Safety Analysis

Lecture 14: CPEN 400P

Karthik Pattabiraman, UBC

(Based on the book "Software Engineering" by Ian Sommerville, 10th edition Chapter 12)

Learning Objectives

- Define safety and safety critical (SC) systems
- Perform hazard analysis on SC systems
- Understand the processes for safety assurance in SC systems
- Construct a safety case for an SC system

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.

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

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 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

Learning Objectives

- Define safety and safety critical (SC) systems
- Perform hazard analysis on SC systems
- Understand the processes for safety assurance in SC systems
- Construct a safety case for an SC system

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

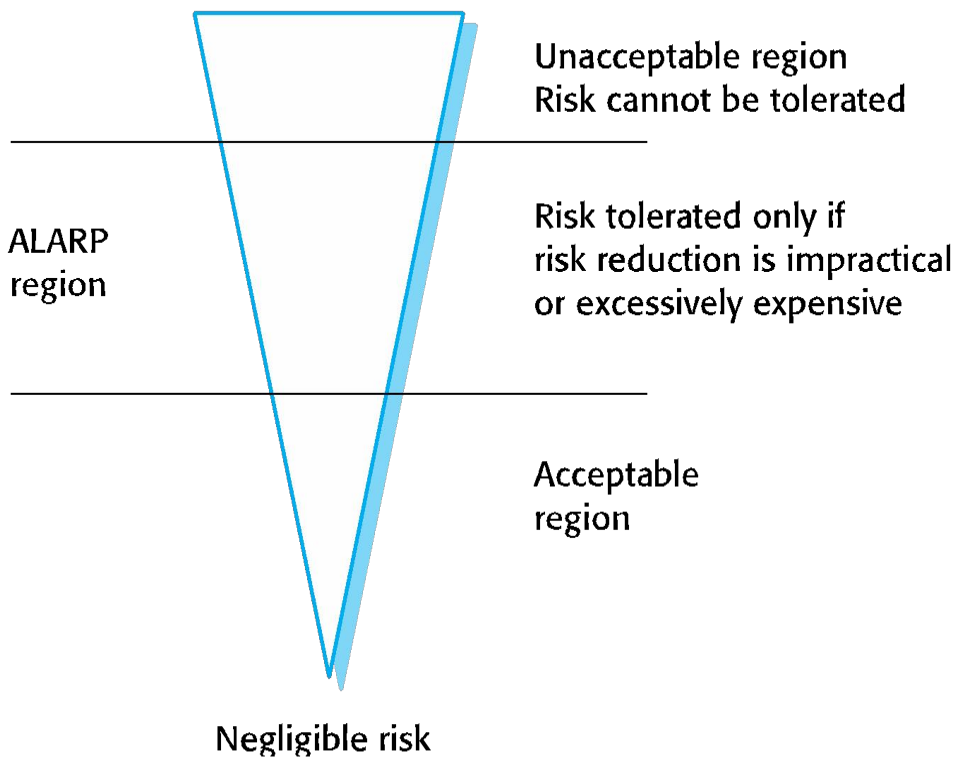
Insulin pump risks

- 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



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

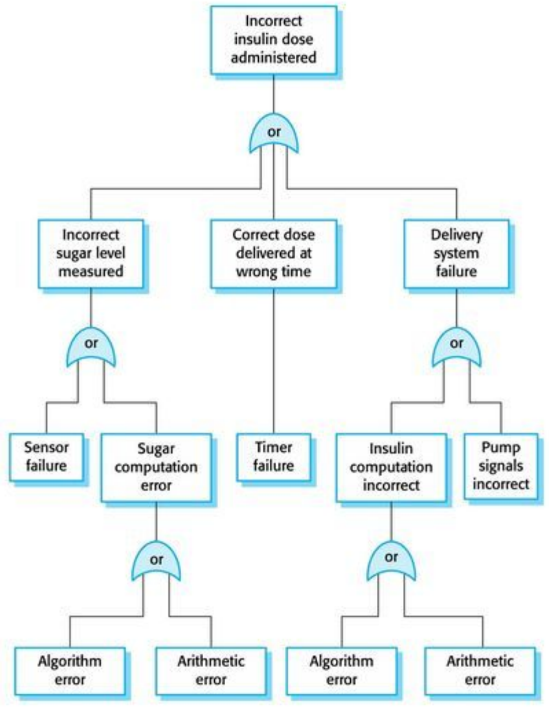
Identified hazard	Hazard probability	Accident severity	Estimated risk	Acceptability
1.Insulin overdose computation	Medium	High	High	Intolerable
2. Insulin underdose computation	Medium	Low	Low	Acceptable
3. Failure of hardware monitoring system	Medium	Medium	Low	ALARP
4. Power failure	High	Low	Low	Acceptable
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

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

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

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.

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.

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.

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.

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.

Learning Objectives

- Define safety and safety critical (SC) systems
- Perform hazard analysis on SC systems
- Understand the processes for safety assurance in SC systems
- Construct a safety case for an SC system

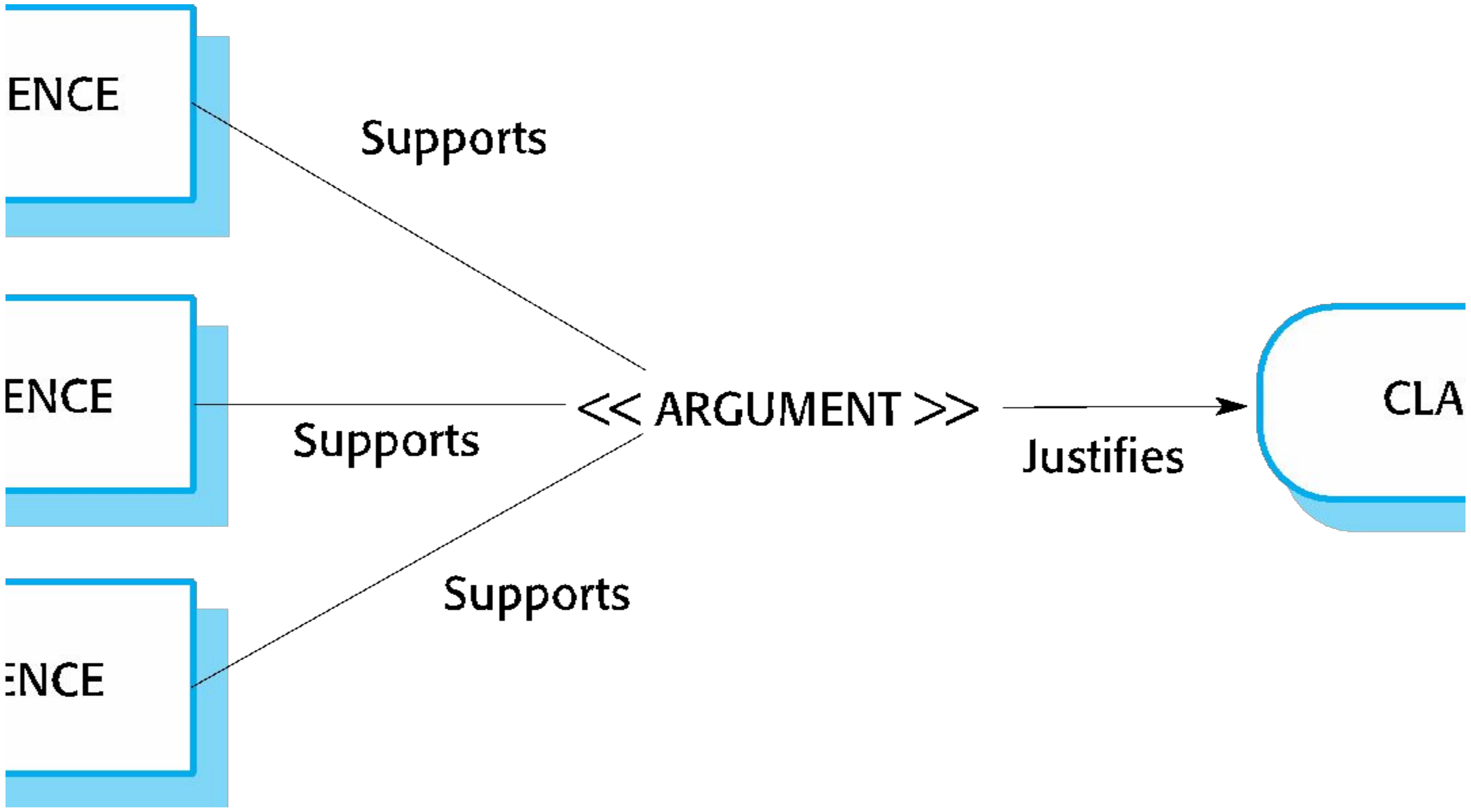
The system safety case

- 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



Insulin pump safety argument

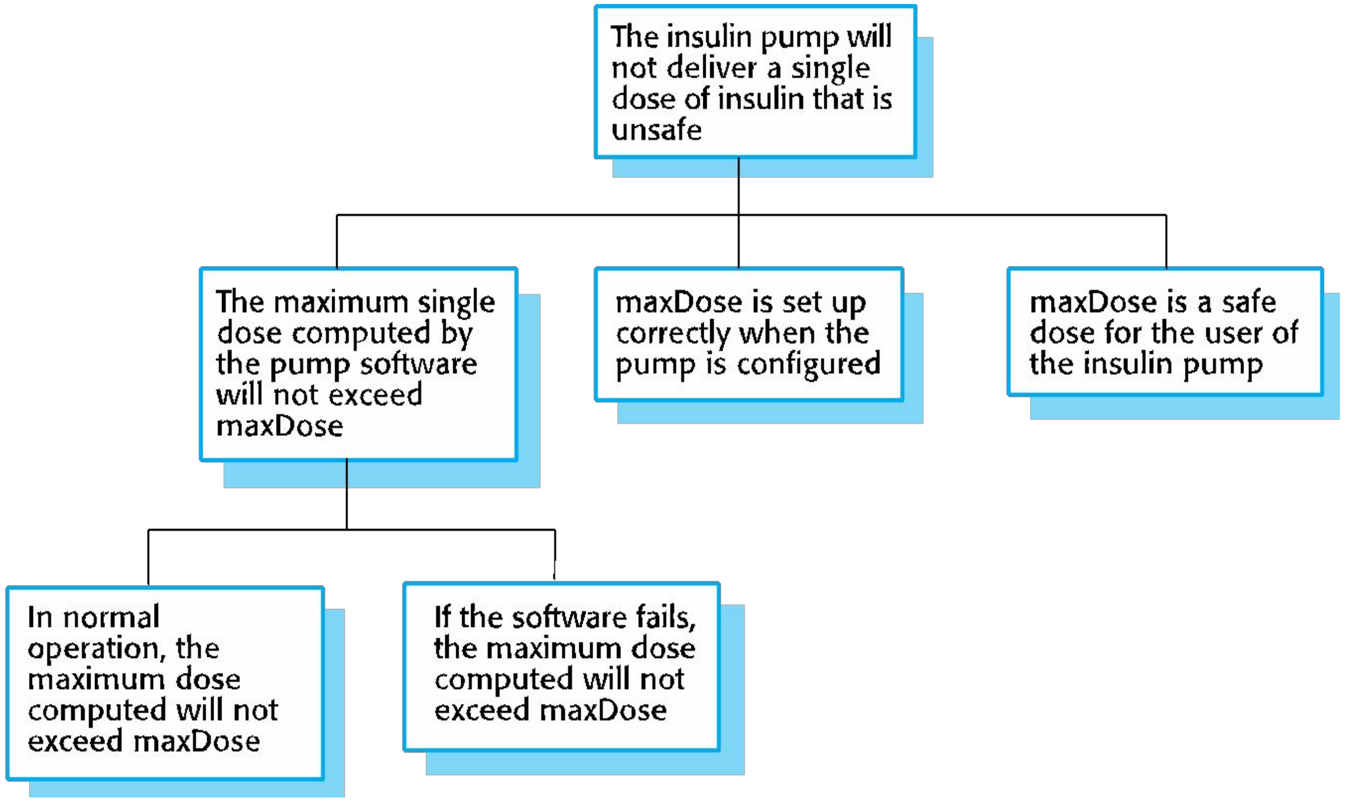
- Arguments are based on claims and evidence.

- 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


The insulin pump will not deliver a single dose of insulin that is unsafe
The maximum single dose computed by the pump software will not exceed maxDose
maxDose is set up correctly when the pump is configured
maxDose is a safe dose for the user of the insulin pump
In normal operation, the maximum dose computed will not exceed maxDose
If the software fails, the maximum dose computed will not exceed maxDose

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.

Class Activity: Insulin dose computation with safety checks

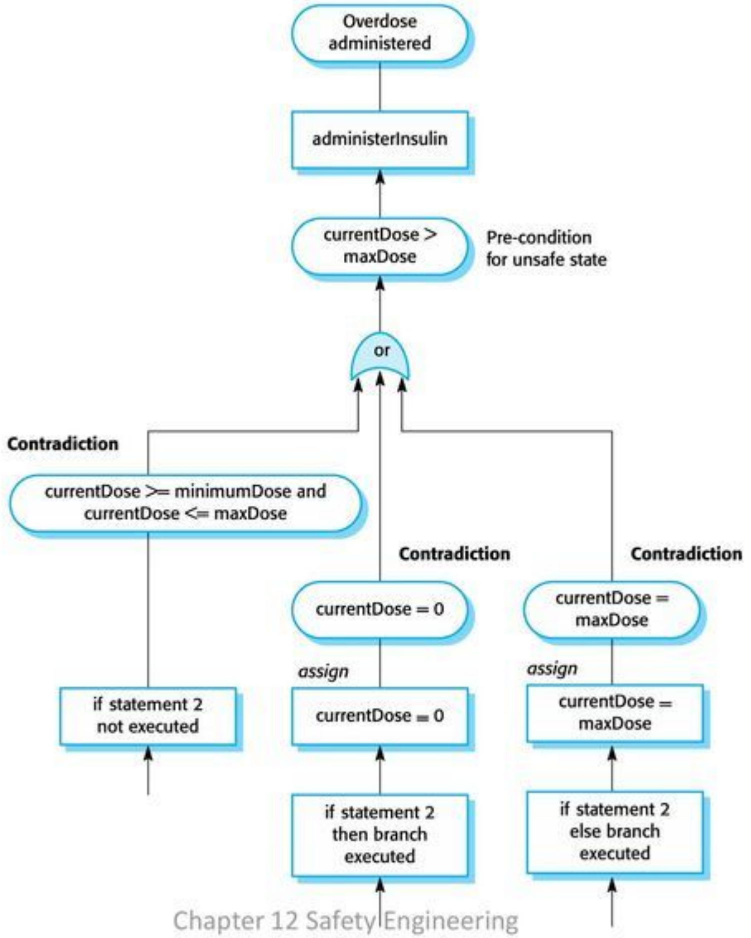
```
-- The insulin dose to be delivered is a function of blood sugar level,
-- the previous dose delivered and the time of delivery of the previous dose

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) ;
```

Solution: 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.

Informal safety argument based on demonstrating contradictions



Learning Objectives

- Define safety and safety critical (SC) systems
- Perform hazard analysis on SC systems
- Understand the processes for safety assurance in SC systems
- Construct a safety case for an SC system