17 Critical systems specification

Objectives

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

- have been introduced to a number of metrics for reliability specification and will understand how these metrics may be used to specify reliability requirements;
- understand how safety requirements for critical systems may be derived from an analysis of hazards and risks;
- understand the similarities between the processes for specifying safety and security requirements.

Contents

I have discussed requirements processes and techniques for the development of system specifications in Chapters 5–9. This chapter supplements these chapters with a discussion of particular issues that arise in the specification of critical systems. Because of the high potential costs of system failure, it is important to ensure that the specification for critical systems is high quality and accurately reflects the real needs of users of the system.

The need for dependability in critical systems generates both functional and non-functional system requirements:

1. System functional requirements may be generated to define error checking and recovery facilities and features that provide protection against system failures.
2. Non-functional requirements may be generated to define the required reliability and availability of the system.

In addition to these requirements, safety and security considerations can generate a further type of requirement that is difficult to classify as a functional or a non-functional requirement. These are perhaps best described as the 'shall not' requirements. By contrast with normal functional requirements that define what the system shall do, 'shall not' requirements define system behaviour 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 are sometimes decomposed into more specific software functional requirements. Alternatively, implementation decisions may be deferred until the system is designed.

The user requirements for critical systems will always be specified using natural language and system models. However, as I discuss in Chapter 9, formal specification and associated verification are most likely to be cost-effective in critical systems development (Hall, 1996; Wordsworth, 1996). Formal specifications are not just a basis for a verification of the design and implementation. They are the most precise way of specifying systems, so reduce the scope for misunderstanding. Furthermore, constructing a formal specification forces a detailed analysis of the requirements and this is an effective way of discovering problems in the specification.

17.1 Software reliability specification

Reliability is a complex concept which should always be considered at the system rather than the individual component level. Because the components in a system are interdependent, a failure in one component can be propagated through the system and affect the operation of other components. In a computer-based system, you have to consider three dimensions when specifying the overall system reliability:

1. *Hardware reliability* What is the probability of a hardware component failing and how long does it take to repair that component?
2. *Software reliability* How likely is it that a software component will produce an incorrect output? Software failures are different from hardware failures in that software does not wear out. It can continue operating correctly after an incorrect result has been produced.
3. *Operator reliability* How likely is it that the operator of a system will make an error?

All of these are closely linked. Hardware failure can cause spurious signals to be generated that are outside the range of inputs expected by software. The software can then behave unpredictably. Unexpected system behaviour may confuse the operator and may result in operator stress. Operator error is most likely in conditions of stress. The operator may then act incorrectly and supply inputs that are inappropriate for the current failure situation. These inputs further confuse the system and more errors are generated. Therefore, a situation can occur where a single sub-system failure that is recoverable can rapidly develop into a serious problem requiring a complete system shutdown.

Software reliability engineering (Lyu, 1996) is a specialised sub-discipline of systems engineering that is concerned with making overall judgements on system reliability. It takes into account the probabilities of failure of different components in a system and how these are combined to affect the overall reliability of a system. Simplistically, if a system depends on component A and component B with failure probabilities of P_A and P_B, then the overall probability of system failure P_S is:

$$P_S = P_A + P_B$$

As the number of dependent components increases, the overall probability of system failure increases. If there are very many critical components in a system then these individual components must be very reliable to ensure that P_S is relatively low. To increase reliability, components may be replicated as discussed in Chapter 18. A number of identical components work together and the component group is operational so long as any one component works correctly. This means (simplistically

again) that if the probability of failure of a component is P_A and if all failures are independent then the overall failure probability P_S of a group of n identical replicated components is computed by multiplying the probability of failure of each component:

$$P_S = P_A^n$$

Systems reliability should be specified as a non-functional requirement that is expressed quantitatively using one of the metrics discussed in the next section. To meet the non-functional reliability requirements, it may be necessary to specify additional functional and design requirements on the system that specify how failures may be avoided or tolerated. Examples of these reliability requirements are:

1. A predefined range for all values that are input by the operator shall be defined and the system shall check that all operator inputs fall within this predefined range.
2. As part of the initialisation process, the system shall check all disks for bad blocks.
3. N-version programming shall be used to implement the braking control system.
4. The system must be implemented in a safe subset of Ada and checked using static analysis (see Chapter 19).

There are no simple rules that may be used to derive functional reliability requirements. In organisations that develop critical systems, there is usually organisational knowledge about possible reliability requirements and how these impact the actual reliability of a system. These organisations may specialise in specific types of system such as railway control systems, so the reliability requirements, once derived, are reused across a range of systems.

17.1.1 Reliability metrics

Reliability metrics were first devised for hardware components. Hardware component failure is inevitable due to physical factors such as mechanical abrasion, electrical heating, etc. Components have some average lifetime and this is reflected in the most widely used hardware reliability metric, 'Mean Time to Failure' (MTTF). The MTTF is the mean time for which a component is expected to be operational. Once a hardware component fails then the failure is usually permanent, so the 'Mean Time to Repair' (MTTR) which reflects the time taken to repair or replace the component is also significant.

However, these hardware metrics are not always applicable for software reliability specification because of the differing nature of software and hardware failures. Software component failures are often transient rather than permanent. They only manifest themselves with some inputs. If the data is undamaged, the system can often continue in operation after a failure has occurred.

Metrics which have been used for specifying software reliability and availability are shown in Figure 17.1. The choice of which metric should be used depends

Figure 17.1
Reliability metrics

Metric	Explanation
POFOD Probability of failure on demand	The likelihood that the system will fail when a service request is made. A POFOD of 0.001 means that one out of a thousand service requests may result in failure.
ROCOF Rate of failure occurrence	The frequency of occurrence with which unexpected behaviour is likely to occur. A ROCOF of 2/100 means that two failures are likely to occur in each 100 operational time units. This metric is sometimes called the failure intensity.
MTTF Mean time to failure	The average time between observed system failures. An MTTF of 500 means that one failure can be expected every 500 time units.
AVAIL Availability	The probability that the system is available for use at a given time. Availability of 0.998 means that in every 1000 time units, the system is likely to be available for 998 of these.

on the type of system to which it applies and the requirements of the application domain. Some examples of the types of system where these different metrics may be used are:

1. *Probability of failure on demand* This metric is most appropriate for systems where services are demanded at unpredictable or at relatively long time intervals and where there are serious consequences if the service is not delivered. It might be used to specify protection systems such as the reliability of a pressure relief system in a chemical plant or an emergency shutdown system in a power plant.
2. *Rate of occurrence of failures* This metric should be used where regular demands are made on system services and where it is important that these services are correctly delivered. It might be used in the specification of a bank teller system that processes customer transactions or in an hotel reservation system.
3. *Mean time to failure* This metric should be used in systems where there are long transactions, i.e. where people use the system for a long time. The mean time to failure should be longer than the average length of transaction. Examples of systems where this metric may be used are word processor systems or CAD systems.
4. *Availability* This metric should be used in non-stop systems where users expect the system to deliver a continuous service. Examples of such systems are telephone switching systems and railway signalling systems.

There are three kinds of measurement which can be made when assessing the reliability of a system:

1. The number of system failures given a number of requests for system services. This is used to measure the POFOD.
2. The time (or number of transactions) between system failures. This is used to measure ROCOF and MTTF.
3. The elapsed repair or restart time when a system failure occurs. Given that the system must be continuously available, this is used to measure AVAIL.

Time units which may be used in these metrics are calendar time, processor time or may be some discrete unit such as number of transactions. In systems that spend much of their time waiting to respond to a service request, such as telephone switching systems, the time unit that should be used is processor time. Basing reliability on calendar time would give an optimistic figure.

Calendar time is an appropriate time unit to use for systems which are in continuous operation. For example, monitoring systems, such as alarm systems, and other types of process control systems fall into this category. Systems which process transactions such as bank ATMs or airline reservation systems have variable loads placed on them depending on the time of day. In these cases, the unit of 'time' used should be the number of transactions, i.e. the ROCOF would be number of failed transactions per N thousand transactions.

17.1.2 Non-functional reliability requirements

In many system requirements documents, reliability requirements are not carefully specified. The reliability specifications are subjective and unmeasurable. For example, statements such as 'The software shall be reliable under normal conditions of use' are meaningless. Quasi-quantitative statements such as 'The software shall exhibit no more than N faults/1000 lines' are equally useless. It is impossible to measure the number of faults/1000 lines of code as you can't tell when all faults have been discovered. Furthermore, the statement means nothing in terms of the dynamic behaviour of the system. It is software failures not software faults that affect the reliability of a system.

The types of failure that can occur are system specific and the consequences of a system failure depend on the nature of that failure. When writing a reliability specification, the specifier should identify different types of failure and consider whether these should be treated differently in the specification. Examples of different types of failure are shown in Figure 17.2. Obviously, combinations of these such as a failure which is transient, recoverable and corrupting can occur.

Most large systems are composed of several sub-systems with different reliability requirements. Because very highly reliable software is expensive, it is usually sensible to assess the reliability requirements of each sub-system separately rather than impose the same reliability requirement on all sub-systems. This avoids placing unnecessarily high demands for reliability on those sub-systems where it is unnecessary.

The steps involved in establishing a reliability specification are:

Figure 17.2 Failure classification

Failure class	Description
Transient	Occurs only with certain inputs
Permanent	Occurs with all inputs
Recoverable	System can recover without operator intervention
Unrecoverable	Operator intervention needed to recover from failure
Non-corrupting	Failure does not corrupt system state or data
Corrupting	Failure corrupts system state or data

1. For each identified sub-system, identify the different types of system failure which may occur and analyse the consequences of these failures.
2. From the system failure analysis, partition failures into appropriate classes. A reasonable starting point is to use the failure types shown in Figure 17.2.
3. For each failure class identified, define the reliability requirement using an appropriate reliability metric. It is not necessary to use the same metric for different classes of failure. If a failure requires some intervention to recover from it, the probability of that failure occurring on demand might be the most appropriate metric. When automatic recovery is possible and the effect of the failure is user inconvenience, ROCOF might be more appropriate.
4. Where appropriate, identify functional reliability requirements that define system functionality to reduce the probability of critical failures.

As an example of a reliability specification, consider the reliability requirements for a bank auto-teller machine (ATM). Assume that each machine in the network is used about 300 times per day. The lifetime of the system hardware is eight years and the software is normally upgraded every two years. Therefore, during the lifetime of a software release, each machine will handle about 200,000 transactions. A bank has 1000 machines in its network. This means that there are 300,000 transactions on the central database per day (say 100 million per year).

Failures fall into two broad classes: those that affect a single machine in the network and those that affect the database and therefore all ATMs in the network. Clearly, the latter type of failure is less acceptable than those failures which are local to an ATM.

Figure 17.3 shows possible failure classes and possible reliability specifications for different types of system failure. The reliability requirements state that it is acceptable for a permanent failure to occur in a machine roughly once per three years. This means that, on average, one machine in the banking network might be affected each day. By contrast, faults which simply mean that a transaction has to be aborted

Figure 17.3 Reliability specification for an ATM

Failure class	Example	Reliability metric
Permanent, non-corrupting	The system fails to operate with any card which is input. Software must be restarted to correct failure.	ROCOF 1 occurrence/1000 days
Transient, non-corrupting	The magnetic stripe data cannot be read on an undamaged card which is input.	ROCOF 1 in 1000 transactions
Transient, corrupting	A pattern of transactions across the network causes database corruption.	Unquantifiable! Should never happen in the lifetime of the system

and the user must start again can occur relatively frequently. Their only effect is to cause minor user inconvenience.

Ideally, faults that corrupt the database should never occur in the lifetime of the software. Therefore, the reliability requirement which might be placed on this is that the probability of a corrupting failure occurring when a demand is made is less than 1 in 200 million transactions. That is, in the lifetime of an ATM software release, there should never be an error which causes database corruption.

However, a reliability requirement like this cannot actually be tested. Say each transaction takes 1 second of machine time and a simulator can be built for the ATM network. Simulating the transactions which take place in a single day across the network will take 300,000 seconds. This is approximately 3.5 days. Clearly this period could be reduced by reducing the transaction time and using multiple simulators but it is still very difficult to test the system to validate the reliability specification.

It is impossible to validate qualitative requirements which demand a very high level of reliability. For example, say a system was intended for use in a safety-critical application, so it should never fail over the total lifetime of the system. Assume that 1000 copies of the system are to be installed and the system is 'executed' 1000 times per second. The projected lifetime of the system is 10 years. The total estimated number of system executions is therefore approximately $3 * 10^{14}$. There is no point in specifying that the rate of occurrence of failure should be $1/10^{15}$ executions (this allows for some safety factor) as you cannot test the system for long enough to validate this level of reliability.

As a further example of reliability specification, consider the insulin pump system that was introduced in Chapter 16. This system delivers insulin a number of times per day and the user's blood glucose is monitored several times per hour. Because the use of the system is intermittent and failure consequences are serious, the most appropriate reliability metric is POFOD (Probability of Failure on Demand).

As I discuss in the following section, failure to deliver insulin does not have immediate safety implications so commercial factors rather than safety factors govern the level of reliability required. Service costs are high because users need a very

fast repair and replacement service. It is in the manufacturer's interest to limit the number of permanent failures that require repair.

Again, two types of failure can be identified:

1. Transient failures that can be repaired by user actions such as resetting or recalibrating the machine. For these types of failure, a relatively low value of POFOD (say 0.002) may be acceptable. This means that one failure may occur in every 500 demands made on the machine. This is approximately once every 3.5 days.
2. Permanent failures that require the machine to be repaired by the manufacturer. The probability of this type of failure should be much lower. Roughly once a year is the minimum figure, so POFOD should be no more than 0.00002.

The cost of developing and validating a system reliability specification can be very high. Organisations must be realistic about whether these costs are worthwhile. They are clearly justified in systems where reliable operation is critical, such as telephone switching systems, or where system failure may result in large economic losses. They are probably not justified for many types of business or scientific system. These usually have relatively modest reliability requirements as the costs of failure are simply processing delays and it is straightforward and relatively inexpensive to recover from these.

17.2 Safety specification

Safe operation is the required characteristic of a safety-related software system. This means that you must consider, during the requirements engineering process, potential hazards that might arise. Each hazard should be assessed for the risk it poses and the specification may either describe how the software should behave to minimise the risk or might require that the hazard should never arise.

The process of safety specification and assurance is part of an overall 'safety life cycle' that is illustrated in Figure 17.4. Figure 17.4 is a simplified form of Redmill's presentation of the safety life cycle (Redmill, 1998) as proposed in an international standard for safety management IEC 61508 (IEC, 1998). As you can see from Figure 17.4, this standard covers all aspects of safety management from initial scope definition through planning and system development to system decommissioning.

The first stages of the IEC standard 61508 safety life cycle define the scope of the system, assess the potential system hazards and estimate the risk they pose. This is followed by safety requirements specification and the allocation of these safety requirements to different sub-systems. The development activity involves planning

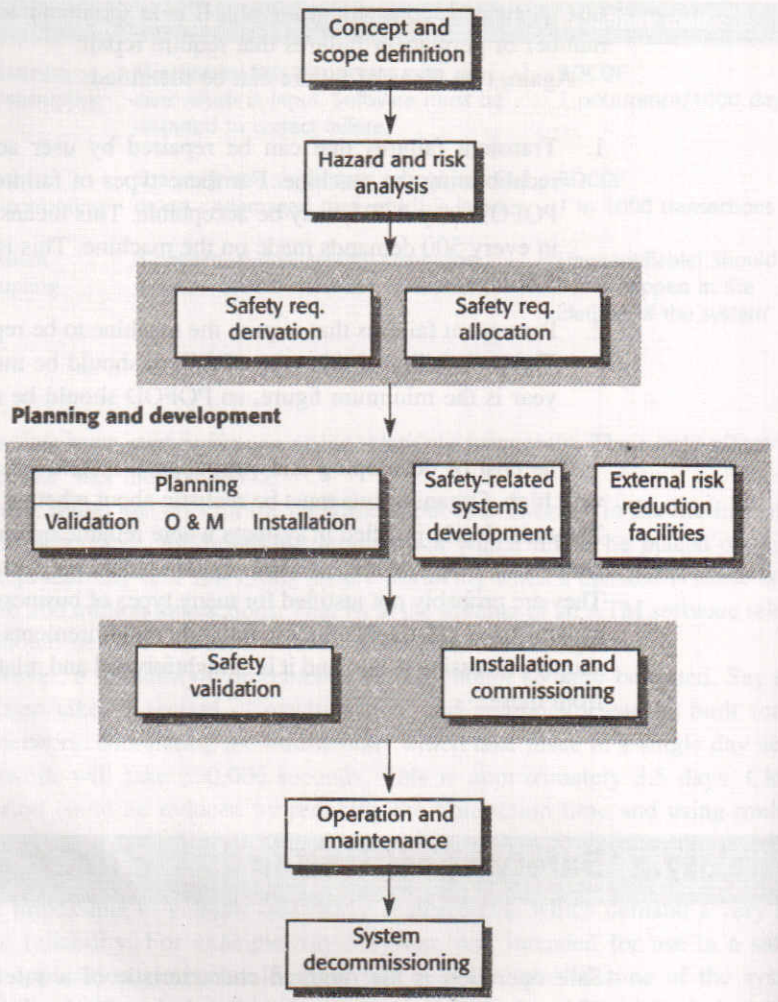

Figure 17.4 The IEC 61508 safety life cycle

and implementation. The safety-critical system itself is designed and implemented, as are related external systems that may provide additional protection. In parallel with this, the safety validation, the installation and the operation and maintenance of the system are planned.

Safety management does not stop on delivery of the system. After delivery, the system must be installed as planned so that the hazard analysis remains valid. Safety validation is then carried out before the system is put into use. Safety must also be managed during the operation and (particularly) the maintenance of the system. Many safety-related systems problems arise because of a poor maintenance process, so it is particularly important that the system is designed for maintainability. Finally, safety considerations that may apply during decommissioning (e.g. disposal of hazardous material in circuit boards) should also be taken into account.

17.2.1 Hazard and risk analysis

Hazard and risk analysis involves analysing the system and its operational environment. Its objective is to discover potential hazards that might arise in that environment, the root causes of these hazards and the risks associated with them. This is a complex and difficult process which requires lateral thinking and input from many different sources of expertise. It should be undertaken by experienced engineers in conjunction with domain experts and professional safety advisers. Group working techniques such as brainstorming may be used to identify hazards. Hazards may also be identified because one of the analysts involved has direct experience of some previous incident which resulted in a hazard.

Figure 17.5 shows the iterative process of hazard and risk analysis:

1. *Hazard identification* Potential hazards which might arise are identified. These are dependent on the environment in which the system is to be used.
2. *Risk analysis and hazard classification* The hazards are considered separately. Those which are potentially serious and not implausible are selected for further analysis. At this stage, some hazards may be eliminated simply because they are very unlikely ever to arise (e.g. simultaneous lightning strike and earthquake).
3. *Hazard decomposition* Each hazard is analysed individually to discover potential causes of that hazard. Techniques such as fault-tree analysis (discussed later in this section) may be used.
4. *Risk reduction assessment* Proposals for ways in which the identified risks may be reduced or eliminated are made. These are input to a more detailed safety requirements specification activity as shown in the safety life cycle model (Figure 17.4).

For large systems, hazard and risk analysis is usually structured into a number of phases (Leveson, 1986). These include:

- preliminary hazard analysis where major risks are identified;
- more detailed system and sub-system hazard analysis;
- software hazard analysis where the risks of software failure are considered;
- operational hazard analysis which is concerned with the system user interface and risks that arise from operator errors.

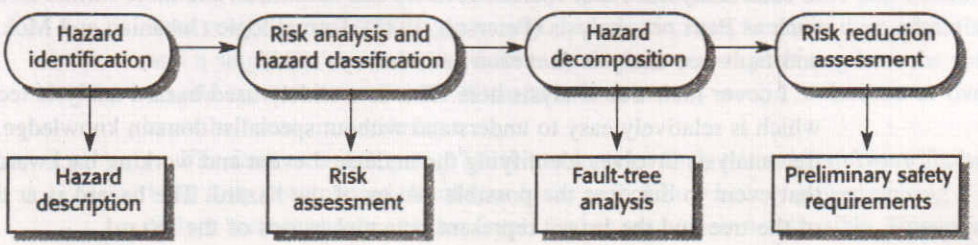

Figure 17.5 Hazard and risk analysis

These analyses identify hazards and associate a risk with each hazard. This involves estimating the probability that the hazard will arise, estimating the probability that the hazard will cause a mishap and estimating the likely severity of that mishap. Engineering judgement is used to make these risk assessments.

The process of hazard analysis generally involves considering different classes of hazard such as physical hazards, electrical hazards, biological hazards, radiation hazards (where appropriate), hazards due to service failure and so on. Each of these classes is then analysed in detail to discover associated hazards.

For the insulin delivery system, introduced above, the hazards and their associated classes are:

1. insulin overdose (sẽrvice failure);
2. insulin underdose (service failure);
3. power failure due to exhausted battery (electrical);
4. machine interferes electrically with other medical equipment such as a heart pacemaker (electrical);
5. poor sensor and actuator contact caused by incorrect fitting (physical);
6. parts of machine break off in patient's body (physical);
7. infection caused by introduction of machine (biological);
8. allergic reaction to the materials or insulin used in the machine (biological).

The risks associated with these hazards are covered in section 17.2.3. Because this is a small system, multiple phases of hazard analysis are not necessary. Many safety-critical systems, however, are very large (e.g. a chemical plant) and hazard analysis is a long, complex and expensive process.

17.2.2 Fault-tree analysis

For each identified hazard, a detailed analysis should be carried out to discover the conditions which might cause that hazard. Hazard analysis techniques can be either deductive or inductive. Deductive techniques, which tend to be easier to use, start with the hazard and work from that to the possible system failure; inductive techniques start with a proposed system failure and identify which hazards might arise. Wherever possible, both inductive and deductive techniques should be used for hazard analysis.

There are various techniques which have been proposed as possible approaches to such analyses. These include reviews and checklists, and more formal techniques such as Petri net analysis (Peterson, 1981), formal logic (Jahanian and Mok, 1986) and fault-tree analysis (Leveson and Harvey, 1983).

I cover fault-tree analysis here. This is a widely used hazard analysis technique which is relatively easy to understand without specialist domain knowledge. Fault-tree analysis involves identifying the undesired event and working backwards from that event to discover the possible causes of the hazard. The hazard is at the root of the tree and the leaves represent potential causes of the hazard.

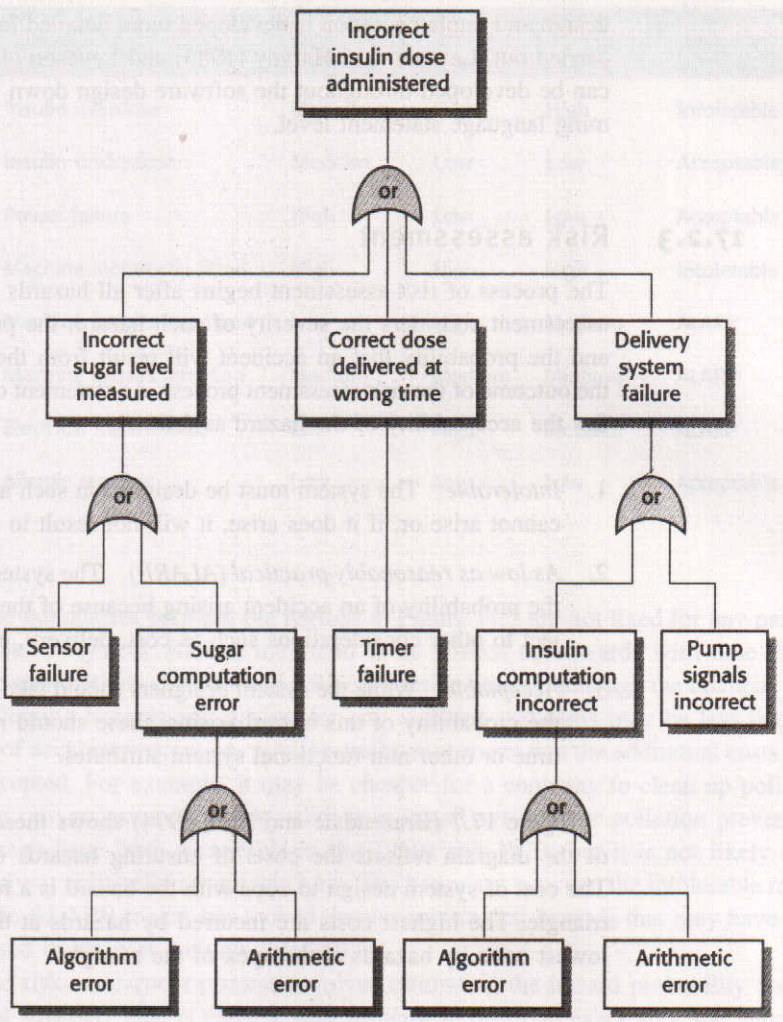

Figure 17.6 Fault tree for insulin delivery system

Software-related hazards are normally concerned with failure to deliver a system service or with the failure of monitoring systems. Monitoring systems may detect potentially hazardous conditions such as power failures.

Figure 17.6 is the fault tree which can be identified for the possible software-related hazards in the insulin delivery system. Insulin underdose and insulin overdose really represent a single hazard, namely 'incorrect insulin dose administered' and a single fault tree can be drawn. Of course, when specifying how the software should react to hazards, the distinction between an insulin underdose or overdose must be taken into account.

The fault tree in Figure 17.6 is incomplete. Only potential software faults have been fully decomposed. Hardware faults such as low battery power causing a sensor failure are not shown. At this level, further analysis is not possible. However, as a

design and implementation is developed more detailed fault tree analyses should be carried out. Leveson and Harvey (1983) and Leveson (1985) show how fault trees can be developed throughout the software design down to the individual programming language statement level.

17.2.3 Risk assessment

The process of risk assessment begins after all hazards have been identified. Risk assessment considers the severity of each hazard, the probability that it will arise and the probability that an accident will result from the hazard. For each hazard, the outcome of the risk assessment process is a statement of acceptability. This classifies the acceptability of the hazard as follows:

1. *Intolerable* The system must be designed in such a way that either the hazard cannot arise or, if it does arise, it will not result in an accident.
2. *As low as reasonably practical (ALARP)* The system must be designed so that the probability of an accident arising because of the hazard is minimised, subject to other considerations such as cost, delivery, etc.
3. *Acceptable* While the system designers should take all possible steps to reduce the probability of this hazard arising, these should not increase costs, delivery time or other non-functional system attributes.

Figure 17.7 (Brazendale and Bell, 1994) shows these three regions. The shape of the diagram reflects the costs of ensuring hazards do not result in accidents. The cost of system design to cope with the hazard is a function of the width of the triangle. The highest costs are incurred by hazards at the top of the diagram, the lowest costs by hazards at the apex of the triangle.

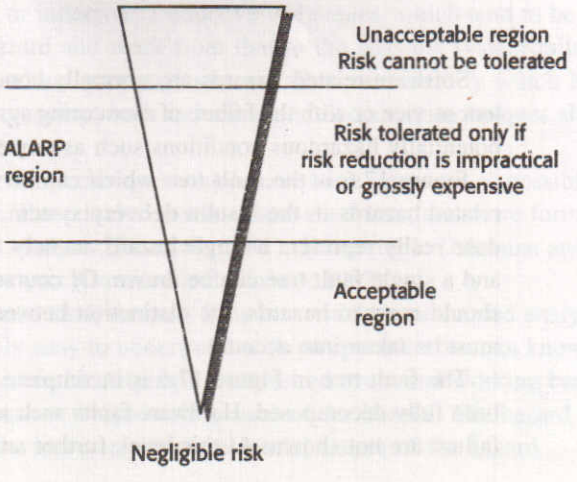

Figure 17.7
Levels of risk

Figure 17.8 Risk analysis of identified hazards

Identified hazard	Hazard probability	Hazard severity	Estimated risk	Acceptability
1. Insulin overdose	Medium	High	High	Intolerable
2. Insulin underdose	Medium	Low	Low	Acceptable
3. Power failure	High	Low	Low	Acceptable
4. Machine incorrectly fitted	High	High	High	Intolerable
5. Machine breaks in patient	Low	High	Medium	ALARP
6. Machine causes infection	Medium	Medium	Medium	ALARP
7. Electrical interference	Low	High	Medium	ALARP
8. Allergic reaction	Low	Low	Low	Acceptable

The boundaries between the regions in Figure 17.7 are not fixed for any particular class of system. Rather, they tend to be pushed downwards with time due to public expectations of safety and political considerations. Although the financial costs of accepting hazards and paying for any resulting accidents may be less than the costs of accident prevention, public opinion may mean that the additional costs must be accepted. For example, it may be cheaper for a company to clean up pollution on the rare occasion it occurs rather than install systems for pollution prevention. This may have been acceptable in the 1960s and 1970s but it is not likely to be publicly or politically acceptable now. The boundary between the intolerable region and the ALARP region has moved downwards so that hazards that may have been accepted in the past are now intolerable.

The risk assessment process involves estimating the hazard probability and the hazard severity. This is usually very difficult to do in an exact way and generally depends on making engineering judgements. Probabilities and severities are assigned using relative terms such as 'probable', 'unlikely' and 'rare', and 'high', 'medium' and 'low'. Previous system experience may allow some numeric value to be associated with these terms. However, because accidents are relatively uncommon, it is usually very difficult to validate the accuracy of this value.

Figure 17.8 shows a risk classification for the hazards identified in the previous section for the insulin delivery system. As I am not a physician, the estimates in that table are for illustration only. Notice that an insulin overdose is potentially more serious than an insulin underdose in the short term.

Hazards 3–8 are not software related so I don't discuss them further here. To counter these hazards, the machine should have built-in self-checking software which should monitor the system state and warn of some of these hazards. The warning will often allow the hazard to be detected before it causes an accident. Examples of hazards which might be detected are power failure and incorrect placement of

machine. The monitoring software is, of course, safety-related as failure to detect a hazard could result in an accident.

17.2.4 Risk reduction

Once potential hazards and their causes have been identified, the system specification should be formulated so that these hazards are unlikely to result in an accident. In Chapter 16, I identified three possible strategies that may be used:

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

Normally, the designers of safety-critical systems use a combination of these approaches. For example, intolerable hazards may be handled by reducing their probability as far as possible and adding a protection system should the hazard arise.

In an insulin delivery system, a 'safe state' is a shutdown state where no insulin is injected. Over a short period this will not pose a threat to the diabetic's health. If the potential software problems identified in Figure 17.6 are considered, the following 'solutions' might be developed:

1. *Arithmetic error* This arises when some arithmetic computation causes a representation failure. The specification must identify all possible arithmetic errors which may occur. These depend on the algorithm used. The specification might state that an exception handler must be included for each identified arithmetic error. The specification should set out the action to be taken for each of these errors if they arise. A safe action is to shut down the delivery system and activate a warning alarm.

2. *Algorithmic error* This is a more difficult situation as no definite anomalous situation can be detected. It might 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 wrongly. 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.

Fault trees are also used to identify potential hardware problems. It 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 than this. There is, therefore, available processor capacity in the system 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.

Figure 17.9
Examples of safety requirements for an insulin pump

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 for a system user.

SR3: The system shall include a hardware diagnostic facility that shall be executed at least 4 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 anomaly is discovered and a diagnostic message as defined in Table 4 should be displayed.

Some of the resulting safety requirements for the insulin pump system are shown in Figure 17.9. These are user requirements and, naturally, they would be expressed in more detail in a final system specification. In these requirements, the references to Tables 3 and 4 relate to tables that would be included in the requirements document.

17.3 Security specification

The specification of security requirements for systems has something in common with safety requirements. It is impractical to specify them quantitatively and security requirements are often 'shall not' requirements that define unacceptable system behaviour rather than required system functionality. However, there are important differences between these types of requirements:

1. The notion of a safety life cycle that covers all aspects of safety management is well developed. The area of security specification and management is immature and there is no accepted equivalent of a security life cycle.
2. The set of security threats faced by systems is fairly generic. All systems must protect themselves against intrusion, denial of service, etc. By contrast, hazards in safety-critical systems are normally domain specific.
3. Security techniques and technologies such as encryption and authentication devices are fairly mature. However, much security technology has been developed for specialised systems (such as military or financial systems) and there are problems in transferring it to more general use. Software safety techniques are still the subject of research.

The conventional (non-computerised) approach to security analysis is based around the assets to be protected and their value to an organisation. Therefore, a

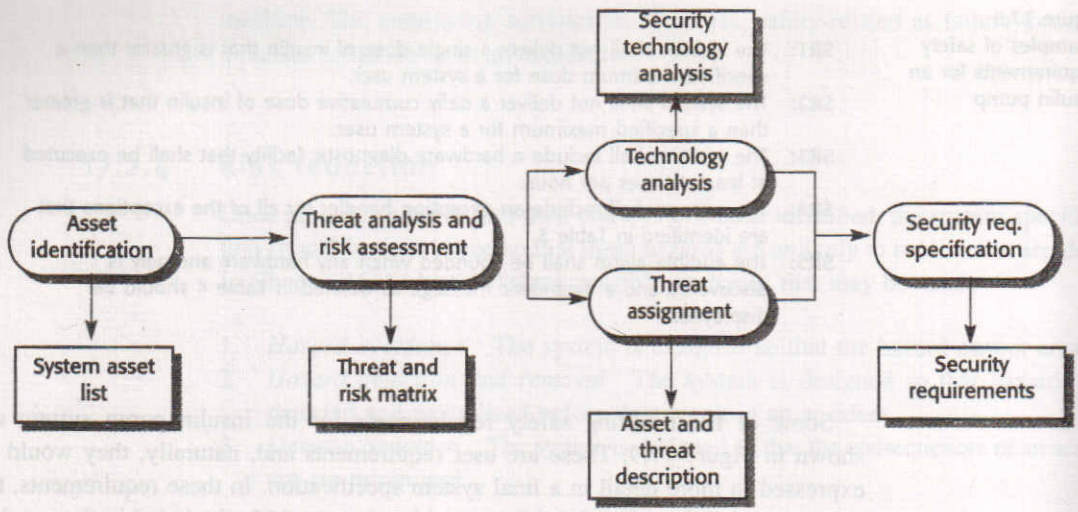

Figure 17.10
Security specification

bank will provide high security in an area where large amounts of money are stored compared to other public areas (say) where the potential losses are limited. The same approach can be used for specifying security for computer-based systems. A possible security specification process is shown in Figure 17.10.

The stages in this process are:

1. *Asset identification and evaluation* The assets (data and programs) and their required degree of protection are identified. Note that the degree of required protection depends on the asset value, so that a password file (say) is normally more valuable than a set of public web pages as a successful attack on the password file has serious system-wide consequences.
2. *Threat analysis and risk assessment* Possible security threats are identified and the risks associated with each of these threats are estimated.
3. *Threat assignment* Identified threats are related to the assets so that, for each identified asset, there is a list of associated threats.
4. *Technology analysis* Available security technologies and their applicability against the identified threats are assessed.
5. *Security requirements specification* The security requirements are specified. Where appropriate, these will explicitly identify the security technologies that may be used to protect against different threats to the system.

Security specification and security management are currently important research areas. Standards for security management are under development (ISO/IEC, 1998) and it is likely that these will be agreed in the next few years.

KEY POINTS

- Reliability requirements should be defined quantitatively in the system requirements specification.
- There are several different reliability metrics such as probability of failure on demand, rate of occurrence of failure, mean time to failure and availability. The most appropriate metric for a specific system depends on the type of system and application domain. Different metrics may be used for different sub-systems.
- Non-functional reliability specifications can lead to functional system requirements that define system features whose function is to reduce the number of system failures and hence increase reliability.
- Hazard analysis is a key activity in the safety specification process. It involves identifying hazardous conditions which can compromise system safety. System requirements are then generated to ensure that these hazards do not arise or, if they occur, they do not result in an accident.
- Risk analysis is the process of assessing the likelihood that a hazard will result in an accident. Risk analysis identifies critical hazards which must be avoided in the system and classifying risks according to their seriousness.
- To specify security requirements, you should identify the assets that are to be protected and define how security techniques and technology should be used to protect these assets.

FURTHER READING

'Requirements definition for survivable network systems'. Discusses the problems of defining requirements for survivable systems where survivability relates to both availability and security. (R. C. Linger, N. R. Mead and H. F. Lipson, *Proc. ICRE'98*, IEEE Press.)

Requirements Engineering: A Good Practice Guide. This book includes a section on the specification of critical systems and a discussion of the use of formal methods in critical systems specification. (I. Sommerville and P. Sawyer, 1997, John Wiley and Sons.)

Safeware: System Safety and Computers. This is a thorough discussion of all aspects of safety-critical systems. It is particularly strong in its description of hazard analysis and the derivation of requirements from this. (N. Leveson, 1995, Addison-Wesley.)

EXERCISES

17.1 Why is it sometimes inappropriate to use hardware reliability metrics in estimating software systems reliability? Illustrate your answer with an example.

17.2 Suggest appropriate reliability metrics for the following classes of software system. Give reasons for your choice of metric. Predict the usage of these systems and suggest appropriate values for the reliability metrics:

- a system which monitors patients in a hospital intensive care unit;
- a word processor;
- an automated vending machine control system;
- a system to control braking in a car;
- a system to control a refrigeration unit;
- a management report generator.

17.3 You are responsible for writing the specification for a software system that controls a network of EPOS (electronic point of sale) terminals in a store. The system accepts bar code information from a terminal, queries a product database and returns the item name and its price to the terminal for display. The system must be continuously available during the store's opening hours.

Giving reasons for your choice, choose appropriate reliability metrics for specifying the reliability of such a system and write a plausible reliability specification that takes into account the fact that some faults are more serious than others.

17.4 Suggest four functional requirements that might be generated for this store system to help improve system reliability.

17.5 Explain why the boundaries in the risk triangle shown in Figure 17.7 are liable to change with time and with changing social attitudes.

17.6 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.

17.7 A safety-critical software system for treating cancer patients has two principal components:

- A radiation therapy machine that delivers controlled doses of radiation to tumour sites. This machine is controlled by an embedded software system.
- A treatment database which 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 which will reduce the probability that these hazards will result in

an accident. Explain why your suggested defence is likely to reduce the risk associated with the hazard.

17.8 Suggest how fault-tree analysis could be modified for use in security specification. Threats in a security-critical system are analogous to hazards in a safety-critical system.

17.9 Should software engineers working on the specification and development of safety-related systems be professionally certified in some way?