Chapter 12 – Safety Engineering

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

Safety engineering processes

Safety engineering processes

- Reliability engineering processes
  - Reviews and checks at each stage in the process
  - General goal of fault avoidance and fault detection
  - Safety reviews and explicit identification of hazards

Regulation

- Evidence that safety engineering processes used.
- For example:
  - The specification and records of the checks.
  - Evidence of the verification and validation the results.
  - Organizations for dependable software processes.

Agile methods and safety

- Agile methods are not usually used for safety-critical systems engineering
  - Extensive documentation.
  - A detailed safety analysis of a complete system specification.
- Test-driven development may be used

Safety assurance processes

- Defining and ensuring a dependable process.
- Process assurance focuses on:
  - The processes are appropriate for dependability required.
  - The processes are followed by the development team.
- Should generate documentation
  - Agile processes are not used for critical systems.

Processes for safety assurance

- Process assurance is important for safety-critical systems development:
  - Testing may not find all problems.
- Safety assurance activities
  - Record the analyses.
  - Personal responsibility.

Safety related process activities

- A hazard logging and monitoring system.
- Safety engineers who responsible for safety.
- Extensive use of safety reviews.
- A safety certification system.
- Detailed configuration management

Hazard analysis

- Identifying hazards and their root causes.
- Traceability from identified hazards
  - Analysis to to ensure that the hazards have been covered.
- A hazard log may be used to track hazards.

Safety reviews

- Driven by the hazard register.
- Assess the system and judge whether to cope with hazards in a safe way.

Formal verification

- A mathematical specification of the system is produced.
- Static verification technique used in development:
  - A formal specification -- mathematically analyzed for consistency.
    - Discover specification errors and omissions.
  - A program conforms to its mathematical specification
    - Programming and design errors.

Arguments for formal methods

- A mathematical specification requires a detailed analysis
- Concurrent systems
  - Discover race conditions.
  - Testing is difficult.
- Detect implementation errors before testing
  - Program is analyzed alongside the specification.

Arguments against formal methods

- ✧ Require specialized notations
  - Cannot be understood by domain experts.
- ✧ Expensive to develop a specification
- ✧ Proofs may contain errors.
- ✧ More cheaply using other V & V techniques.
- ✧ The proof making incorrect assumptions
  - System's behavior lies outside the scope of the proof.

Model checking

- ✧ Create a state model of using a specialized system
  - Checking the model for errors.
- ✧ The model checker explores all possible paths
  - Checks that a user-specified property is valid for each path.
- ✧ Verifying concurrent systems, which are hard to test.
- ✧ Model checking is computationally very expensive
  - Verification of small to medium sized critical systems.

Static program analysis

- ✧ Tools for source text processing.
- ✧ Parse the program text to discover erroneous conditions.
- ✧ Effective as an aid to inspections
  - A supplement to 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
  - Check for patterns in the code for characteristic of errors.
- ✧ User-defined error checking
  - Define error patterns, extending error types by specific rules
- ✧ Assertion checking
  - Formal assertions in their program
  - Symbolically executes to find potential problems.

Example for Symbolic Execution

```
x = readInput();
y = x * 2;

if (y == 12) {
   fail();
} else {
   print("OK");
}
```

Use of static analysis

- ✧ Particularly valuable when a language has weak typing
  - Many errors are undetected by the compiler.
- ✧ Security checking
  - Discover areas of vulnerability such as buffer overflows.
- ✧ The development of safety and security critical systems.

Safety cases

Sep 19

Safety and dependability cases

- ✧ Safety and dependability cases
  - Structured documents
  - Evidence of a required level of safety or dependability.
- ✧ Regulators check a system is as safe or dependable.
- ✧ Regulators and developers work together for a system safety/dependability case.

The system safety case

- ✧ A safety case is:
  - A documented body of evidence.
  - A system is adequately safe for a given environment.
- ✧ Formal proof, design rationale, safety proofs, etc.
- ✧ Wider system safety case that takes hardware and operational issues into account.

The contents of a software safety case

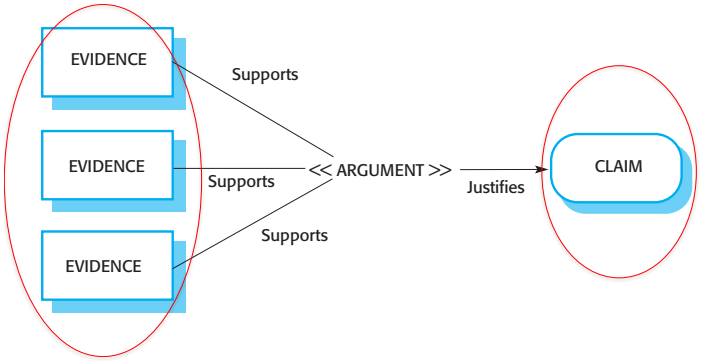
Chapter	Description
System description	An overview of the system and a description of its critical components.
Safety requirements	The safety requirements abstracted from the system requirements specification. Details of other relevant system requirements may also be included.
Hazard and risk analysis	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 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 be based on structured arguments
- ✧ Claims of safety and security justified by evidences.

Structured arguments



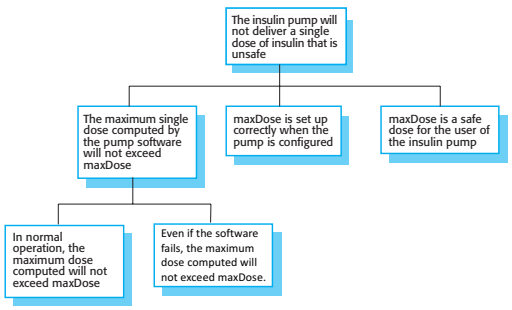
Insulin pump safety argument

- ✧ 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
  - 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 of a system safety obligations.
- ✧ Generally based on a claim hierarchy.

A safety claim hierarchy for the insulin pump



Software safety arguments

- ✧ 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.

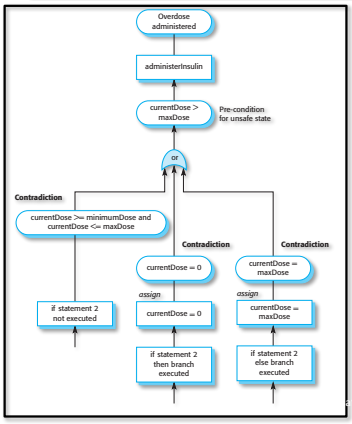
Insulin dose computation with safety checks

```
-- 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




```
...

/* if statement 2 */
if ( currentDose < minimumDose ) {
    currentDose = 0 ;
}
else if ( currentDose > maxDose ) {
    currentDose = maxDose ;
}

administerInsulin (currentDose) ;
...
```


Program paths

```
if ( currentDose < minimumDose ) {
    currentDose = 0 ;
}
else if ( currentDose > maxDose ) {
    currentDose = maxDose ;
}
administerInsulin (currentDose) ;
```

- 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.