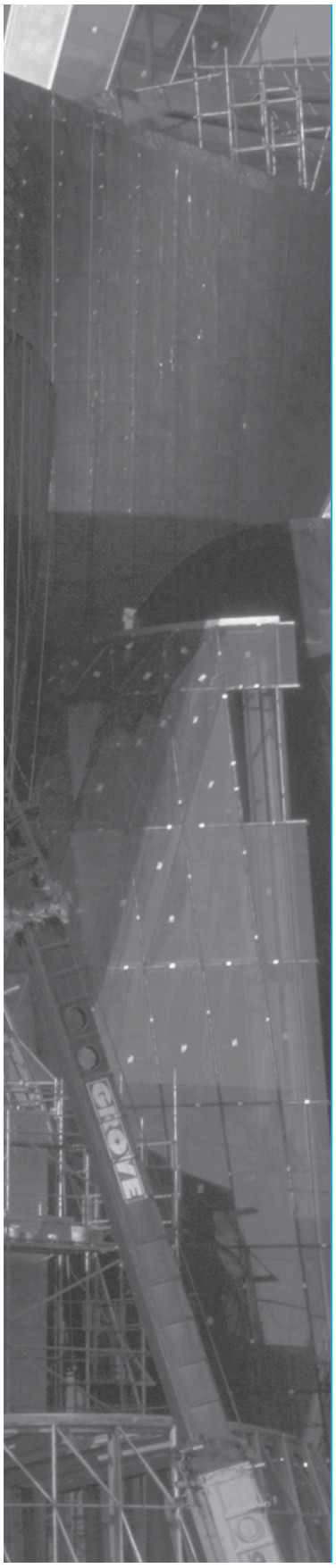

11

Dependability and security

Objectives

The objective of this chapter is to introduce software dependability and security. When you have read this chapter, you will:

- understand why dependability and security are usually more important than the functional characteristics of a software system;
- understand the four principal dimensions of dependability, namely availability, reliability, safety, and security;
- be aware of the specialized terminology that is used when discussing security and dependability;
- understand that to achieve secure, dependable software, you need to avoid mistakes during the development of a system, to detect and remove errors when the system is in use, and to limit the damage caused by operational failures.

Contents

As computer systems have become deeply embedded in our business and personal lives, the problems that result from system and software failure are increasing. A failure of server software in an e-commerce company could lead to a major loss of revenue, and possibly also customers for that company. A software error in an embedded control system in a car could lead to expensive recalls of that model for repair and, in the worst case, could be a contributory factor in accidents. The infection of company PCs with malware requires expensive cleanup operations to sort out the problem and could result in the loss or damage to sensitive information.

Because software-intensive systems are so important to governments, companies, and individuals, it is essential that widely used software is trustworthy. The software should be available when required and should operate correctly and without undesirable side effects, such as unauthorized information disclosure. The term 'dependability' was proposed by Laprie (1995) to cover the related systems attributes of availability, reliability, safety, and security. As I discuss in Section 11.1, these properties are inextricably linked, so having a single term to cover them all makes sense.

The dependability of systems is now usually more important than their detailed functionality for the following reasons:

1. *System failures affect a large number of people.* Many systems include functionality that is rarely used. If this functionality were left out of the system, only a small number of users would be affected. System failures, which affect the availability of a system, potentially affect all users of the system. Failure may mean that normal business is impossible.
2. *Users often reject systems that are unreliable, unsafe, or insecure.* If users find that a system is unreliable or insecure, they will refuse to use it. Furthermore, they may also refuse to buy or use other products from the same company that produced the unreliable system, because they believe that these products are also likely to be unreliable or insecure.
3. *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.
4. *Undependable systems may cause information loss.* Data is very expensive to collect and maintain; it is usually worth much more than the computer system on which it is processed. The cost of recovering lost or corrupt data is usually very high.

As I discussed in Chapter 10, software is always part of a broader system. It executes in an operational environment that includes the hardware on which the software executes, the human users of that software, and the organizational or business processes where the software is used. When designing a dependable system, you therefore have to consider:

1. *Hardware failure* System hardware may fail because of mistakes in its design, because components fail as a result of manufacturing errors, or because the components have reached the end of their natural life.

Critical systems

Some classes of system are 'critical systems' where system failure may result in injury to people, damage to the environment, or extensive economic losses. Examples of critical systems include embedded systems in medical devices, such as an insulin pump (safety-critical), spacecraft navigation systems (mission-critical), and online money transfer systems (business critical).

Critical systems are very expensive to develop. Not only must they be developed so that failures are very rare but they must also include recovery mechanisms that are used if and when failures occur.

http://www.SoftwareEngineering-9.com/Web/Dependability/CritSys.html

2. *Software failure* System software may fail because of mistakes in its specification, design, or implementation.
3. *Operational failure* Human users may fail to use or operate the system correctly. As hardware and software have become more reliable, failures in operation are now, perhaps, the largest single cause of system failures.

These failures are often interrelated. A failed hardware component may mean system operators have to cope with an unexpected situation and additional workload. This puts them under stress and people under stress often make mistakes. This can cause the software to fail, which means more work for the operators, even more stress, and so on.

As a result, it is particularly important that designers of dependable, software-intensive systems take a holistic systems perspective, rather than focus on a single aspect of the system such as its software or hardware. If hardware, software, and operational processes are designed separately, without taking into account the potential weaknesses of other parts of the system, then it is more likely that errors will occur at the interfaces between the different parts of the system.

11.1 Dependability properties

All of us are familiar with the problem of computer system failure. For no obvious reason, our computers sometimes crash or go wrong in some way. 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 but 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 reflects its trustworthiness. Trustworthiness here essentially means the degree of confidence a user has that the system will operate as they expect, and that the system will not 'fail' in normal use. It is not meaningful to express dependability numerically.

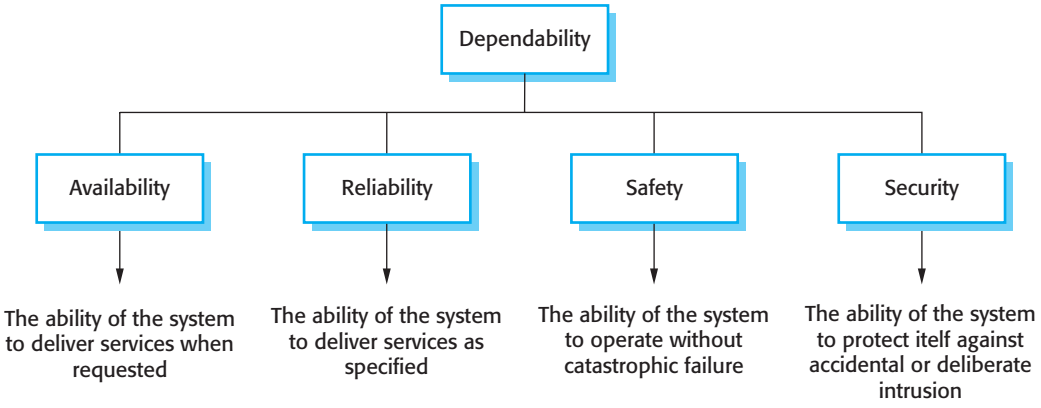

Figure 11.1 Principal dependability properties

Rather, we use relative terms such as 'not dependable,' 'very dependable,' and 'ultra-dependable' to reflect the degrees of trust that we might have in a system.

Trustworthiness and usefulness are not, of course, the same thing. I don't think that the word processor that I used to write this book is a very dependable system. It sometimes freezes and has to be restarted. Nevertheless, because it is very useful, I am prepared to tolerate occasional failure. However, to reflect my lack of trust in the system I save my work frequently and keep multiple backup copies of it. I compensate for the lack of system dependability by actions that limit the damage that could result from system failure.

There are four principal dimensions to dependability, as shown in Figure 11.1.

1. *Availability* Informally, the availability of a system is the probability that it will be up and running and able to deliver useful services to users 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 judgment 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 judgment of how likely it is that the system can resist accidental or deliberate intrusions.

The dependability properties shown in Figure 11.1 are complex properties that can be broken down into a number of other, simpler properties. For example, security includes 'integrity' (ensuring that the systems program and data are not damaged) and 'confidentiality' (ensuring that information can only be accessed by people who are authorized). Reliability includes 'correctness' (ensuring the system services are as specified), 'precision' (ensuring information is delivered at an appropriate level of detail), and 'timeliness' (ensuring that information is delivered when it is required).

Of course, these dependability properties are not all applicable to all systems. For the insulin pump system, introduced in Chapter 1, the most important properties are availability (it must work when required), reliability (it must deliver the correct dose of insulin), and safety (it must never deliver a dangerous dose of insulin). Security is not an issue as the pump will not maintain confidential information. It is not networked and so cannot be maliciously attacked. For the wilderness weather system, availability and reliability are the most important properties because the costs of repair may be very high. For the patient information system, security is particularly important because of the sensitive private data that is maintained.

As well as these four main dependability properties, you may also think of other system properties as dependability properties:

1. *Repairability* System failures are inevitable, but the disruption caused by failure can be minimized if the system can be repaired quickly. For that to happen, it must be possible to diagnose the problem, access the component that has failed, and make changes to fix that component. Repairability in software is enhanced when the organization using the system has access to the source code and has the skills to make changes to it. Open source software makes this easier but the reuse of components can make it more difficult.

2. *Maintainability* As systems are used, new requirements emerge and it is important to maintain the usefulness of a system by changing it to accommodate these new requirements. Maintainable software is software that can be adapted economically to cope with new requirements, and where there is a low probability that making changes will introduce new errors into the system.

3. *Survivability* A very important attribute for Internet-based systems is survivability (Ellison et al., 1999b). Survivability is the ability of a system to continue to deliver service whilst under attack and, potentially, whilst part of the system is disabled. Work on survivability focuses on identifying key system components and ensuring that they can deliver a minimal service. Three strategies are used to enhance survivability—resistance to attack, attack recognition, and recovery from the damage caused by an attack (Ellison et al., 1999a; Ellison et al., 2002). I discuss this in more detail in Chapter 14.

4. *Error tolerance* This property can be considered as part of usability and reflects the extent to which the system has been designed so that user input errors are avoided and tolerated. When user errors occur, the system should, as far as possible, detect these errors and either fix them automatically or request the user to reinput their data.

The notion of system dependability as an encompassing property was introduced because the dependability properties of availability, security, reliability, and safety are closely related. Safe system operation usually depends on the system being available and operating reliably. A system may become unreliable because an intruder has corrupted its data. Denial of service attacks on a system are intended to compromise the

system's availability. If a system is infected with a virus, you cannot then be confident in its reliability or safety because the virus may change its behavior.

To develop dependable software, you therefore need to ensure that:

1. You avoid the introduction of accidental errors into the system during software specification and development.
2. You design verification and validation processes that are effective in discovering residual errors that affect the dependability of the system.
3. You design protection mechanisms that guard against external attacks that can compromise the availability or security of the system.
4. You configure the deployed system and its supporting software correctly for its operating environment.

In addition, you should usually assume that your software is not perfect and that software failures may occur. Your system should therefore include recovery mechanisms that make it possible to restore normal system service as quickly as possible.

The need for fault tolerance means that dependable systems have to include redundant code to help them monitor themselves, detect erroneous states, and recover from faults before failures occur. This affects the performance of systems, as additional checking is required each time the system executes. Therefore, designers usually have to trade off performance and dependability. You may need to leave checks out of the system because these slow the system down. However, the consequential risk here is that some failures occur because a fault has not been detected.

Because of extra design, implementation, and validation costs, increasing the dependability of a system significantly increases development costs. In particular, validation costs are high for systems that must be ultra-dependable such as safety-critical control systems. As well as validating that the system meets its requirements, the validation process may have to prove to an external regulator that the system is safe. For example, aircraft systems have to demonstrate to regulators, such as the Federal Aviation Authority, that the probability of a catastrophic system failure that affects aircraft safety is extremely low.

Figure 11.2 shows that the relationship between costs and incremental improvements in dependability. If your software is not very dependable, you can get significant improvements at relatively low costs by using better software engineering. However, if you are already using good practice, the costs of improvement are much greater and the benefits from that improvement are less. There is also the problem of testing your software to demonstrate that it is dependable. This relies on running many tests and looking at the number of failures that occur. As your software becomes more dependable, you see fewer and fewer failures. Consequently, more and more tests are needed to try and assess how many problems remain in the software. As testing is very expensive, this dramatically increases the cost of high-dependability systems.

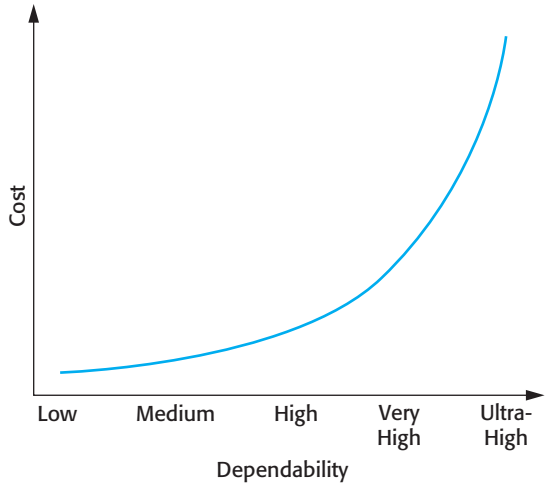

Figure 11.2 Cost/dependability curve

11.2 Availability and reliability

System availability and reliability are closely related properties that can both be expressed as numerical probabilities. The availability of a system is the probability that the system will be up and running to deliver these services to users on request. The reliability of a system is the probability that the system's services will be delivered as defined in the system specification. If, on average, 2 inputs in every 1,000 cause failures, then the reliability, expressed as a rate of occurrence of failure, is 0.002. If the availability is 0.999, this means that, over some time period, the system is available for 99.9% of that time.

Reliability and availability are closely related but sometimes one is more important than the other. If users expect continuous service from a system then the system has a high availability requirement. It must be available whenever a demand is made. However, if the losses that result from a system failure are low and the system can recover quickly then failures don't seriously affect system users. In such systems, the reliability requirements may be relatively low.

A telephone exchange switch that routes phone calls is an example of a system where availability is more important than reliability. Users expect a dial tone when they pick up a phone, so the system has high availability requirements. If a system fault occurs while a connection is being set up, this is often quickly recoverable. Exchange switches can usually reset the system and retry the connection attempt. This can be done very quickly and phone users may not even notice that a failure has occurred. Furthermore, even if a call is interrupted, the consequences are usually not serious. Therefore, availability rather than reliability is the key dependability requirement for this type of system.

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.

One of the practical problems in developing reliable systems is that our intuitive notions of reliability and availability are sometimes broader than these limited definitions. The definition of reliability states that the environment in which the system is used and the purpose that it is used for must be taken into account. If you measure system reliability in one environment, you can't assume that the reliability will be the same if the system is used in a different way.

For example, let's say that you measure the reliability of a word processor 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. If you then measure the reliability of the same system in a university environment, then the reliability may be quite different. Here, students may explore the boundaries of the system and use the system in unexpected ways. This may result in system failures that did not occur in the more constrained office environment.

These standard definitions of availability and reliability do not take into account the severity of failure or the consequences of unavailability. People often accept minor system failures but are very concerned about serious failures that have high consequential costs. For example, computer failures that corrupt stored data are less acceptable than failures that freeze the machine and that can be resolved by restarting the computer.

A strict definition of reliability relates the system implementation to its specification. That is, the system is behaving reliably if its behavior 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 behavior that users expect. It is also true, of course, that users don't read system specifications. They may therefore have unrealistic expectations of the system.

Availability and reliability are obviously linked as system failures may crash the system. However, availability does not just depend on the number of system crashes, but also on the time needed to repair the faults that have caused the failure. Therefore, if system A fails once a year and system B fails once a month then A is clearly more reliable then B. However, assume that system A takes three days to restart after a failure, whereas system B takes 10 minutes to restart. The availability of system B over the year (120 minutes of down time) is much better than that of system A (4,320 minutes of down time).

The disruption caused by unavailable systems is not reflected in the simple availability metric that specifies the percentage of time that the system is available. The time when the system fails is also significant. If a system is unavailable for an hour each day between 3 am and 4 am, this may not affect many users. However, if the same system is unavailable for 10 minutes during the working day, system unavailability will probably have a much greater effect.

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

Figure 11.3
Reliability terminology

System reliability and availability problems are mostly caused by system failures. Some of these failures are a consequence of specification errors or failures in other related systems such as a communications system. However, many failures are a consequence of erroneous system behavior that derives from faults in the system. When discussing reliability, it is helpful to use precise terminology and distinguish between the terms 'fault,' 'error,' and 'failure.' I have defined these terms in Figure 11.3 and have illustrated each definition with an example from the wilderness weather system.

When an input or a sequence of inputs causes faulty code in a system to be executed, an erroneous state is created that may lead to a software failure. Figure 11.4, derived from Littlewood (1990), shows a software system as a mapping of a set of inputs to a set of outputs. Given an input or input sequence, the program responds by producing a corresponding output. For example, given an input of a URL, a web browser produces an output that is the display of the requested web page.

Most inputs do not lead to system failure. However, some inputs or input combinations, shown in the shaded ellipse I_e in Figure 11.4, cause system failures or erroneous outputs to be generated. The program's reliability depends on the number of system inputs that are members of the set of inputs that lead to an erroneous output. If inputs in the set I_e are executed by frequently used parts of the system, then failures will be frequent. However, if the inputs in I_e are executed by code that is rarely used, then users will hardly ever see failures.

Because each user of a system uses it in different ways, they have different perceptions of its reliability. Faults that affect the reliability of the system for one user may never be revealed under someone else's mode of working (Figure 11.5). In Figure 11.5, the set of erroneous inputs correspond to the ellipse labeled I_e in Figure 11.4. The set of inputs produced by User 2 intersects with this erroneous input set. User 2 will

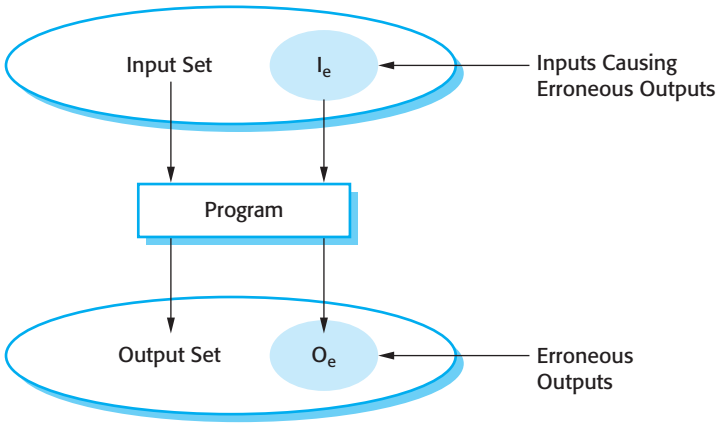

Figure 11.4 A system as an input/output mapping

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.

The practical reliability of a program depends on the number of inputs causing erroneous outputs (failures) during normal use of the system by most users. Software faults that only occur in exceptional situations have little practical effect on the system's reliability. Consequently, removing software faults may not significantly improve the overall reliability of the system. Mills et al. (1987) found that removing 60% of known errors in their software led to a 3% reliability improvement. Adams (1984), in a study of IBM software products, noted that many defects in the products were only likely to cause failures after hundreds or thousands of months of product usage.

System faults do not always result in system errors and system errors do not necessarily result in system failures. The reasons for this are as follows:

1. Not all code in a program is executed. The code that includes a fault (e.g., the failure to initialize a variable) may never be executed because of the way that the software is used.

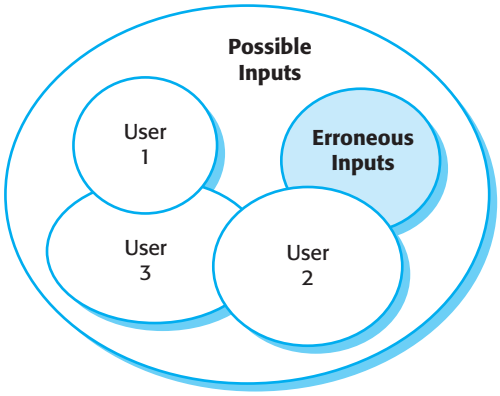

Figure 11.5 Software usage patterns

2. Errors are transient. A state variable may have an incorrect value caused by the execution of faulty code. However, before this is accessed and causes a system failure, some other system input may be processed that resets the state to a valid value.
3. The system may include fault detection and protection mechanisms. These ensure that the erroneous behavior is discovered and corrected before the system services are affected.

Another reason why the faults in a system may not lead to system failures is that, in practice, users adapt their behavior to avoid using inputs that they know cause program failures. Experienced users 'work around' software features that they have found to be unreliable. For example, I avoid certain features, such as automatic numbering in the word processing system that I used to write this book. When I used auto-numbering, it often went wrong. Repairing the faults in unused features makes no practical difference to the system reliability. As users share information on problems and work-arounds, the effects of software problems are reduced.

The distinction between faults, errors, and failures, explained in Figure 11.3, helps identify three complementary approaches that are used to improve the reliability of a system:

1. *Fault avoidance* Development techniques are used that either minimize the possibility of human errors and/or that trap mistakes before they 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.
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 testing and debugging is an example of a fault-detection technique.
3. *Fault tolerance* These are techniques that ensure that faults in a system do not result in system errors or 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 practical application of these techniques is discussed in Chapter 13, which covers techniques for dependable software engineering.

11.3 Safety

Safety-critical systems are systems where it is essential that system operation is always safe; that is, the system should never damage people or the system's environment even if the system fails. Examples of safety-critical systems include 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 analyze than software control. However, we now build systems of such complexity that they cannot be controlled by hardware alone. Software control is essential because of the need to manage large numbers of sensors and actuators with complex control laws. For example, advanced, aerodynamically unstable, military aircraft 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 that 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. The insulin pump software, introduced in Chapter 1, is an example of a primary safety-critical system. System failure may lead to user injury.
2. *Secondary safety-critical software* This is software that can indirectly result in an injury. An example of such software is a computer-aided engineering design system whose malfunctioning might result in a design fault in the object being designed. This fault may cause injury to people if the designed system malfunctions. Another example of a secondary safety-critical system is the mental health care management system, MHC-PMS. Failure of this system, whereby an unstable patient may not be treated properly, could lead to that patient injuring themselves or others.

System reliability and system safety are related but a reliable system can be unsafe and vice versa. The software may still behave in such a way that the resultant system behavior leads to an accident. There are four reasons why software systems that are reliable are not necessarily safe:

1. We can never be 100% certain that a software system is fault-free and fault-tolerant. Undetected faults can be dormant for a long time and software failures can occur after many years of reliable operation.
2. The specification may be incomplete in that it does not describe the required behavior of the system in some critical situations. A high percentage of system malfunctions (Boehm et al., 1975; Endres, 1975; Lutz, 1993; Nakajo and Kume, 1991) 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.

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

Term	Definition
Accident (or mishap)	An unplanned event or sequence of events which results in human death or injury, damage to property, or to the environment. An overdose of insulin is an example of an accident.
Hazard	A condition with the potential for causing or contributing to an accident. A failure of the sensor that measures blood glucose is an example of a hazard.
Damage	A measure of the loss resulting from a mishap. Damage can range from many people being killed as a result of an accident to minor injury or property damage. Damage resulting from an overdose of insulin could be serious injury or the death of the user of the insulin pump.
Hazard severity	An assessment of the worst possible damage that could result from a particular hazard. Hazard severity can range from catastrophic, where many people are killed, to minor, where only minor damage results. When an individual death is a possibility, a reasonable assessment of hazard severity is 'very high.'
Hazard probability	The probability of the events occurring which create a hazard. Probability values tend to be arbitrary but range from 'probable' (say 1/100 chance of a hazard occurring) to 'implausible' (no conceivable situations are likely in which the hazard could occur). The probability of a sensor failure in the insulin pump that results in an overdose is probably low.
Risk	This is a measure of the probability that the system will cause an accident. The risk is assessed by considering the hazard probability, the hazard severity, and the probability that the hazard will lead to an accident. The risk of an insulin overdose is probably medium to low.

Figure 11.6
Safety terminology

4. The system operators may generate inputs that are not individually incorrect but which, in some situations, can lead to a system malfunction. An anecdotal example of this occurred when an aircraft undercarriage collapsed whilst the aircraft was on the ground. Apparently, a technician pressed a button that instructed the utility management software to raise the undercarriage. The software carried out the mechanic's instruction perfectly. However, the system should have disallowed the command unless the plane was in the air.

A specialized vocabulary has evolved to discuss safety-critical systems and it is important to understand the specific terms used. Figure 11.6 summarizes some definitions of important terms, with examples taken from the insulin pump system.

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 an operator to use two hands to press

separate buttons simultaneously 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 minimize 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 aircraft.

Accidents most often occur when several things go wrong at the same time. An analysis of serious accidents (Perrow, 1984) suggests that they were almost all due to a combination of failures in different parts of a system. Unanticipated combinations of subsystem failures led to interactions that resulted in overall system failure. For example, failure of an air-conditioning system could lead to overheating, which then causes the system hardware to generate incorrect signals. Perrow also suggests that it is impossible to anticipate all possible combinations of failures. Accidents are therefore an inevitable part of using complex systems.

Some people have used this as an argument against software control. Because of the complexity of software, there are more interactions between the different parts of a system. This means that there will probably be more combinations of faults that could lead to system failure.

However, software-controlled systems can monitor a wider range of conditions than electro-mechanical systems. They can be adapted relatively easily. They use 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 that reduce the amount of time people need to spend in hazardous environments. Although software control may introduce more ways in which a system can go wrong, it also allows better monitoring and protection and hence can contribute to improvements in system safety.

In all cases, it is important to maintain a sense of proportion about system safety. It is impossible to make a system 100% safe and society has to decide whether or not the consequences of an occasional accident are worth the benefits that come from the use of advanced technologies. It is also a social and political decision about how to deploy limited national resources to reduce risk to the population as a whole.

11.4 Security

Security is a system attribute that reflects the ability of the system to protect itself from external attacks, which may be accidental or deliberate. These external attacks are possible because most general-purpose computers are now networked and are

Term	Definition
Asset	Something of value which has to be protected. The asset may be the software system itself or data used by that system.
Exposure	Possible loss or harm to a computing system. This can be loss or damage to data, or can be a loss of time and effort if recovery is necessary after a security breach.
Vulnerability	A weakness in a computer-based system that may be exploited to cause loss or harm.
Attack	An exploitation of a system's vulnerability. Generally, this is from outside the system and is a deliberate attempt to cause some damage.
Threats	Circumstances that have potential to cause loss or harm. You can think of these as a system vulnerability that is subjected to an attack.
Control	A protective measure that reduces a system's vulnerability. Encryption is an example of a control that reduces a vulnerability of a weak access control system.

Figure 11.7
Security terminology

therefore accessible by outsiders. Examples of attacks might be the installation of viruses and Trojan horses, unauthorized use of system services or unauthorized modification of a system or its data. If you really want a secure system, it is best not to connect it to the Internet. Then, your security problems are limited to ensuring that authorized users do not abuse the system. In practice, however, there are huge benefits from networked access for most large systems so disconnecting from the Internet is not cost effective.

For some systems, 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 is unavailable, for example, this causes inconvenience and some delays in issuing tickets. However, if the system is insecure then an attacker could delete all bookings and it would be practically impossible for normal airline operations to continue.

As with other aspects of dependability, there is a specialized terminology associated with security. Some important terms, as discussed by Pfleeger (Pfleeger and Pfleeger, 2007), are defined in Figure 11.7. Figure 11.8 takes the security concepts described in Figure 11.7 and shows how they relate to the following scenario taken from the MHC-PMS:

> *Clinic staff log on to the MHC-PMS with a username and password. The system requires passwords to be at least eight letters long but allows any password to be set without further checking. A criminal finds out that a well-paid sports star is receiving treatment for mental health problems. He would like to gain illegal access to information in this system so that he can blackmail the star.*

Term	Example
Asset	The records of each patient that is receiving or has received treatment.
Exposure	Potential financial loss from future patients who do not seek treatment because they do not trust the clinic to maintain their data. Financial loss from legal action by the sports star. Loss of reputation.
Vulnerability	A weak password system which makes it easy for users to set guessable passwords. User ids that are the same as names.
Attack	An impersonation of an authorized user.
Threat	An unauthorized user will gain access to the system by guessing the credentials (login name and password) of an authorized user.
Control	A password checking system that disallows user passwords that are proper names or words that are normally included in a dictionary.

Figure 11.8 Examples of security terminology

By posing as a concerned relative and talking with the nurses in the mental health clinic, he discovers how to access the system and personal information about the nurses. By checking name badges, he discovers the names of some of the people allowed access. He then attempts to log on to the system by using these names and systematically guessing possible passwords (such as children's names).

In any networked system, there are three main types of security threats:

1. *Threats to the confidentiality of the system and its data* These can disclose information to people or programs that are not authorized to have access to that information.
2. *Threats to the integrity of the system and its data* These threats can damage or corrupt the software or its data.
3. *Threats to the availability of the system and its data* These threats can restrict access to the software or its data for authorized users.

These threats are, of course, interdependent. If an attack makes the system unavailable, then you will not be able to update information that changes with time. This means that the integrity of the system may be compromised. If an attack succeeds and the integrity of the system is compromised, then it may have to be taken down to repair the problem. Therefore, the availability of the system is reduced.

In practice, most vulnerabilities in sociotechnical systems result from human failings rather than technical problems. People choose easy-to-guess passwords or write

down their passwords in places where they can be found. System administrators make errors in setting up access control or configuration files and users don't install or use protection software. However, as I discussed in Section 10.5, we have to be very careful when classifying a problem as a user error. Human problems often reflect poor systems design decisions that require, for example, frequent password changes (so that users write down their passwords) or complex configuration mechanisms.

The controls that you might put in place to enhance system security are comparable to those for reliability and safety:

1. *Vulnerability avoidance* Controls that are intended to ensure that attacks are unsuccessful. The strategy here is to design the system so that security problems are avoided. For example, sensitive military systems are not connected to public networks so that external access is impossible. You should also think of encryption as a control based on avoidance. Any unauthorized access to encrypted data means that it cannot be read by the attacker. In practice, it is very expensive and time consuming to crack strong encryption.
2. *Attack detection and neutralization* Controls that are intended to detect and repel attacks. These controls involve including functionality in a system that monitors its operation and checks for unusual patterns of activity. If these are detected, then action may be taken, such as shutting down parts of the system, restricting access to certain users, etc.
3. *Exposure limitation and recovery* Controls that support recovery from problems. These can range from automated backup strategies and information 'mirroring' to insurance policies that cover the costs associated with a successful attack on the system.

Without a reasonable level of security, we cannot be confident in a system's availability, reliability, and safety. Methods for certifying availability, reliability, and security assume that the operational software is the same as the software that was originally installed. If the system has been attacked and the software has been compromised in some way (for example, if the software has been modified to include a worm), then the reliability and safety arguments no longer hold.

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. Major security incidents such as the original Internet worm (Spafford, 1989) and the Code Red worm more than 10 years later (Berghel, 2001) took advantage of the same vulnerability. Programs in C# do not include array bound checking, so it is possible to overwrite part of memory with code that allows unauthorized access to the system.

KEY POINTS

- Failure of critical computer systems can lead to large economic losses, serious information loss, physical damage, or threats to human life.
- The dependability of a computer system is a system property that reflects the user's degree of trust in the system. The most important dimensions of dependability are availability, reliability, safety, and security.
- The availability of a system is the probability that the system will be able to deliver services to its users when requested to do so. Reliability is the probability that system services will be delivered as specified.
- Perceived reliability is related to the probability of an error occurring in operational use. A program may contain known faults but may still be experienced as reliable by its users. They may never use features of the system that are affected by the faults.
- The safety of a system is a system attribute that reflects the system's ability to operate, normally or abnormally, without injury to people or damage to the environment.
- Security reflects the ability of a system to protect itself against external attacks. Security failures may lead to loss of availability, damage to the system or its data, or the leakage of information to unauthorized people.
- Without a reasonable level of security, the availability, reliability, and safety of the system may be compromised if external attacks damage the system. If a system is unreliable, it is difficult to ensure system safety or security, as they may be compromised by system failures.

FURTHER READING

'The evolution of information assurance'. An excellent article discussing the need to protect critical information in an organization from accidents and attacks. (R. Cummings, *IEEE Computer*, **35** (12), December 2002.) http://dx.doi.org/10.1109/MC.2002.1106181.

'Designing Safety Critical Computer Systems'. This is a good introduction to the field of safety-critical systems, which discusses the fundamental concepts of hazards and risks. More accessible than Dunn's book on safety-critical systems. (W. R. Dunn, *IEEE Computer*, **36** (11), November 2003.) http://dx.doi.org/10.1109/MC.2003.1244533.

Secrets and Lies: Digital Security in a Networked World. An excellent, very readable book on computer security which approaches the subject from a sociotechnical perspective. Schneier's columns on security issues in general (URL below) are also very good. (B. Schneier, John Wiley & Sons, 2004.) http://www.schneier.com/essays.html.

EXERCISES

11.1. Suggest six reasons why software dependability is important in most sociotechnical systems.

11.2. What are the most important dimensions of system dependability?

11.3. Why do the costs of assuring dependability increase exponentially as the reliability requirement increases?

11.4. Giving reasons for your answer, suggest which dependability attributes are likely to be most critical for the following systems:

An Internet server provided by an ISP with thousands of customers

A computer-controlled scalpel used in keyhole surgery

A directional control system used in a satellite launch vehicle

An Internet-based personal finance management system

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

11.6. Reliability and safety are related but distinct dependability attributes. Describe the most important distinction between these attributes and explain why it is possible for a reliable system to be unsafe and vice versa.

11.7. In a medical system that is designed to deliver radiation to treat tumors, 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.

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

11.9. Using the MHC-PMS as an example, identify three threats to this system (in addition to the threat shown in Figure 11.8). Suggest controls that might be put in place to reduce the chances of a successful attack based on these threats.

11.10. As an expert in computer security, you have been approached by an organization that campaigns for the rights of torture victims and have been asked to help the organization gain unauthorized access to the computer systems of an American company. This will help them confirm or deny that this company is selling equipment that is used directly in the torture of political prisoners. Discuss the ethical dilemmas that this request raises and how you would react to this request.