
LUT
University

Software Engineering Models and Modelling

Safety, Risks & Modelling

Antti Knutas

Software Defects and Faults

Defect perspective to software quality: Is it free of errors?

One perspective to software quality is defect perspective – once the software is fully testable, then those tests must be passed

What is the defect severity? (one classification)

- Critical defects – cause failures
- Major defects – caught by failure tolerance
- Minor defects – nuisances (such as typos)

Faults, errors and failures

Term	Description
Human error or mistake	Human behavior that results in the introduction of faults into a system. For example, in the wilderness weather system, a programmer might decide that the way to compute the time for the next transmission is to add 1 hour to the current time. This works except when the transmission time is between 23.00 and midnight (midnight is 00.00 in the 24-hour clock).
System fault	A characteristic of a software system that can lead to a system error. The fault is the inclusion of the code to add 1 hour to the time of the last transmission, without a check if the time is greater than or equal to 23.00.
System error	An erroneous system state that can lead to system behavior that is unexpected by system users. The value of transmission time is set incorrectly (to 24.XX rather than 00.XX) when the faulty code is executed.
System failure	An event that occurs at some point in time when the system does not deliver a service as expected by its users. No weather data is transmitted because the time is invalid.

Defect terminology

- Error or bug – program malfunctioning due to programming error
- Defect – pre-existing condition in a finished product
- Fault – fault occurs when software encounters a defect during operation
- Fault tolerance – a set of mechanisms that allow software to self-correct
- Failure – software product encounters a fault that cannot be corrected
- Accident – unplanned event or sequence which results in human death or injury
- Hazard – condition that can cause or contribute to an accident
- Risk – Probability that the system will cause an accident (hazard probability & severity; probability that hazard will lead to an accident)

Safety-critical system: A system where a failure can lead to human injury or death.

Quality software does not contain **faults** and its operation should not end in **failure**. Safety-critical system's **failure** should not lead to **hazards**, even if there is a **risk** for **accidents** or operator makes an **error**.

Fault tolerance

- In critical situations, software systems must be fault tolerant.
- Fault tolerance is required where there are high availability requirements or where system failure costs are very high.
- Fault tolerance means that the system can continue in operation in spite of software failure.
- Even if the system has been proved to conform to its specification, it must also be fault tolerant as there may be specification errors or the validation may be incorrect.

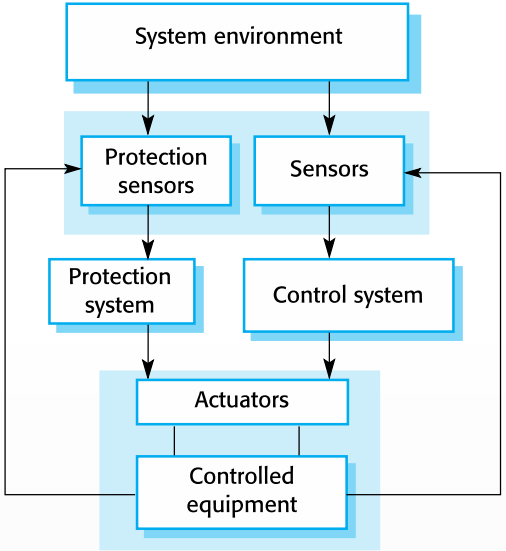
Fault-tolerant system architectures

- Fault-tolerant systems architectures are used in situations where fault tolerance is essential. These architectures are generally all based on redundancy and diversity.
- Examples of situations where dependable architectures are used:
  - Flight control systems, where system failure could threaten the safety of passengers
  - Reactor systems where failure of a control system could lead to a chemical or nuclear emergency
  - Telecommunication systems, where there is a need for 24/7 availability.

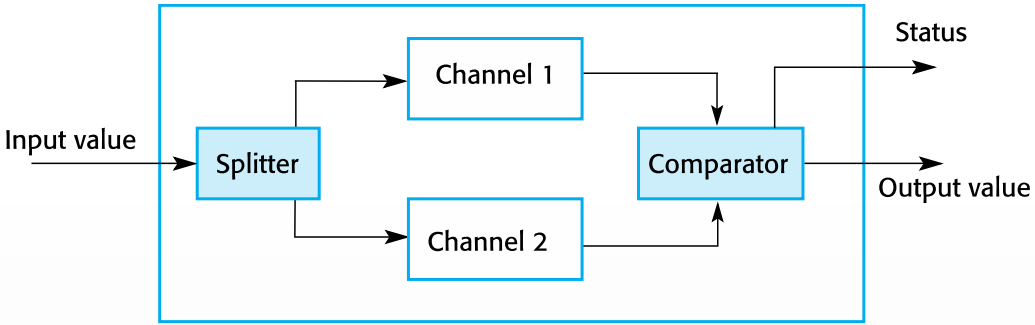
Protection systems

- A specialized system that is associated with some other control system, which can take emergency action if a failure occurs.
  - System to stop a train if it passes a red light
  - System to shut down a reactor if temperature/pressure are too high
- Protection systems independently monitor the controlled system and the environment.
- If a problem is detected, it issues commands to take emergency action to shut down the system and avoid a catastrophe.

Protection system architecture



Self-monitoring architecture



Safety and Reliability

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

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.

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.

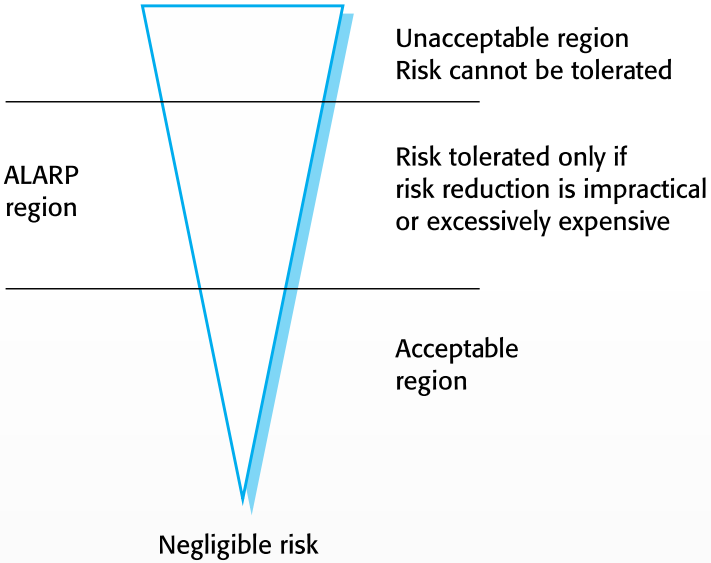
How hazardous faults are?

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 categorized 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



Software criticality – one example

A scale from 1 to 5, expressing how bad things will happen if the software fails.

- 1: None or at most user irritation; for example “user has to reboot the game system”
- 2: Small economical losses; “the ticket fails to print and money is lost”, “no record of sale is made”
- 3: Significant economical losses; “Store has to be closed for a couple of days”, “product stock has to be scrapped”.
- 4: Bodily harm or great economical losses; “Operator loses hand”, “production line has to be closed for repairs.”
- 5: Loss of human life; Operator or people depending on the software system are killed.

Roughly: The more critical the more design-oriented approach, and or “modelled before developed”

Modeling and Reducing Risks & Hazards

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

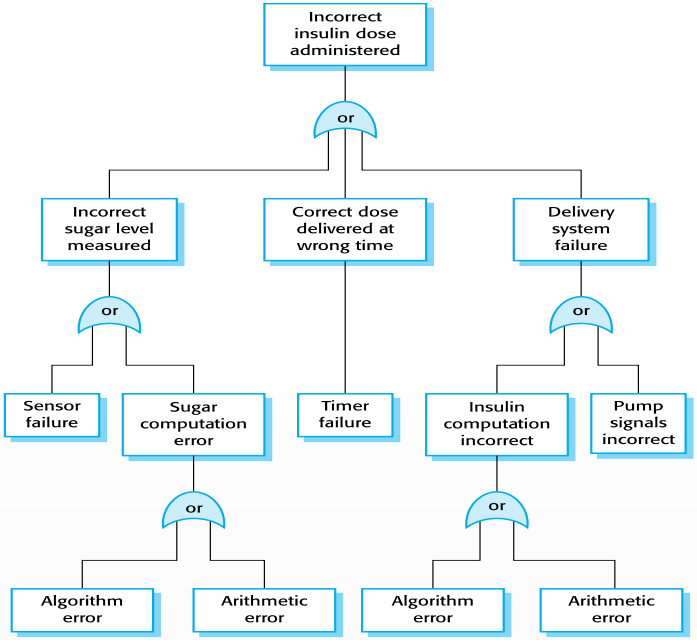
- Concerned with discovering the root causes of risks in a particular system.
- Techniques have been mostly derived from safety-critical systems and can be
  - Inductive, bottom-up techniques. Start with a proposed system failure and assess the hazards that could arise from that failure;
  - Deductive,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

(Reminder: Hazard is 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.)

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



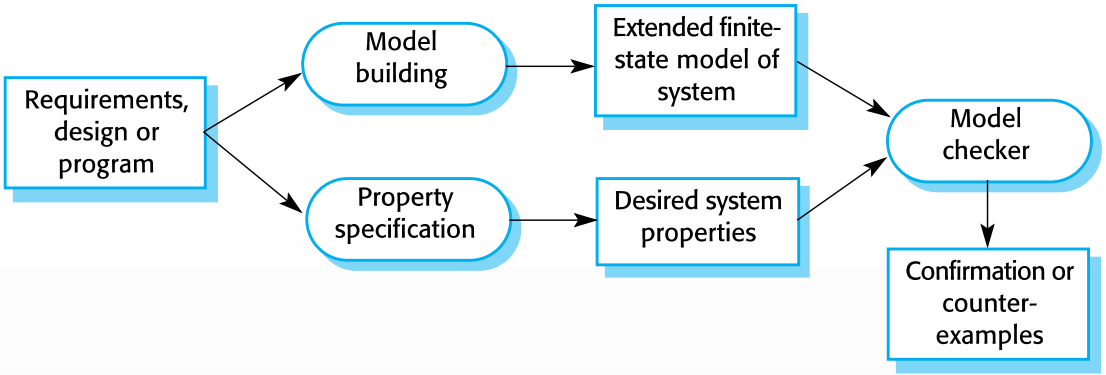
Fault-tree analysis example

- 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

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



Safety Cases and Modeling Them

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, pt. 1

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.

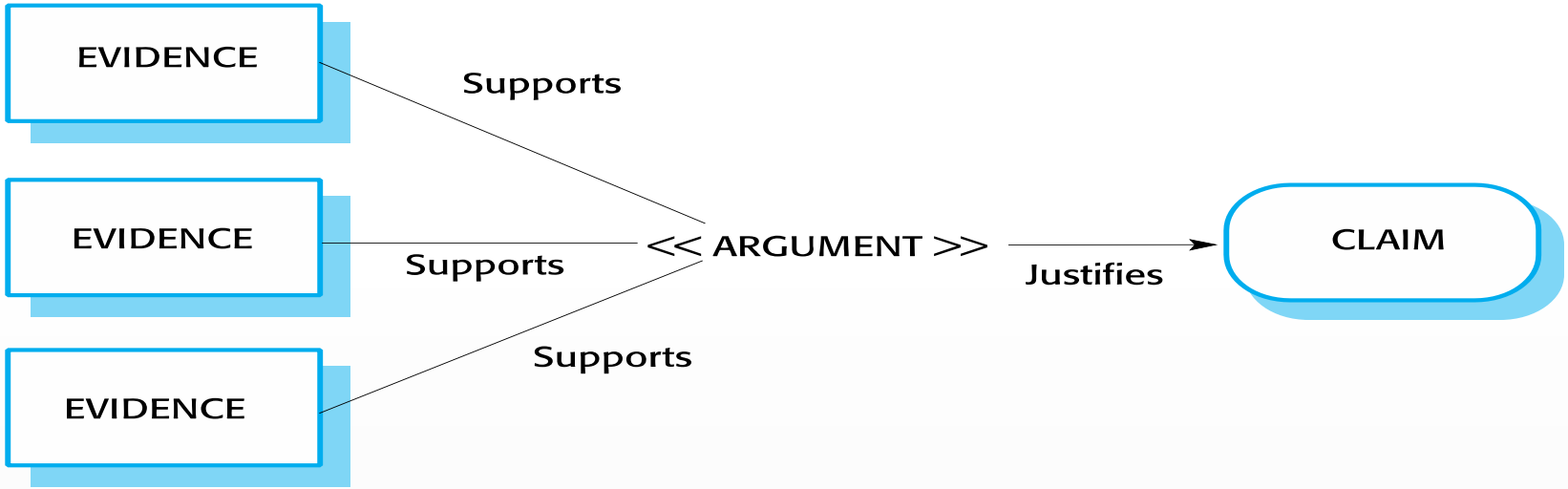
Safety case contents, pt. 2

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



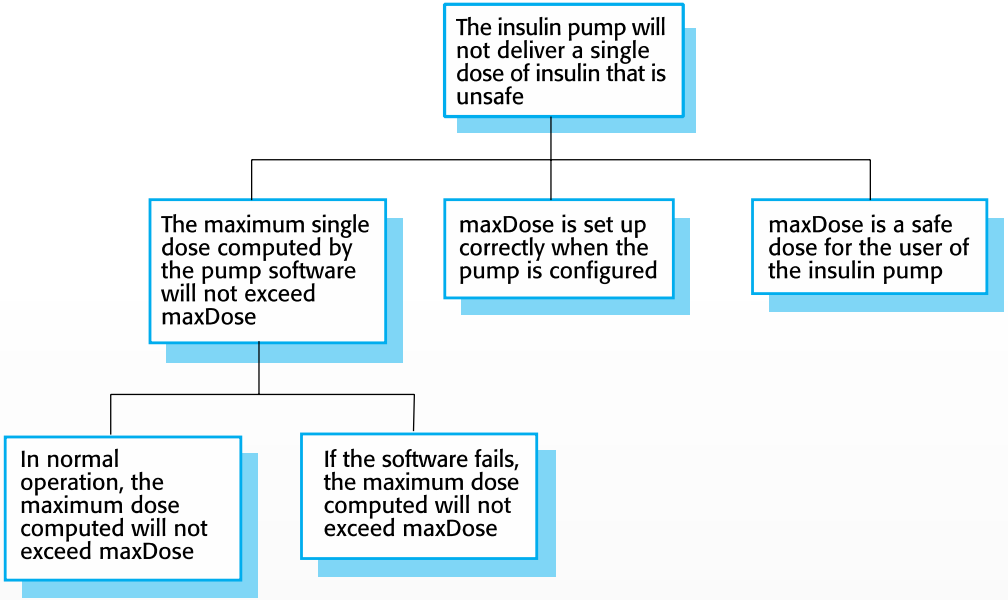
Insulin pump safety argument

- 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



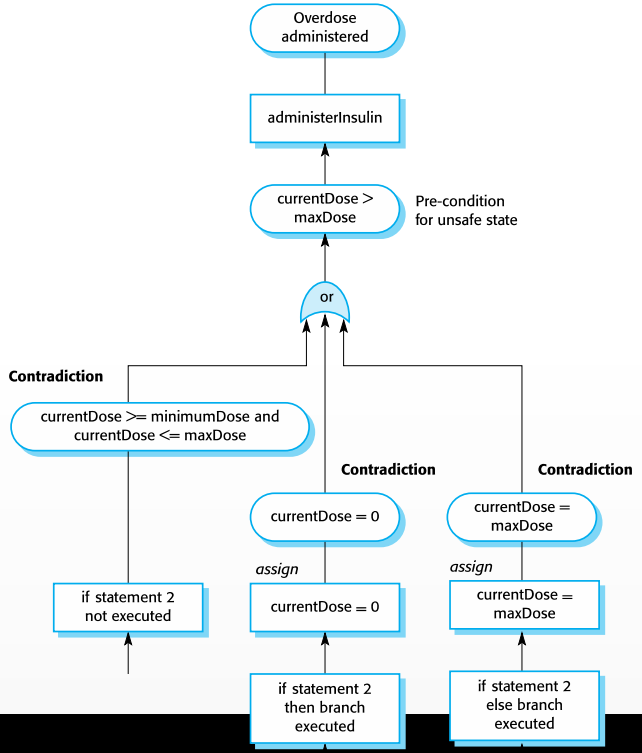
Software safety arguments

- 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.

How to: 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.

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.

Summary

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.