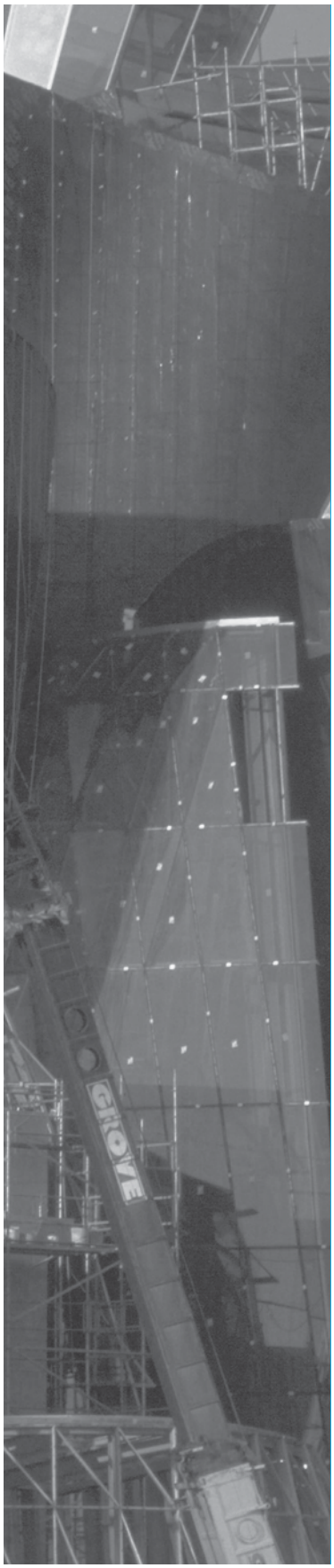

12

Dependability and security specification

Objectives

The objective of this chapter is to explain how to specify functional and non-functional dependability and security requirements. When you have read this chapter, you will:

- understand how a risk-driven approach can be used for identifying and analyzing safety, reliability, and security requirements;
- understand how fault trees can be used to help analyze risks and derive safety requirements;
- have been introduced to metrics for reliability specification and how these are used to specify measurable reliability requirements;
- know the different types of security requirements that may be required in a complex system;
- be aware of the advantages and disadvantages of using formal, mathematical specifications of a system.

Contents

In September 1993, a plane landed at Warsaw airport in Poland during a thunderstorm. For nine seconds after landing, the brakes on the computer-controlled braking system did not work. The braking system had not recognized that the plane had landed and assumed that the aircraft was still airborne. A safety feature on the aircraft had stopped the deployment of the reverse thrust system, which slows down the aircraft, because this can be dangerous if the plane is in the air. The plane ran off the end of the runway, hit an earth bank, and caught fire.

The inquiry into the accident showed that the braking system software had operated according to its specification. There were no errors in the program. However, the software specification was incomplete and had not taken into account a rare situation, which arose in this case. The software worked but the system failed.

This illustrates that system dependability does not just depend on good engineering. It also requires attention to detail when the system requirements are derived and the inclusion of special software requirements that are geared to ensuring the dependability and security of a system. Those dependability and security requirements are of two types:

1. Functional requirements, which define checking and recovery facilities that should be included in the system and features that provide protection against system failures and external attacks.
2. Non-functional requirements, which define the required reliability and availability of the system.

The starting point for generating functional dependability and security requirements is often high-level business or domain rules, policies, or regulations. These are high-level requirements that are perhaps best described as ‘shall not’ requirements. By contrast, with normal functional requirements that define what the system shall do, ‘shall not’ requirements define system behavior that is unacceptable. Examples of ‘shall not’ requirements are:

> “The system shall not allow users to modify access permissions on any files that they have not created.” (security)
>
> “The system shall not allow reverse thrust mode to be selected when the aircraft is in flight.” (safety)
>
> “The system shall not allow the simultaneous activation of more than three alarm signals.” (safety)

These ‘shall not’ requirements cannot be implemented directly but have to be decomposed into more specific software functional requirements. Alternatively, they may be implemented through system design decisions such as a decision to use particular types of equipment in the system.

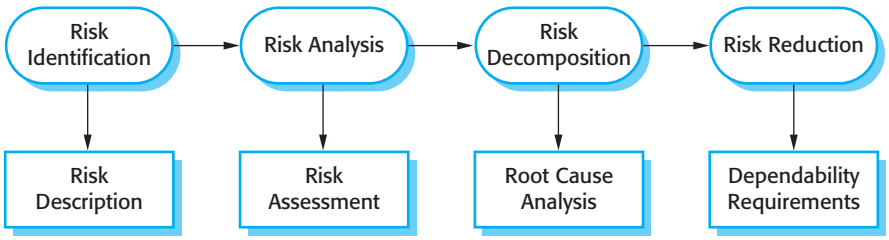

Figure 12.1 Risk-driven specification

12.1 Risk-driven requirements specification

Dependability and security requirements can be thought of as protection requirements. These specify how a system should protect itself from internal faults, stop system failures causing damage to its environment, stop accidents or attacks from the system's environment damaging the system, and facilitate recovery in the event of failure. To discover these protection requirements, you need to understand the risks to the system and its environment. A risk-driven approach to requirements specification takes into account the dangerous events that may occur, the probability that these will actually occur, the probability that damage will result from such an event, and the extent of the damage caused. Security and dependability requirements can then be established, based on an analysis of possible causes of dangerous events.

Risk-driven 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. In safety-critical systems, the risks are associated with hazards that can result in accidents; in security-critical systems, the risks come from insider and outsider attacks on a system that are intended to exploit possible vulnerabilities.

A general risk-driven specification process (Figure 12.1) involves understanding the risks faced by the system, discovering their root causes, and generating requirements to manage these risks. The stages in this process are:

1. *Risk identification* Potential risks to the system are identified. These are dependent on the environment in which the system is to be used. Risks may arise from interactions between the system and rare conditions in its operating environment. The Warsaw accident that I discussed earlier happened when crosswinds generated during a thunderstorm caused the plane to tilt so that (unusually) it landed on one wheel rather than two wheels.

2. *Risk analysis and classification* Each risk is considered separately. Those that are potentially serious and not implausible are selected for further analysis.

At this stage, risks may be eliminated because they are unlikely to arise or because they cannot be detected by the software (e.g., an allergic reaction to the sensor in the insulin pump system).

3. *Risk decomposition* Each risk is analyzed to discover potential root causes of that risk. Root causes are the reasons why a system may fail. They may be software or hardware errors or inherent vulnerabilities that result from system design decisions.
4. *Risk reduction* Proposals for ways in which the identified risks may be reduced or eliminated are made. These contribute to the system dependability requirements that define the defenses against the risk and how the risk will be managed.

For large systems, risk analysis may be structured into phases (Leveson, 1995), where each phase considers different types of risks:

1. Preliminary risk analysis, where major risks from the system's environment are identified. These are independent from the technology used for system development. The aim of preliminary risk analysis is to develop an initial set of security and dependability requirements for the system.
2. Life-cycle risk analysis, which takes place during system development and which is mostly concerned with risks that arise from system design decisions. Different technologies and system architectures have their own associated risks. At this stage, you should extend the requirements to protect against these risks.
3. Operational risk analysis, which is concerned with the system user interface and risks from operator errors. Again, once decisions have been made on the user interface design, further protection requirements may have to be added.

These phases are necessary because it is impossible to make all dependability and security decisions without complete information about the system implementation. Security and dependability requirements are particularly affected by technology choices and design decisions. System checks may have to be included to ensure that third-party components have operated correctly. Security requirements may have to be modified because they conflict with the security features that are provided by an off-the-shelf system.

For example, a security requirement may be that users should identify themselves to a system using a pass phrase rather than a password. Pass phrases are considered to be more secure than passwords. They are harder for an attacker to guess or to discover using an automated password cracking system. However, if a decision is made to use an existing system that only supports password-based authentication, then this security requirement cannot be supported. It may then be necessary to include additional functionality in the system to compensate for the increased risks of using passwords rather than pass phrases.

The IEC standard for safety management

The IEC (International Electrotechnical Commission) has defined a standard for safety management for protection systems (i.e., systems that are intended to trigger safeguards when some dangerous situation arises). An example of a protection system is a system that automatically stops a train if it goes through a red signal. This standard includes extensive guidance on the process of safety specification.

http://www.SoftwareEngineering-9.com/Web/SafetyLifeCycle/

12.2 Safety specification

Safety-critical systems are systems in which failures may affect the environment of the system and cause injury or death to the people in that environment. The principal concern of safety specification is to identify requirements that will minimize the probability that such system failures will occur. Safety requirements are primarily protection requirements and are not concerned with normal system operation. They may specify that the system should be shut down so that safety is maintained. In deriving safety requirements, you therefore need to find an acceptable balance between safety and functionality and avoid overprotection. There is no point in building a very safe system if it does not operate in a cost-effective way.

Recall from the discussion in Chapter 10 that safety-critical systems use a specialized terminology where a hazard is something that could (but need not) result in death or injury to a person, and a risk is the probability that the system will enter a hazardous state. Therefore safety specification is usually focused on the hazards that may arise in a given situation, and the events that can lead to these hazards.

The activities in the general risk-based specification process, shown in Figure 12.1, map onto the safety specification process as follows:

1. *Risk identification* In safety specification, this is the hazard identification process that identifies hazards that may threaten the system.
2. *Risk analysis* This is a process of hazard assessment to decide which hazards are the most dangerous and/or the most likely to occur. These should be prioritized when deriving safety requirements.
3. *Risk decomposition* This process is concerned with discovering the events that can lead to the occurrence of a hazard. In safety specification, the process is known as hazard analysis.
4. *Risk reduction* This process is based on the outcome of hazard analysis and leads to identification of safety requirements. These may be concerned with ensuring that a hazard does not arise or lead to an accident or that if an accident does occur, the associated damage is minimized.

12.2.1 Hazard identification

In safety-critical systems, the principal risks come from hazards that can lead to an accident. You can tackle the hazard identification problem by considering different types of hazards, such as physical hazards, electrical hazards, biological hazards, radiation hazards, service failure hazards, and so on. Each of these classes can then be analyzed to discover specific hazards that could occur. Possible combinations of hazards that are potentially dangerous must also be identified.

The insulin pump system that I have used as an example in earlier chapters is a safety-critical system, because failure can cause injury or even death to the system user. Accidents that may occur when using this machine include the user suffering from long-term consequences of poor blood sugar control (eye, heart, and kidney problems); cognitive dysfunction as a result of low blood sugar levels; or the occurrence of some other medical conditions, such as an allergic reaction.

Some of the hazards in the insulin pump system are:

- insulin overdose computation (service failure);
- insulin underdose computation (service failure);
- failure of the hardware monitoring system (service failure);
- power failure due to exhausted battery (electrical);
- electrical interference with other medical equipment such as a heart pacemaker (electrical);
- poor sensor and actuator contact caused by incorrect fitting (physical);
- parts of machine breaking off in patient's body (physical);
- infection caused by introduction of machine (biological);
- allergic reaction to the materials or insulin used in the machine (biological).

Experienced engineers, working with domain experts and professional safety advisers, identify hazards from previous experience and from an analysis of the application domain. Group working techniques such as brainstorming may be used, where a group of people exchange ideas. For the insulin pump system, people who may be involved include doctors, medical physicists, and engineers and software designers.

Software-related hazards are normally concerned with failure to deliver a system service, or with the failure of monitoring and protection systems. Monitoring and protection systems are included in a device to detect conditions, such as low battery levels, which could lead to device failure.

12.2.2 Hazard assessment

The hazard assessment process focuses on understanding the probability that a hazard will occur and the consequences if an accident or incident associated with that hazard should occur. You need to make this analysis to understand whether a hazard

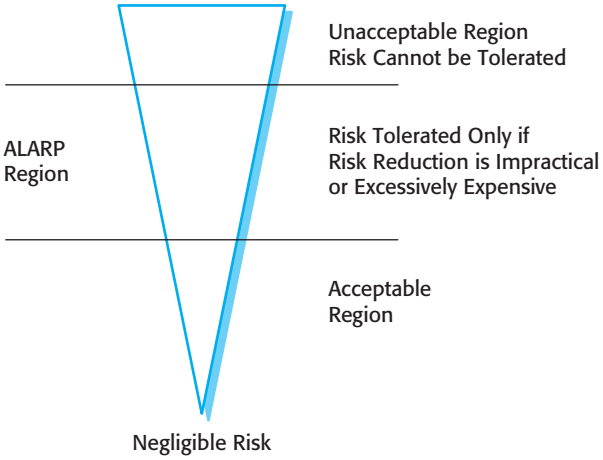

Figure 12.2
The risk triangle

is a serious threat to the system or environment. The analysis also provides a basis for deciding on how to manage the risk associated with the hazard.

For each hazard, the outcome of the analysis and classification process is a statement of acceptability. This is expressed in terms of risk, where the risk takes into account the likelihood of an accident and its consequences. There are three risk categories that you can use in hazard assessment:

1. Intolerable risks in safety-critical systems are those that threaten human life. The system must be designed so that such hazards either cannot arise or, that if they do, features in the system will ensure that they are detected before they cause an accident. In the case of the insulin pump, an intolerable risk is that an overdose of insulin should be delivered.
2. As low as reasonably practical (ALARP) risks are those that have less serious consequences or that are serious but have a very low probability of occurrence. The system should be designed so that the probability of an accident arising because of a hazard is minimized, subject to other considerations such as cost and delivery. An ALARP risk for an insulin pump might be the failure of the hardware monitoring system. The consequences of this are, at worst, a short-term insulin underdose. This is a situation that would not lead to a serious accident.
3. Acceptable risks are those where the associated accidents normally result in minor damage. System designers should take all possible steps to reduce 'acceptable' risks, so long as these do not increase costs, delivery time, or other non-functional system attributes. An acceptable risk in the case of the insulin pump might be the risk of an allergic reaction arising in the user. This usually causes only minor skin irritation. It would not be worth using special, more expensive materials in the device to reduce this risk.

Figure 12.2 (Brazendale and Bell, 1994), developed for safety-critical systems, shows these three regions. The shape of the diagram reflects the costs of ensuring risks do not result in incidents or accidents. The cost of system design to cope with

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 12.3 Risk classification for the insulin pump

the risk is indicated by the width of the triangle. The highest costs are incurred by risks at the top of the diagram, the lowest costs by risks at the apex of the triangle.

The boundaries between the regions in Figure 12.2 are not technical but rather depend on social and political factors. Over time, society has become more risk-averse so the boundaries have moved downwards. Although the financial costs of accepting risks and paying for any resulting accidents may be less than the costs of accident prevention, public opinion may demand that money be spent to reduce the likelihood of a system accident, thus incurring additional costs.

For example, it may be cheaper for a company to clean up pollution on the rare occasion it occurs, rather than to install systems for pollution prevention. However, because the public and the press will not tolerate such accidents, clearing up the damage rather than preventing the accident is no longer acceptable. Such events may also lead to a reclassification of risk. For example, risks that were thought to be improbable (and hence in the ALARP region) may be reclassified as intolerable because of events, such as terrorist attacks, or accidents that have occurred.

Hazard assessment involves estimating hazard probability and risk severity. This is usually difficult as hazards and accidents are uncommon so the engineers involved may not have direct experience of previous incidents or accidents. Probabilities and severities are assigned using relative terms such as ‘probable,’ ‘unlikely,’ and ‘rare’ and ‘high,’ ‘medium,’ and ‘low’. It is only possible to quantify these terms if enough accident and incident data is available for statistical analysis.

Figure 12.3 shows a risk classification for the hazards identified in the previous section for the insulin delivery system. I have separated the hazards that relate to the

incorrect computation of insulin into an insulin overdose and an insulin underdose. An insulin overdose is potentially more serious than an insulin underdose in the short term. Insulin overdose can result in cognitive dysfunction, coma, and ultimately death. Insulin underdoses lead to high levels of blood sugar. In the short term, these cause tiredness but are not very serious; in the longer term, however, they can lead to serious heart, kidney, and eye problems.

Hazards 4–9 in Figure 12.3 are not software related, but software nevertheless has a role to play in hazard detection. The hardware monitoring software should monitor the system state and warn of potential problems. The warning will often allow the hazard to be detected before it causes an accident. Examples of hazards that might be detected are power failure, which is detected by monitoring the battery, and incorrect placement of machine, which may be detected by monitoring signals from the blood sugar sensor.

The monitoring software in the system is, of course, safety related. Failure to detect a hazard could result in an accident. If the monitoring system fails but the hardware is working correctly then this is not a serious failure. However, if the monitoring system fails and hardware failure cannot then be detected, then this could have more serious consequences.

12.2.3 Hazard analysis

Hazard analysis is the process of discovering the root causes of hazards in a safety-critical system. Your aim is to find out what events or combination of events could cause a system failure that results in a hazard. To do this, you can use either a top-down or a bottom-up approach. Deductive, top-down techniques, which tend to be easier to use, start with the hazard and work up from that to the possible system failure. Inductive, bottom-up techniques start with a proposed system failure and identify what hazards might result from that failure.

Various techniques have been proposed as possible approaches to hazard decomposition or analysis. These are summarized by Storey (1996). They include reviews and checklists, formal techniques such as Petri net analysis (Peterson, 1981), formal logic (Jahanian and Mok, 1986), and fault tree analysis (Leveson and Stolzy, 1987; Storey, 1996). As I don't have space to cover all of these techniques here, I focus on a widely used approach to hazard analysis based on fault trees. This technique is fairly easy to understand without specialist domain knowledge.

To do a fault tree analysis, you start with the hazards that have been identified. For each hazard, you then work backwards to discover the possible causes of that hazard. You put the hazard at the root of the tree and identify the system states that can lead to that hazard. For each of these states, you then identify further system states that can lead to them. You continue this decomposition until you reach the root cause(s) of the risk. Hazards that can only arise from a combination of root causes are usually less likely to lead to an accident than hazards with a single root cause.

Figure 12.4 is a fault tree for the software-related hazards in the insulin delivery system that could lead to an incorrect dose of insulin being delivered. In this case, I have

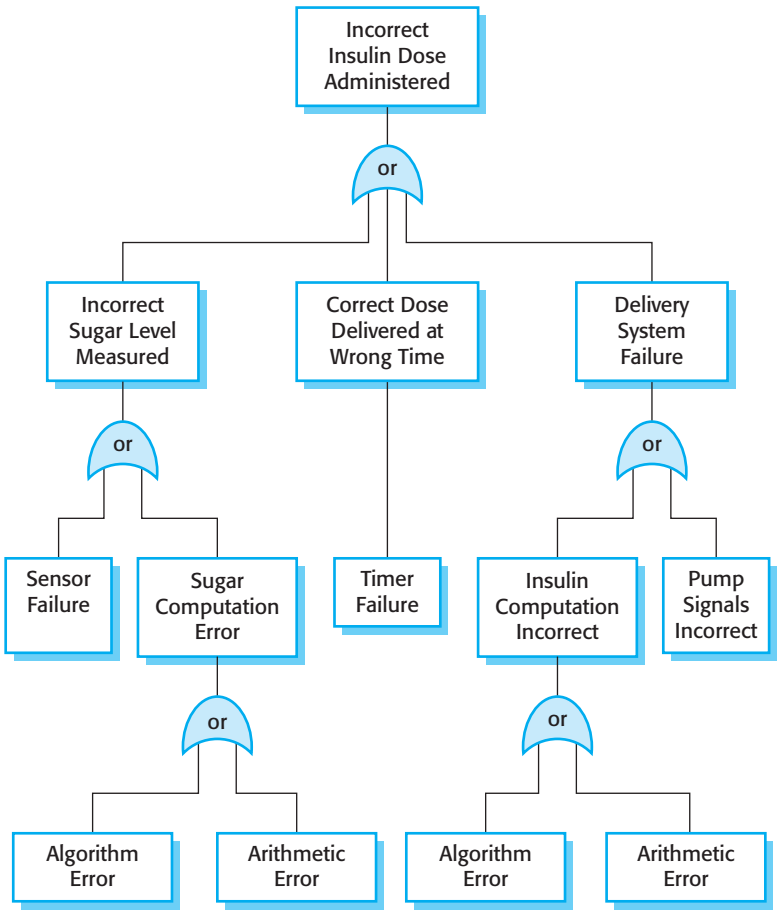

Figure 12.4 An example of a fault tree

merged insulin underdose and insulin overdose into a single hazard, namely 'incorrect insulin dose administered.' This reduces the number of fault trees that are required. Of course, when you specify how the software should react to this hazard, you have to distinguish between an insulin underdose and an insulin overdose. As I have said, they are not equally serious—in the short term, an overdose is the more serious hazard.

From Figure 12.4, you can see that:

1. There are three conditions that could lead to the administration of an incorrect dose of insulin. The level of blood sugar may have been incorrectly measured so the insulin requirement has been computed with an incorrect input. The delivery system may not respond correctly to commands specifying the amount of insulin to be injected. Alternatively, the dose may be correctly computed but it is delivered too early or too late.
2. The left branch of the fault tree, concerned with incorrect measurement of the blood sugar level, looks at how this might happen. This could occur either

because the sensor that provides an input to calculate the sugar level has failed or because the calculation of the blood sugar level has been carried out incorrectly. The sugar level is calculated from some measured parameter, such as the conductivity of the skin. Incorrect computation can result from either an incorrect algorithm or an arithmetic error that results from the use of floating point numbers.

3. The central branch of the tree is concerned with timing problems and concludes that these can only result from system timer failure.
4. The right branch of the tree, concerned with delivery system failure, examines possible causes of this failure. These could result from an incorrect computation of the insulin requirement, or from a failure to send the correct signals to the pump that delivers the insulin. Again, an incorrect computation can result from algorithm failure or arithmetic errors.

Fault trees are also used to identify potential hardware problems. Hardware fault trees may provide insights into requirements for software to detect and, perhaps, correct these problems. For example, insulin doses are not administered at a very high frequency, no more than two or three times per hour and sometimes less often than this. Therefore, processor capacity is available to run diagnostic and self-checking programs. Hardware errors such as sensor, pump, or timer errors can be discovered and warnings issued before they have a serious effect on the patient.

12.2.4 Risk reduction

Once potential risks and their root causes have been identified, you are then able to derive safety requirements that manage the risks and ensure that incidents or accidents do not occur. There are three possible strategies that you can use:

1. *Hazard avoidance* The system is designed so that the hazard cannot occur.
2. *Hazard detection and removal* The system is designed so that hazards are detected and neutralized before they result in an accident.
3. *Damage limitation* The system is designed so that the consequences of an accident are minimized.

Normally, designers of critical systems use a combination of these approaches. In a safety-critical system, intolerable hazards may be handled by minimizing their probability and adding a protection system that provides a safety backup. For example, in a chemical plant control system, the system will attempt to detect and avoid excess pressure in the reactor. However, there may also be an independent protection system that monitors the pressure and opens a relief valve if high pressure is detected.

In the insulin delivery system, a ‘safe state’ is a shutdown state where no insulin is injected. Over a short period this is not a threat to the diabetic’s health. For the

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 12.5 Examples of safety requirements

software failures that could lead to an incorrect dose of insulin are considered, the following 'solutions' might be developed:

1. *Arithmetic error* This may occur when an arithmetic computation causes a representation failure. The specification should identify all possible arithmetic errors that may occur and state that an exception handler must be included for each possible error. The specification should set out the action to be taken for each of these errors. The default safe action is to shut down the delivery system and activate a warning alarm.
2. *Algorithmic error* This is a more difficult situation as there is no clear program exception that must be handled. This type of error could be detected by comparing the required insulin dose computed with the previously delivered dose. If it is much higher, this may mean that the amount has been computed incorrectly. The system may also keep track of the dose sequence. After a number of above-average doses have been delivered, a warning may be issued and further dosage limited.

Some of the resulting safety requirements for the insulin pump software are shown in Figure 12.5. These are user requirements and, naturally, they would be expressed in more detail in the system requirements specification. In Figure 12.5, the references to Tables 3 and 4 relate to tables that are included in the requirements document—they are not shown here.

12.3 Reliability specification

As I discussed in Chapter 10, the overall reliability of a system depends on the hardware reliability, the software reliability, and the reliability of the system operators. The system software has to take this into account. As well as including requirements

that compensate for software failure, there may also be related reliability requirements to help detect and recover from hardware failures and operator errors.

Reliability is different from safety and security in that it is a measurable system attribute. That is, it is possible to specify the level of reliability that is required, monitor the system's operation over time, and check if the required reliability has been achieved. For example, a reliability requirement might be that system failures that require a reboot should not occur more than once per week. Every time such a failure occurs, it can be logged and you can check if the required level of reliability has been achieved. If not, you either modify your reliability requirement or submit a change request to address the underlying system problems. You may decide to accept a lower level of reliability because of the costs of changing the system to improve reliability or because fixing the problem may have adverse side effects, such as lower performance or throughput.

By contrast, both safety and security are about avoiding undesirable situations, rather than specifying a desired 'level' of safety or security. Even one such situation in the lifetime of a system may be unacceptable and, if it occurs, system changes have to be made. It makes no sense to make statements like 'system faults should result in fewer than 10 injuries per year.' As soon as one injury occurs, the system problem must be rectified.

Reliability requirements are, therefore, of two kinds:

1. Non-functional requirements, which define the number of failures that are acceptable during normal use of the system, or the time in which the system is unavailable for use. These are quantitative reliability requirements.
2. Functional requirements, which define system and software functions that avoid, detect, or tolerate faults in the software and so ensure that these faults do not lead to system failure.

Quantitative reliability requirements lead to related functional system requirements. To achieve some required level of reliability, the functional and design requirements of the system should specify the faults to be detected and the actions that should be taken to ensure that these faults do not lead to system failures.

The process of reliability specification can be based on the general risk-driven specification process shown in Figure 12.1:

1. *Risk identification* At this stage, you identify the types of system failures that may lead to economic losses of some kind. For example, an e-commerce system may be unavailable so that customers cannot place orders, or a failure that corrupts data may require time to restore the system database from a backup and rerun transactions that have been processed. The list of possible failure types, shown in Figure 12.6, can be used as a starting point for risk identification.
2. *Risk analysis* This involves estimating the costs and consequences of different types of software failure and selecting high-consequence failures for further analysis.

Failure type	Description
Loss of service	The system is unavailable and cannot deliver its services to users. You may separate this into loss of critical services and loss of non-critical services, where the consequences of a failure in non-critical services are less than the consequences of critical service failure.
Incorrect service delivery	The system does not deliver a service correctly to users. Again, this may be specified in terms of minor and major errors or errors in the delivery of critical and non-critical services.
System/data corruption	The failure of the system causes damage to the system itself or its data. This will usually but not necessarily be in conjunction with other types of failures.

Figure 12.6 Types of system failure

3. *Risk decomposition* At this stage, you do a root cause analysis of serious and probable system failures. However, this may be impossible at the requirements stage as the root causes may depend on system design decisions. You may have to return to this activity during design and development.

4. *Risk reduction* At this stage, you should generate quantitative reliability specifications that set out the acceptable probabilities of the different types of failures. These should, of course, take into account the costs of failures. You may use different probabilities for different system services. You may also generate functional reliability requirements. Again, this may have to wait until system design decisions have been made. However, as I discuss in Section 12.3.2, it is sometimes difficult to create quantitative specifications. You may only be able to identify functional reliability requirements.

12.3.1 Reliability metrics

In general terms, reliability can be specified as a probability that a system failure will occur when a system is in use within a specified operating environment. If you are willing to accept, for example, that 1 in any 1,000 transactions may fail, then you can specify the failure probability as 0.001. This doesn't mean, of course, that you will see 1 failure in every 1,000 transactions. It means that if you observe N thousand transactions, the number of failures that you observe should be around N. You can refine this for different kinds of failure or for different parts of the system. You may decide that critical components must have a lower probability of failure than noncritical components.

There are two important metrics that are used to specify reliability plus an additional metric that is used to specify the related system attribute of availability. The choice of metric depends on the type of system that is being specified and the requirements of the application domain. The metrics are:

1. *Probability of failure on demand (POFOD)* If you use this metric, you define the probability that a demand for service from a system will result in a system