Safety Critical Systems

Lecture 15: 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's operation is always safe i.e. the system should never cause damage to people or environment
- Examples
  - Control and monitoring systems in aircraft
  - Process control systems in chemical manufacturing
  - 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.

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

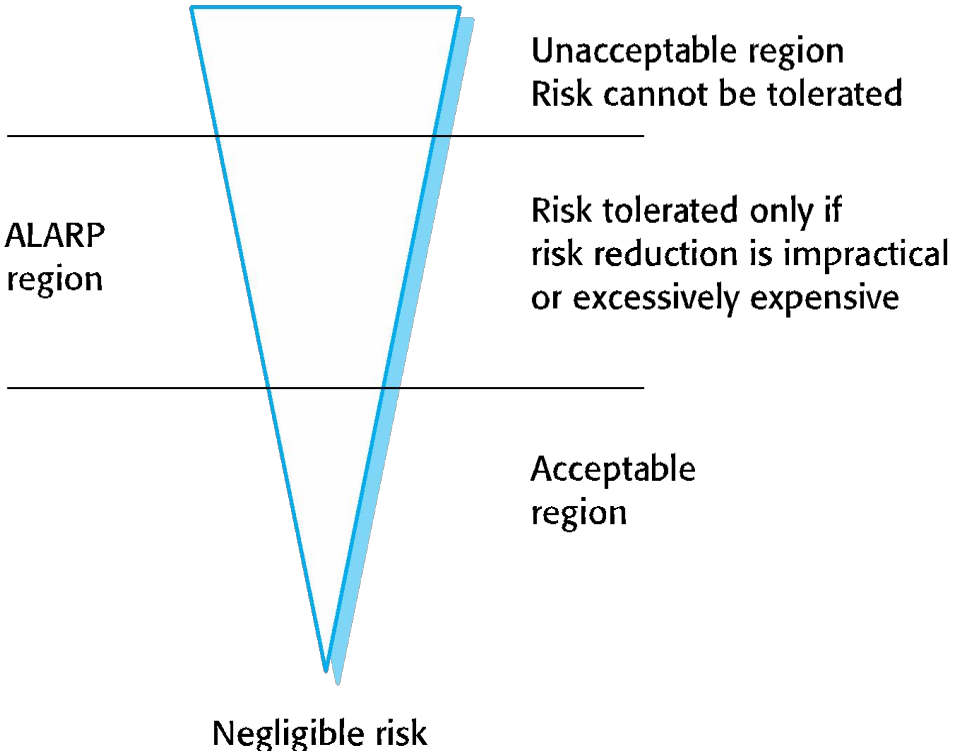
- Insulin overdose (service failure).
- Insulin underdose (service failure).
- Power failure due to exhausted battery (electrical).
- Electrical interference with other medical equipment (electrical).
- Poor sensor and actuator contact (physical).
- Parts of machine break off in body (physical).
- Infection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

Hazard assessment

- 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



Risk classification for the insulin pump

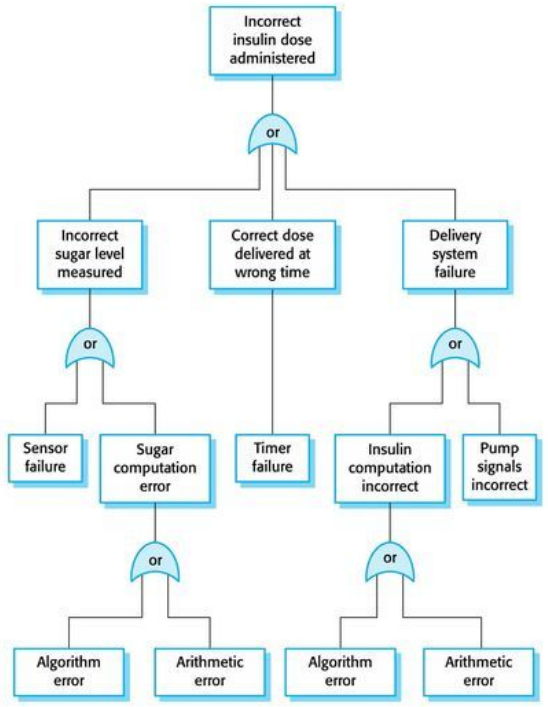
Identified hazard	Hazard probability	Accident severity	Estimated risk	Acceptability
1.Insulin overdose computation	Medium	High	High	Intolerable
2. Insulin underdose computation	Medium	Low	Low	Acceptable
3. Failure of hardware monitoring system	Medium	Medium	Low	ALARP
4. Power failure	High	Low	Low	Acceptable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



Insulin pump - Mitigating 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.

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

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.

- 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.

Formal verification

- Formal methods can be used when a mathematical specification of the system is produced.
- 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.

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

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:
  - Documented body of evidence that provides convincing and valid argument that the system is adequately safe for a given application in a given environment
- Arguments 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 also 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.

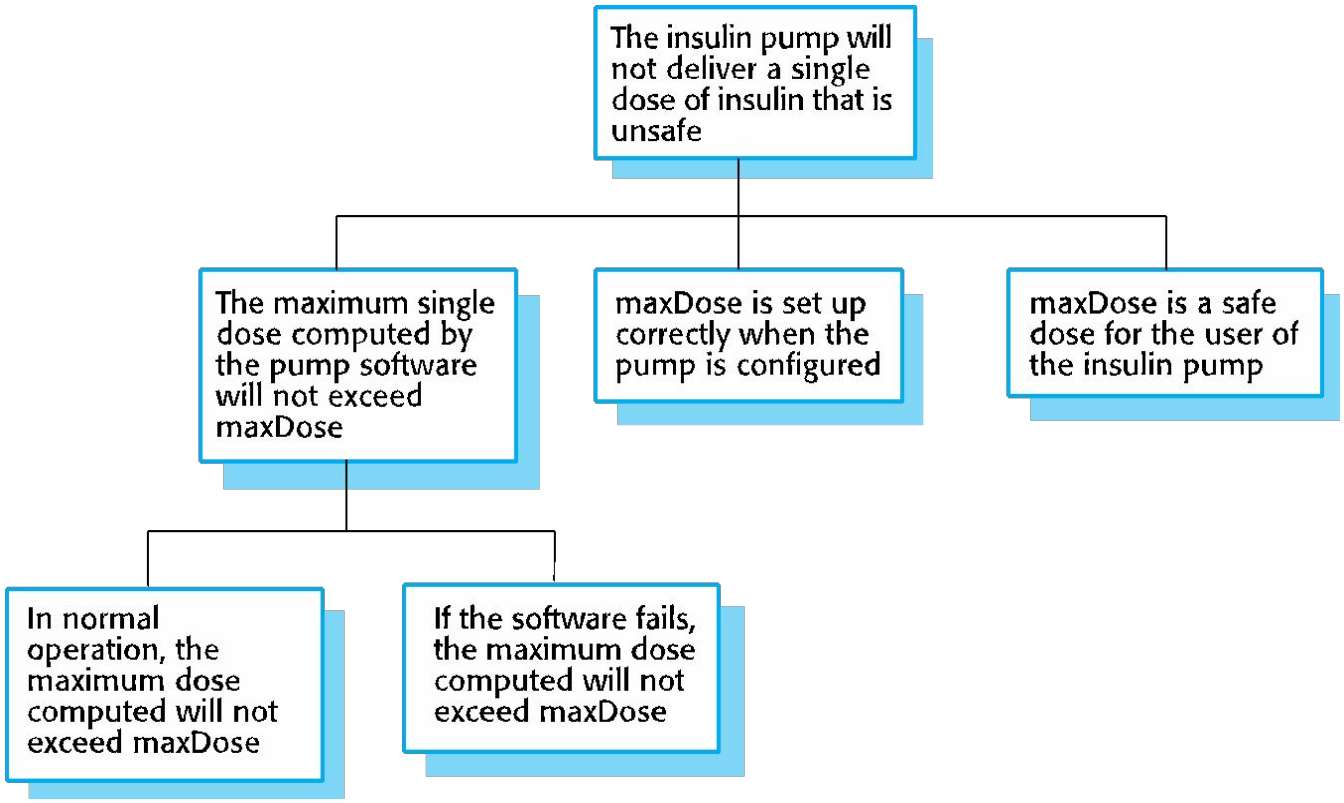
Insulin pump safety argument

- Claim: The maximum single dose of insulin to be delivered (*CurrentDose*) is *MaxDose*.
  - Evidence: Safety argument for insulin pump (later)
  - Evidence: Test data for insulin pump. The value of *CurrentDose* was correctly computed in 400 tests
  - Evidence: Static analysis for insulin pump software no anomalies that affect the value of *CurrentDose*
  - Argument: The evidence demonstrates that the maximum dose of insulin is equal to *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.

Safety claim hierarchy: Insulin Pump



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.