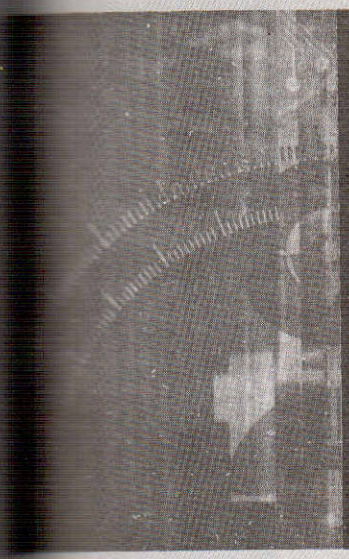
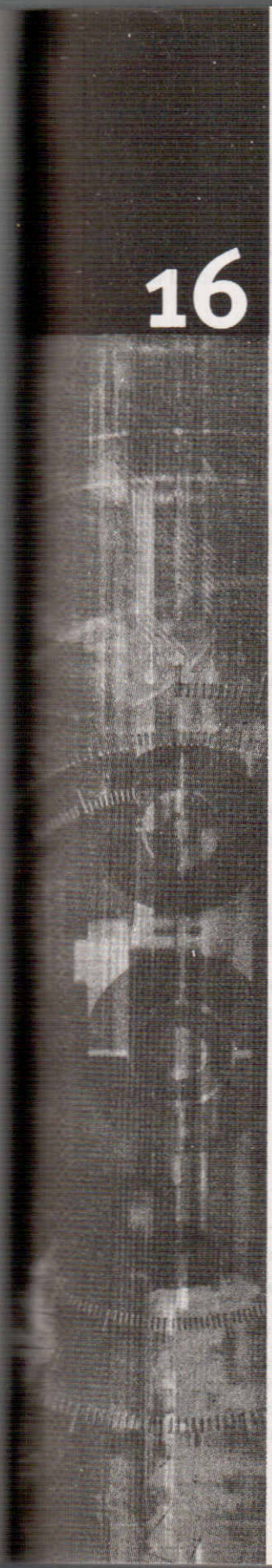


PART FOUR

Critical Systems



16 Dependability

Objectives

The objective of this chapter is to introduce the notion of dependability and its importance to critical systems. When you have read this chapter, you will:

- understand four dimensions of dependability, namely availability, reliability, safety and security;
- have been introduced to the notion of critical systems where system failure can have severe human or economic consequences;
- understand that to achieve dependability you need to avoid mistakes during the development of a system, detect and remove errors when the system is in use and limit the damage caused by operational failures.

Contents

All of us are familiar with the problems of computer system failures. For no obvious reason, computer systems sometimes crash and fail to deliver the services that have been requested. Programs running on these computers may not operate as expected and, occasionally, may corrupt the data that is managed by the system. We have learned to live with these failures and few of us completely trust the personal computers that we normally use.

The dependability of a computer system is a property of the system that equates to its trustworthiness. Trustworthiness essentially means the degree of user confidence that the system will operate as they expect and that the system will not 'fail' in normal use. This property cannot be expressed numerically but we use relative terms such as 'not dependable', 'very dependable' and 'ultra-dependable' to reflect different degrees of trust that we might have in a system.

Trustworthiness and usefulness are not, of course, the same thing. The word processor that I used to write this book is not, in my opinion, a very dependable system but it is very useful. However, to reflect my lack of trust in the system I frequently save my work and keep multiple backup copies of it. Therefore, I compensate for the lack of system dependability by actions that limit the damage that might result from system failure.

There are four principal dimensions to dependability as shown in Figure 16.1:

1. *Availability* Informally, the availability of a system is the probability that it will be up and running and able to deliver useful services at any given time.
2. *Reliability* Informally, the reliability of a system is the probability, over a given period of time, that the system will correctly deliver services as expected by the user.
3. *Safety* Informally, the safety of a system is a judgement of how likely it is that the system will cause damage to people or its environment.
4. *Security* Informally, the security of a system is a judgement of how likely it is that the system can resist accidental or deliberate intrusion.

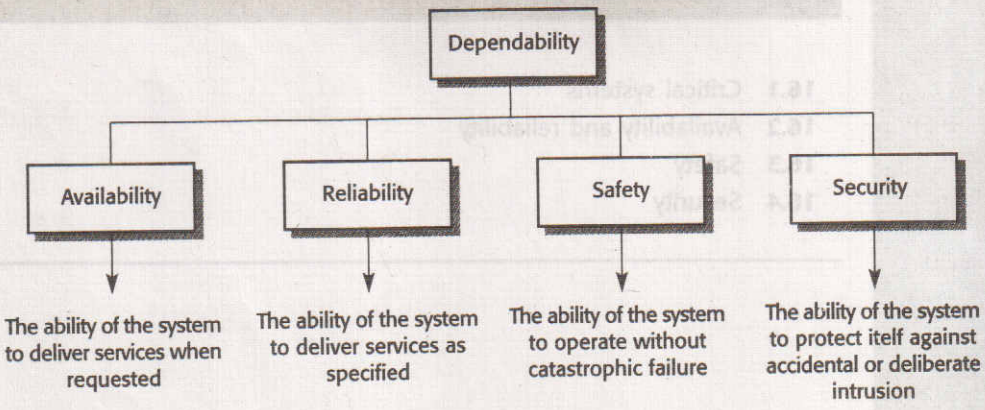


Figure 16.1 Dimensions of dependability

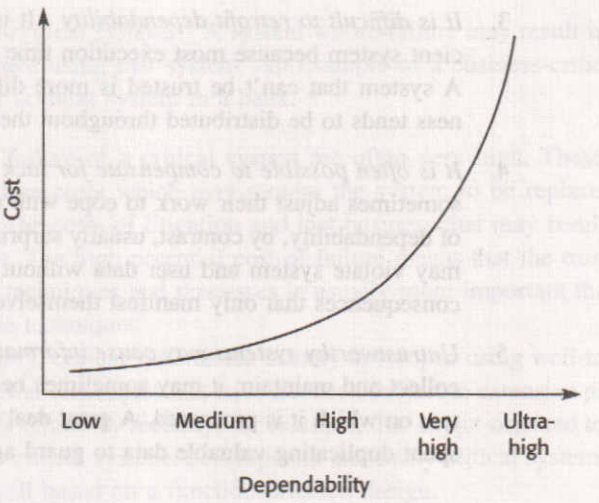


Figure 16.2 Cost/dependability curve

As I discuss in Chapter 17, availability and dependability are, essentially, probabilities and therefore can be expressed quantitatively. Safety and security are judgements that are made on the basis of evidence about the system. These are rarely expressed as numerical values but may be expressed in terms of integrity levels. Therefore, the safety of a level 1 system is less than the safety of a level 2 system that is less than the safety of a level 3 system and so on.

Because of additional design, implementation and validation overheads, increasing the dependability of a system can dramatically increase development costs. Figure 16.2 shows the relationship between costs and incremental improvements in dependability. This graph holds for all aspects of dependability – reliability, availability, safety and security. Because of the exponential nature of this cost/dependability curve, it is not possible to prove that a system is 100 per cent dependable as the costs of dependability assurance would then be infinite.

It is generally true that high levels of dependability can only be achieved at the expense of system performance. Dependable software includes extra, often redundant, code to perform the necessary checking for exceptional system states and to recover from system faults. This reduces system performance and increases the amount of store required by the software. However, there are a number of reasons why dependability is usually a more important attribute than performance:

1. *Systems that are unreliable, unsafe or insecure are often unused* If a system is not trusted by its users, they will often refuse to use it. Furthermore, they may also refuse to use products from the same company as the untrustworthy system as they believe that these may also be untrustworthy.

2. *System failure costs may be enormous* For some applications, such as a reactor control system or an aircraft navigation system, the cost of system failure is orders of magnitude greater than the cost of the control system.

3. *It is difficult to retrofit dependability* It is usually possible to tune an inefficient system because most execution time is spent in small program sections. A system that can't be trusted is more difficult to improve as untrustworthiness tends to be distributed throughout the system.
4. *It is often possible to compensate for lack of system performance* Users can sometimes adjust their work to cope with a poorly performing system. A lack of dependability, by contrast, usually surprises the user. Untrustworthy software may violate system and user data without warning and may fail with serious consequences that only manifest themselves at a later date.
5. *Untrustworthy systems may cause information loss* Data is very expensive to collect and maintain; it may sometimes be worth more than the computer system on which it is processed. A great deal of effort and money may have to be spent duplicating valuable data to guard against data corruption.

The dependability of the software product is influenced by the software process used to develop that product. A repeatable process which is oriented towards defect avoidance is likely to develop a dependable system. However, there is not a simple relationship between product and process quality. Conforming to a particular process does not allow any quantitative product quality assessment to be made. The relationships between process and product quality are discussed in Chapters 24 and 25 which cover quality management and process improvement.

16.1 Critical systems

The failure of many software-controlled systems causes inconvenience but no serious, long-term damage. However, there are some systems where failures can result in significant economic losses, physical damage or threats to human life. These systems are usually called *critical systems*. Dependability is an essential attribute of critical systems and all aspects of dependability (availability, reliability, safety and security) may be important. Achieving a high level of dependability is usually the most important requirement for critical systems.

There are three main types of critical system:

1. *Safety-critical systems* A system whose failure may result in injury, loss of life or major environmental damage. An example of a safety-critical system is a control system for a chemical manufacturing plant.
2. *Mission-critical systems* A system whose failure may result in the failure of some goal-directed activity. An example of a mission-critical system is a navigational system for a spacecraft.

3. *Business-critical systems* A system whose failure may result in the failure of the business using that system. An example of a business-critical system is a customer account system in a bank.

The costs of failure of a critical system are often very high. These costs include the direct failure costs which may require the system to be replaced and indirect costs such as the costs of litigation and lost business that may result if the system is unavailable. The high potential cost of failure means that the trustworthiness of development techniques and processes is usually more important than the costs of applying these techniques.

Consequently, critical systems are usually developed using well-tried techniques rather than newer techniques that have not been subject to extensive practical experience. It is only relatively recently, for example, that object-oriented techniques have been used for critical systems development and many critical systems development projects are still based on a function-oriented design.

However, software engineering techniques that are not normally cost-effective may be used for critical systems development; for example, the use of formal specification and formal verification of a program against its specification. One reason why this is used is that it helps reduce the amount of testing required. For critical systems, the costs of verification and validation are usually very high and may consume more than 50 per cent of the total system development costs.

Although this book focuses on software engineering and these chapters concentrate on software issues, dependability is really a systems concept. When considering dependability in critical systems, there are three types of system 'component' that are prone to failure:

1. system hardware that may fail because of mistakes in its design, because components fail as a result of manufacturing errors or because hardware components have come to an end of their natural life;
2. system software that may fail because of mistakes in its specification, design or implementation;
3. human operators of the system that may fail to operate the system correctly.

Therefore, if your goal is to improve the dependability of a system, you have to consider all of these aspects and their interactions. I illustrate this in some of the examples that are developed in the other chapters in this part of the book.

16.1.1 A simple safety-critical system

To illustrate the chapters on critical systems, I use an example of a simple safety-critical medical system. This is an insulin delivery system for the control of diabetes. I hope that most readers will have general knowledge of this condition and its treatment. I have already introduced this system in Chapter 9 and formally specified part of its functionality.

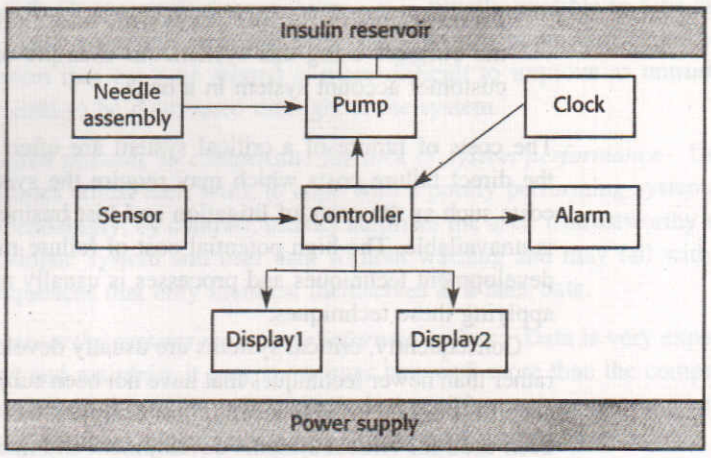


Figure 16.3 Insulin pump structure

Diabetes is a relatively common condition where the human body is unable to produce sufficient quantities of a hormone called insulin. Insulin metabolises glucose in the blood. The conventional treatment of diabetes involves regular injections of genetically engineered insulin.

The problem with this treatment is that the level of insulin in the blood does not depend on the blood glucose level but is a function of the time when the insulin injection was taken. This can lead to very low levels of blood glucose (if there is too much insulin) or very high levels of blood sugar (if there is too little insulin). Low blood sugar is, in the short term, a more serious condition as it can result in temporary brain malfunctioning and, ultimately, unconsciousness and death. In the long term, continual high levels of blood sugar can lead to eye damage, kidney damage, and heart problems.

Current advances in developing miniaturised sensors have meant that it is now possible to develop automated insulin delivery systems. These monitor blood sugar levels and deliver an appropriate dose of insulin when required. Insulin delivery systems like this already exist for the treatment of hospital patients. In future, it may be possible for many diabetics to have such systems permanently attached to their bodies.

An insulin delivery system might work by using a micro-sensor embedded in the patient to measure some blood parameter which is proportional to the sugar level. This is then sent to the pump controller. This controller computes the sugar level, judges how much insulin is required and sends signals to a miniaturised pump to deliver the insulin via a permanently attached needle. Insulin delivery systems are likely to be software controlled. Figure 16.3 shows the components and organisation of the insulin pump. Figure 16.4 is a data-flow model that illustrates how an input blood sugar level is transformed to a sequence of pump control commands.

Three dimensions of dependability apply to this insulin delivery system:

1. *Availability* It is important that the system should be available to deliver insulin when required.

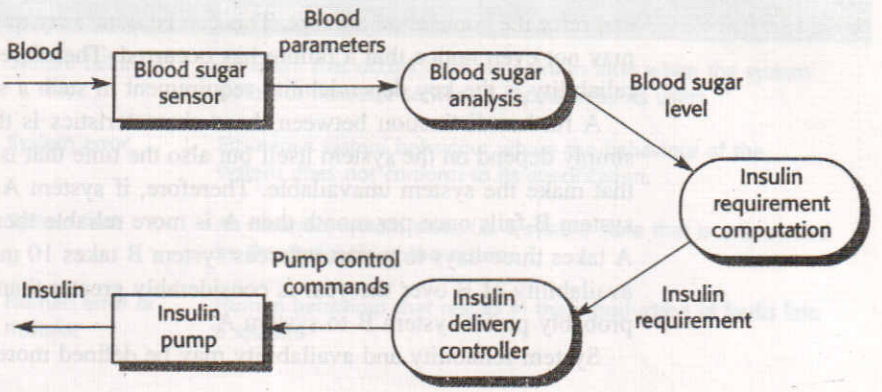


Figure 16.4 Data-flow model of the insulin pump

2. *Reliability* It is important that the system performs reliably and delivers the correct amount of insulin to counter the current level of blood sugar.

3. *Safety* Failure of the system could, in principle, cause excessive doses of insulin to be delivered and this would threaten the life of the user. It is important that this class of system failures should not occur.

I explain in later chapters how these dependability dimensions can be specified and validated.

16.2 Availability and reliability

In this section I discuss two closely related dimensions of dependability, namely availability and reliability. The availability of a system is the probability that it will be able to deliver services to its users when requested to do so. The reliability is the probability that system services will be delivered as specified. Obviously, reliability subsumes availability because if a specified service is not delivered then the system is obviously not behaving according to its specification.

However, it is useful to distinguish between these characteristics because the requirements for availability and reliability may be different. For example, some systems can tolerate relatively frequent failures so long as they can recover quickly from these failures. They therefore have relatively low reliability requirements. The same systems, however, may have high availability requirements because of users' expectations of continuous service.

A good example of such a system is a telephone exchange switch. Users expect a dial tone when they pick up a phone so the system has high availability requirements. However, if a system fault causes a connection to fail, this is often recoverable. Exchange switches usually include repair facilities that can reset the system

and retry the connection attempt. This can be done very quickly and the phone user may not even notice that a failure has occurred. Therefore, availability rather than reliability is the key dependability requirement in such a situation.

A further distinction between these characteristics is that availability does not simply depend on the system itself but also the time that is required to repair faults that make the system unavailable. Therefore, if system A fails once per year and system B fails once per month then A is more reliable then B. However, if system A takes three days to restart whereas system B takes 10 minutes to restart then the availability of B over the year is considerably greater than that of A. Users would probably prefer system B to system A.

System reliability and availability may be defined more precisely as follows:

1. *Reliability* The probability of failure-free operation over a specified time in a given environment for a specific purpose.
2. *Availability* The probability that a system, at a point in time, will be operational and able to deliver the requested services.

These are careful definitions of these terms and one of the practical problems in developing reliable systems is that our intuitive notions of reliability and availability are often sometimes broader than these limited definitions. The definition of reliability states that the environment in which the system is used and the purpose for which it is used must be taken into account when considering its reliability. Therefore, measurements of reliability in one environment are not necessarily transferable to another environment where the system is used in a different way.

For example, the reliability of a software system may be measured in an office environment where most users are uninterested in the operation of the software. They follow the instructions for its use and do not try to experiment with the system. In a university environment, the reliability may be quite different. Here, students explore the boundaries of the system and may use the system in unexpected ways. These may result in system failures that did not occur in the more constrained office environment.

Human perceptions and patterns of use are also significant. For example, consider a situation where a car has a fault in its windscreen wiper system that results in the intermittent failure of the wipers to operate correctly in heavy rain. The reliability of that system as perceived by a driver depends on where the driver lives and uses the car. A driver in Seattle (wet climate) will probably be more affected by this failure than a driver in Las Vegas (dry climate). The Seattle driver's perception will be that the system is unreliable whereas the driver in Las Vegas may never notice the problem.

A further difficulty with these definitions is that they do not take into account the severity of failure or the consequences of unavailability. People, naturally, are more concerned about system failures that have serious consequences and their perception of system reliability is influenced by these consequences. For example, a car engine that cuts out immediately after starting but inevitably operates correctly after restarting is irritating. However, it does not affect the normal operation of the

Figure 16.5 Reliability terminology

Term	Description
System failure	An event that occurs at some point in time when the system does not deliver a service as expected by its users.
System error	Erroneous system behaviour where the behaviour of the system does not conform to its specification.
System fault	An incorrect system state, i.e. a system state that is unexpected by the designers of the system.
Human error or mistake	Human behaviour that results in the introduction of faults into a system.

car and many drivers would not think that it was unreliable. By contrast, most drivers will think that an engine which cuts out while driving at high speed once per month (say) is unreliable.

A strict definition of reliability relates the system implementation to its specification. That is, the system is behaving reliably if its behaviour is consistent with that defined in the specification. However, a common cause of perceived unreliability is that the system specification does not match the expectations of the system users. Unfortunately, many specifications are incomplete or incorrect and it is left to software engineers to interpret how the system should behave. As they are not domain experts, they may not, therefore, implement the behaviour that users expect.

Reliability and availability are usually considered to be the most important dimensions of dependability. If a system is unreliable, it is difficult to ensure system safety or security as they may be compromised by system failures. If a system is unavailable, the consequent economic losses can be very high. Unreliable software results in high costs for end-users. Developers of unreliable systems may acquire a bad reputation for quality and lose future business opportunities.

Reliability is compromised by system failures. These may be a failure to provide a service, a failure to deliver a service as specified or the delivery of a service in such a way that it is unsafe or insecure. Some of these failures are a consequence of specification errors or failures in associated systems such as a telecommunications system. However, many failures are a consequence of erroneous system behaviour that derives from faults in the system. When discussing reliability, it is helpful to distinguish between the terms fault, error and failure. I have defined these terms in Figure 16.5.

System faults do not necessarily result in system errors as the faulty state may be transient and it may be corrected before erroneous behaviour occurs. System errors do not necessarily result in system failures as the behaviour may also be transient and have no observable effect or the system may include protection that ensures that the erroneous behaviour is discovered and corrected before the system services are affected.

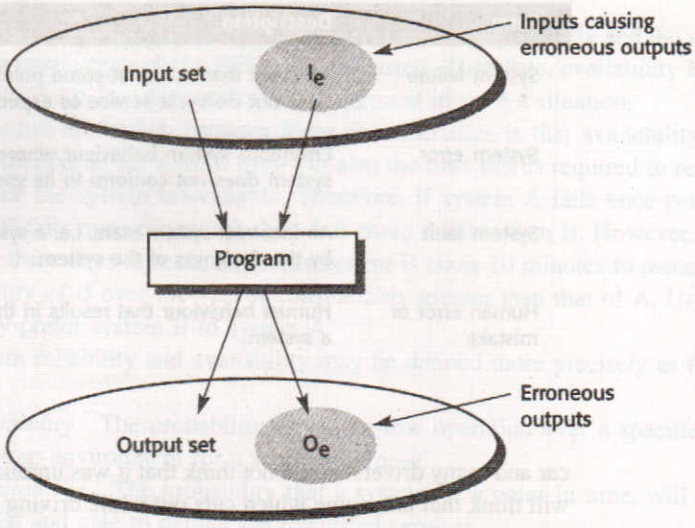


Figure 16.6 A system as an input/output mapping

This distinction between the different terms shown in Figure 16.5 is helpful because it helps us identify three complementary approaches that may be used to improve the reliability of a system:

1. *Fault avoidance* Development techniques are used that either minimise the possibility of mistakes and/or trap mistakes before these result in the introduction of system faults. Examples of such techniques include avoiding error-prone programming language constructs such as pointers and the use of static analysis to detect program anomalies as discussed in Chapter 19.
2. *Fault detection and removal* The use of verification and validation techniques that increase the chances that faults will be detected and removed before the system is used. Systematic system testing and debugging is an example of a fault detection technique.
3. *Fault tolerance* The use of techniques that ensure that faults in a system do not result in system errors or that ensure that system errors do not result in system failures. The incorporation of self-checking facilities in a system and the use of redundant system modules are examples of fault tolerance techniques.

The development of fault-tolerant systems is covered in Chapter 18 where I also briefly discuss some techniques for fault avoidance. Process-based approaches to fault avoidance are covered in Chapter 24. Fault detection is discussed in Chapters 19 and 20.

Software faults cause software failures when the faulty code is executed with a set of inputs which expose the software fault. The code works properly for most inputs. Figure 16.6, derived from Littlewood (1990), shows a software system as a mapping of an input to an output set. A program has many possible inputs (for

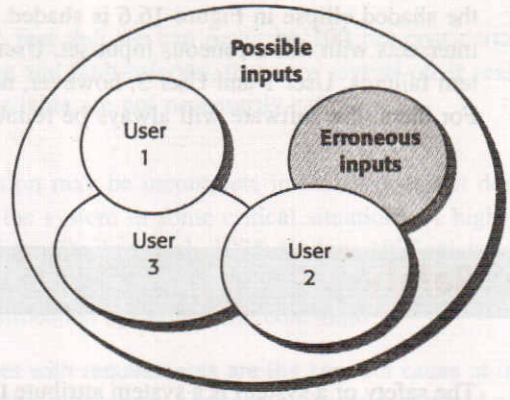


Figure 16.7 Software usage patterns

simplicity, combinations and sequences of inputs are considered as a single input). The program responds to these inputs by producing an output or a set of outputs.

Some of these inputs (shown in the shaded ellipse in Figure 16.6) cause system failures where erroneous outputs are generated by the program. The software reliability is related to the probability that, in a particular execution of the program, the system input will be a member of the set of inputs which cause an erroneous output.

There is a complex relationship between observed system reliability and the number of latent software faults. Mills *et al.* (1987) point out that not all software faults are equally likely to cause software failure. Usually, there are a number of members of I_e which are more likely to be selected than others. If these inputs do not cause the faulty parts of the software to be executed, there will be no failures. The reliability of the program, therefore, mostly depends on the number of inputs causing erroneous outputs during normal use of the system. Faults that only occur in exceptional situations have little effect on the system's reliability.

Reliability is related to the probability of an error occurring in operational use. Removing software faults from parts of the system which are rarely used makes little real difference to the perceived reliability. Mills *et al.* found that, in their software, removing 60 per cent of product defects led to only a 3 per cent reliability improvement. This was confirmed in a separate study of errors in IBM software products. Adams (1984) noted that many defects in the products were only likely to cause failures after hundreds or thousands of months of product usage.

Therefore, a program may contain known faults but may still be seen as reliable by its users. They may never select an erroneous input so program failures never arise. Furthermore, experienced users often 'work around' software faults which are known to cause failures. They deliberately avoid using system features which they know can cause problems for them. Repairing the faults in these features may make no practical difference to the reliability as seen by these users.

Each user of a system uses it in different ways. Faults which affect the reliability of the system for one user may never be revealed under a different mode of working (Figure 16.7). In Figure 16.7, the set of erroneous inputs corresponding to

the shaded ellipse in Figure 16.6 is shaded. The set of inputs produced by User 2 intersects with this erroneous input set. User 2 will therefore experience some system failures. User 1 and User 3, however, never use inputs from the erroneous set. For them, the software will always be reliable.

16.3 Safety

The safety of a system is a system attribute that reflects the system's ability to operate, normally or abnormally, without threatening people or the environment. Where safety is an essential attribute of a critical system, the system is a safety-critical system. Examples of safety-critical systems are control and monitoring systems in aircraft, process control systems in chemical and pharmaceutical plants and automobile control systems.

Hardware control of safety-critical systems is simpler to implement and analyse than software control. However, we are now building systems of such complexity that they cannot be controlled by hardware alone. Some software control is essential because of the need to manage large numbers of sensors and actuators with complex control laws. An example of such complexity is found in advanced military aircraft which are aerodynamically unstable. They require continual software-controlled adjustment of their flight surfaces to ensure that they do not crash.

Safety-critical software falls into two classes:

1. *Primary, safety-critical software* This is software which is embedded as a controller in a system. Malfunctioning of such software can cause a hardware malfunction which results in human injury or environmental damage. I focus on this type of software.
2. *Secondary safety-critical software* This is software which can indirectly result in injury. Examples of such systems are computer-aided engineering design systems whose malfunctioning might result in a design fault in the object being designed. This fault may pose a threat to humans if the designed system malfunctions. Another example of a secondary safety-critical system is a medical database holding details of drugs administered to patients. Errors in this system might result in an incorrect drug dosage being administered.

System reliability and system safety are related but distinct dependability attributes. Of course, a safety-critical system should be reliable in that it should conform to its specification and operate without failures. It may incorporate fault-tolerant features so that it can provide continuous service even if faults occur. However, fault-tolerant systems are not necessarily safe. The software may still malfunction and cause system behaviour which results in an accident.

Apart from the fact that we can never be 100 per cent certain that a software system is fault-free and fault-tolerant, there are several other reasons why software systems that are reliable are not necessarily safe:

1. The specification may be incomplete in that it does not describe the required behaviour of the system in some critical situations. A high percentage of system malfunctions (Boehm *et al.*, 1975; Endres, 1975; Nakajo and Kume, 1991; Lutz, 1993) are the result of specification rather than design errors. In a study of errors in embedded systems, Lutz concludes:

   > ... difficulties with requirements are the key root cause of the safety-related software errors which have persisted until integration and system testing

2. Hardware malfunctions may cause the system to behave in an unpredictable way and may present the software with an unanticipated environment. When components are close to failure they may behave erratically and generate signals that are outside the ranges that can be handled by the software.

3. The operator of the system may generate inputs that are not individually incorrect but which, in particular situations, can lead to a system malfunction. An anecdotal example of this is where a mechanic instructed the utility management software on an aircraft to raise the undercarriage. The software carried out the mechanic's instruction in spite of the fact that the plane was on the ground!

A specialised vocabulary has evolved to discuss safety-critical systems and it is important to understand the specific terms used. In Figure 16.8 I show some definitions that I have adapted from terms initially defined by Leveson (1985).

The key to assuring safety is to ensure either that accidents do not occur or that the consequences of an accident are minimal. This can be achieved in three complementary ways:

1. *Hazard avoidance* The system is designed so that hazards are avoided. For example, a cutting system that requires the operator to press two separate buttons at the same time to operate the machine avoids the hazard of the operator's hands being in the blade pathway.

2. *Hazard detection and removal* The system is designed so that hazards are detected and removed before they result in an accident. For example, a chemical plant system may detect excessive pressure and open a relief valve to reduce these pressures before an explosion occurs.

3. *Damage limitation* The system may include protection features that minimise the damage that may result from an accident. For example, an aircraft engine normally includes automatic fire extinguishers. If a fire occurs, it can often be controlled before it poses a threat to the passengers or crew.

Figure 16.8 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. A computer-controlled machine injuring its operator 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 detects an obstacle in front of a machine is an example of a hazard.
Damage	A measure of the loss resulting from a mishap. Damage can range from many people killed as a result of an accident to minor injury or property damage.
Hazard severity	An assessment of the worst possible damage which could result from a particular hazard. Hazard severity can range from catastrophic where many people are killed to minor where only minor damage results.
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 where the hazard could occur).
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 a hazard will result in an accident.

Accidents generally occur when several things go wrong at the same time. An analysis of serious accidents (Perrow, 1984) suggested that they were almost all due to a combination of malfunctions rather than single failures. The unanticipated combination led to interactions which resulted in system failure. Perrow also suggests that it is impossible to anticipate all possible combinations of system malfunction and that accidents are an inevitable part of using complex systems. Software tends to increase system complexity so using software control *may* increase the probability of system accidents.

This does not mean that software control invariably increases the risk associated with a system. Software control and monitoring can improve the safety of systems. Software-controlled systems can monitor a wider range of conditions than electromechanical systems. They can be adapted relatively easily. They involve the use of computer hardware which has very high inherent reliability and which is physically small and lightweight. Software-controlled systems can provide sophisticated safety interlocks. They can support control strategies which reduce the amount of time people need to spend in hazardous environments.

16.4 Security

The security of a system is an assessment of the extent that the system protects itself from external attacks that may be accidental or deliberate. Examples of attacks might be viruses, unauthorised use of system services, unauthorised modification of the system or its data, etc. Security is important for all critical systems. Without a reasonable level of security, the availability, reliability and safety of the system may be compromised if external attacks cause some damage to the system.

The reason for this is that all methods for assuring availability, reliability and security rely on the fact that the operational system is the same as the system that was originally installed. If this installed system has been compromised in some way (for example, if the software has been modified to include a virus) then the arguments for reliability and safety that were originally made can no longer hold. The system software may be corrupted and may behave in an unpredictable way.

Conversely, errors in the development of a system can lead to security loopholes. If a system does not respond to unexpected inputs or if array bounds are not checked then attackers can exploit these weaknesses to gain access to the system. One major security incident (the Internet worm) (Spafford, 1989) took advantage of the fact that programs in C do not include array bound checking to overwrite part of memory with code that allowed unauthorised access to the system.

Of course, there are some types of critical system where security is the most important dimension of system dependability. Military systems, systems for electronic commerce and systems that involve the processing and interchange of confidential information must be designed so that they achieve a high level of security. If an airline reservation system (say) is unavailable, this causes inconvenience and some delays in issuing tickets. However, if the system is insecure and can accept fake bookings then the airline that owns the system can lose a great deal of money as a result of this problem.

There are three types of damage that may be caused through external attack:

1. *Denial of service* The system may be forced into a state where its normal services become unavailable. This, obviously, then affects the availability of the system.
2. *Corruption of programs or data* The software components of the system may be altered in an unauthorised way. This may affect the system's behaviour and hence its reliability and safety. If damage is severe, the availability of the system may be affected.
3. *Disclosure of confidential information* The information managed by the system may be confidential and the external attack may expose this to unauthorised people. Depending on the type of data, this could affect the safety of the system and may allow later attacks that affect the system availability or reliability.

Figure 16.9 Security terminology

Term	Definition
Exposure	Possible loss or harm in a computing system.
Vulnerability	A weakness in a computer-based system that may be exploited to cause loss or harm.
Attack	An exploitation of a system vulnerability.
Threats	Circumstances that have potential to cause loss or harm.
Control	A protective measure that reduces a system vulnerability.

As with other aspects of dependability, there is a specialised terminology associated with security. Some important terms, as discussed by Pfleeger (1997), are defined in Figure 16.9.

There is a clear analogy here with some of the terminology of safety so that an exposure is analogous to an accident and a vulnerability is analogous to a hazard. Therefore there are comparable approaches that may be used to assure the security of a system:

1. *Vulnerability avoidance* The system is designed so that vulnerabilities do not occur. For example, if a system is not connected to an external public network then there is no possibility of an attack from members of the public.
2. *Attack detection and neutralisation* The system is designed to detect vulnerabilities and remove them before they result in an exposure. An example of vulnerability detection and removal is the use of a virus checker that analyses incoming files for viruses and modifies these files to remove the virus.
3. *Exposure limitation* The consequences of a successful attack are minimised. Examples of exposure limitation are regular system backups and a configuration management policy that allows damaged software to be re-created.

Security has become increasingly important as more and more systems are connected to the Internet. Internet connections provide additional system functionality (e.g. a customer may be able to access his or her bank account directly) but also means that the system can be attacked by people with hostile intentions. The Internet also means that details of specific system vulnerabilities may be easily disseminated so that more people may be able to attack the system.

A related, very important attribute for Internet-based systems is survivability (Ellison *et al.*, 1999a). Survivability is the ability of a system to continue to deliver service while it is under attack and, potentially, while part of the system is disabled. Survivability is clearly related to both security and availability. Work on survivability focuses on identifying key system components and ensuring that they can deliver a minimal service (Ellison *et al.*, 1999b). Three strategies are used to enhance survivability, namely resistance to attack, attack recognition and recovery

from the damage caused by an attack. I do not have space to cover this topic here but have included some links from the book's web pages to information on survivability research.

KEY POINTS

- The dependability of a computer system is a property of the system that reflects the user's degree of trust in the system. The most important dimensions of dependability are availability, reliability, safety and security.
- A critical system is a system where failures can result in significant economic losses, physical damage or threats to human life. Three important classes of critical system are safety-critical systems, mission-critical systems and business-critical systems.
- The availability of a system is the probability that it will be able to deliver services to its users when requested to do so and the reliability is the probability that system services will be delivered as specified.
- Reliability and availability are usually considered to be the most important dimensions of dependability. If a system is unreliable, it is difficult to ensure system safety or security as they may be compromised by system failures.
- Reliability is related to the probability of an error occurring in operational use. A program may contain known faults but may still be seen as reliable by its users. They may never use features of the system that are affected by these faults.
- The safety of a system is a system attribute that reflects the system's ability to operate, normally or abnormally, without threatening people or the environment. Where safety is an essential attribute of a critical system, the system is a safety-critical system.
- Security is important for all critical systems. Without a reasonable level of security, the availability, reliability and safety of the system may be compromised if external attacks cause some damage to the system.

FURTHER READING

'Survivability: Protecting your critical systems'. An accessible introduction to the topic of survivability and why it is important. (R. Ellison *et al.*, *IEEE Internet Computing*, Nov./Dec. 1999.)

Computer Security. This book is an excellent introduction to the area of computer security. It covers basic principles, practice, distributed systems security and security theory. (D. Gollmann, 1999, John Wiley and Sons.)

Handbook of Software Reliability Engineering. This book is a detailed discussion of software reliability from both a practical and a theoretical stand-point. (M. R. Lyu, 1996, McGraw-Hill.)

Computer-related Risks. This is a collection drawn from the Internet Risks Forum of incidents that have occurred in automated systems. It shows how much can actually go wrong in safety-related systems. (P. G. Neumann, 1995, Addison-Wesley.)

EXERCISES

16.1 What are the most important dimensions of system dependability? Why is it the case that the cost of assuring dependability is exponential?

16.2 Suggest six reasons why dependability is important in critical systems.

16.3 Using an example, explain the difficulties of describing what software reliability means.

16.4 Assess the reliability of some software system which you use regularly by keeping a log of system failures and observed faults. Write a user's handbook which describes how to make effective use of the system in the presence of these faults.

16.5 Identify six consumer products which may contain, or which may contain in future, safety-critical software systems.

16.6 Explain why ensuring system reliability is not a guarantee of system safety.

16.7 In a medical system that is designed to deliver radiation to treat tumours, suggest one hazard that may arise and propose one software feature that may be used to ensure that the identified hazard does not result in an accident.

16.8 Explain why there is a close relationship between system availability and system security.

16.9 In computer security terms, explain the differences between an attack and a threat.

16.10 Is it ethical for an engineer to agree to deliver a software system with known faults to a customer? Does it make any different if the customer is told of the existence of these faults in advance? Would it be reasonable to make claims about the reliability of the software in such circumstances?

16.11 Assume you were part of a team which developed software for a chemical plant which went wrong and caused a serious pollution incident. Your boss is interviewed on television and states that there are no faults in the software and that the problems must be due to poor operational procedures. You are approached by a newspaper for your opinion. Discuss how you should handle such an interview.