Software Engineering

Lesson #10 - Lecture

Lesson #10 - Lecture

Your KBTU 202309 Software Engineering class information is updating …

Lesson #10 update is in progress

This will take around 2 hours to complete

Please, don’t turn off your computer

Part 2. System Dependability and Security

Part 2: Dependable systems

Agenda: **Lesson #10 - Software Engineering - Lecture**

Reliability engineering

1. Availability and reliability
2. Reliability requirements
3. Fault-tolerant architectures
4. Programming for reliability
5. Reliability measurement

Agenda: **Lesson #10 - Software Engineering - Lecture**

Safety engineering

1	Safety-critical systems
2	Safety requirements
3	Safety engineering processes
4	Safety cases
5	Q & A

Part 2. System Dependability and Security

```
#include<iostream>
Using namespace std;

int main()
{
	cout << “Reliability engineering” << endl;

	return 0;
}
```

Agenda: Lesson #10 - Software Engineering - Lecture

Reliability engineering

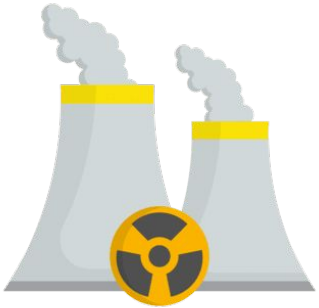
Software reliability

In general, software customers expect all software to be dependable. However, for non-critical applications, they may be willing to accept some system failures.

Software reliability

Some applications (critical systems) have very high reliability requirements and special software engineering techniques may be used to achieve this.

- Medical systems
- Telecommunications and power systems
- Aerospace systems
- Nuclear systems



Faults, errors and failures

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.

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

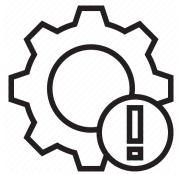
- The erroneous system state resulting from the fault may be transient and ‘corrected’ before an error arises
- The faulty code may never be executed

Faults and failures

Failures are a usually a result of system errors that are derived from faults in the system

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



Fault management

Fault avoidance

- The system 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 system are detected and repaired before delivery to the customer

Fault management

Fault detection

- Verification and validation techniques are used to discover and remove faults in a system before it is deployed

Fault tolerance

- The system is designed so that faults in the delivered software do not result in system failure

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

Reliability achievement

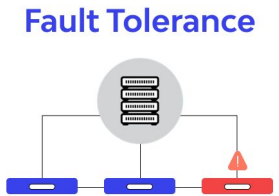
Fault detection and removal

- Verification and validation techniques are used that increase the probability of detecting and correcting errors before the system goes into service are used

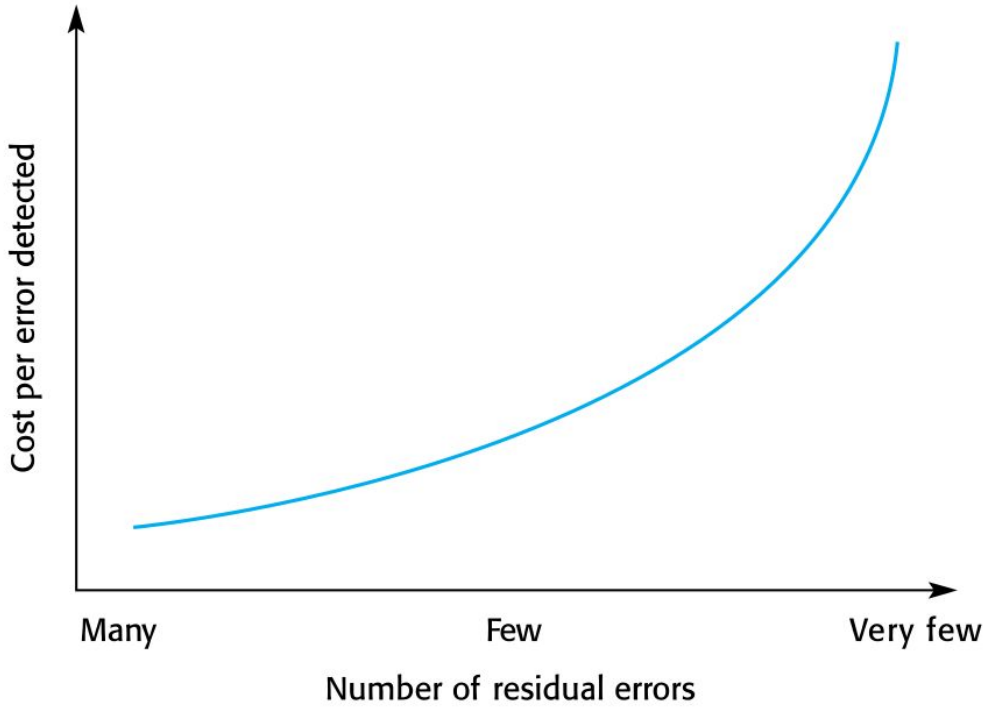
Reliability achievement

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



The increasing costs of residual fault removal



Agenda: **Lesson #10 - Software Engineering - Lecture**

Reliability engineering

1. **Availability and reliability**
2. Reliability requirements
3. Fault-tolerant architectures
4. Programming for reliability
5. Reliability measurement

%99.95

Web App

%99.99

SQL database

WHAT YOUR SLA MEANS

99.999%

(five nines)

unavailable for up to
6 minutes per year

99.99%

(four nines)

unavailable for up to
60 minutes per year

99.9%

(three nines)

unavailable for up to
10 hours per year

Availability and reliability

Reliability

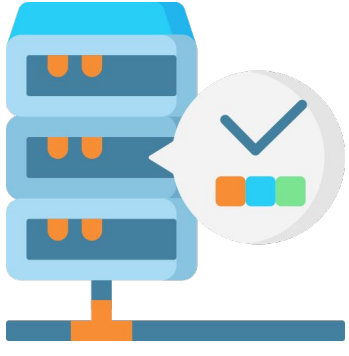
- The probability of failure-free system operation over a specified time in a given environment for a given purpose

Availability

- The probability that a system, at a point in time, will be operational and able to deliver the requested services

Availability and reliability

Both of these attributes can be expressed quantitatively e.g. availability of 0.999 means that the system is up and running for 99.9% of the time



Reliability and specifications

Reliability can only be defined formally with respect to a system specification i.e. a failure is a deviation from a specification

However, many specifications are incomplete or incorrect – hence, a system that conforms to its specification may ‘fail’ from the perspective of system users

Reliability and specifications

Furthermore, users don’t read specifications so don’t know how the system is supposed to behave

Therefore perceived reliability is more important in practice

Agenda: **Lesson #10 - Software Engineering - Lecture**

Reliability engineering

1. Availability and reliability
2. **Reliability requirements**
3. Fault-tolerant architectures
4. Programming for reliability
5. Reliability measurement

System reliability requirements

Functional reliability requirements define system and software functions that avoid, detect or tolerate faults in the software and so ensure that these faults do not lead to system failure

Software reliability requirements may also be included to cope with hardware failure or operator error

System reliability requirements

Reliability is a measurable system attribute so non-functional reliability requirements may be specified quantitatively. These define the number of failures that are acceptable during normal use of the system or the time in which the system must be available

Reliability metrics

Reliability metrics are units of measurement of system reliability

System reliability is measured by counting the number of operational failures and, where appropriate, relating these to the demands made on the system and the time that the system has been operational

Reliability metrics

A long-term measurement programme is required to assess the reliability of critical systems

Metrics

- Probability of failure on demand
- Rate of occurrence of failures/Mean time to failure
- Availability

Agenda: **Lesson #10 - Software Engineering - Lecture**

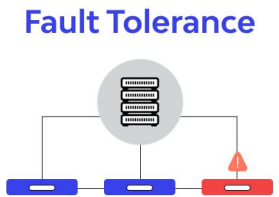
Reliability engineering

1. Availability and reliability
2. Reliability requirements
3. **Fault-tolerant architectures**
4. Programming for reliability
5. Reliability measurement

Fault tolerance

In critical situations, software systems must be fault tolerant

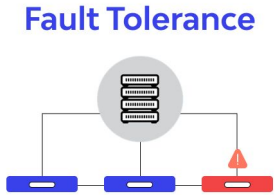
Fault tolerance is required where there are high availability requirements or where system failure costs are very high



Fault tolerance

Fault tolerance means that the system can continue in operation in spite of software failure

Even if the system has been proved to conform to its specification, it must also be fault tolerant as there may be specification errors or the validation may be incorrect



Fault-tolerant system architectures

Fault-tolerant systems architectures are used in situations where fault tolerance is essential. These architectures are generally all based on redundancy and diversity

Fault-tolerant system architectures

Examples of situations where dependable architectures are used:

- Flight control systems, where system failure could threaten the safety of passengers
- Reactor systems where failure of a control system could lead to a chemical or nuclear emergency
- Telecommunication systems, where there is a need for 24/7 availability

Agenda: **Lesson #10 - Software Engineering - Lecture**

Reliability engineering

1. Availability and reliability
2. Reliability requirements
3. Fault-tolerant architectures
4. **Programming for reliability**
5. Reliability measurement

Dependable programming

Good programming practices can be adopted that help reduce the incidence of program faults

These programming practices support

- Fault avoidance
- Fault detection
- Fault tolerance

Good practice guidelines for dependable programming

Dependable programming guidelines

1. Limit the visibility of information in a program
2. Check all inputs for validity
3. Provide a handler for all exceptions
4. Minimize the use of error-prone constructs

continues …

Good practice guidelines for dependable programming

Dependable programming guidelines

5. Provide restart capabilities
6. Check array bounds
7. Include timeouts when calling external components
8. Name all constants that represent real-world values

Agenda: **Lesson #10 - Software Engineering - Lecture**

Reliability engineering

1	Availability and reliability
2	Reliability requirements
3	Fault-tolerant architectures
4	Programming for reliability
5	**Reliability measurement**

Reliability measurement

To assess the reliability of a system, you have to collect data about its operation. The data required may include:

- The number of system failures given a number of requests for system services. This is used to measure the POFOD. This applies irrespective of the time over which the demands are made.

continues …

Reliability measurement

To assess the reliability of a system, you have to collect data about its operation. The data required may include:

- The time or the number of transactions between system failures plus the total elapsed time or total number of transactions. This is used to measure ROCOF and MTTF

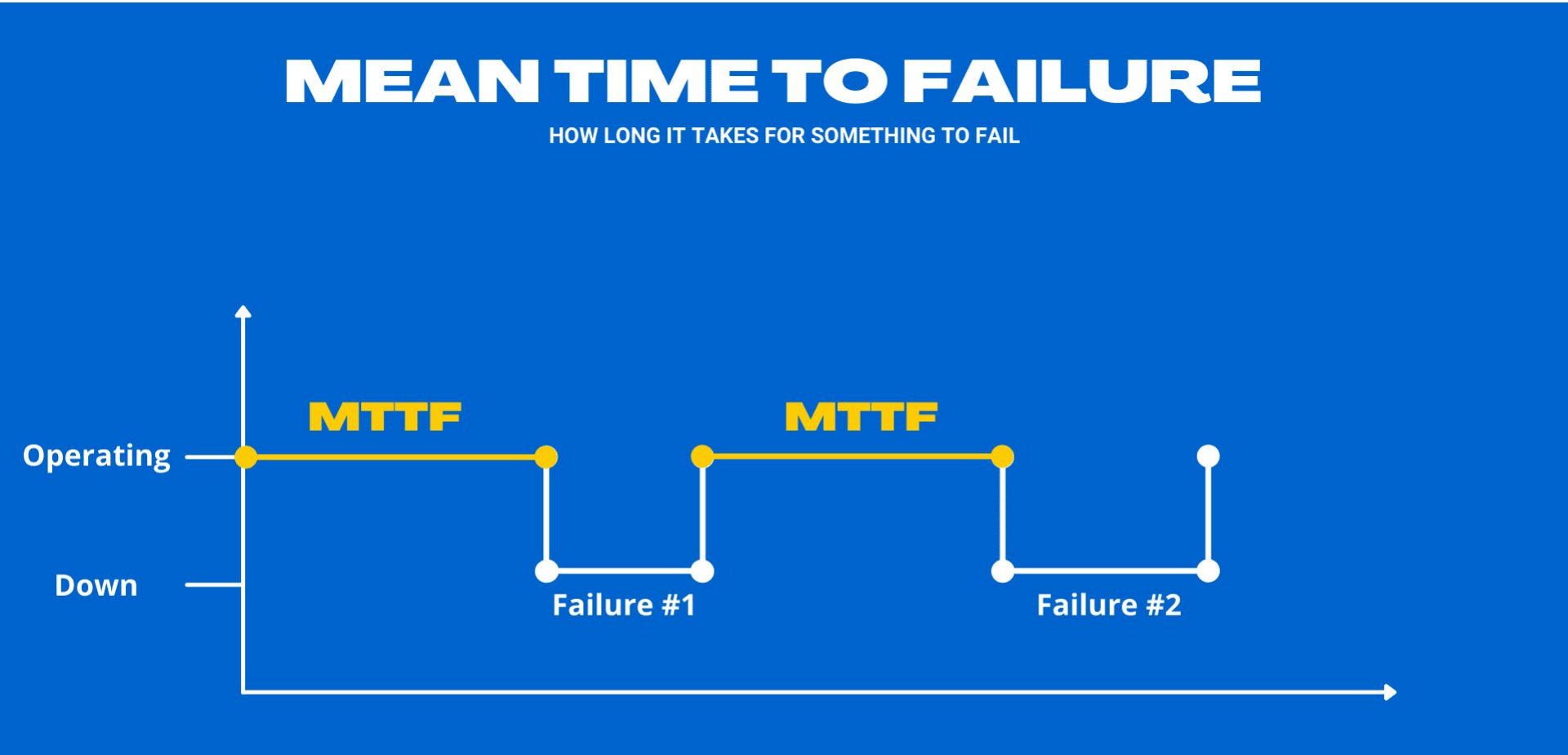
continues …

Reliability measurement

To assess the reliability of a system, you have to collect data about its operation. The data required may include:

- The repair or restart time after a system failure that leads to loss of service. This is used in the measurement of availability. Availability does not just depend on the time between failures but also on the time required to get the system back into operation

Reliability measurement

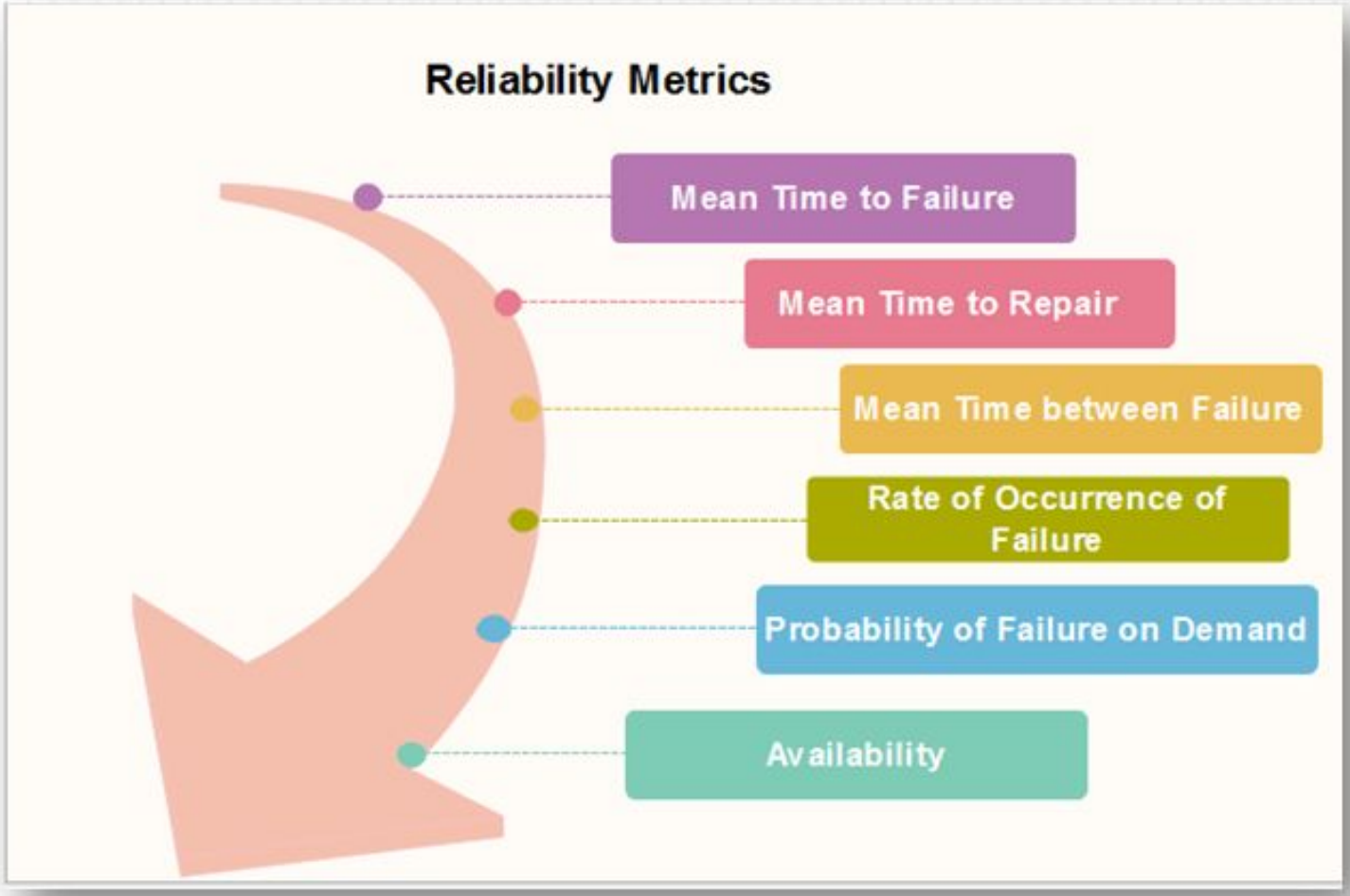


Reliability testing

Reliability testing (Statistical testing) involves running the program to assess whether or not it has reached the required level of reliability

This cannot normally be included as part of a normal defect testing process because data for defect testing is (usually) atypical of actual usage data

Reliability measurement therefore requires a specially designed data set that replicates the pattern of inputs to be processed by the system


Reliability Metrics
Mean Time to Failure
Mean Time to Repair
Mean Time between Failure
Rate of Occurrence of Failure
Probability of Failure on Demand
Availability

Part 2. System Dependability and Security

```
#include<iostream>
Using namespace std;

int main()
{
	cout << “Safety engineering” << endl;

	return 0;
}
```

Agenda: Lesson #10 - Software Engineering - Lecture

Safety engineering

1	Safety-critical systems
2	Safety requirements
3	Safety engineering processes
4	Safety cases
5	Q & A

Agenda: Lesson #10 - Software Engineering - Lecture

Safety engineering

A problem has been detected and Windows has been shut down to prevent damage
to your computer.

UNMOUNTABLE_BOOT_VOLUME

If this is the first time you've seen this error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

Technical Information:

*** STOP: 0x000000ED (0x80F128D0, 0xc000009c, 0x00000000, 0x00000000)

Safety engineering

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 important to consider software safety as most devices whose failure is critical now incorporate software-based control systems

Software in safety-critical systems

The system may be software-controlled so that the decisions made by the software and subsequent actions are safety-critical. Therefore, the software behaviour is directly related to the overall safety of the system.

Software in safety-critical systems

Software is extensively used for checking and monitoring other safety-critical components in a system. For example, all aircraft engine components are monitored by software looking for early indications of component failure. This software is safety-critical because, if it fails, other components may fail and cause an accident.

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 and reliability

Safety is concerned with ensuring system cannot cause damage irrespective of whether or not it conforms to its specification

- System reliability is essential for safety but is not enough
- Reliable systems can be unsafe

Unsafe reliable systems

There may be dormant faults in a system that are undetected for many years and only rarely arise

Specification errors

- If the system specification is incorrect then the system can behave as specified but still cause an accident

Unsafe reliable systems

There may be dormant faults in a system that are undetected for many years and only rarely arise

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

Agenda: Lesson #10 - Software Engineering - Lecture

Safety engineering

1. **Safety-critical systems**
2. Safety requirements
3. Safety engineering processes
4. Safety cases
5. Q & A

Safety critical systems

Systems where it is essential that system operation is always safe i.e. the system should never cause damage to people or the system's environment

Examples

- Control and monitoring systems in aircraft
- Process control systems in chemical manufacture
- Automobile control systems such as braking and engine management systems

Safety criticality

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.

Safety criticality

Secondary safety-critical systems

- Systems whose failure results in faults in other (socio-technical) systems, which can then have safety consequences
    - For example, the Mentcare system is safety-critical as failure may lead to inappropriate treatment being prescribed
    - Infrastructure control systems are also secondary safety-critical systems

Hazards

Situations or events that can lead to an accident

- Stuck valve in reactor control system
- Incorrect computation by software in navigation system
- Failure to detect possible allergy in medication prescribing system

Hazards do not inevitably result in accidents – accident prevention actions can be taken

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

Agenda: **Lesson #10 - Software Engineering - Lecture**

Safety engineering

1	Safety-critical systems
2	**Safety requirements**
3	Safety engineering processes
4	Safety cases
5	Q & A

Safety specification

The goal of safety requirements engineering is to identify protection requirements that ensure that system failures do not cause injury or death or environmental damage

Safety requirements may be ‘shall not’ requirements i.e. they define situations and events that should never occur

Safety specification

Functional safety requirements define:

- Checking and recovery features that should be included in a system
- Features that provide protection against system failures and external attacks

Hazard-driven analysis

- Hazard identification
- Hazard assessment
- Hazard analysis
- Safety requirements specification

Agenda: **Lesson #10 - Software Engineering - Lecture**

Safety engineering

1	Safety-critical systems
2	Safety requirements
3	**Safety engineering processes**
4	Safety cases
5	Q & A

Safety engineering processes

Safety engineering processes are based on reliability engineering processes

- Plan-based approach with reviews and checks at each stage in the process
- General goal of fault avoidance and fault detection
- Must also include safety reviews and explicit identification and tracking of hazards

Regulation

Regulators may require evidence that safety engineering processes have been used in system development

Regulation

For example:

- The specification of the system that has been developed and records of the checks made on that specification
- Evidence of the verification and validation processes that have been carried out and the results of the system verification and validation

Regulation

For example:

- Evidence that the organizations developing the system have defined and dependable software processes that include safety assurance reviews. There must also be records that show that these processes have been properly enacted

Agenda: **Lesson #10 - Software Engineering - Lecture**

Safety engineering

1	Safety-critical systems
2	Safety requirements
3	Safety engineering processes
4	**Safety cases**
5	Q & A

The system safety case

A safety case is:

- A documented body of evidence that provides a convincing and valid argument that a system is adequately safe for a given application in a given environment.

Arguments in a safety case can be based on formal proof, design rationale, safety proofs, etc. Process factors may also be included

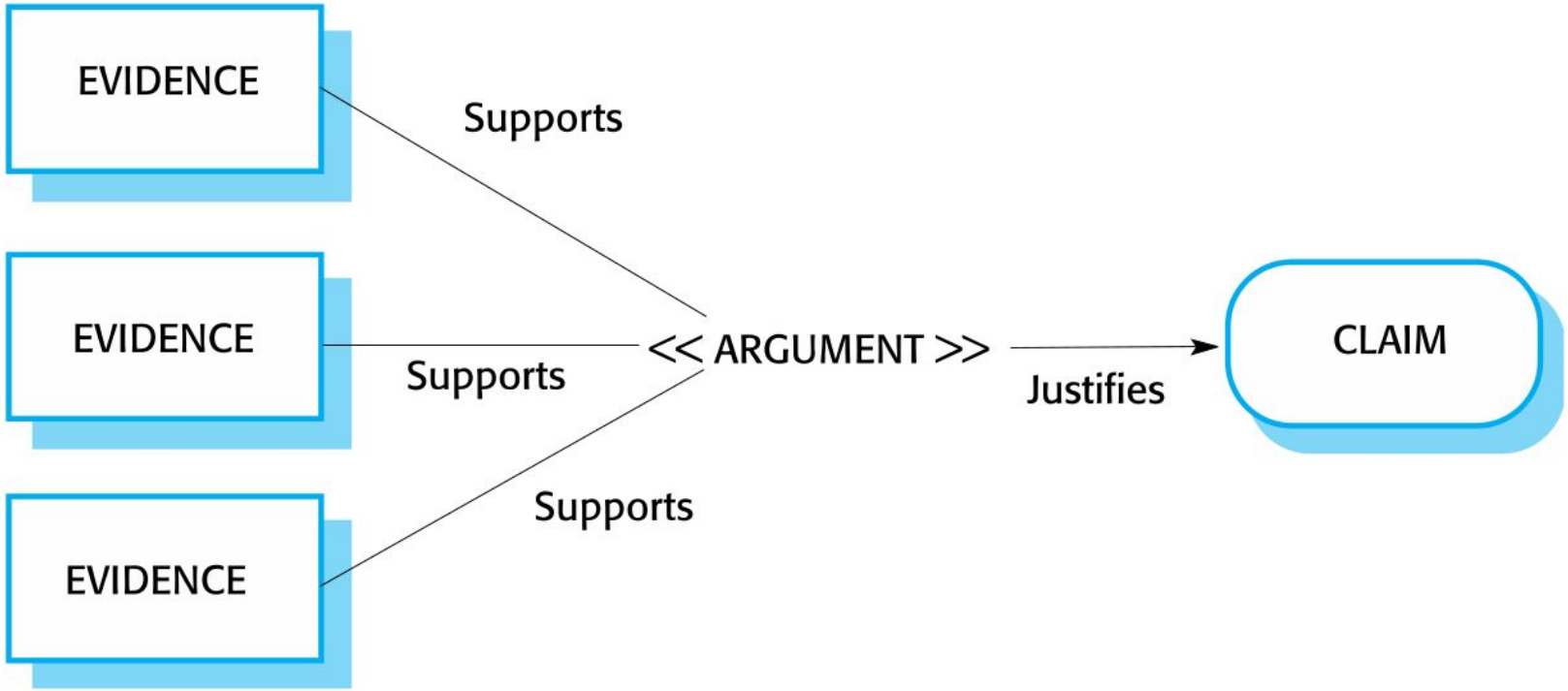
A software safety case is usually part of a wider system safety case that takes hardware and operational issues into account

Structured arguments

Safety cases should be based around structured arguments that present evidence to justify the assertions made in these arguments

The argument justifies why a claim about system safety and security is justified by the available evidence

Structured arguments



Agenda: **Lesson #10 - Software Engineering - Lecture**

Safety engineering

1	Safety-critical systems
2	Safety requirements
3	Safety engineering processes
4	Safety cases
5	**Q & A**

Agenda: Lesson #10 - Software Engineering - Lecture

Q & A