12 Safety Engineering

Objectives

The objective of this chapter is to explain techniques that are used to ensure safety when developing critical systems. When

et al.

12.1 Safety-critical systems

Primary safety-critical software

Risk-based requirements specification

Risk-based specification is an approach that has been widely used by safety and security-critical systems developers. It focuses on those events that could cause the most damage or that are likely to occur frequently. Events that have only minor consequences or that are extremely rare may be ignored. The risk-based specification process involves understanding the risks faced by the system, discovering their root causes and generating requirements to manage these risks.

http://software-engineering-book.com/web/risk-based-specification/

12.2 Safety requirements

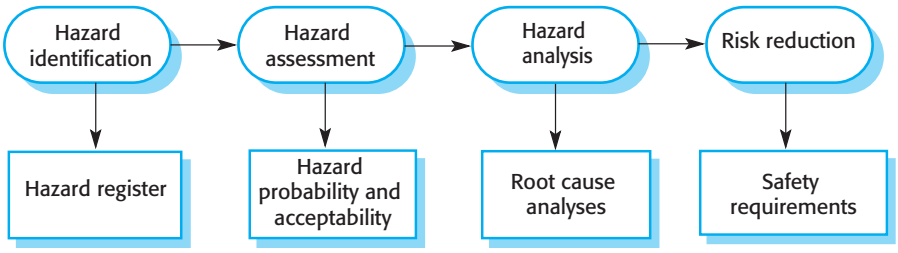


Figure 12.2 Hazard-driven requirements specification

Hazard identification

Hazard assessment

Hazard analysis

Risk reduction

12.2.1 Hazard identification

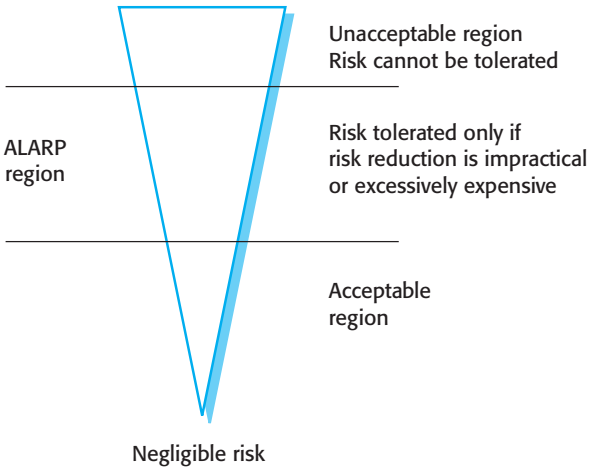
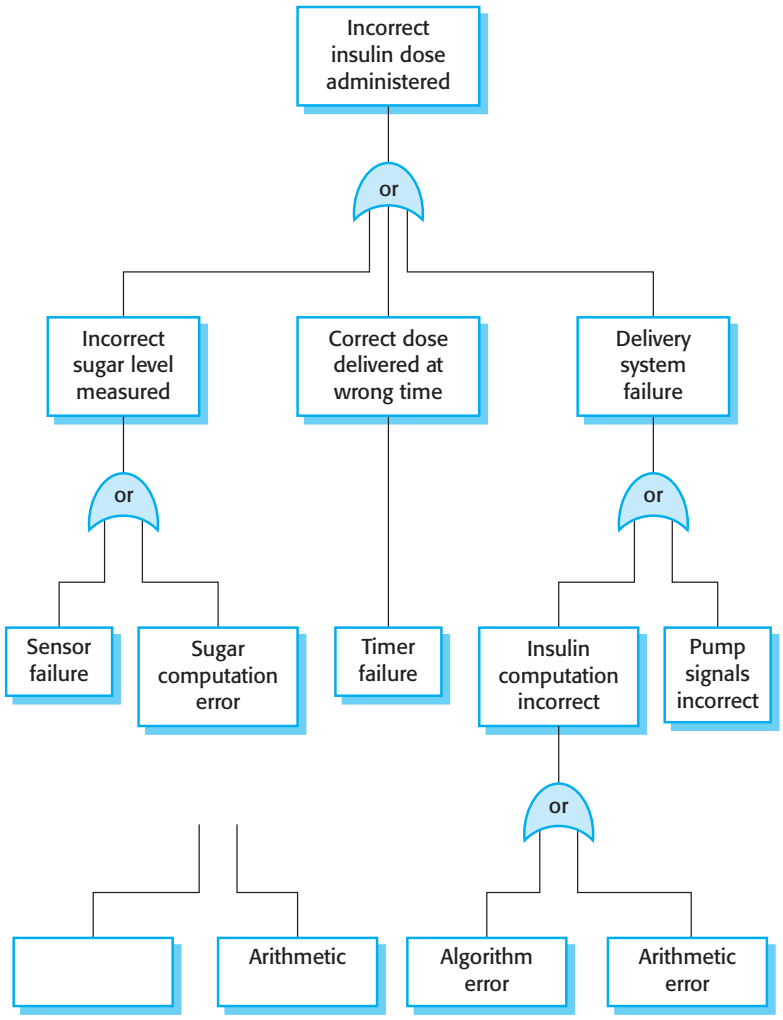


Figure 12.3 The riek triangle

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

Figure 12.4 Risk classification for the insulin pump

12.2.3 Hazard analysis


Incorrect insulin dose administered
or
Incorrect sugar level measured
Correct dose delivered at wrong time
Delivery system failure
or
or
Sensor failure
Sugar computation error
Timer failure
Insulin computation incorrect
Pump signals incorrect
or
Arithmetic
Algorithm error
Arithmetic error

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.

Figure 12.6 Examples of safety requirements

Algorithmic error

12.3 Safety engineering processes

12.3.1 Safety assurance processes

Hazard Register. Page 4: Printed 20.02.2012

System: Insulin Pump System			*File*: InsulinPump/Safety/HazardLog		
Safety Engineer: James Brown			*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*	24.01.11	*Location*	Hazard register, Page 5
Fault tree creators	Jane Williams and Bill Smith				
Fault tree checked	YES	*Date*	28.01.11	*Checker*	James Brown

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.

Figure 12.7 A simplified hazard register entry

12.3.3 Model checking

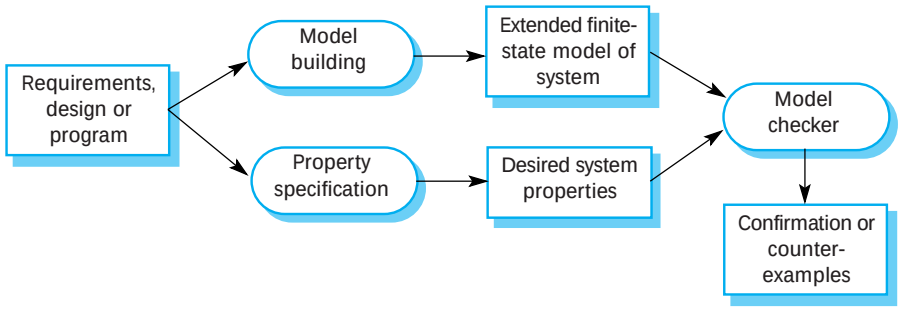


Figure 12.8 Model checking

12.3.4 Static program analysis

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

Figure 12.9
Automated static analysis checks

Characteristic error checking

User-defined error checking

Assertion checking

12.4 Safety cases

Chapter	Description
System description	An overview of the system and a description of its critical components.
Safety requirements	The safety requirements taken 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 12.4.1) 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.
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 on the safety case.

Figure 12.10 Possible contents of a software safety

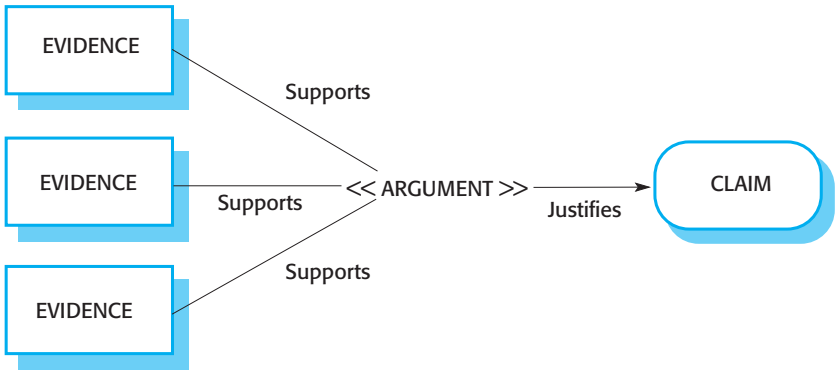


Figure 12.11 Structured arguments

12.4.1 Structured arguments

Claim:

maxDose maxDose

Evidence:

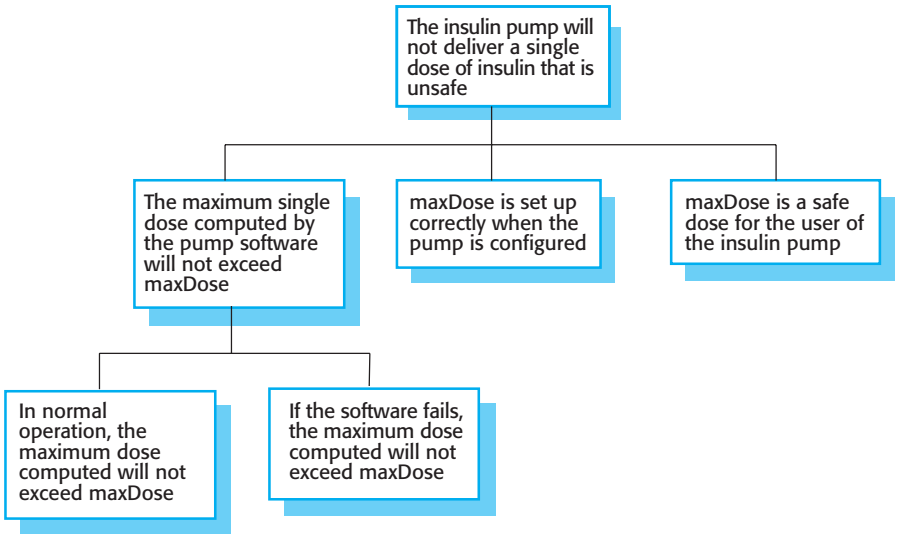


Figure 12.12 A safety claim hierarchy for the insulin pump

Evidence:

currentDose, maxDose

Evidence:

currentDose

Argument:

maxDose

12.4.2 Software safety arguments

```
-- 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) ;
```

Figure 12.13 Insulin dose computation with safety checks

maxDose

currentDose > maxDose

administerInsulin

maxDose

administerInsulin are considered:

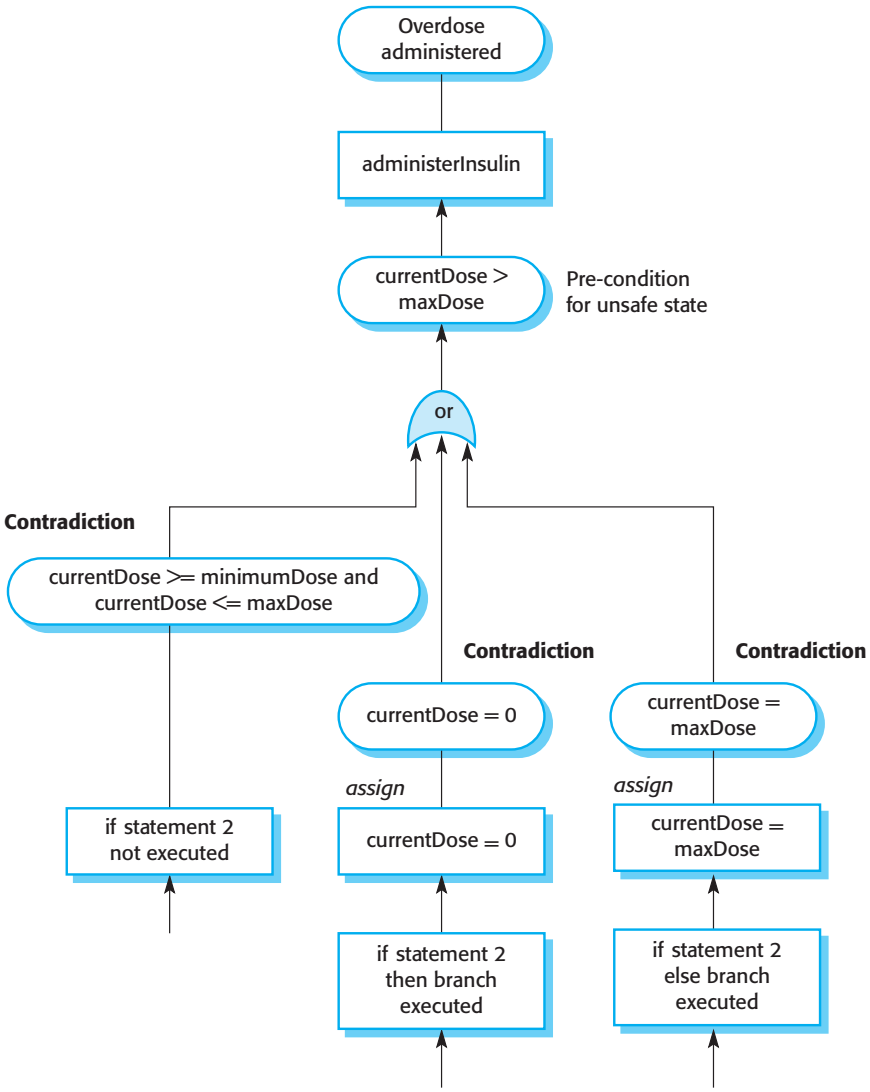


Figure 12.14 Informal safety argument based on demonstrating contradictions

currentDose minimumDose..maxDose

currentDose >= minimumDose and currentDose <= maxDose

currentDose
currentDose = 0

currentDose maxDose

currentDose =

maxDose

currentDose > maxDose.

currentDose

administerInsulin

KEY POINTS

Safety-critical systems are systems whose failure can lead to human injury or death.

A hazard-driven approach may be 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. The process should include the identification and monitoring of potential hazards.

Static analysis is an approach to V & V that examines the source code (or other representation) 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 all of 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.

FURTHER READING

Safeware: System Safety and Computers. Although now 20 years old, this book is still offers the best and most thorough coverage of safety-critical systems. It is particularly strong in its description of hazard analysis and the derivation of requirements from this. (N. Leveson, Addison Wesley, 1995)

'Safety-critical software'. A special edition of *IEEE Software* magazine that focuses on safety-critical systems. It includes papers on model-based development of safety-critical systems, model checking and formal methods. (*IEEE Software*, 30 (3), May/June 2013)

'Constructing safety assurance cases for medical devices'. This short paper gives a practical example of how a safety case can be created for an analgesic pump. (A. Ray and R. Cleaveland, Proc. Workshop on Assurance Cases for Software-Intensive Systems, San Francisco, 2013)
http://dx.doi.org/10.1109/ASSURE.2013.6614270

WEBSITE

PowerPoint slides for this chapter:

http://software-engineering-book.com/slides/chap12/

Links to supporting videos:

http://software-engineering-book.com/videos/reliability-and-safety/

EXERCISES

12. 1 Identify six consumer products that are likely to be controlled by safety-critical software systems.

12.2 Explain why the boundaries in the risk triangle shown in Figure 12.3 are liable to change with time and changing social attitudes.

12.3 In the insulin pump system, the user has to change the needle and insulin supply at regular intervals and may also change the maximum single dose and the maximum daily dose that may be administered. Suggest three user errors that might occur and propose safety requirements that would avoid these errors resulting in an accident.

12.4 A safety-critical software system for treating cancer patients has two main components:

A radiation therapy machine that delivers controlled doses of radiation to tumor sites. This machine is controlled by an embedded software system.

A treatment database that includes details of the treatment given to each patient. Treatment requirements are entered in this database and are automatically downloaded to the radiation therapy machine.

Identify three hazards that may arise in this system. For each hazard, suggest a defensive requirement that will reduce the probability that these hazards will result in an accident. Explain why your suggested defense is likely to reduce the risk associated with the hazard.

12.5 A train protection system automatically applies the brakes of a train if the speed limit for a segment of track is exceeded, or if the train enters a track segment that is currently signaled with a red light (i.e. the segment should not be entered). There are two critical safety requirements for this train protection system:

The train shall not enter a segment of track that is signaled with a red light.

The train shall not exceed the specified speed limit for a section of track.

Assuming that the signal status and the speed limit for the track segment are transmitted to on-board software on the train before it enters the track segment, propose five possible functional system requirements for the onboard software that may be generated from the system safety requirements.

12.6 Explain when it may be cost-effective to use formal specification and verification in the development of safety-critical software systems. Why do you think that some critical systems engineers are against the use of formal methods?

12.7 Explain why using model checking is sometimes a more cost-effective approach to verification than verifying a program's correctness against a formal specification.

12.8 List four types of systems that may require software safety cases, explaining why safety cases are required.

12.9 The door lock control mechanism in a nuclear waste storage facility is designed for safe operation. It ensures that entry to the storeroom is only permitted when radiation shields are in place or when the radiation level in the room falls below some given value (dangerLevel). So:

(i) If remotely controlled radiation shields are in place within a room, an authorized operator may open the door.

(ii) If the radiation level in a room is below a specified value, an authorized operator may open the door.

(iii) An authorized operator is identified by the input of an authorized door entry code.

The code shown in Figure 12.15 (see below) controls the door-locking mechanism. Note that the safe state is that entry should not be permitted. Using the approach discussed in this chapter, develop a safety argument for this code. Use the line numbers to refer to specific statements. If you find that the code is unsafe, suggest how it should be modified to make it safe.

```
1    entryCode = lock.getEntryCode () ;
2    if (entryCode == lock.authorizedCode)
3    {
4         shieldStatus = Shield.getStatus ();
5         radiationLevel = RadSensor.get ();
6         if (radiationLevel < dangerLevel)
7              state = safe;
8         else
9              state = unsafe;
10        if (shieldStatus == Shield.inPlace() )
11             state = safe;
12        if (state == safe)
13             {
14                  Door.locked = false ;
15                  Door.unlock ();
16             }
17        else
18        {
19             Door.lock ( );
20             Door.locked := true ;
21        }
22   }
```

Figure 12.15 Door entry code

12.10 Should software engineers working on the specification and development of safety-related systems be professionally certified or licensed in some way? Explain your reasoning.

REFERENCES

Modeling in Event-B: System and Software Engineering

Communications of the ACM

Formal methods'99

Proc. Safety-Critical Systems Symposium

Proc. 31st International Conf. on Software Engineering, Companion Volume

Applications of Formal Methods

Proc. 37th Annual IEEE Conf. on Dependable Systems and Networks

Comm ACM

Computing Surveys

Proc. 23rd Int. Conf. on Computer Aided Verification

IEEE Software

Proceedings of DASIA 2009 Data Systems in Aerospace

RE'93

IEEE Software

Normal Accidents: Living with High-Risk Technology

IEEE Computer

Concurrent and Real-Time Systems: The CSP Approach

Formal Methods'09: Proceedings of the 2nd World Congress on Formal Methods

Safety-Critical Computer Systems