Chapter 12 – Safety Engineering

Topics covered

- ✧ Safety-critical systems
- ✧ Safety requirements
- ✧ Safety engineering processes
- ✧ Safety cases

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 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 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 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.
- 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-critical systems

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.
- ✧ 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.

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. 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.

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 rather than single failures.
- It is probably the case that anticipating all problem combinations, especially, in software controlled systems is impossible so achieving complete safety is impossible. Accidents are inevitable.

Software safety benefits

- Although software failures can be safety-critical, the use of software control systems contributes to increased system safety
  - Software monitoring and control allows a wider range of conditions to be monitored and controlled than is possible using electro-mechanical safety systems.
  - Software control allows safety strategies to be adopted that reduce the amount of time people spend in hazardous environments.
  - Software can detect and correct safety-critical operator errors.

Safety requirements

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.
- 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

Hazard identification

- Identify the hazards that may threaten the system.
- Hazard identification may be based on different types of hazard:
  - Physical hazards
  - Electrical hazards
  - Biological hazards
  - Service failure hazards
  - Etc.

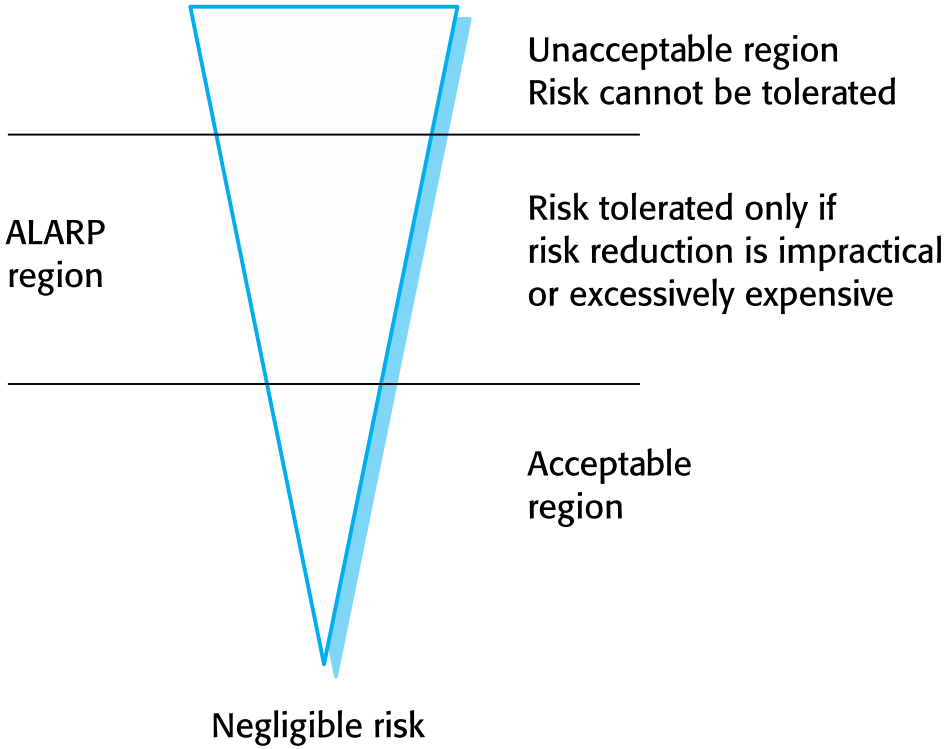
Insulin pump risks

- 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).

Hazard assessment

- The process is concerned with understanding the likelihood that a risk will arise and the potential consequences if an accident or incident should occur.
- Risks may be categorised as:
  - Intolerable. Must never arise or result in an accident
  - As low as reasonably practical(ALARP). Must minimise the possibility of risk given cost and schedule constraints
  - Acceptable. The consequences of the risk are acceptable and no extra costs should be incurred to reduce hazard probability

The risk triangle



Social acceptability of risk

- The acceptability of a risk is determined by human, social and political considerations.
- In most societies, the boundaries between the regions are pushed upwards with time i.e. society is less willing to accept risk
  - For example, the costs of cleaning up pollution may be less than the costs of preventing it but this may not be socially acceptable.
- Risk assessment is subjective
  - Risks are identified as probable, unlikely, etc. This depends on who is making the assessment.

Hazard assessment

- Estimate the risk probability and the risk severity.
- It is not normally possible to do this precisely so relative values are used such as ‘unlikely’, ‘rare’, ‘very high’, etc.
- The aim must be to exclude risks that are likely to arise or that have high severity.

Risk classification for the insulin pump

Identified hazard	Hazard probability	Accident severity	Estimated risk	Acceptability
1.Insulin overdose computation	Medium	High	High	Intolerable
2. Insulin underdose computation	Medium	Low	Low	Acceptable
3. Failure of hardware monitoring system	Medium	Medium	Low	ALARP
4. Power failure	High	Low	Low	Acceptable
5. Machine incorrectly fitted	High	High	High	Intolerable
6. Machine breaks in patient	Low	High	Medium	ALARP
7. Machine causes infection	Medium	Medium	Medium	ALARP
8. Electrical interference	Low	High	Medium	ALARP
9. Allergic reaction	Low	Low	Low	Acceptable

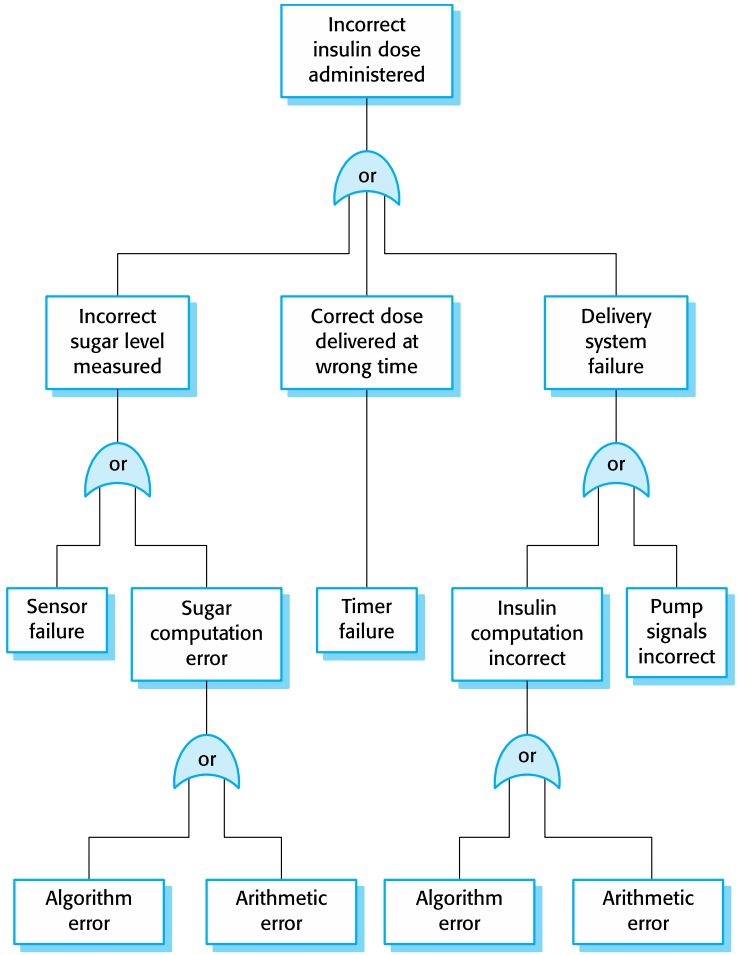
Hazard analysis

- Concerned with discovering the root causes of risks in a particular system.
- Techniques have been mostly derived from safety-critical systems and can be
  - Inductive, bottom-up techniques. Start with a proposed system failure and assess the hazards that could arise from that failure;
  - Deductive, top-down techniques. Start with a hazard and deduce what the causes of this could be.

Fault-tree analysis

- A deductive top-down technique.
- Put the risk or hazard at the root of the tree and identify the system states that could lead to that hazard.
- Where appropriate, link these with ‘and’ or ‘or’ conditions.
- A goal should be to minimise the number of single causes of system failure.

An example of a software fault tree



Fault tree analysis

- Three possible conditions that can lead to delivery of incorrect dose of insulin
  - Incorrect measurement of blood sugar level
  - Failure of delivery system
  - Dose delivered at wrong time
- By analysis of the fault tree, root causes of these hazards related to software are:
  - Algorithm error
  - Arithmetic error

Risk reduction

- The aim of this process is to identify dependability requirements that specify how the risks should be managed and ensure that accidents/incidents do not arise.
- Risk reduction strategies
  - Hazard avoidance;
  - Hazard detection and removal;
  - Damage limitation.

Strategy use

- Normally, in critical systems, a mix of risk reduction strategies are used.
- In a chemical plant control system, the system will include sensors to detect and correct excess pressure in the reactor.
- However, it will also include an independent protection system that opens a relief valve if dangerously high pressure is detected.

Insulin pump - software risks

- Arithmetic error
  - A computation causes the value of a variable to overflow or underflow;
  - Maybe include an exception handler for each type of arithmetic error.
- Algorithmic error
  - Compare dose to be delivered with previous dose or safe maximum doses. Reduce dose if too high.

Examples of safety requirements

SR1: The system shall not deliver a single dose of insulin that is greater than a specified maximum dose for a system user.

SR2: The system shall not deliver a daily cumulative dose of insulin that is greater than a specified maximum daily dose for a system user.

SR3: The system shall include a hardware diagnostic facility that shall be executed at least four times per hour.

SR4: The system shall include an exception handler for all of the exceptions that are identified in Table 3.

SR5: The audible alarm shall be sounded when any hardware or software anomaly is discovered and a diagnostic message, as defined in Table 4, shall be displayed.

SR6: In the event of an alarm, insulin delivery shall be suspended until the user has reset the system and cleared the alarm.

Safety engineering processes

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
- 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.
  - 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.

Agile methods and safety

- Agile methods are not usually used for safety-critical systems engineering
  - Extensive process and product documentation is needed for system regulation. Contradicts the focus in agile methods on the software itself.
  - A detailed safety analysis of a complete system specification is important. Contradicts the interleaved development of a system specification and program.
- Some agile techniques such as test-driven development may be used

Safety assurance processes

- Process assurance involves defining a dependable process and ensuring that this process is followed during the system development.
- Process assurance focuses on:
  - Do we have the right processes? Are the processes appropriate for the level of dependability required. Should include requirements management, change management, reviews and inspections, etc.
  - Are we doing the processes right? Have these processes been followed by the development team.
- Process assurance generates documentation
  - Agile processes therefore are rarely used for critical systems.

Processes for safety assurance

- Process assurance is important for safety-critical systems development:
  - Accidents are rare events so testing may not find all problems;
  - Safety requirements are sometimes ‘shall not’ requirements so cannot be demonstrated through testing.
- Safety assurance activities may be included in the software process that record the analyses that have been carried out and the people responsible for these.
  - Personal responsibility is important as system failures may lead to subsequent legal actions.

Safety related process activities

- Creation of a hazard logging and monitoring system.
- Appointment of project safety engineers who have explicit responsibility for system safety.
- Extensive use of safety reviews.
- Creation of a safety certification system where the safety of critical components is formally certified.
- Detailed configuration management (see Chapter 25).

Hazard analysis

- Hazard analysis involves identifying hazards and their root causes.
- There should be clear traceability from identified hazards through their analysis to the actions taken during the process to ensure that these hazards have been covered.
- A hazard log may be used to track hazards throughout the process.

A simplified hazard log entry

Hazard Log	Page 4: Printed 20.02.2012				
System: Insulin Pump System *Safety Engineer:* James Brown				*File*: InsulinPump/Safety/HazardLog *Log version*: 1/3	
Identified Hazard	Insulin overdose delivered to patient				
Identified by	Jane Williams				
Criticality class	1				
Identified risk	High				
Fault tree identified	YES	*Date*	24.01.07	*Location*	Hazard Log, Page 5
Fault tree creators	Jane Williams and Bill Smith				
Fault tree checked	YES	*Date*	28.01.07	*Checker*	James Brown

Hazard log (2)

System safety design requirements
1. The system shall include self-testing software that will test the sensor system, the clock, and the insulin delivery system.
2. The self-checking software shall be executed once per minute.
3. In the event of the self-checking software discovering a fault in any of the system components, an audible warning shall be issued and the pump display shall indicate the name of the component where the fault has been discovered. The delivery of insulin shall be suspended.
4. The system shall incorporate an override system that allows the system user to modify the computed dose of insulin that is to be delivered by the system.
5. The amount of override shall be no greater than a pre-set value (maxOverride), which is set when the system is configured by medical staff.

Safety reviews

- Driven by the hazard register.
- For each identified hazrd, the review team should assess the system and judge whether or not the system can cope with that hazard in a safe way.

Formal verification

- Formal methods can be used when a mathematical specification of the system is produced.
- They are the ultimate static verification technique that may be used at different stages in the development process:
  - A formal specification may be developed and mathematically analyzed for consistency. This helps discover specification errors and omissions.
  - Formal arguments that a program conforms to its mathematical specification may be developed. This is effective in discovering programming and design errors.

Arguments for formal methods

- Producing a mathematical specification requires a detailed analysis of the requirements and this is likely to uncover errors.
- Concurrent systems can be analysed to discover race conditions that might lead to deadlock. Testing for such problems is very difficult.
- They can detect implementation errors before testing when the program is analyzed alongside the specification.

Arguments against formal methods

- Require specialized notations that cannot be understood by domain experts.
- Very expensive to develop a specification and even more expensive to show that a program meets that specification.
- Proofs may contain errors.
- It may be possible to reach the same level of confidence in a program more cheaply using other V & V techniques.

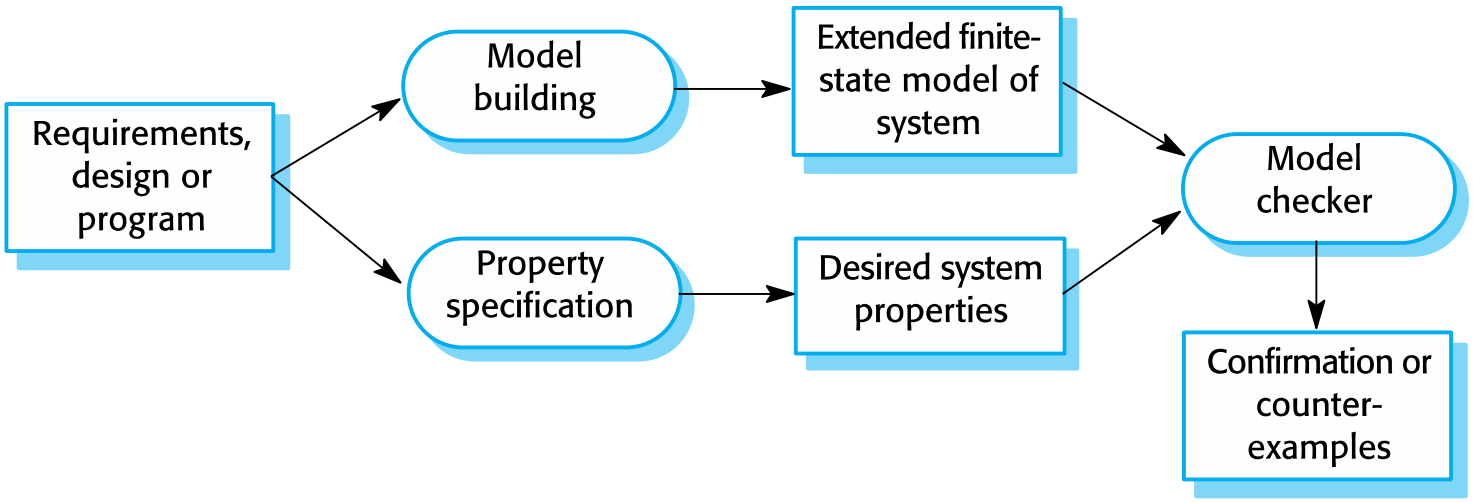
Formal methods cannot guarantee safety

- The specification may not reflect the real requirements of system users. Users rarely understand formal notations so they cannot directly read the formal specification to find errors and omissions.
- The proof may contain errors. Program proofs are large and complex, so, like large and complex programs, they usually contain errors.
- The proof may make incorrect assumptions about the way that the system is used. If the system is not used as anticipated, then the system's behavior lies outside the scope of the proof.

Model checking

- Involves creating an extended finite state model of a system and, using a specialized system (a model checker), checking that model for errors.
- The model checker explores all possible paths through the model and checks that a user-specified property is valid for each path.
- Model checking is particularly valuable for verifying concurrent systems, which are hard to test.
- Although model checking is computationally very expensive, it is now practical to use it in the verification of small to medium sized critical systems.

Model checking



Static program analysis

- Static analysers are software tools for source text processing.
- They parse the program text and try to discover potentially erroneous conditions and bring these to the attention of the V & V team.
- They are very effective as an aid to inspections - they are a supplement to but not a replacement for inspections.

Automated static analysis checks

Fault class	Static analysis check
Data faults	Variables used before initialization Variables declared but never used Variables assigned twice but never used between assignments Possible array bound violations Undeclared variables
Control faults	Unreachable code Unconditional branches into loops
Input/output faults	Variables output twice with no intervening assignment
Interface faults	Parameter-type mismatches Parameter number mismatches Non-usage of the results of functions Uncalled functions and procedures
Storage management faults	Unassigned pointers Pointer arithmetic Memory leaks

Levels of static analysis

- ✧ Characteristic error checking
  - The static analyzer can check for patterns in the code that are characteristic of errors made by programmers using a particular language.
- ✧ User-defined error checking
  - Users of a programming language define error patterns, thus extending the types of error that can be detected. This allows specific rules that apply to a program to be checked.
- ✧ Assertion checking
  - Developers include formal assertions in their program and relationships that must hold. The static analyzer symbolically executes the code and highlights potential problems.

Use of static analysis

- Particularly valuable when a language such as C is used which has weak typing and hence many errors are undetected by the compiler.
- Particularly valuable for security checking – the static analyzer can discover areas of vulnerability such as buffer overflows or unchecked inputs.
- Static analysis is now routinely used in the development of many safety and security critical systems.

Safety cases

Safety and dependability cases

- Safety and dependability cases are structured documents that set out detailed arguments and evidence that a required level of safety or dependability has been achieved.
- They are normally required by regulators before a system can be certified for operational use. The regulator's responsibility is to check that a system is as safe or dependable as is practical.
- Regulators and developers work together and negotiate what needs to be included in a system safety/dependability case.

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.

The contents of a software safety case

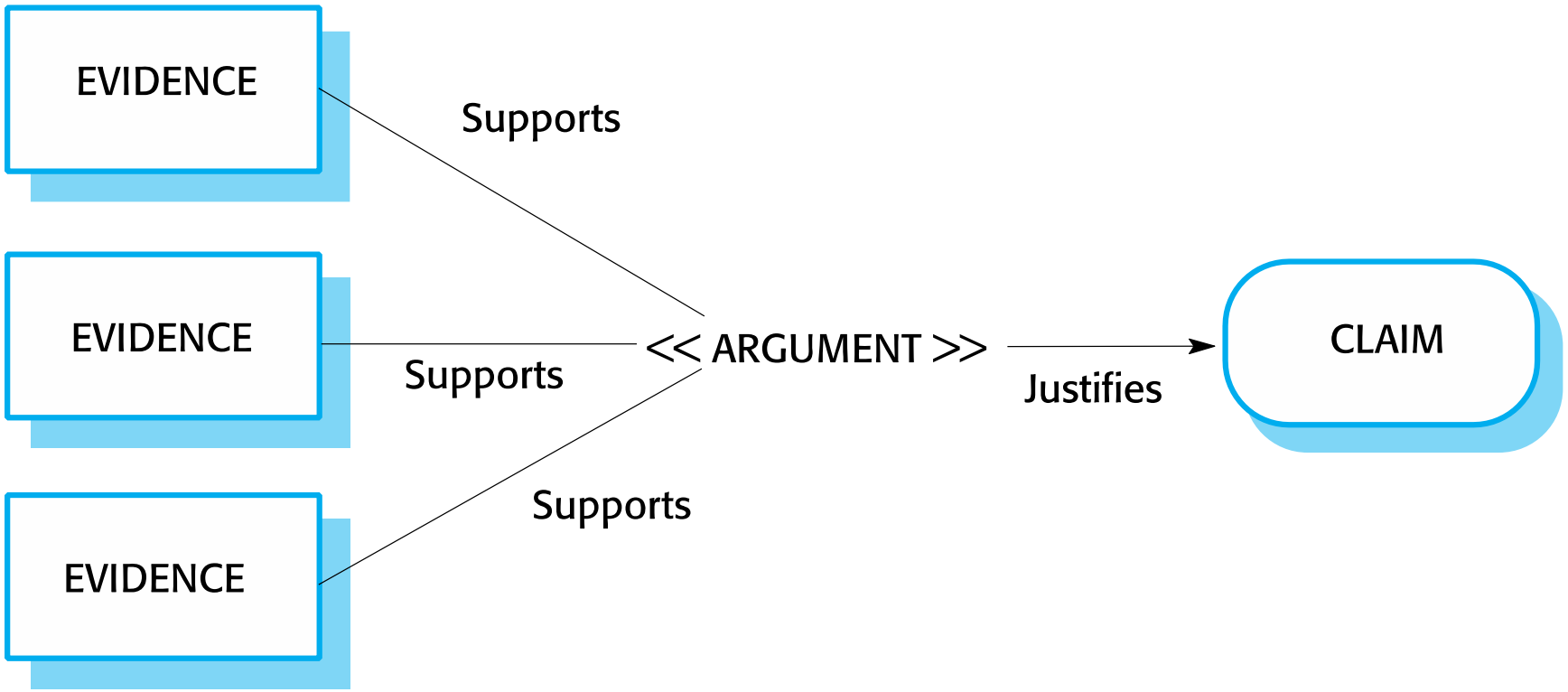
Chapter	Description
System description	An overview of the system and a description of its critical components.
Safety requirements	The safety requirements abstracted from the system requirements specification. Details of other relevant system requirements may also be included.
Hazard and risk analysis	Documents describing the hazards and risks that have been identified and the measures taken to reduce risk. Hazard analyses and hazard logs.
Design analysis	A set of structured arguments (see Section 15.5.1) that justify why the design is safe.
Verification and validation	A description of the V & V procedures used and, where appropriate, the test plans for the system. Summaries of the test results showing defects that have been detected and corrected. If formal methods have been used, a formal system specification and any analyses of that specification. Records of static analyses of the source code.

Chapter	Description
Review reports	Records of all design and safety reviews.
Team competences	Evidence of the competence of all of the team involved in safety-related systems development and validation.
Process QA	Records of the quality assurance processes (see Chapter 24) carried out during system development.
Change management processes	Records of all changes proposed, actions taken and, where appropriate, justification of the safety of these changes. Information about configuration management procedures and configuration management logs.
Associated safety cases	References to other safety cases that may impact the safety case.

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



Insulin pump safety argument

✧ Arguments are based on claims and evidence.

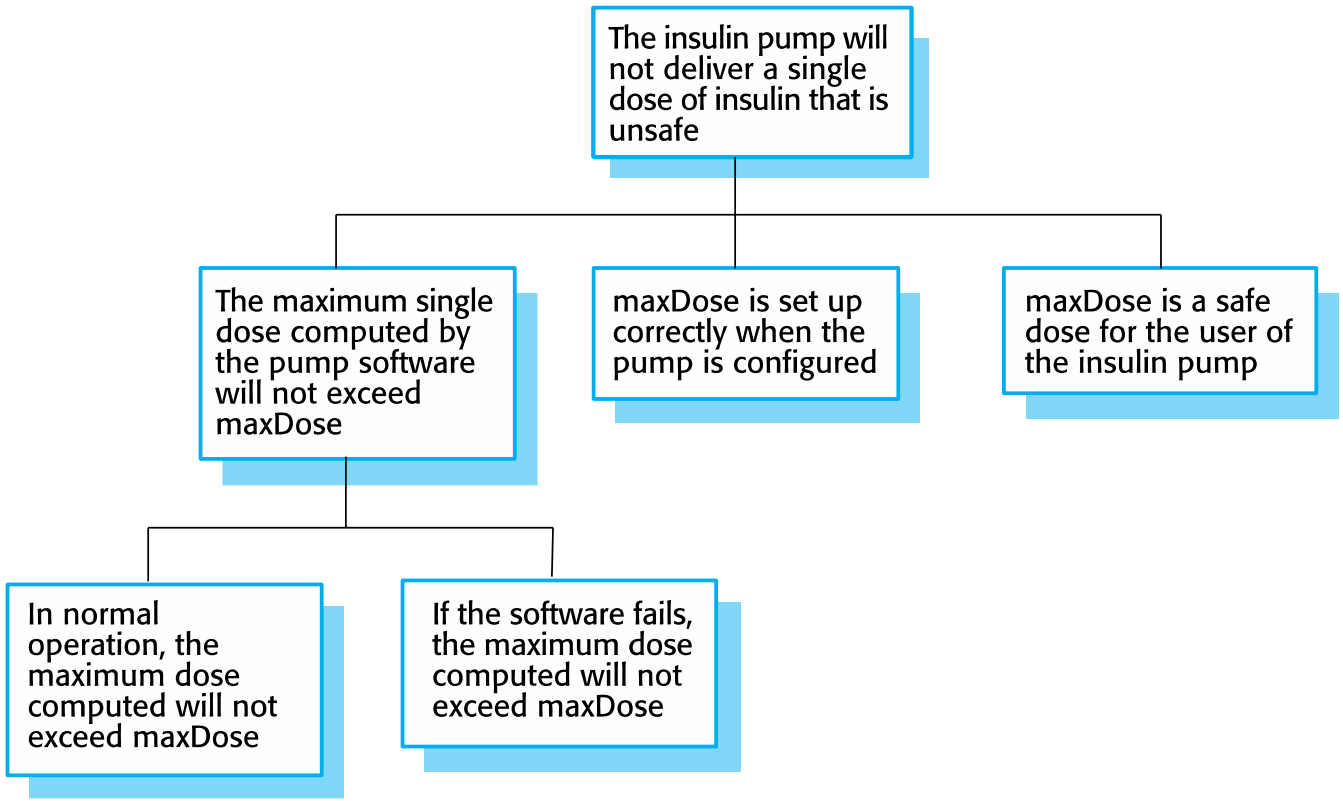
✧ Insulin pump safety:

- Claim: The maximum single dose of insulin to be delivered (CurrentDose) will not exceed MaxDose.
- Evidence: Safety argument for insulin pump (discussed later)
- Evidence: Test data for insulin pump. The value of currentDose was correctly computed in 400 tests
- Evidence: Static analysis report for insulin pump software revealed no anomalies that affected the value of CurrentDose
- Argument: The evidence presented demonstrates that the maximum dose of insulin that can be computed = MaxDose.

Structured safety arguments

- Structured arguments that demonstrate that a system meets its safety obligations.
- It is not necessary to demonstrate that the program works as intended; the aim is simply to demonstrate safety.
- Generally based on a claim hierarchy.
  - You start at the leaves of the hierarchy and demonstrate safety. This implies the higher-level claims are true.

A safety claim hierarchy for the insulin pump



Software safety arguments

- Safety arguments are intended to show that the system cannot reach in unsafe state.
- These are weaker than correctness arguments which must show that the system code conforms to its specification.
- They are generally based on proof by contradiction
  - Assume that an unsafe state can be reached;
  - Show that this is contradicted by the program code.
- A graphical model of the safety argument may be developed.

Construction of a safety argument

- Establish the safe exit conditions for a component or a program.
- Starting from the END of the code, work backwards until you have identified all paths that lead to the exit of the code.
- Assume that the exit condition is false.
- Show that, for each path leading to the exit that the assignments made in that path contradict the assumption of an unsafe exit from the component.

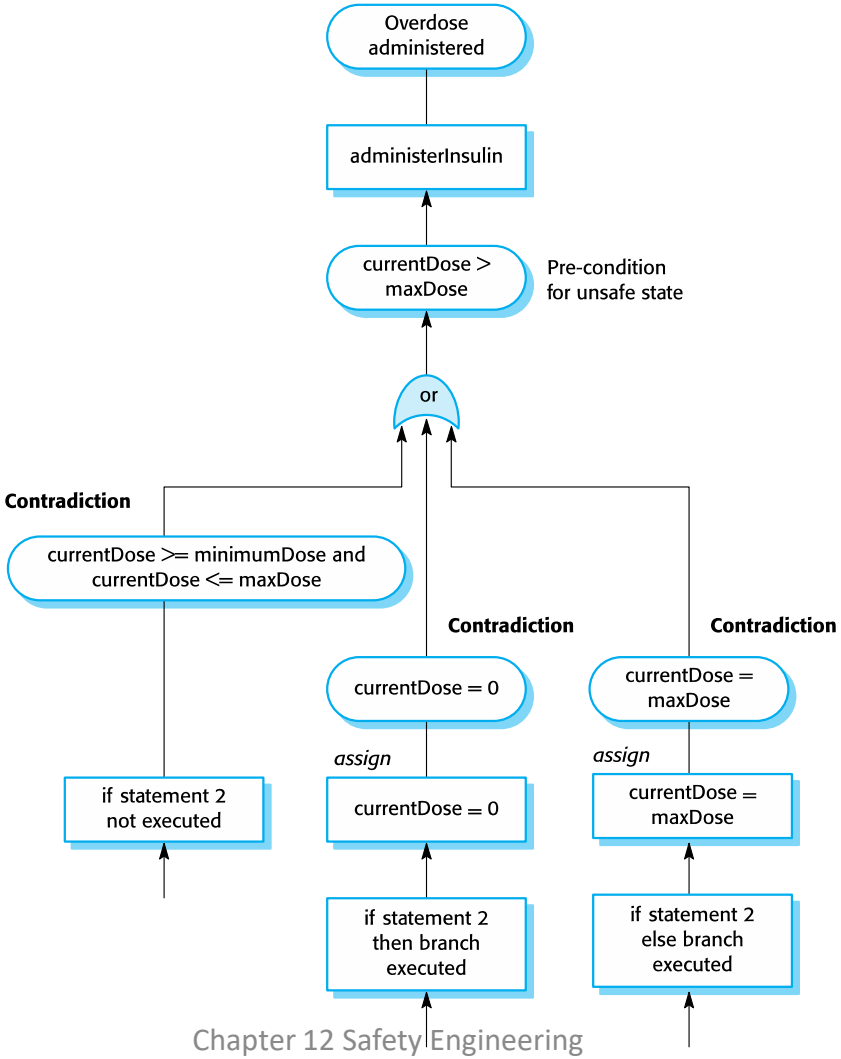
Insulin dose computation with safety checks


```
-- The insulin dose to be delivered is a function of blood sugar level,
-- the previous dose delivered and the time of delivery of the previous dose

currentDose = computeInsulin () ;

// Safety check—adjust currentDose if necessary.
// if statement 1
if (previousDose == 0)
{
        if (currentDose > maxDose/2)
        currentDose = maxDose/2 ;
}
else
        if (currentDose > (previousDose * 2) )
                currentDose = previousDose * 2 ;
// if statement 2
if ( currentDose < minimumDose )
        currentDose = 0 ;
else if ( currentDose > maxDose )
        currentDose = maxDose ;
administerInsulin (currentDose) ;
```

Informal safety argument based on demonstrating contradictions



Program paths

- ✧ Neither branch of if-statement 2 is executed
  - Can only happen if CurrentDose is >= minimumDose and <= maxDose.
- ✧ then branch of if-statement 2 is executed
  - currentDose = 0.
- ✧ else branch of if-statement 2 is executed
  - currentDose = maxDose.
- ✧ In all cases, the post conditions contradict the unsafe condition that the dose administered is greater than maxDose.

Key points

- Safety-critical systems are systems whose failure can lead to human injury or death.
- A hazard-driven approach is used to understand the safety requirements for safety-critical systems. You identify potential hazards and decompose these (using methods such as fault tree analysis) to discover their root causes. You then specify requirements to avoid or recover from these problems.
- It is important to have a well-defined, certified process for safety-critical systems development. This should include the identification and monitoring of potential hazards.

Key points

- Static analysis is an approach to V & V that examines the source code of a system, looking for errors and anomalies. It allows all parts of a program to be checked, not just those parts that are exercised by system tests.
- Model checking is a formal approach to static analysis that exhaustively checks all states in a system for potential errors.
- Safety and dependability cases collect the evidence that demonstrates a system is safe and dependable. Safety cases are required when an external regulator must certify the system before it is used.