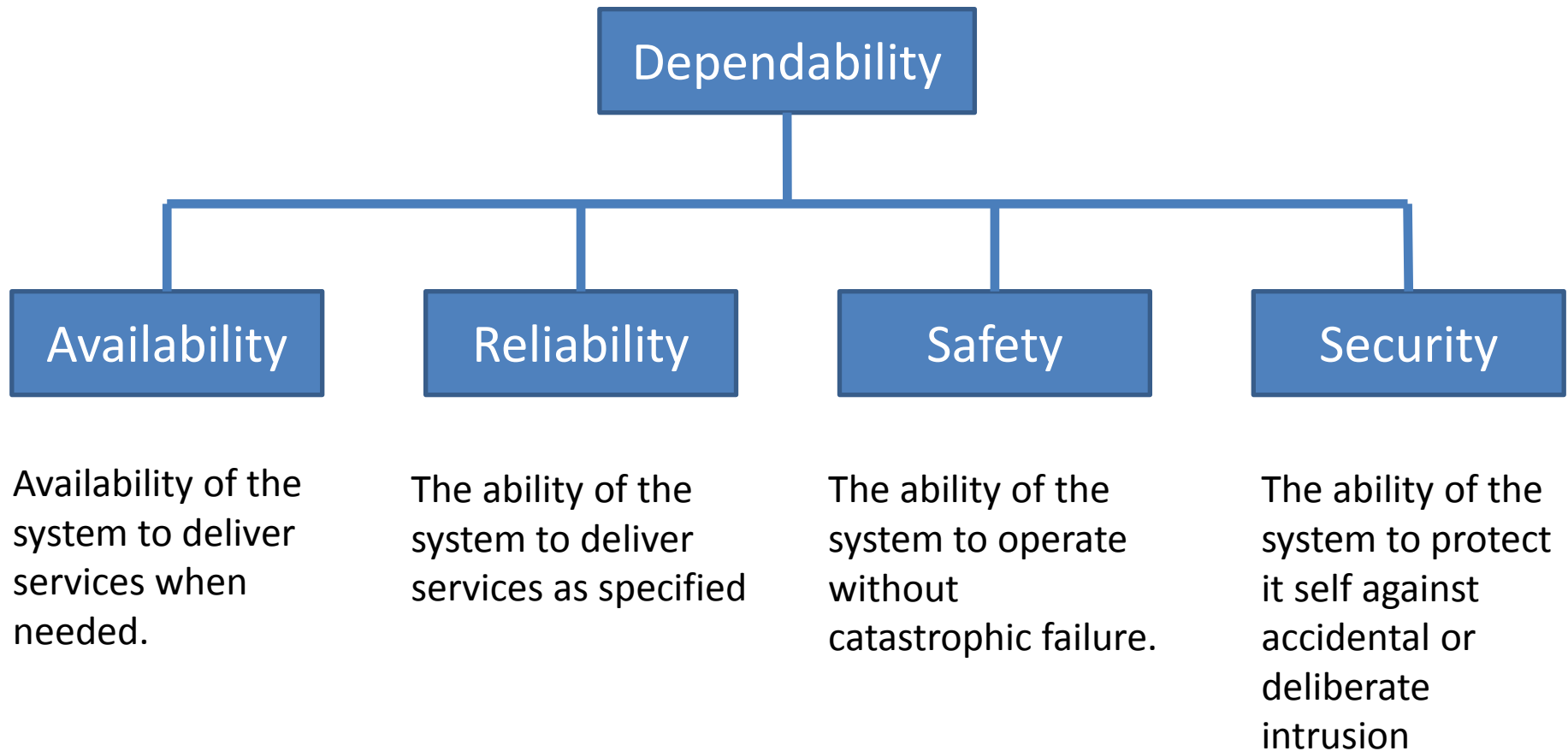
- Critical software/system
- Dependable software/system
- Safety-critical software

# Sources of problems

- 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

# Dependability

## (Sommeville Fig 11.1)



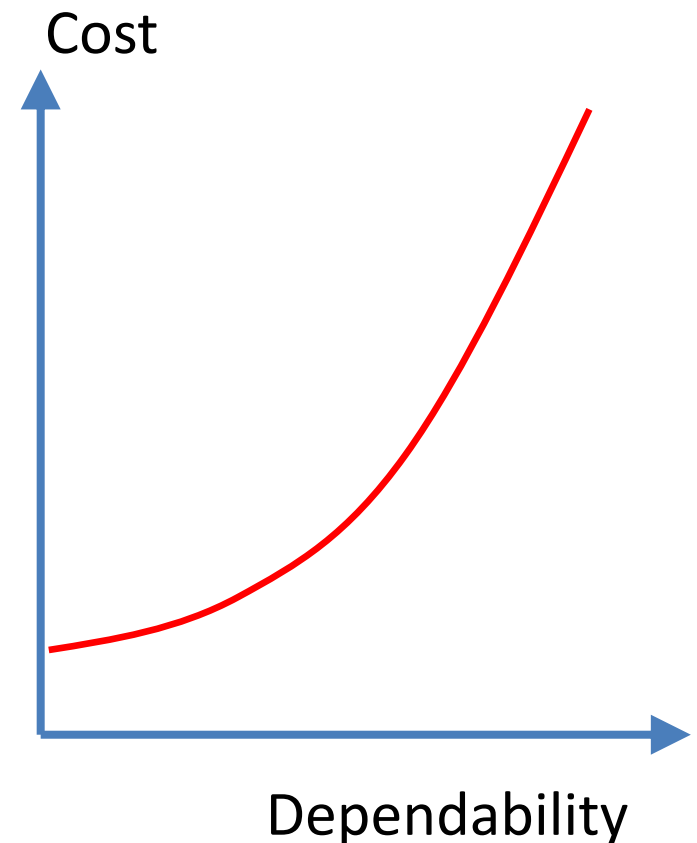
# Other dependability properties

- **Reparability**
  - 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.

# In the course of distributed system we have used

- Availability
- Reliability
- Safety
- Maintainability
  
- Fault tolerance

- 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 and regulators that the required levels of dependability have been achieved.



# About availability

- Usually on percentage
  - For example 99.95% means that system is down 0.05% of the time
    - Means about 4 hours and 20 minutes per year
- However, we also need to consider
  - Number of users affected
  - Length of single break
  - ... criticality of the system

# About reliability

## (Sommerville Figure 11.3)

Term	Description
Human error or mistake	Human behavior that results in the introduction of faults into a system. For example, in the wilderness weather system, a programmer might decide that the way to compute the time for the next transmission is to add 1 hour to the current time. This works except when the transmission time is between 23.00 and midnight (midnight is 00.00 in the 24-hour clock).
System fault	A characteristic of a software system that can lead to a system error. The fault is the inclusion of the code to add 1 hour to the time of the last transmission, without a check if the time is greater than or equal to 23.00.
System error	An erroneous system state that can lead to system behavior that is unexpected by system users. The value of transmission time is set incorrectly (to 24.XX rather than 00.XX) when the faulty code is executed.
System failure	An event that occurs at some point in time when the system does not deliver a service as expected by its users. No weather data is transmitted because the time is invalid.



# Some faults are more relevant than others

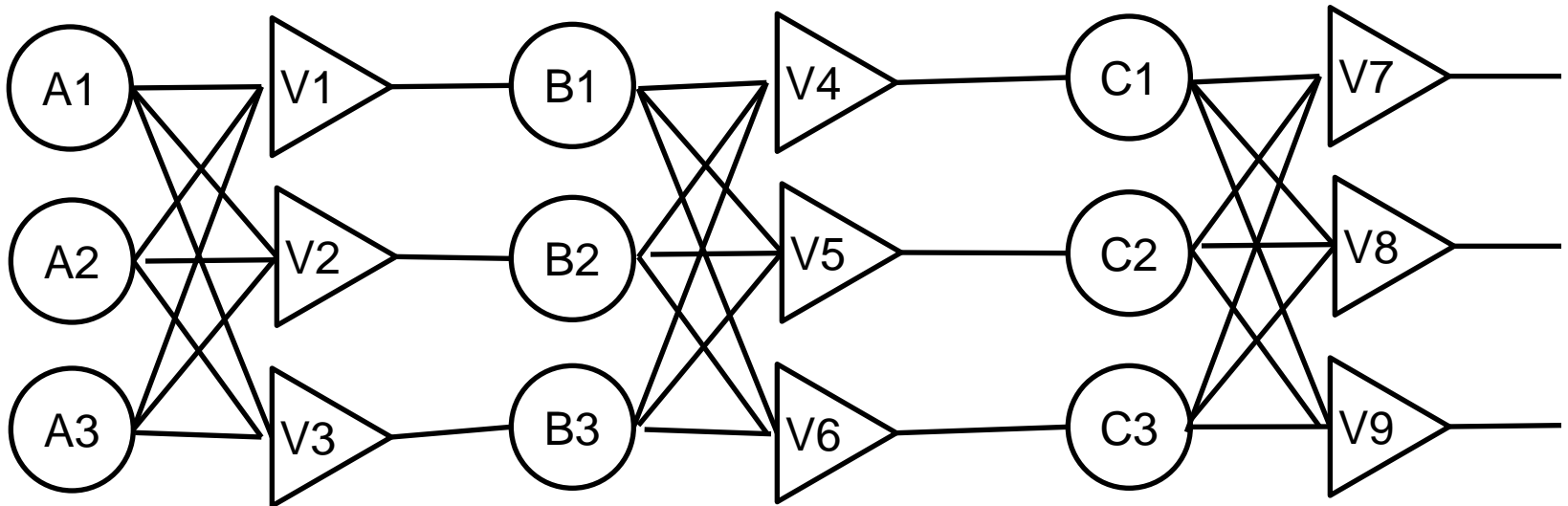
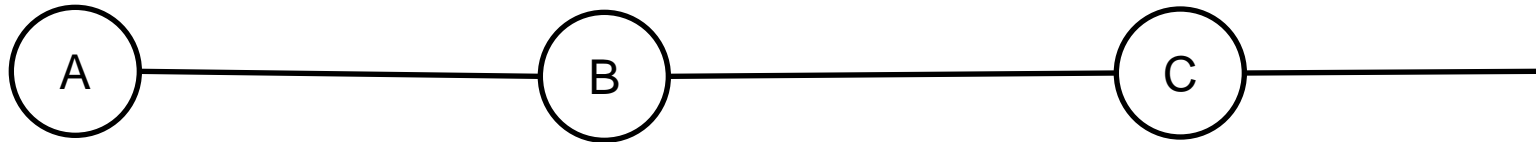
- I study at IBM showed that removing of 60% of known bugs increased reliability only by 3%
  - Many of the faults are likely to cause failures only after using the system for thousands of months
  - Some faults never lead to failures
- Users adapt their behavior to avoid system features that may fail for them.
- A program with known faults may therefore still be perceived as reliable by its users.

# Techniques for failure prevention

- Fault avoidance
  - Development techniques 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.

# Physical redundancy

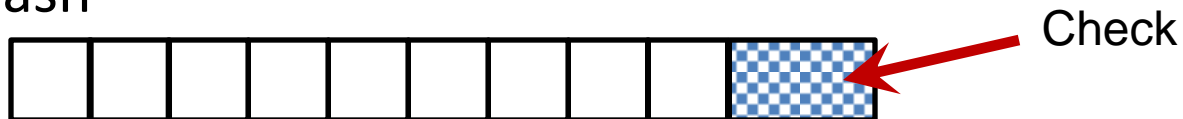
(Example "Triple modular redundancy")



# Redundant information

- Error detection

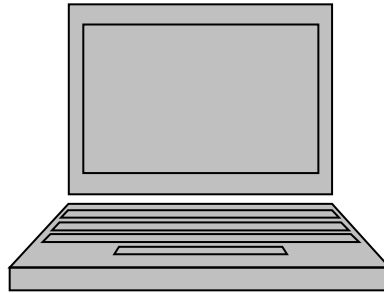
- Parity bit is the simplest and most known
- cyclic-redundancy check)
- Hash



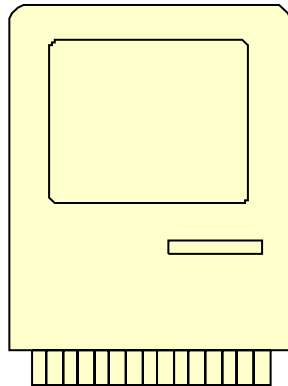
- error correction

- More check data
- "Hamming code"

# But how to use redundancy in SW?



C++



Java



# Safety

- “system’s ability to operate, normally or abnormally, without danger of causing human injury or death and without damage to the system’s environment.
- Important as most devices whose failure is critical now incorporate software-based control systems.
  - Transport
  - Medical

# Categories in Sommerville

- Primary safety-critical systems
  - Embedded software systems whose failure can cause the associated hardware to fail and directly threaten people. Example is the insulin pump control system.
- Secondary safety-critical systems
  - Systems whose failure results in faults in other (socio-technical) systems, which can then have safety consequences. For example, a patient management system in hospital is safety-critical as failure may lead to inappropriate treatment being prescribed.

Firefox

SE9 web index

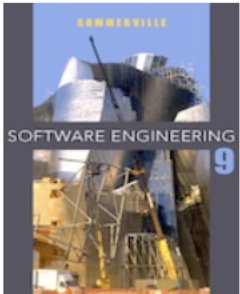
http://ifs.host.cs.st-andrews.ac.uk/Books/SE9/CaseStudies/MHCPMS/index.html

MHC-PMS

# SOFTWARE ENGINEERING 9

---

[Home](#) [Presentations](#) [Instructor's Guide](#) [Case Studies](#) [Figures](#) [Sample Chapters](#) [Web Chapters](#)



## MHC-PMS: A patient management system for mental health care

This case study describes a system that I have called MHC-PMS, which is a real system (although that is not its real name) which was used (and may still be used) in a number of UK hospitals, including hospitals in Scotland. The system is designed for use in clinics attended by patients suffering from mental health problems and records details of their consultations and conditions. It is separate from a more general patient records system as more detailed information has to be maintained and the system has to be set up to generate letters and reports of different types and to help ensure that the laws pertaining to mental health are maintained by staff treating patients.

This is a secondary safety-critical system as system failure can lead to decisions that compromise the safety of the patient or the medical staff caring for the patient.

### Use of this case study in teaching

I use this case study to discuss general issues of around the design of information systems where the system dependability is important and security is a significant concern. It is particularly useful for highlighting requirements conflicts as there is a clear conflict between requirements for patient privacy and safety requirements for maintaining the safety of the patient and their carers.

### Supporting documents

Case study description ([PDF](#), from Chapter 1)

System overview ([PPTX](#))

System requirements specification and discussion of requirements conflicts ([PDF](#))

An integrated approach to dependability requirements engineering. This is a supporting paper that I wrote on a method for deriving dependability requirements that uses this system as an example ([PDF](#))

17.3.2014

TIE-21100/21101; K.Systä

24



# Safety terminology

## (Sommerville Figure 11.6)

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. An overdose of insulin 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 measures blood glucose is an example of a hazard.
Damage	A measure of the loss resulting from a mishap. Damage can range from many people being killed as a result of an accident to minor injury or property damage. Damage resulting from an overdose of insulin could be serious injury or the death of the user of the insulin pump.
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. When an individual death is a possibility, a reasonable assessment of hazard severity is 'very high'.
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 in which the hazard could occur). The probability of a sensor failure in the insulin pump that results in an overdose is probably low.
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 the hazard will lead to an accident. The risk of an insulin overdose is probably medium to low.

# 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

# Software engineering for safety and high reliability

- **Requirements engineering**
- Risk-analysis
- Dependable programming
- Testing and validation
- Formal methods
- Software process concerns

# Requirements engineering for dependable systems

- In general, requirements work is especially important for dependable systems
- Dependability adds new requirements:
  - Functional requirements to define error checking and recovery facilities and protection against system failures.
  - Non-functional requirements defining the required reliability and availability of the system.
  - Excluding requirements that define states and conditions that must not arise.

# Software engineering for safety and high reliability

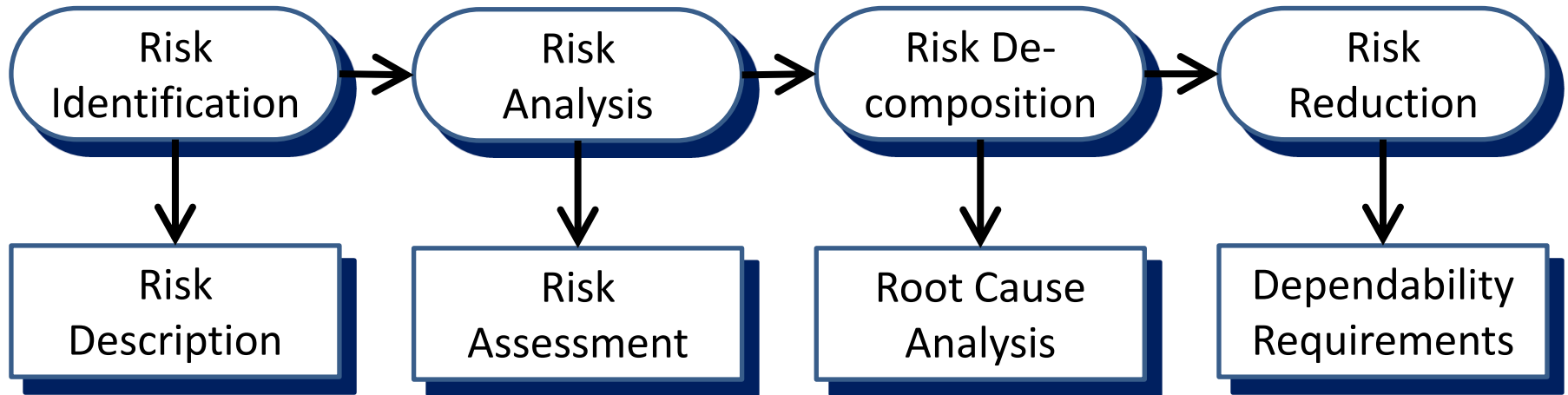
- Requirements engineering
- **Risk-analysis**
- Dependable programming
- Testing and validation
- Formal methods
- Software process concerns

# Risk analysis is an important part of development of critical systems

- Bring "what all can go wrong"-attitude to development
- Very common and recommended practice
- Compulsory task in many regulated domains
- Drives specification, implementation and testing
  - and also planning

# Risk analysis

(Figure 12.1 in Sommerville)



Risk identification = Hazard identification

Risk analysis = Hazard assessment

Risk decomposition = Hazard analysis

Risk reduction = safety requirements specification

# Different hazards

Physical hazards

Electrical hazards

Biological hazards

Service failure hazards

User/operation hazards

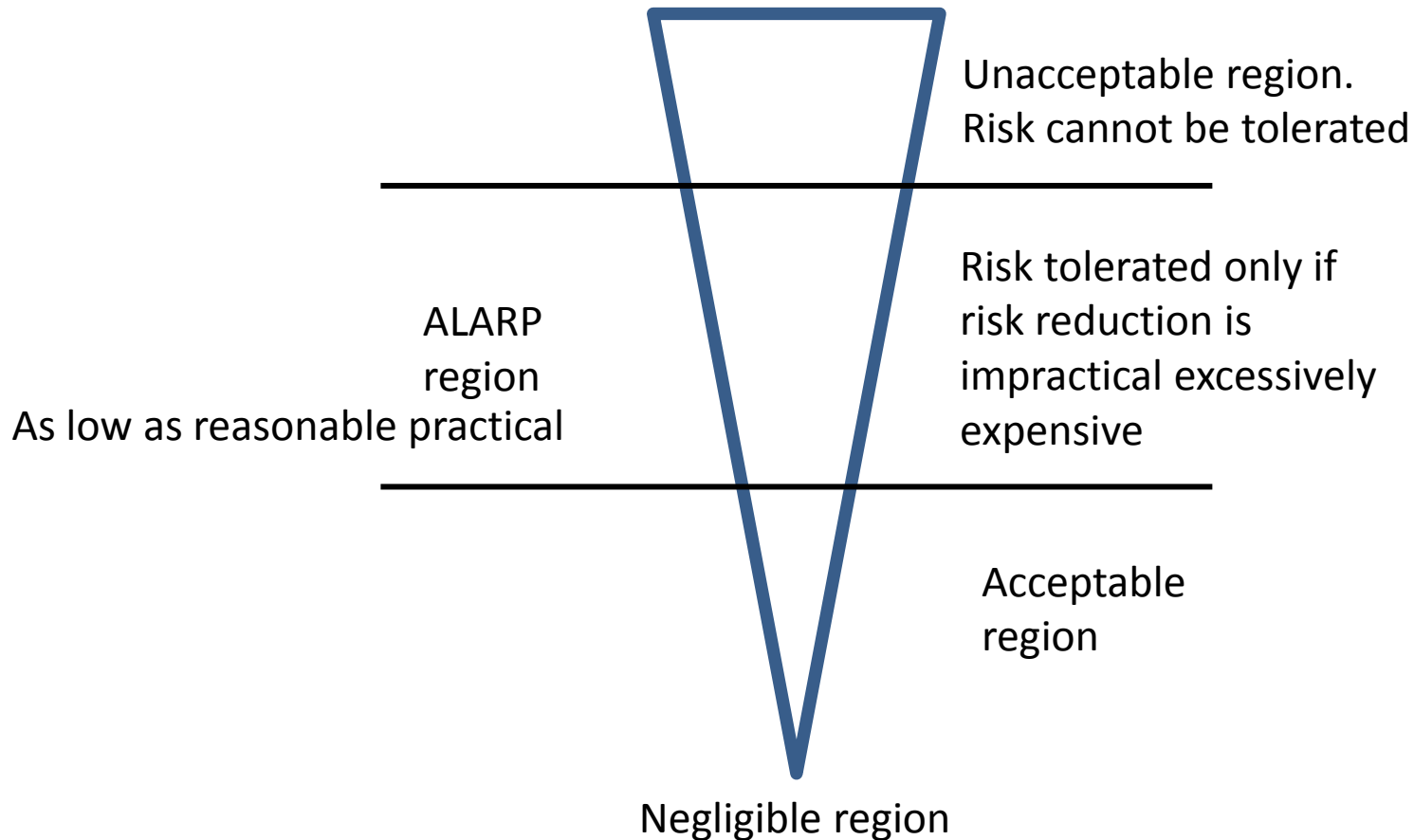
Etc.



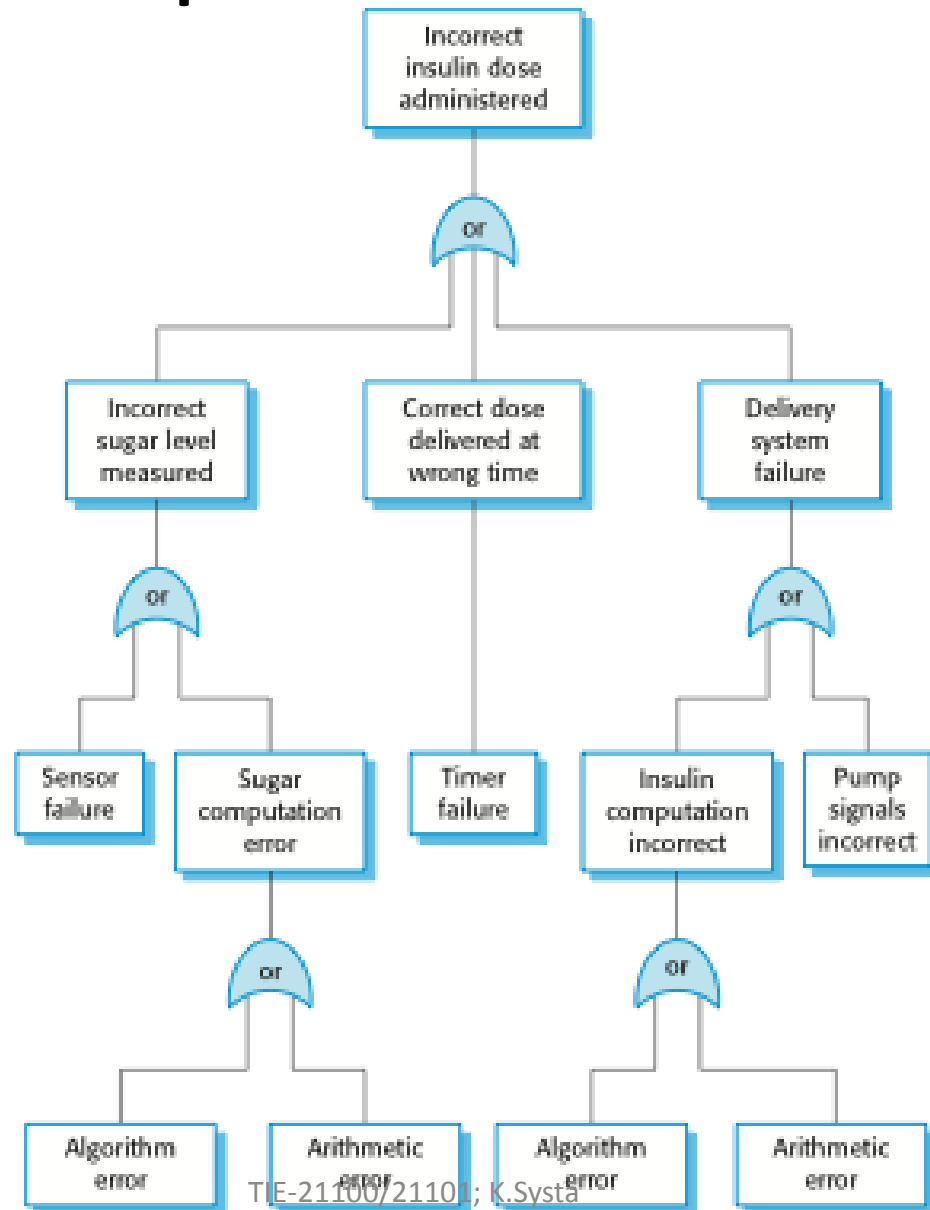
# Example (insulin pump)

- Insulin overdose (service failure).
- Insulin underdose (service failure).
- Power failure due to exhausted battery (electrical).
- Electrical interference with other medical equipment (electrical).
- Poor sensor and actuator contact (physical).
- Parts of machine break off in body (physical).
- Infection caused by introduction of machine (biological).
- Allergic reaction to materials or insulin (biological).

# Risk triangle



# Example of fault tree



# Software engineering for safety and high reliability

- Requirements engineering
- Risk-analysis
- **Dependable programming**
- Testing and validation
- Formal methods
- Software process concerns

# Guidelines for dependable programming

- Limit visibility
- Check all inputs (and return values)
- Provide handler for all exceptions
- Minimize use of error-prone constructs
- Provide restart capabilities
- Check array bounds
- Use timeouts when calling external components
- Name all constants that represent real-world values

# Software engineering for safety and high reliability

- Requirements engineering
- Risk-analysis
- Dependable programming
- **Testing and validation**
- Formal methods
- Software process concerns

# Testing and validation

- Test a lot ... and more
  - Plan test carefully
  - Document test plan and results
  - Measure test coverage
- 
- Many test cases should come from risk analysis

# Software engineering for safety and high reliability

- Requirements engineering
- Risk-analysis
- Dependable programming
- Testing and validation
- **Formal methods**
- Software process concerns



# Formal methods

- Mathematic- based techniques for the specification, development and verification of software and hardware systems.
- Formal specification
  - Precise, unambiguous, ...
- Specification analysis and proof
  - Consistency, missing of errors
- Transformational development
  - From correct specification to correct programs
- Program verification.

# Acceptance problems according to Sommerville

- Problem owners cannot understand a formal specification and so cannot assess if it is an accurate representation of their requirements.
- It is easy to assess the costs of developing a formal specification but harder to assess the benefits. Managers may therefore be unwilling to invest in formal methods.
- Software engineers are unfamiliar with this approach and are therefore reluctant to propose the use of FM.
- Formal methods are still hard to scale up to large systems.
- Formal specification is not really compatible with agile development methods.

# Software engineering for safety and high reliability

- Requirements engineering
- Risk-analysis
- Dependable programming
- Testing and validation
- Formal methods
- **Software process concerns**

# Notes on software engineering processes and safety-critical software

- It is often claimed that waterfall suits better to safety-critical systems than agile
  - Partly true, but there are successful adaptations of agile methods to safety critical systems
  - For example role of documentation is bigger in safety-critical systems
- Risk-analysis need to critical part of the process
- More discipline is needed
- Need to provide evidence for auditing and certification

# Example: medical devices

- Standards:
  - IEC 60601-1-4 (Safety of medical equipment)
  - ISO 14971 (Risk management)
  - IEC 62304 (SW life cycle)
  - IEC 62366 (Medical devices).
  - IEC 82304 (Healthcare SW systems) – work in progress
- 62304 defines three classes of systems:
  - A: No injury or damage to health is possible
  - B: Non-serious injury is possible
  - C: Death or serious injury is possible

# Additional issues

- While medical systems rely on ISO 60601 other areas are based on IEC 61508
  - 26262 (Automotive), 62279(Rail), 61511 (Process industries), 62513 (Nuclear), 62061 (Machinery), ...
- Many practices are HW based and talk about probabilities, "mean time between failure",
  - This concepts are hard to apply to SW

# An example

- London Ambulance Dispatching system
- Around 1992
- System was tried, but users returned to manual systems
- Managers required to resign

# Analysis discovered some causes

- Poor system design - Was designed for perfect world
  - Technologies always work, people do what they are told to do, and unexpected never happens
- Management problems
  - Changes, efforts to organize for higher productivity
  - "Fear of failure" culture
- Procurement process
- Timetable
- Inadequate testing and A&A
- Inadequate project management



# Safety-critical and dependable systems

## Learning goals

- Understand role of software in critical systems
- Basic understanding of issues and methods
- Sommerville chapters 11-13

Week	Lecture	Exercise
10.3	Quality in general; Quality management systems	Patterns
17.3	Dependable and safety-critical systems	ISO9001
24.3	Work planning; effort estimation	Code inspections
31.3	Version and configuration management	Effort estimation
7.4	Role of software architecture; product families; software evolution	?
14.4	Specifics of some domains, e.g. web system and/or embedded and real time systems	Break?
21.4	Easter	Break?
28.4	Software business, software start-ups	?
5.5	Last lecture; summary; recap for exam	?