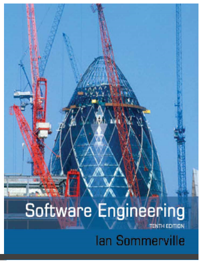
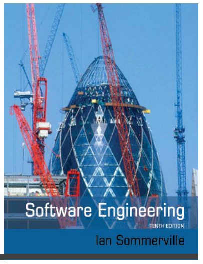


Chapter 12 – Safety Engineering

Topics covered



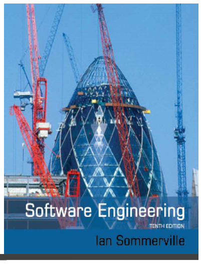
✧ Safety-critical systems

✧ Safety requirements

✧ Safety engineering processes

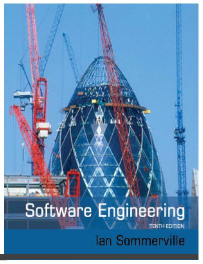
✧ Safety cases

Safety



- ✧ Safety is a property of a system that reflects the system's ability to operate, normally or abnormally, without danger of causing human injury or death and without damage to the system's environment.
- ✧ It is important to consider software safety as most devices whose failure is critical now incorporate software-based control systems.

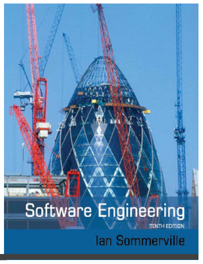
Software in safety-critical systems



✧ The system may be software-controlled so that the decisions made by the software and subsequent actions are safety-critical. Therefore, the software behaviour is directly related to the overall safety of the system.

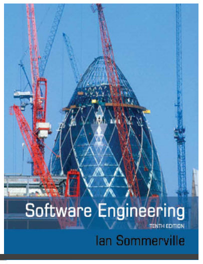
✧ Software is extensively used for checking and monitoring other safety-critical components in a system. For example, all aircraft engine components are monitored by software looking for early indications of component failure. This software is safety-critical because, if it fails, other components may fail and cause an accident.

Safety and reliability

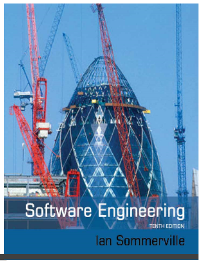


- ✧ Safety and reliability are related but distinct
  - In general, reliability and availability are necessary but not sufficient conditions for system safety
- ✧ Reliability is concerned with conformance to a given specification and delivery of service
- ✧ Safety is concerned with ensuring system cannot cause damage irrespective of whether or not it conforms to its specification.
  - System reliability is essential for safety but is not enough
  - Reliable systems can be unsafe

Unsafe reliable systems

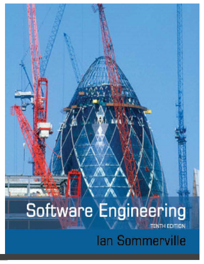


- There may be dormant faults in a system that are undetected for many years and only rarely arise.
- Specification errors
  - If the system specification is incorrect then the system can behave as specified but still cause an accident.
- Hardware failures generating spurious inputs
  - Hard to anticipate in the specification.
- Context-sensitive commands i.e. issuing the right command at the wrong time
  - Often the result of operator error.



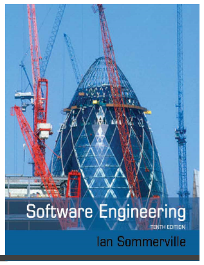
Safety-critical systems

Safety critical systems



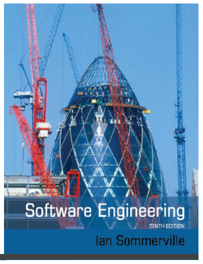
- ✧ Systems where it is essential that system operation is always safe i.e. the system should never cause damage to people or the system's environment
- ✧ Examples
  - Control and monitoring systems in aircraft
  - Process control systems in chemical manufacture
  - Automobile control systems such as braking and engine management systems

Safety criticality



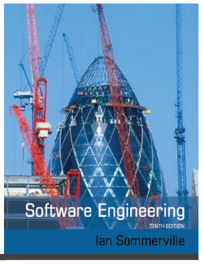
- ✧ Primary safety-critical systems
  - Embedded software systems whose failure can cause the associated hardware to fail and directly threaten people. Example is the insulin pump control system.
- ✧ Secondary safety-critical systems
  - Systems whose failure results in faults in other (socio-technical) systems, which can then have safety consequences.
    - For example, the Mentcare system is safety-critical as failure may lead to inappropriate treatment being prescribed.
    - Infrastructure control systems are also secondary safety-critical systems.

Hazards



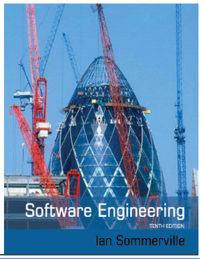
- ✧ Situations or events that can lead to an accident
  - Stuck valve in reactor control system
  - Incorrect computation by software in navigation system
  - Failure to detect possible allergy in medication prescribing system
- ✧ Hazards do not inevitably result in accidents – accident prevention actions can be taken.

Safety achievement



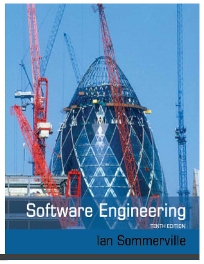
- ✧ Hazard avoidance
  - The system is designed so that some classes of hazard simply cannot arise.
- ✧ Hazard detection and removal
  - The system is designed so that hazards are detected and removed before they result in an accident.
- ✧ Damage limitation
  - The system includes protection features that minimise the damage that may result from an accident.

Safety terminology



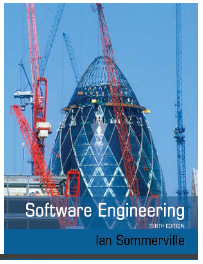
Term	Definition
Accident (or mishap)	An unplanned event or sequence of events which results in human death or injury, damage to property, or to the environment. An overdose of insulin is an example of an accident.
Hazard	A condition with the potential for causing or contributing to an accident. A failure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

Normal accidents

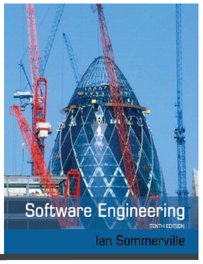


- ✧ Accidents in complex systems rarely have a single cause as these systems are designed to be resilient to a single point of failure
  - ▪ Designing systems so that a single point of failure does not cause an accident is a fundamental principle of safe systems design.
- ✧ Almost all accidents are a result of combinations of malfunctions rather than single failures.
- ✧ It is probably the case that anticipating all problem combinations, especially, in software controlled systems is impossible so achieving complete safety is impossible. Accidents are inevitable.

Software safety benefits

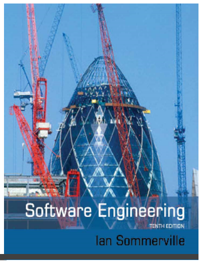


✧ Although software failures can be safety-critical, the use of software control systems contributes to increased system safety
  - Software monitoring and control allows a wider range of conditions to be monitored and controlled than is possible using electro-mechanical safety systems.
  - Software control allows safety strategies to be adopted that reduce the amount of time people spend in hazardous environments.
  - Software can detect and correct safety-critical operator errors.



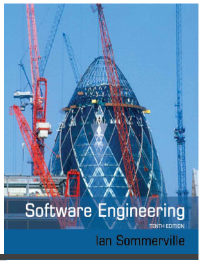
Safety requirements

Safety specification



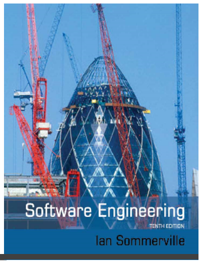
- The goal of safety requirements engineering is to identify protection requirements that ensure that system failures do not cause injury or death or environmental damage.
- Safety requirements may be ‘shall not’ requirements i.e. they define situations and events that should never occur.
- Functional safety requirements define:
  - Checking and recovery features that should be included in a system
  - Features that provide protection against system failures and external attacks

Hazard-driven analysis



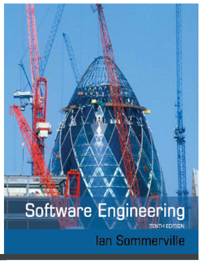
- ✧ Hazard identification
- ✧ Hazard assessment
- ✧ Hazard analysis
- ✧ Safety requirements specification

Hazard identification



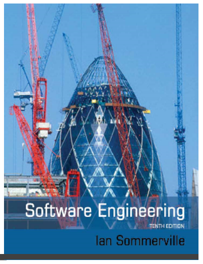
- Identify the hazards that may threaten the system.
- Hazard identification may be based on different types of hazard:
  - Physical hazards
  - Electrical hazards
  - Biological hazards
  - Service failure hazards
  - Etc.

Insulin pump risks



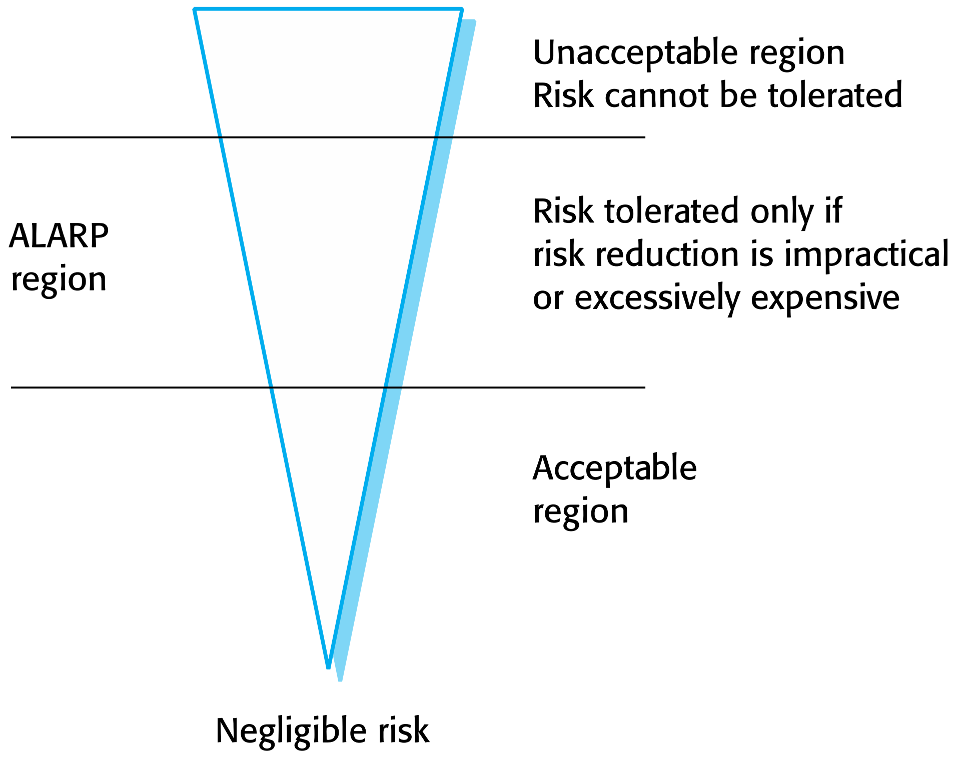
- ✧ Insulin overdose (service failure).
- ✧ Insulin underdose (service failure).
- ✧ Power failure due to exhausted battery (electrical).
- ✧ Electrical interference with other medical equipment (electrical).
- ✧ Poor sensor and actuator contact (physical).
- ✧ Parts of machine break off in body (physical).
- ✧ Infection caused by introduction of machine (biological).
- ✧ Allergic reaction to materials or insulin (biological).

Hazard assessment

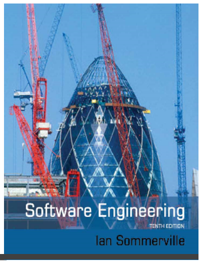


- ✧ The process is concerned with understanding the likelihood that a risk will arise and the potential consequences if an accident or incident should occur.
- ✧ Risks may be categorised as:
  - Intolerable. Must never arise or result in an accident
  - As low as reasonably practical(ALARP). Must minimise the possibility of risk given cost and schedule constraints
  - Acceptable. The consequences of the risk are acceptable and no extra costs should be incurred to reduce hazard probability

The risk triangle

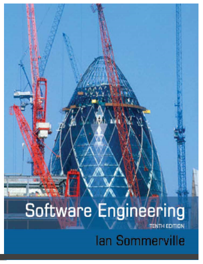


Social acceptability of risk



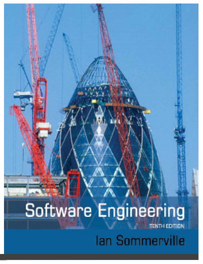
- ✧ The acceptability of a risk is determined by human, social and political considerations.
- ✧ In most societies, the boundaries between the regions are pushed upwards with time i.e. society is less willing to accept risk
  - For example, the costs of cleaning up pollution may be less than the costs of preventing it but this may not be socially acceptable.
- ✧ Risk assessment is subjective
  - Risks are identified as probable, unlikely, etc. This depends on who is making the assessment.

Hazard assessment



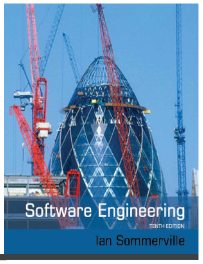
- ✧ Estimate the risk probability and the risk severity.
- ✧ It is not normally possible to do this precisely so relative values are used such as ‘unlikely’, ‘rare’, ‘very high’, etc.
- ✧ The aim must be to exclude risks that are likely to arise or that have high severity.

Risk classification for the insulin pump



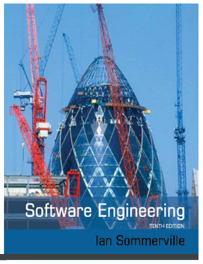
Identified hazard	Hazard probability	Accident severity	Estimated risk	Acceptability
1.Insulin overdose computation	Medium	High	High	Intolerable
2. Insulin underdose computation	Medium	Low	Low	Acceptable
3. Failure of hardware monitoring system	Medium	Medium	Low	ALARP
4. Power failure	High	Low	Low	Acceptable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

Hazard analysis



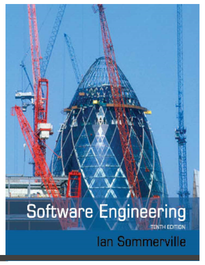
- ✧ Concerned with discovering the root causes of risks in a particular system.
- ✧ Techniques have been mostly derived from safety-critical systems and can be
  - Inductive, bottom-up techniques. Start with a proposed system failure and assess the hazards that could arise from that failure;
  - Deductive,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

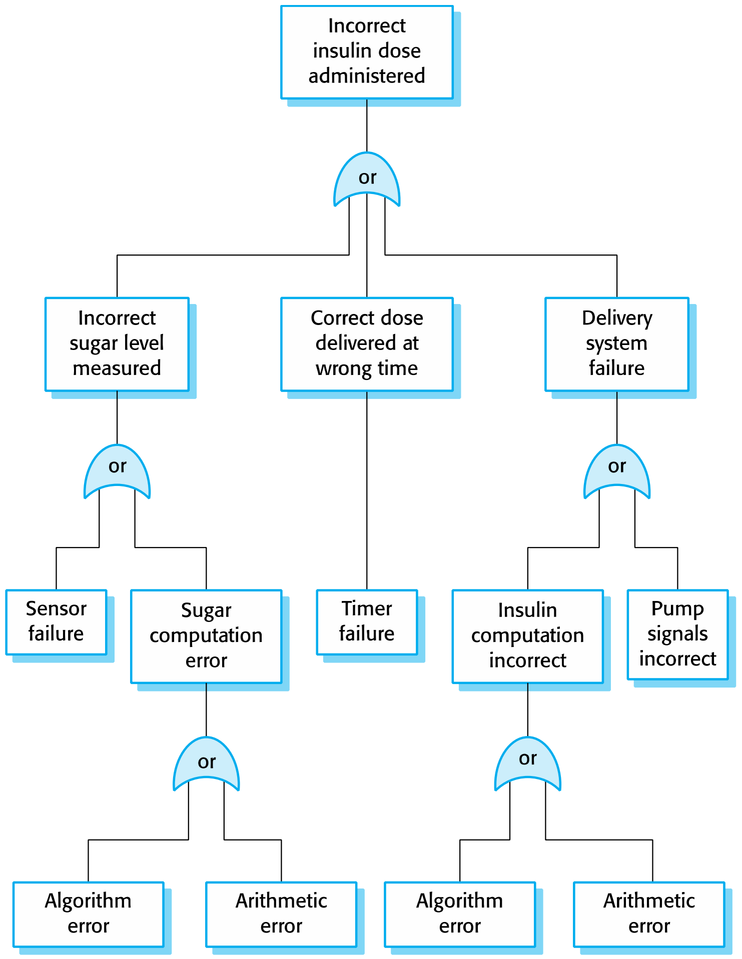
Fault-tree analysis



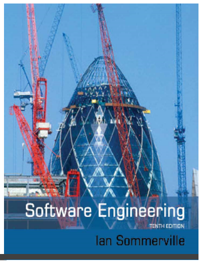
- ✧ A deductive top-down technique.
- ✧ Put the risk or hazard at the root of the tree and identify the system states that could lead to that hazard.
- ✧ Where appropriate, link these with ‘and’ or ‘or’ conditions.
- ✧ A goal should be to minimise the number of single causes of system failure.

An example of a software fault tree





Fault tree analysis



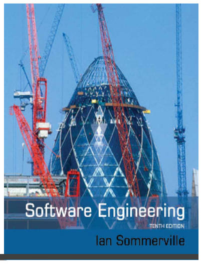
✧ Three possible conditions that can lead to delivery of incorrect dose of insulin

- Incorrect measurement of blood sugar level
- Failure of delivery system
- Dose delivered at wrong time

✧ By analysis of the fault tree, root causes of these hazards related to software are:

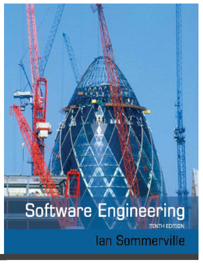
- Algorithm error
- Arithmetic error

Risk reduction



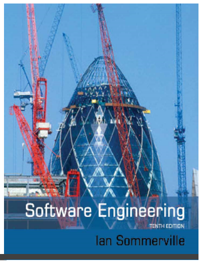
- ✧ The aim of this process is to identify dependability requirements that specify how the risks should be managed and ensure that accidents/incidents do not arise.
- ✧ Risk reduction strategies
  - Hazard avoidance;
  - Hazard detection and removal;
  - Damage limitation.

Strategy use



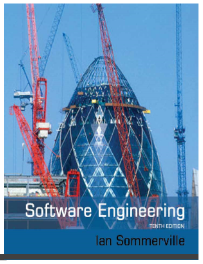
- ✧ Normally, in critical systems, a mix of risk reduction strategies are used.
- ✧ In a chemical plant control system, the system will include sensors to detect and correct excess pressure in the reactor.
- ✧ However, it will also include an independent protection system that opens a relief valve if dangerously high pressure is detected.

Insulin pump - software risks



- ✧ Arithmetic error
  - A computation causes the value of a variable to overflow or underflow;
  - Maybe include an exception handler for each type of arithmetic error.
- ✧ Algorithmic error
  - Compare dose to be delivered with previous dose or safe maximum doses. Reduce dose if too high.

Examples of safety requirements



SR1: The system shall not deliver a single dose of insulin that is greater than a specified maximum dose for a system user.

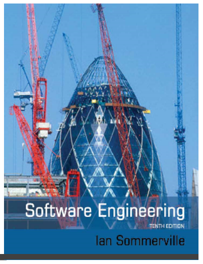
SR2: The system shall not deliver a daily cumulative dose of insulin that is greater than a specified maximum daily dose for a system user.

SR3: The system shall include a hardware diagnostic facility that shall be executed at least four times per hour.

SR4: The system shall include an exception handler for all of the exceptions that are identified in Table 3.

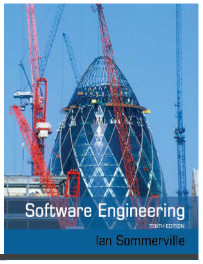
SR5: The audible alarm shall be sounded when any hardware or software anomaly is discovered and a diagnostic message, as defined in Table 4, shall be displayed.

SR6: In the event of an alarm, insulin delivery shall be suspended until the user has reset the system and cleared the alarm.



Safety engineering processes

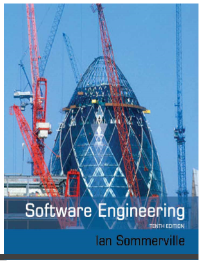
Safety engineering processes



✧ Safety engineering processes are based on reliability engineering processes

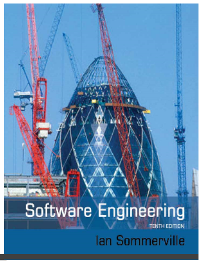
- Plan-based approach with reviews and checks at each stage in the process
- General goal of fault avoidance and fault detection
- Must also include safety reviews and explicit identification and tracking of hazards

Regulation



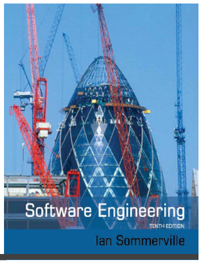
- ✧ Regulators may require evidence that safety engineering processes have been used in system development
- ✧ For example:
  - The specification of the system that has been developed and records of the checks made on that specification.
  - Evidence of the verification and validation processes that have been carried out and the results of the system verification and validation.
  - Evidence that the organizations developing the system have defined and dependable software processes that include safety assurance reviews. There must also be records that show that these processes have been properly enacted.

Agile methods and safety



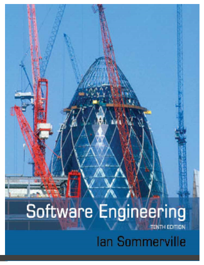
- Agile methods are not usually used for safety-critical systems engineering
  - Extensive process and product documentation is needed for system regulation. Contradicts the focus in agile methods on the software itself.
  - A detailed safety analysis of a complete system specification is important. Contradicts the interleaved development of a system specification and program.
- Some agile techniques such as test-driven development may be used

Safety assurance processes



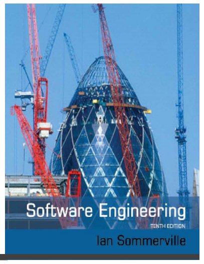
- Process assurance involves defining a dependable process and ensuring that this process is followed during the system development.
- Process assurance focuses on:
  - Do we have the right processes? Are the processes appropriate for the level of dependability required. Should include requirements management, change management, reviews and inspections, etc.
  - Are we doing the processes right? Have these processes been followed by the development team.
- Process assurance generates documentation
  - Agile processes therefore are rarely used for critical systems.

Processes for safety assurance



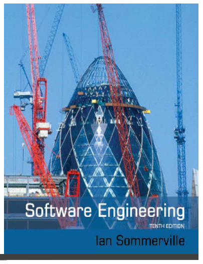
- ✧ Process assurance is important for safety-critical systems development:
  - Accidents are rare events so testing may not find all problems;
  - Safety requirements are sometimes ‘shall not’ requirements so cannot be demonstrated through testing.
- ✧ Safety assurance activities may be included in the software process that record the analyses that have been carried out and the people responsible for these.
  - Personal responsibility is important as system failures may lead to subsequent legal actions.

Safety related process activities



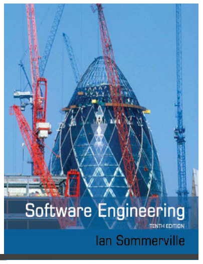
- ✧ Creation of a hazard logging and monitoring system.
- ✧ Appointment of project safety engineers who have explicit responsibility for system safety.
- ✧ Extensive use of safety reviews.
- ✧ Creation of a safety certification system where the safety of critical components is formally certified.
- ✧ Detailed configuration management (see Chapter 25).

Hazard analysis



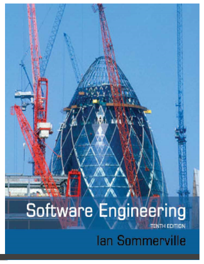
- ✧ Hazard analysis involves identifying hazards and their root causes.
- ✧ There should be clear traceability from identified hazards through their analysis to the actions taken during the process to ensure that these hazards have been covered.
- ✧ A hazard log may be used to track hazards throughout the process.

A simplified hazard log entry



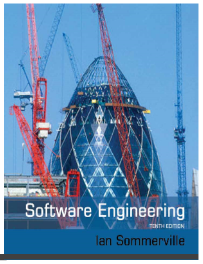
Hazard Log	**Page 4: Printed 20.02.2012**				
System: Insulin Pump System *Safety Engineer:* James Brown				*File*: InsulinPump/Safety/HazardLog *Log version*: 1/3	
Identified Hazard	Insulin overdose delivered to patient				
Identified by	Jane Williams				
Criticality class	1				
Identified risk	High				
Fault tree identified	YES	*Date*	24.01.07	*Location*	Hazard Log, Page 5
Fault tree creators	Jane Williams and Bill Smith				
Fault tree checked	YES	*Date*	28.01.07	*Checker*	James Brown

Hazard log (2)



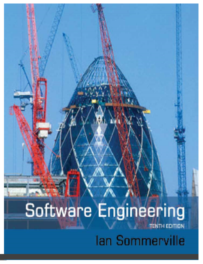
System safety design requirements
1. The system shall include self-testing software that will test the sensor system, the clock, and the insulin delivery system.
2. The self-checking software shall be executed once per minute.
3. In the event of the self-checking software discovering a fault in any of the system components, an audible warning shall be issued and the pump display shall indicate the name of the component where the fault has been discovered. The delivery of insulin shall be suspended.
4. The system shall incorporate an override system that allows the system user to modify the computed dose of insulin that is to be delivered by the system.
5. The amount of override shall be no greater than a pre-set value (maxOverride), which is set when the system is configured by medical staff.

Safety reviews



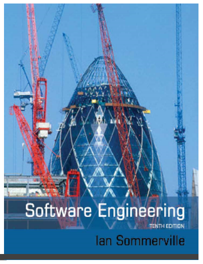
- ✧ Driven by the hazard register.
- ✧ For each identified hazrd, the review team should assess the system and judge whether or not the system can cope with that hazard in a safe way.

Formal verification



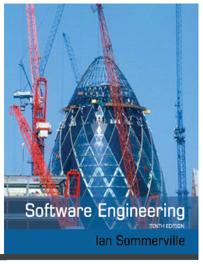
- ✧ Formal methods can be used when a mathematical specification of the system is produced.
- ✧ They are the ultimate static verification technique that may be used at different stages in the development process:
  - A formal specification may be developed and mathematically analyzed for consistency. This helps discover specification errors and omissions.
  - Formal arguments that a program conforms to its mathematical specification may be developed. This is effective in discovering programming and design errors.

Arguments for formal methods



- ✧ Producing a mathematical specification requires a detailed analysis of the requirements and this is likely to uncover errors.
- ✧ Concurrent systems can be analysed to discover race conditions that might lead to deadlock. Testing for such problems is very difficult.
- ✧ They can detect implementation errors before testing when the program is analyzed alongside the specification.

Arguments against formal methods



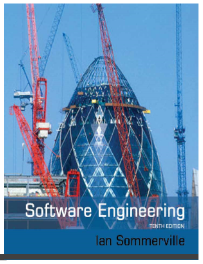
✧ Require specialized notations that cannot be understood by domain experts.

✧ Very expensive to develop a specification and even more expensive to show that a program meets that specification.

✧ Proofs may contain errors.

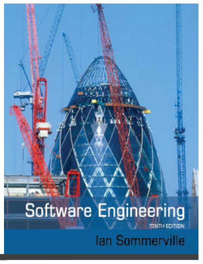
✧ It may be possible to reach the same level of confidence in a program more cheaply using other V & V techniques.

Formal methods cannot guarantee safety



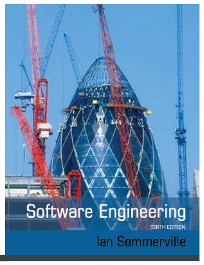
- ✧ The specification may not reflect the real requirements of system users. Users rarely understand formal notations so they cannot directly read the formal specification to find errors and omissions.
- ✧ The proof may contain errors. Program proofs are large and complex, so, like large and complex programs, they usually contain errors.
- ✧ The proof may make incorrect assumptions about the way that the system is used. If the system is not used as anticipated, then the system's behavior lies outside the scope of the proof.

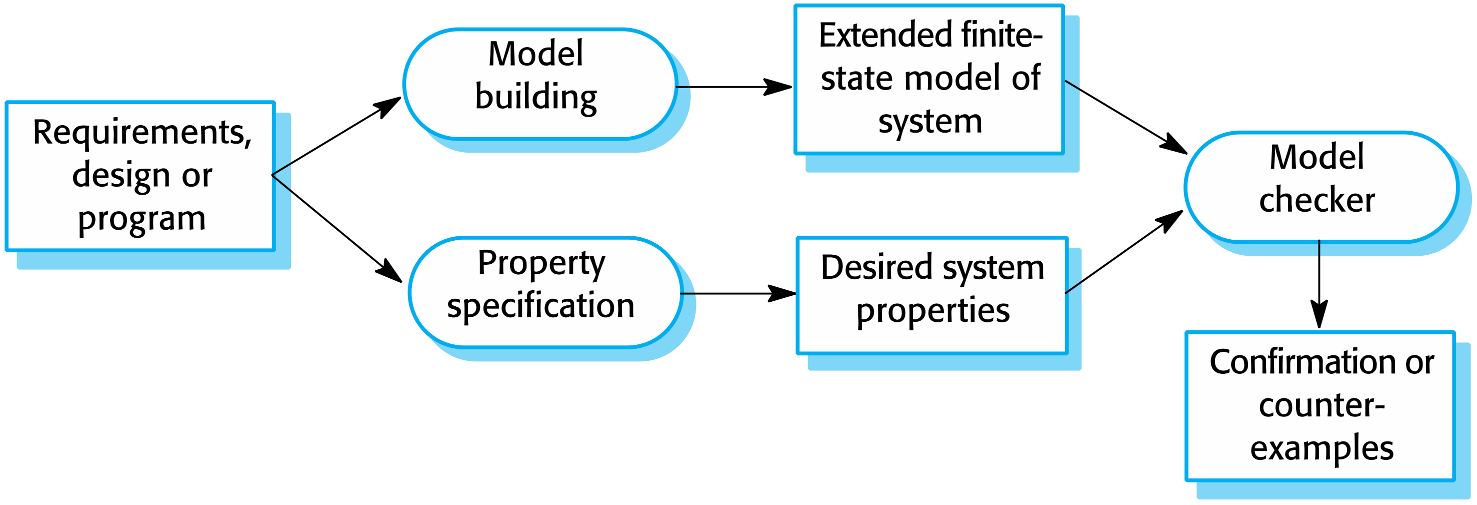
Model checking



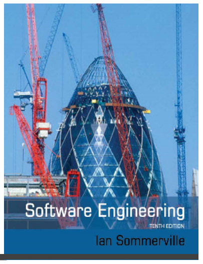
- Involves creating an extended finite state model of a system and, using a specialized system (a model checker), checking that model for errors.
- The model checker explores all possible paths through the model and checks that a user-specified property is valid for each path.
- Model checking is particularly valuable for verifying concurrent systems, which are hard to test.
- Although model checking is computationally very expensive, it is now practical to use it in the verification of small to medium sized critical systems.

Model checking




Requirements, design or program
Model building
Property specification
Extended finite-state model of system
Desired system properties
Model checker
Confirmation or counter-examples

Static program analysis

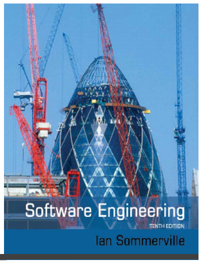


✧ Static analysers are software tools for source text processing.

✧ They parse the program text and try to discover potentially erroneous conditions and bring these to the attention of the V & V team.

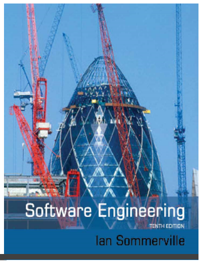
✧ They are very effective as an aid to inspections - they are a supplement to but not a replacement for inspections.

Automated static analysis checks



Fault class	Static analysis check
Data faults	Variables used before initialization Variables declared but never used Variables assigned twice but never used between assignments Possible array bound violations Undeclared variables
Control faults	Unreachable code Unconditional branches into loops
Input/output faults	Variables output twice with no intervening assignment
Interface faults	Parameter-type mismatches Parameter number mismatches Non-usage of the results of functions Uncalled functions and procedures
Storage management faults	Unassigned pointers Pointer arithmetic Memory leaks

Levels of static analysis



✧ Characteristic error checking

- The static analyzer can check for patterns in the code that are characteristic of errors made by programmers using a particular language.

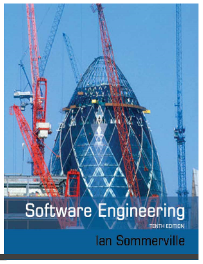
✧ User-defined error checking

- Users of a programming language define error patterns, thus extending the types of error that can be detected. This allows specific rules that apply to a program to be checked.

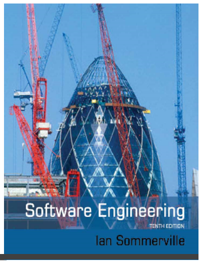
✧ Assertion checking

- Developers include formal assertions in their program and relationships that must hold. The static analyzer symbolically executes the code and highlights potential problems.

Use of static analysis

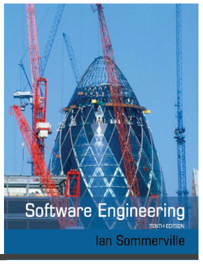


- ✧ Particularly valuable when a language such as C is used which has weak typing and hence many errors are undetected by the compiler.
- ✧ Particularly valuable for security checking – the static analyzer can discover areas of vulnerability such as buffer overflows or unchecked inputs.
- ✧ Static analysis is now routinely used in the development of many safety and security critical systems.



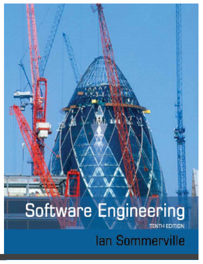
Safety cases

Safety and dependability cases



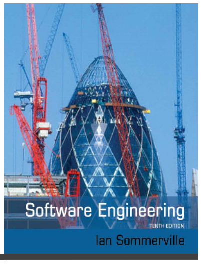
- ✧ Safety and dependability cases are structured documents that set out detailed arguments and evidence that a required level of safety or dependability has been achieved.
- ✧ They are normally required by regulators before a system can be certified for operational use. The regulator's responsibility is to check that a system is as safe or dependable as is practical.
- ✧ Regulators and developers work together and negotiate what needs to be included in a system safety/dependability case.

The system safety case

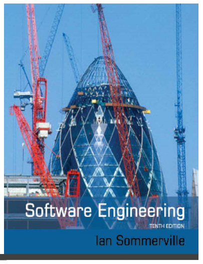


- A safety case is:
  - A documented body of evidence that provides a convincing and valid argument that a system is adequately safe for a given application in a given environment.
- Arguments in a safety case can be based on formal proof, design rationale, safety proofs, etc. Process factors may also be included.
- A software safety case is usually part of a wider system safety case that takes hardware and operational issues into account.

The contents of a software safety case

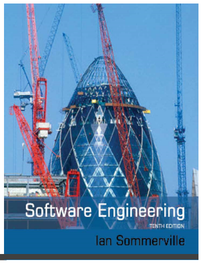


Chapter	Description
System description	An overview of the system and a description of its critical components.
Safety requirements	The safety requirements abstracted from the system requirements specification. Details of other relevant system requirements may also be included.
Hazard and risk analysis	Documents describing the hazards and risks that have been identified and the measures taken to reduce risk. Hazard analyses and hazard logs.
Design analysis	A set of structured arguments (see Section 15.5.1) that justify why the design is safe.
Verification and validation	A description of the V & V procedures used and, where appropriate, the test plans for the system. Summaries of the test results showing defects that have been detected and corrected. If formal methods have been used, a formal system specification and any analyses of that specification. Records of static analyses of the source code.



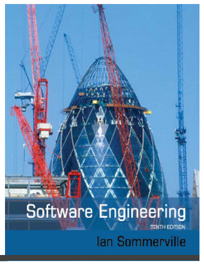
Chapter	Description
Review reports	Records of all design and safety reviews.
Team competences	Evidence of the competence of all of the team involved in safety-related systems development and validation.
Process QA	Records of the quality assurance processes (see Chapter 24) carried out during system development.
Change management processes	Records of all changes proposed, actions taken and, where appropriate, justification of the safety of these changes. Information about configuration management procedures and configuration management logs.
Associated safety cases	References to other safety cases that may impact the safety case.

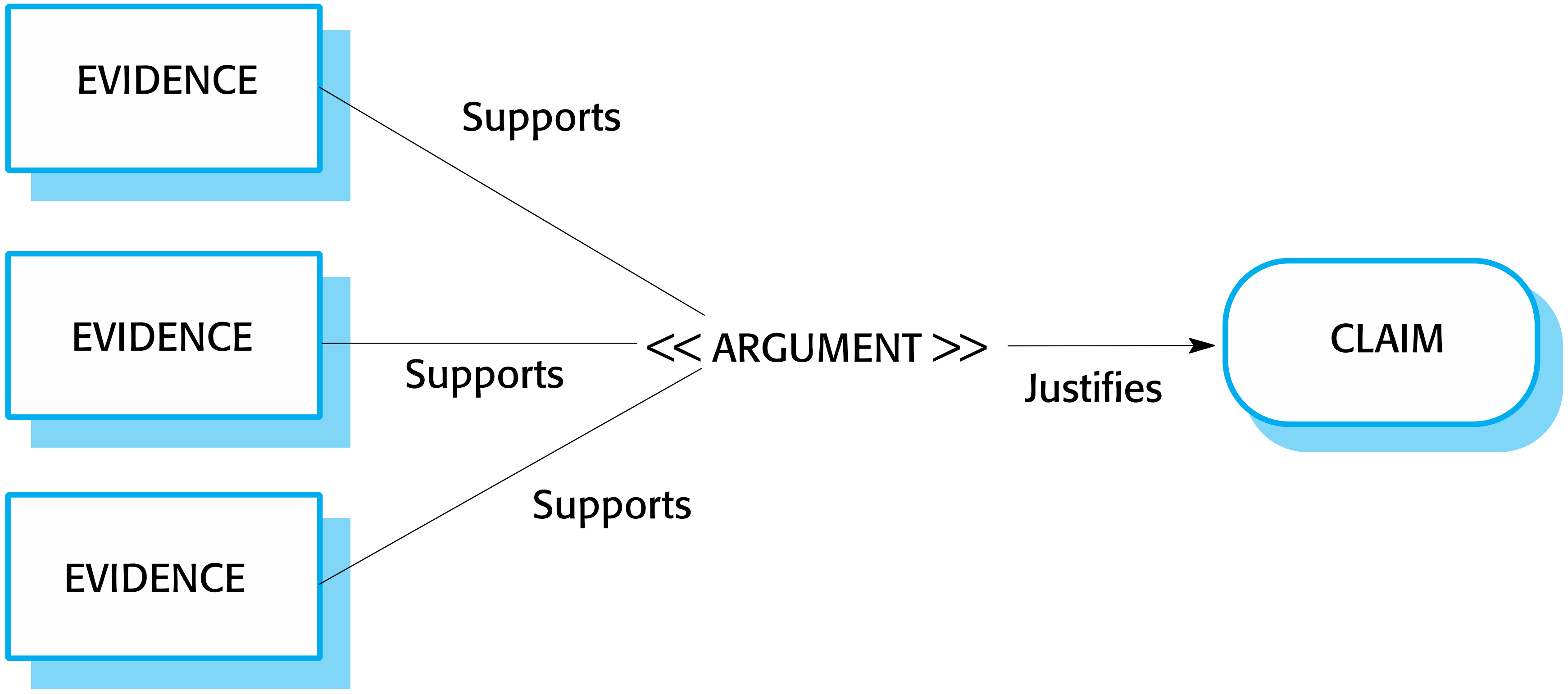
Structured arguments



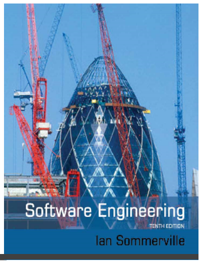
- ✧ Safety cases should be based around structured arguments that present evidence to justify the assertions made in these arguments.
- ✧ The argument justifies why a claim about system safety and security is justified by the available evidence.

Structured arguments





Insulin pump safety argument

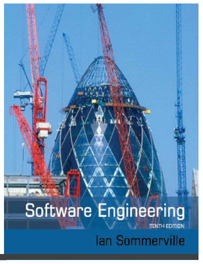


✧ Arguments are based on claims and evidence.

✧ Insulin pump safety:

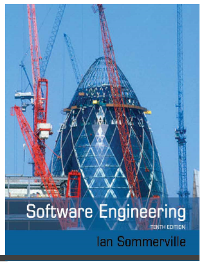
- Claim: The maximum single dose of insulin to be delivered (CurrentDose) will not exceed MaxDose.
- Evidence: Safety argument for insulin pump (discussed later)
- Evidence: Test data for insulin pump. The value of currentDose was correctly computed in 400 tests
- Evidence: Static analysis report for insulin pump software revealed no anomalies that affected the value of CurrentDose
- Argument: The evidence presented demonstrates that the maximum dose of insulin that can be computed = MaxDose.

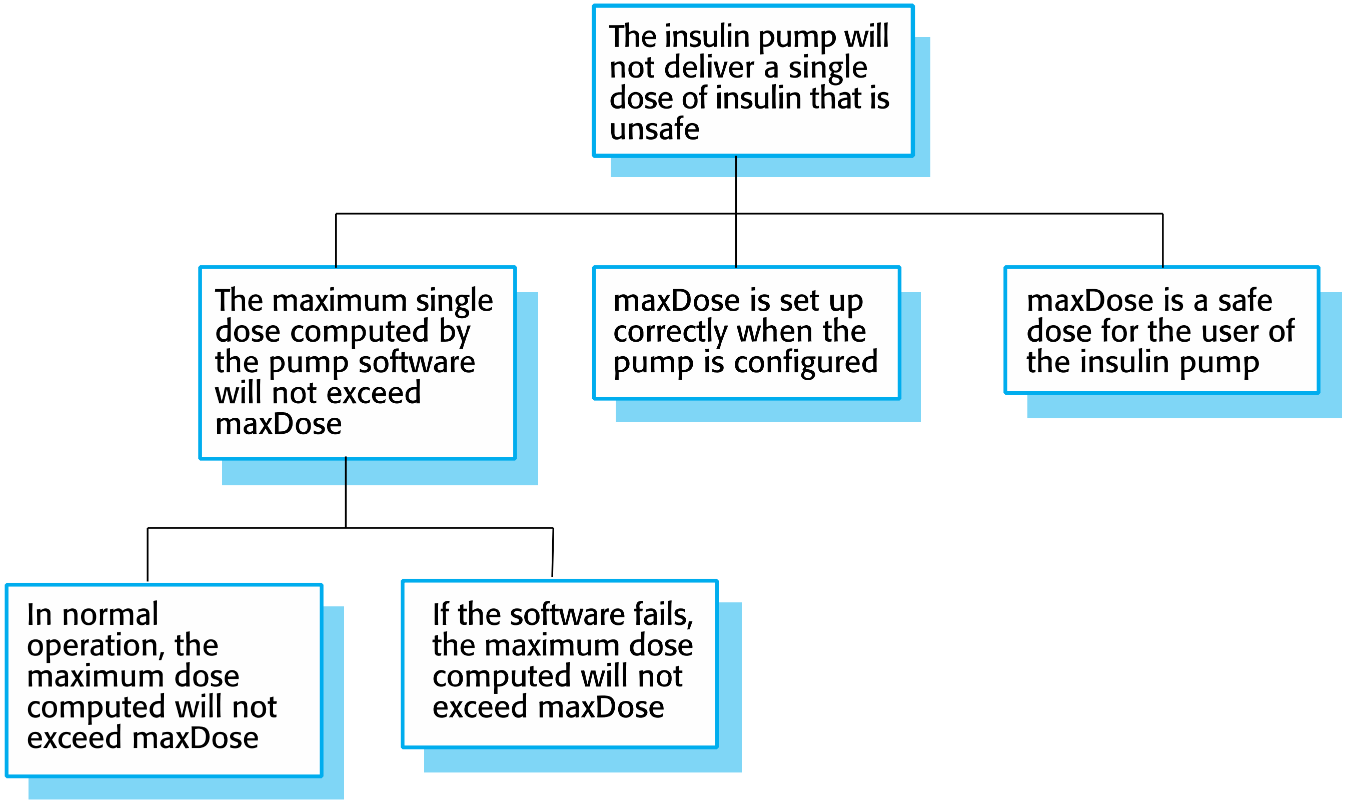
Structured safety arguments



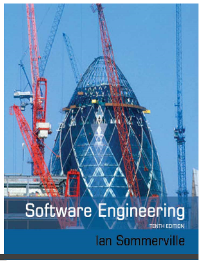
- ✧ Structured arguments that demonstrate that a system meets its safety obligations.
- ✧ It is not necessary to demonstrate that the program works as intended; the aim is simply to demonstrate safety.
- ✧ Generally based on a claim hierarchy.
  - ▪ You start at the leaves of the hierarchy and demonstrate safety. This implies the higher-level claims are true.

A safety claim hierarchy for the insulin pump



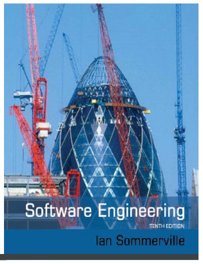

The insulin pump will not deliver a single dose of insulin that is unsafe
The maximum single dose computed by the pump software will not exceed maxDose
maxDose is set up correctly when the pump is configured
maxDose is a safe dose for the user of the insulin pump
In normal operation, the maximum dose computed will not exceed maxDose
If the software fails, the maximum dose computed will not exceed maxDose

Software safety arguments



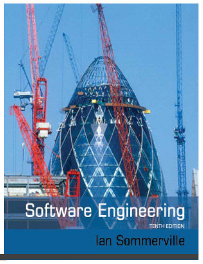
- ✧ Safety arguments are intended to show that the system cannot reach in unsafe state.
- ✧ These are weaker than correctness arguments which must show that the system code conforms to its specification.
- ✧ They are generally based on proof by contradiction
  - Assume that an unsafe state can be reached;
  - Show that this is contradicted by the program code.
- ✧ A graphical model of the safety argument may be developed.

Construction of a safety argument



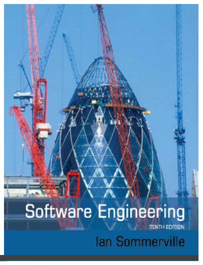
- ✧ Establish the safe exit conditions for a component or a program.
- ✧ Starting from the END of the code, work backwards until you have identified all paths that lead to the exit of the code.
- ✧ Assume that the exit condition is false.
- ✧ Show that, for each path leading to the exit that the assignments made in that path contradict the assumption of an unsafe exit from the component.

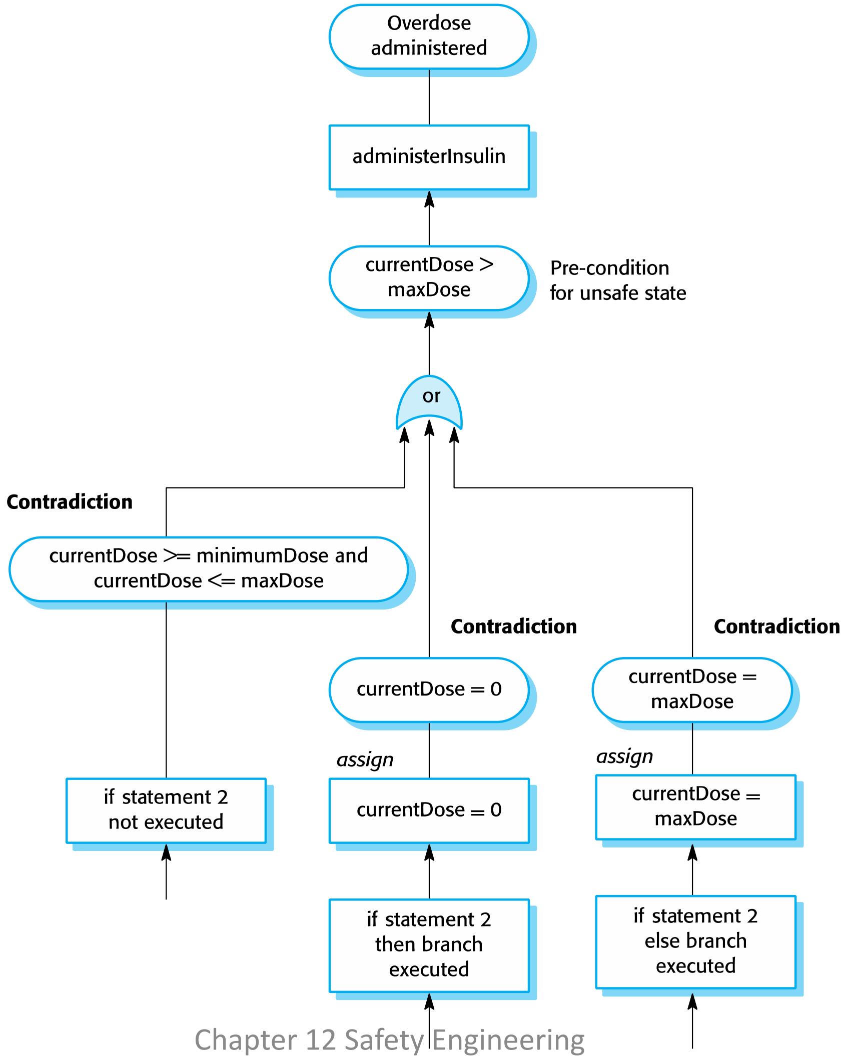
Insulin dose computation with safety checks



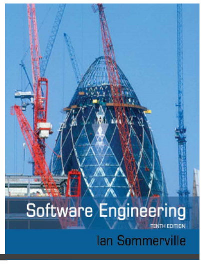
```
-- The insulin dose to be delivered is a function of blood sugar level,
-- the previous dose delivered and the time of delivery of the previous dose

currentDose = computeInsulin () ;

// Safety check—adjust currentDose if necessary.
// if statement 1
if (previousDose == 0)
{
        if (currentDose > maxDose/2)
        currentDose = maxDose/2 ;
}
else
        if (currentDose > (previousDose * 2) )
                currentDose = previousDose * 2 ;
// if statement 2
if ( currentDose < minimumDose )
        currentDose = 0 ;
else if ( currentDose > maxDose )
        currentDose = maxDose ;
administerInsulin (currentDose) ;
```

Informal safety argument based on demonstrating contradictions



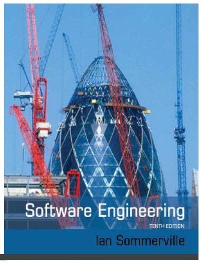


Program paths



- ✧ Neither branch of if-statement 2 is executed
  - Can only happen if CurrentDose is >= minimumDose and <= maxDose.
- ✧ then branch of if-statement 2 is executed
  - currentDose = 0.
- ✧ else branch of if-statement 2 is executed
  - currentDose = maxDose.
- ✧ In all cases, the post conditions contradict the unsafe condition that the dose administered is greater than maxDose.

Key points

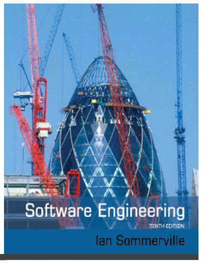


✧ Safety-critical systems are systems whose failure can lead to human injury or death.

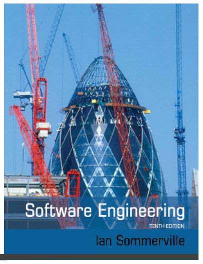
✧ A hazard-driven approach is used to understand the safety requirements for safety-critical systems. You identify potential hazards and decompose these (using methods such as fault tree analysis) to discover their root causes. You then specify requirements to avoid or recover from these problems.

✧ It is important to have a well-defined, certified process for safety-critical systems development. This should include the identification and monitoring of potential hazards.

Key points

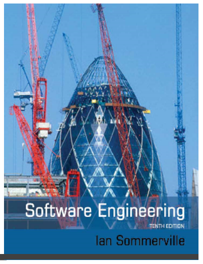


- ✧ Static analysis is an approach to V & V that examines the source code of a system, looking for errors and anomalies. It allows all parts of a program to be checked, not just those parts that are exercised by system tests.
- ✧ Model checking is a formal approach to static analysis that exhaustively checks all states in a system for potential errors.
- ✧ Safety and dependability cases collect the evidence that demonstrates a system is safe and dependable. Safety cases are required when an external regulator must certify the system before it is used.



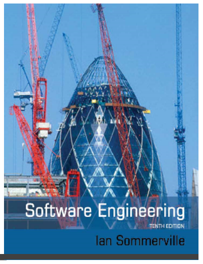
Chapter 13 – Security Engineering

Topics covered



- ✧ Security and dependability
- ✧ Security and organizations
- ✧ Security requirements
- ✧ Secure systems design
- ✧ Security testing and assurance

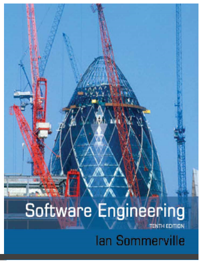
Security engineering



✧ Tools, techniques and methods to support the development and maintenance of systems that can resist malicious attacks that are intended to damage a computer-based system or its data.

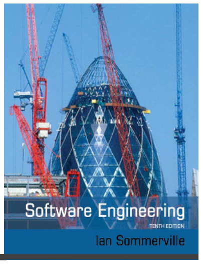
✧ A sub-field of the broader field of computer security.

Security dimensions



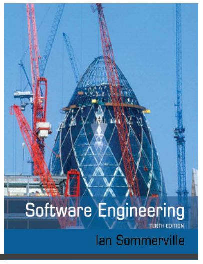
- *Confidentiality*
  - Information in a system may be disclosed or made accessible to people or programs that are not authorized to have access to that information.
- *Integrity*
  - Information in a system may be damaged or corrupted making it unusual or unreliable.
- *Availability*
  - Access to a system or its data that is normally available may not be possible.

Security levels



- ✧ Infrastructure security, which is concerned with maintaining the security of all systems and networks that provide an infrastructure and a set of shared services to the organization.
- ✧ Application security, which is concerned with the security of individual application systems or related groups of systems.
- ✧ Operational security, which is concerned with the secure operation and use of the organization's systems.

System layers where security may be compromised



Application

Reusable components and libraries

Middleware

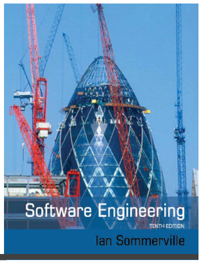
Database management

Generic, shared applications (browsers, e--mail, etc)

Operating System

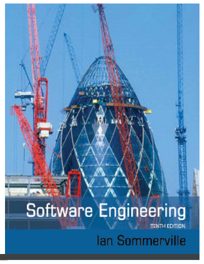
Network   Computer hardware

Application/infrastructure security



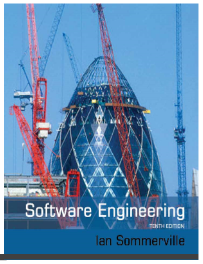
- ✧ Application security is a software engineering problem where the system is designed to resist attacks.
- ✧ Infrastructure security is a systems management problem where the infrastructure is configured to resist attacks.
- ✧ The focus of this chapter is application security rather than infrastructure security.

System security management

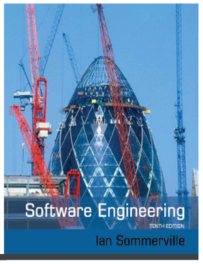


- User and permission management
  - Adding and removing users from the system and setting up appropriate permissions for users
- Software deployment and maintenance
  - Installing application software and middleware and configuring these systems so that vulnerabilities are avoided.
- Attack monitoring, detection and recovery
  - Monitoring the system for unauthorized access, design strategies for resisting attacks and develop backup and recovery strategies.

Operational security

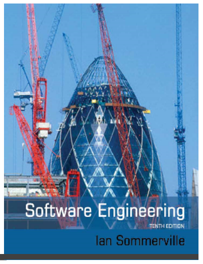


- ✧ Primarily a human and social issue
- ✧ Concerned with ensuring the people do not take actions that may compromise system security
  - E.g. Tell others passwords, leave computers logged on
- ✧ Users sometimes take insecure actions to make it easier for them to do their jobs
- ✧ There is therefore a trade-off between system security and system effectiveness.



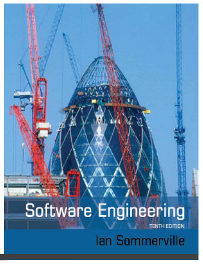
Security and dependability

Security



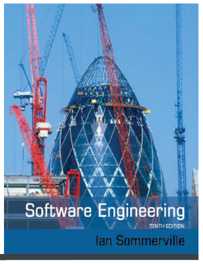
- ✧ The security of a system is a system property that reflects the system's ability to protect itself from accidental or deliberate external attack.
- ✧ Security is essential as most systems are networked so that external access to the system through the Internet is possible.
- ✧ Security is an essential pre-requisite for availability, reliability and safety.

Fundamental security



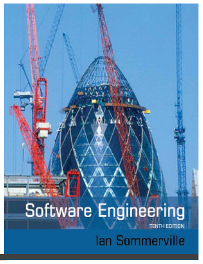
- ✧ If a system is a networked system and is insecure then statements about its reliability and its safety are unreliable.
- ✧ These statements depend on the executing system and the developed system being the same. However, intrusion can change the executing system and/or its data.
- ✧ Therefore, the reliability and safety assurance is no longer valid.

Security terminology



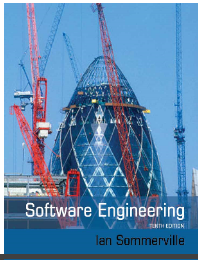
Term	Definition
Asset	Something of value which has to be protected. The asset may be the software system itself or data used by that system.
Attack	An exploitation of a system's vulnerability. Generally, this is from outside the system and is a deliberate attempt to cause some damage.
Control	A protective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

Examples of security terminology (Mentcare)



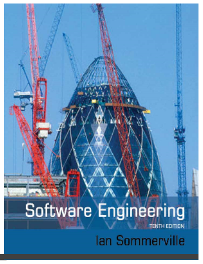
Term	Example
Asset	The records of each patient that is receiving or has received treatment.
Exposure	Potential financial loss from future patients who do not seek treatment because they do not trust the clinic to maintain their data. Financial loss from legal action by the sports star. Loss of reputation.
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.

Threat types



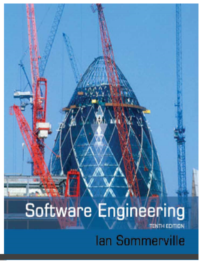
- Interception threats that allow an attacker to gain access to an asset.
  - A possible threat to the Mentcare system might be a situation where an attacker gains access to the records of an individual patient.
- Interruption threats that allow an attacker to make part of the system unavailable.
  - A possible threat might be a denial of service attack on a system database server so that database connections become impossible.

Threat types



- Modification threats that allow an attacker to tamper with a system asset.
  - In the Mentcare system, a modification threat would be where an attacker alters or destroys a patient record.
- Fabrication threats that allow an attacker to insert false information into a system.
  - This is perhaps not a credible threat in the Mentcare system but would be a threat in a banking system, where false transactions might be added to the system that transfer money to the perpetrator's bank account.

Security assurance



✧ Vulnerability avoidance

- The system is designed so that vulnerabilities do not occur. For example, if there is no external network connection then external attack is impossible

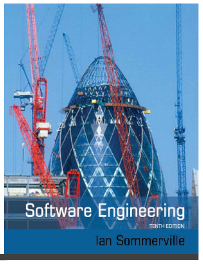
✧ Attack detection and elimination

- The system is designed so that attacks on vulnerabilities are detected and neutralised before they result in an exposure. For example, virus checkers find and remove viruses before they infect a system

✧ Exposure limitation and recovery

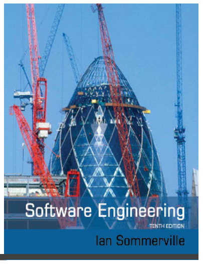
- The system is designed so that the adverse consequences of a successful attack are minimised. For example, a backup policy allows damaged information to be restored

Security and dependability



- *Security and reliability*
  - If a system is attacked and the system or its data are corrupted as a consequence of that attack, then this may induce system failures that compromise the reliability of the system.
- *Security and availability*
  - A common attack on a web-based system is a denial of service attack, where a web server is flooded with service requests from a range of different sources. The aim of this attack is to make the system unavailable.

Security and dependability

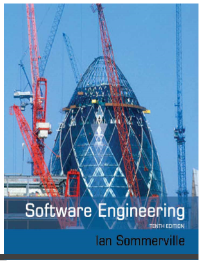


✧ *Security and safety*

- An attack that corrupts the system or its data means that assumptions about safety may not hold. Safety checks rely on analysing the source code of safety critical software and assume the executing code is a completely accurate translation of that source code. If this is not the case, safety-related failures may be induced and the safety case made for the software is invalid.

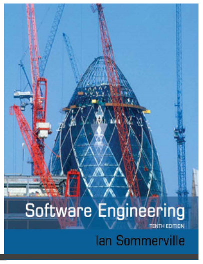
✧ *Security and resilience*

- Resilience is a system characteristic that reflects its ability to resist and recover from damaging events. The most probable damaging event on networked software systems is a cyberattack of some kind so most of the work now done in resilience is aimed at deterring, detecting and recovering from such attacks.



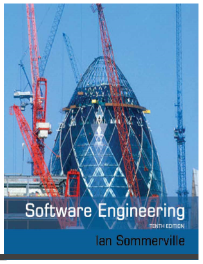
Security and organizations

Security is a business issue



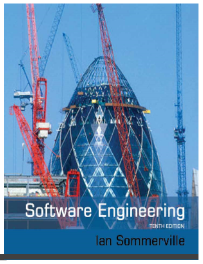
- ✧ Security is expensive and it is important that security decisions are made in a cost-effective way
  - ▪ There is no point in spending more than the value of an asset to keep that asset secure.
- ✧ Organizations use a risk-based approach to support security decision making and should have a defined security policy based on security risk analysis
- ✧ Security risk analysis is a business rather than a technical process

Organizational security policies



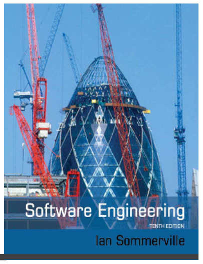
- ✧ Security policies should set out general information access strategies that should apply across the organization.
- ✧ The point of security policies is to inform everyone in an organization about security so these should not be long and detailed technical documents.
- ✧ From a security engineering perspective, the security policy defines, in broad terms, the security goals of the organization.
- ✧ The security engineering process is concerned with implementing these goals.

Security policies



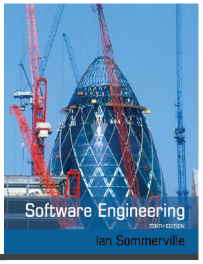
- ✧ *The assets that must be protected*
  - It is not cost-effective to apply stringent security procedures to all organizational assets. Many assets are not confidential and can be made freely available.
- ✧ *The level of protection that is required for different types of asset*
  - For sensitive personal information, a high level of security is required; for other information, the consequences of loss may be minor so a lower level of security is adequate.

Security policies



- ✧ *The responsibilities of individual users, managers and the organization*
  - The security policy should set out what is expected of users e.g. strong passwords, log out of computers, office security, etc.
- ✧ *Existing security procedures and technologies that should be maintained*
  - For reasons of practicality and cost, it may be essential to continue to use existing approaches to security even where these have known limitations.

Security risk assessment and management



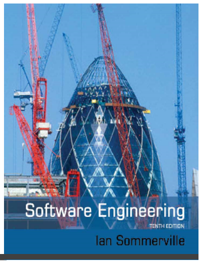
✧ Risk assessment and management is concerned with assessing the possible losses that might ensue from attacks on the system and balancing these losses against the costs of security procedures that may reduce these losses.

✧ Risk management should be driven by an organisational security policy.

✧ Risk management involves

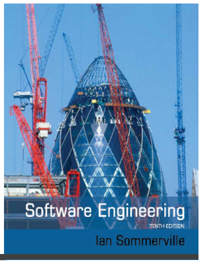
- Preliminary risk assessment
- Life cycle risk assessment
- Operational risk assessment

Preliminary risk assessment



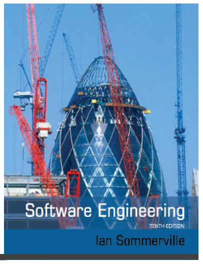
- ✧ The aim of this initial risk assessment is to identify generic risks that are applicable to the system and to decide if an adequate level of security can be achieved at a reasonable cost.
- ✧ The risk assessment should focus on the identification and analysis of high-level risks to the system.
- ✧ The outcomes of the risk assessment process are used to help identify security requirements.

Design risk assessment

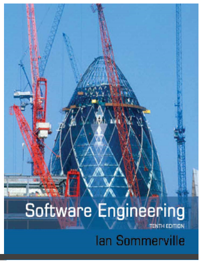


- ✧ This risk assessment takes place during the system development life cycle and is informed by the technical system design and implementation decisions.
- ✧ The results of the assessment may lead to changes to the security requirements and the addition of new requirements.
- ✧ Known and potential vulnerabilities are identified, and this knowledge is used to inform decision making about the system functionality and how it is to be implemented, tested, and deployed.

Operational risk assessment

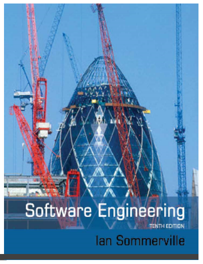


- ✧ This risk assessment process focuses on the use of the system and the possible risks that can arise from human behavior.
- ✧ Operational risk assessment should continue after a system has been installed to take account of how the system is used.
- ✧ Organizational changes may mean that the system is used in different ways from those originally planned. These changes lead to new security requirements that have to be implemented as the system evolves.



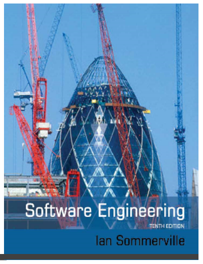
Security requirements

Security specification



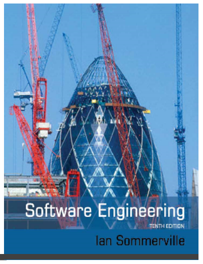
- ✧ Security specification has something in common with safety requirements specification – in both cases, your concern is to avoid something bad happening.
- ✧ Four major differences
  - Safety problems are accidental – the software is not operating in a hostile environment. In security, you must assume that attackers have knowledge of system weaknesses
  - When safety failures occur, you can look for the root cause or weakness that led to the failure. When failure results from a deliberate attack, the attacker may conceal the cause of the failure.
  - Shutting down a system can avoid a safety-related failure. Causing a shut down may be the aim of an attack.
  - Safety-related events are not generated from an intelligent adversary. An attacker can probe defenses over time to discover weaknesses.

Types of security requirement



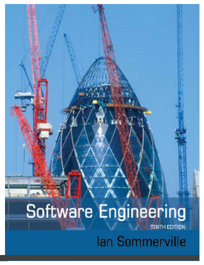
- ✧ Identification requirements.
- ✧ Authentication requirements.
- ✧ Authorisation requirements.
- ✧ Immunity requirements.
- ✧ Integrity requirements.
- ✧ Intrusion detection requirements.
- ✧ Non-repudiation requirements.
- ✧ Privacy requirements.
- ✧ Security auditing requirements.
- ✧ System maintenance security requirements.

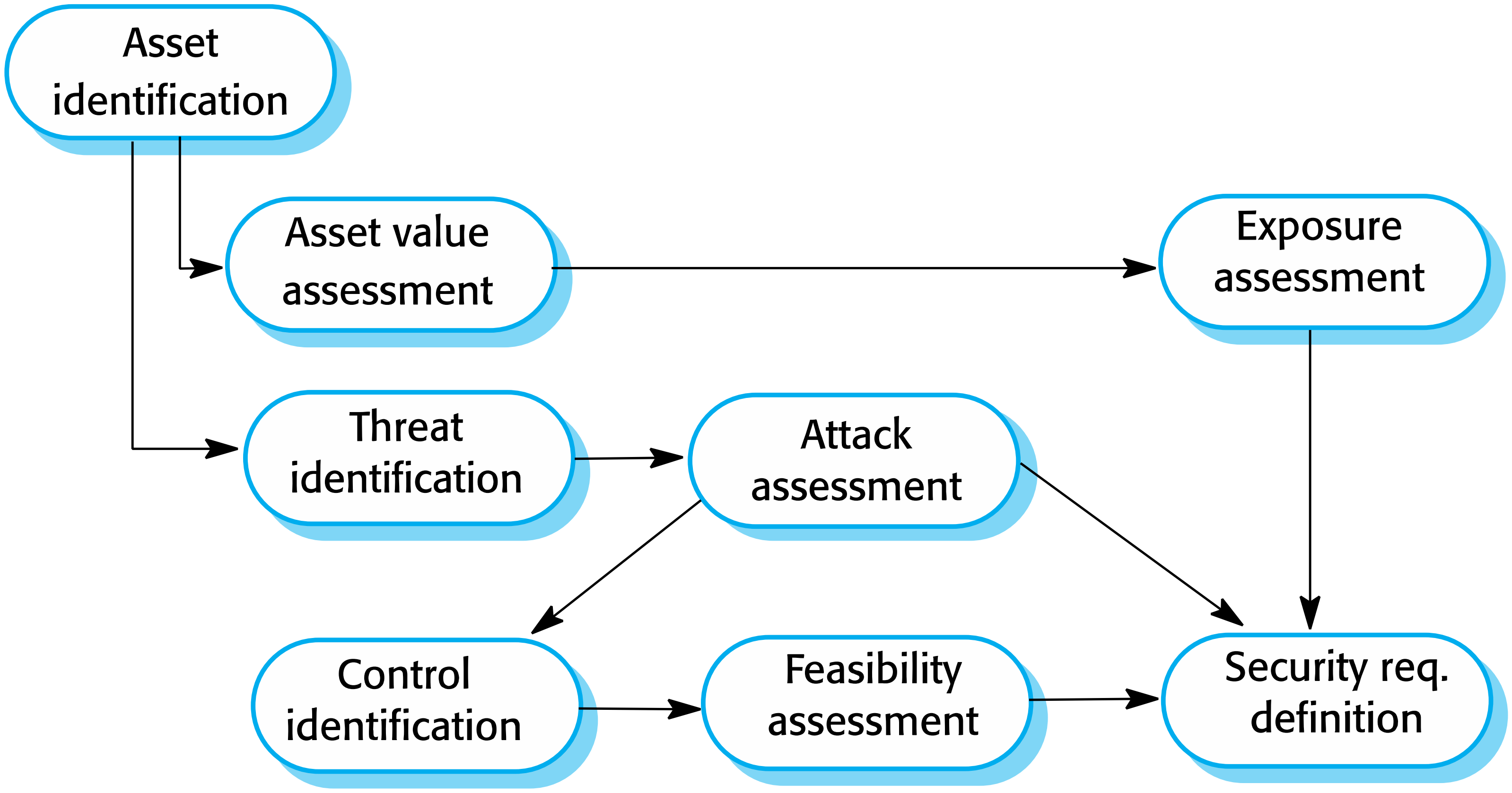
Security requirement classification



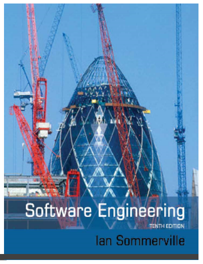
- ✧ Risk avoidance requirements set out the risks that should be avoided by designing the system so that these risks simply cannot arise.
- ✧ Risk detection requirements define mechanisms that identify the risk if it arises and neutralise the risk before losses occur.
- ✧ Risk mitigation requirements set out how the system should be designed so that it can recover from and restore system assets after some loss has occurred.

The preliminary risk assessment process for security requirements



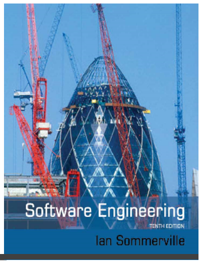


Security risk assessment



- ✧ Asset identification
  - Identify the key system assets (or services) that have to be protected.
- ✧ Asset value assessment
  - Estimate the value of the identified assets.
- ✧ Exposure assessment
  - Assess the potential losses associated with each asset.
- ✧ Threat identification
  - Identify the most probable threats to the system assets

Security risk assessment



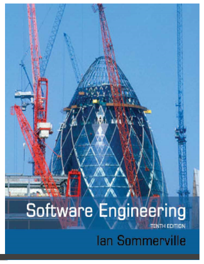
✧ Attack assessment
  - Decompose threats into possible attacks on the system and the ways that these may occur.

✧ Control identification
  - Propose the controls that may be put in place to protect an asset.

✧ Feasibility assessment
  - Assess the technical feasibility and cost of the controls.

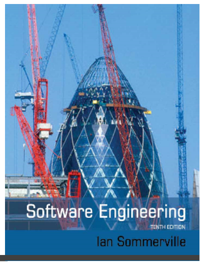
✧ Security requirements definition
  - Define system security requirements. These can be infrastructure or application system requirements.

Asset analysis in a preliminary risk assessment report for the Mentcare system



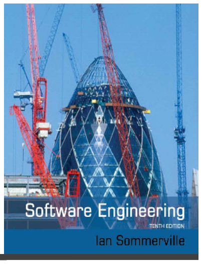
Asset	Value	Exposure
The information system	High. Required to support all clinical consultations. Potentially safety-critical.	High. Financial loss as clinics may have to be canceled. Costs of restoring system. Possible patient harm if treatment cannot be prescribed.
The patient database	High. Required to support all clinical consultations. Potentially safety-critical.	High. Financial loss as clinics may have to be canceled. Costs of restoring system. Possible patient harm if treatment cannot be prescribed.
An individual patient record	Normally low although may be high for specific high-profile patients.	Low direct losses but possible loss of reputation.

Threat and control analysis in a preliminary risk assessment report



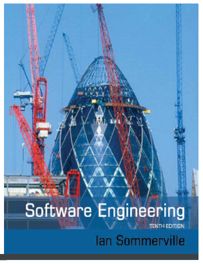
Threat	Probability	Control	Feasibility
An unauthorized user gains access as system manager and makes system unavailable	Low	Only allow system management from specific locations that are physically secure.	Low cost of implementation but care must be taken with key distribution and to ensure that keys are available in the event of an emergency.
An unauthorized user gains access as system user and accesses confidential information	High	Require all users to authenticate themselves using a biometric mechanism. Log all changes to patient information to track system usage.	Technically feasible but high-cost solution. Possible user resistance. Simple and transparent to implement and also supports recovery.

Security requirements for the Mentcare system



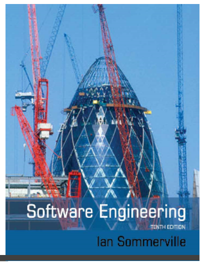
- ✧ Patient information shall be downloaded at the start of a clinic session to a secure area on the system client that is used by clinical staff.
- ✧ All patient information on the system client shall be encrypted.
- ✧ Patient information shall be uploaded to the database after a clinic session has finished and deleted from the client computer.
- ✧ A log on a separate computer from the database server must be maintained of all changes made to the system database.

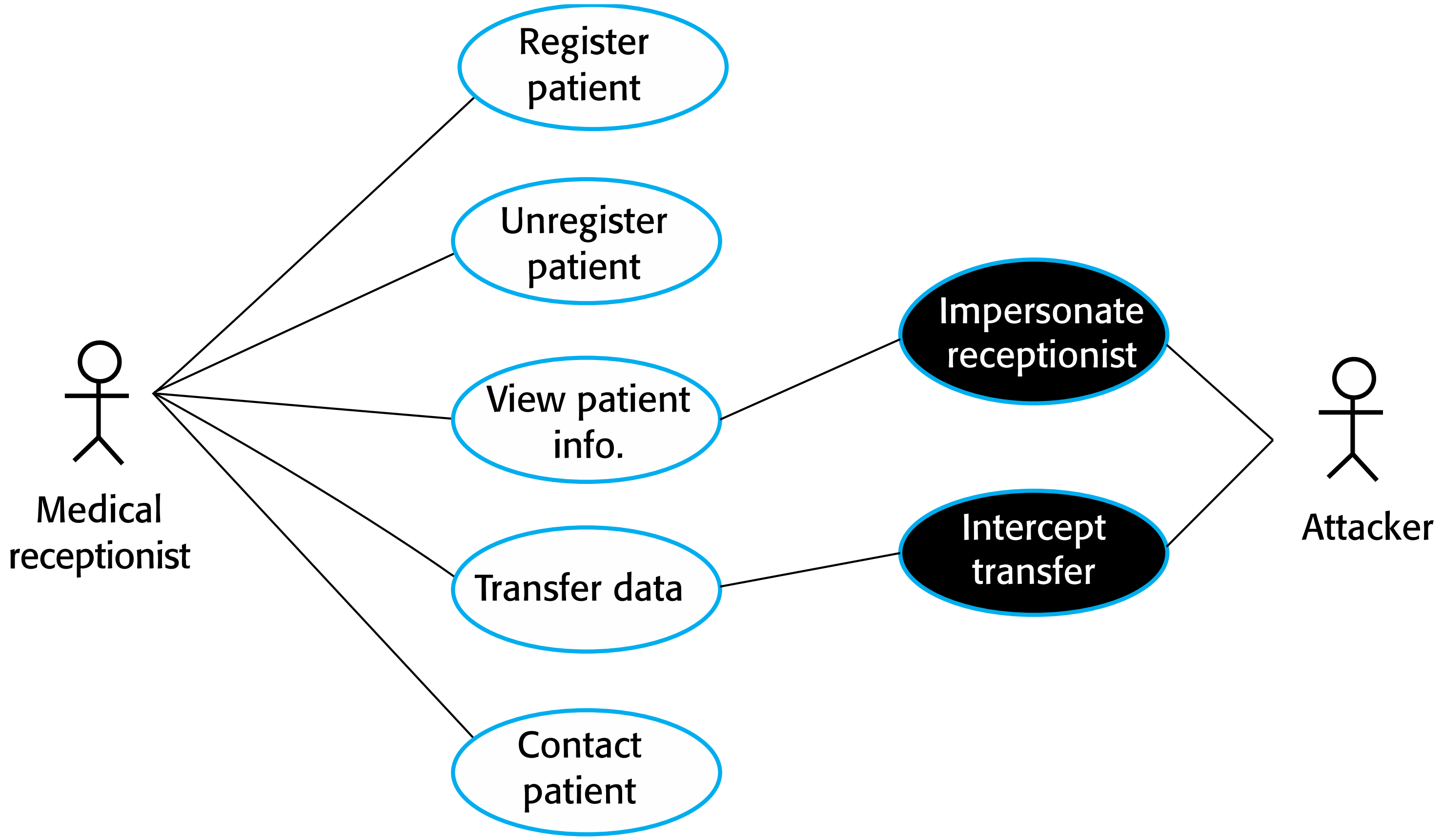
Misuse cases



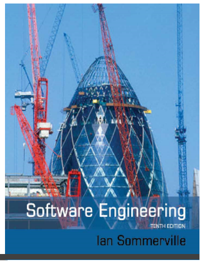
- ✧ Misuse cases are instances of threats to a system
- ✧ Interception threats
  - Attacker gains access to an asset
- ✧ Interruption threats
  - Attacker makes part of a system unavailable
- ✧ Modification threats
  - A system asset if tampered with
- ✧ Fabrication threats
  - False information is added to a system

Misuse cases



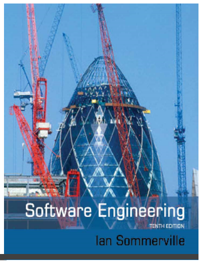


Mentcare use case – Transfer data

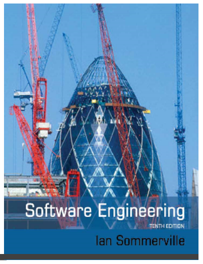


Mentcare system: Transfer data	
Actors	Medical receptionist, Patient records system (PRS)
Description	A receptionist may transfer data from the Mentcare system to a general patient record database that is maintained by a health authority. The information transferred may either be updated personal information (address, phone number, etc.) or a summary of the patient's diagnosis and treatment.
Data	Patient's personal information, treatment summary.
Stimulus	User command issued by medical receptionist.
Response	Confirmation that PRS has been updated.
Comments	The receptionist must have appropriate security permissions to access the patient information and the PRS.

Mentcare misuse case: Intercept transfer

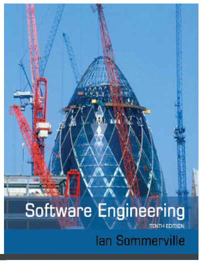


Mentcare system: Intercept transfer (Misuse case)	
Actors	Medical receptionist, Patient records system (PRS), Attacker
Description	A receptionist transfers data from his or her PC to the Mentcare system on the server. An attacker intercepts the data transfer and takes a copy of that data.
Data (assets)	Patient's personal information, treatment summary
Attacks	A network monitor is added to the system and packets from the receptionist to the server are intercepted. A spoof server is set up between the receptionist and the database server so that receptionist believes they are interacting with the real system.



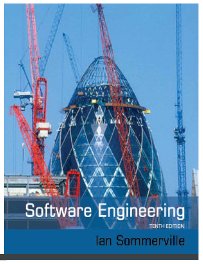
Misuse case: Intercept transfer

Mentcare system: Intercept transfer (Misuse case)	
Mitigations	All networking equipment must be maintained in a locked room. Engineers accessing the equipment must be accredited. All data transfers between the client and server must be encrypted. Certificate-based client-server communication must be used
Requirements	All communications between the client and the server must use the Secure Socket Layer (SSL). The https protocol uses certificate based authentication and encryption.



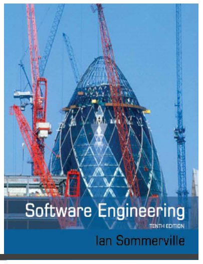
Secure systems design

Secure systems design



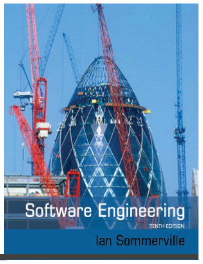
- ✧ Security should be designed into a system – it is very difficult to make an insecure system secure after it has been designed or implemented
- ✧ Architectural design
  - how do architectural design decisions affect the security of a system?
- ✧ Good practice
  - what is accepted good practice when designing secure systems?

Design compromises



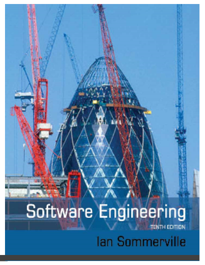
- ✧ Adding security features to a system to enhance its security affects other attributes of the system
- ✧ Performance
  - Additional security checks slow down a system so its response time or throughput may be affected
- ✧ Usability
  - Security measures may require users to remember information or require additional interactions to complete a transaction. This makes the system less usable and can frustrate system users.

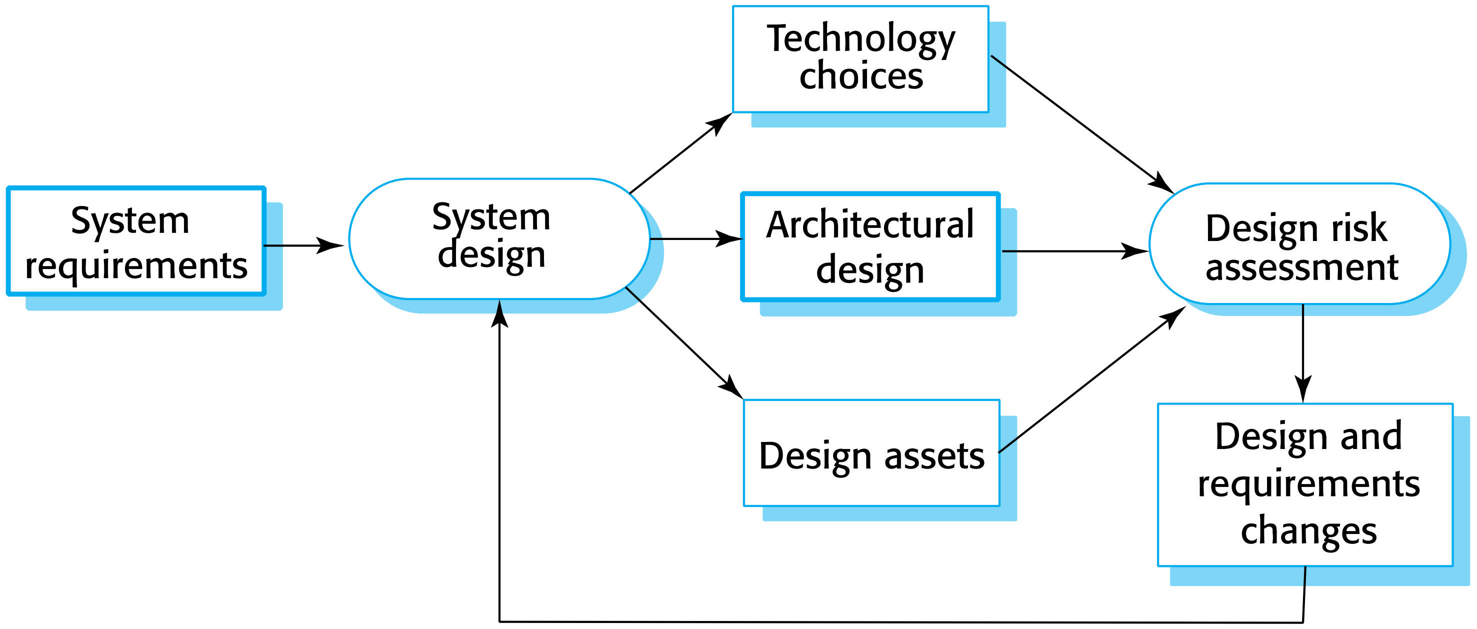
Design risk assessment



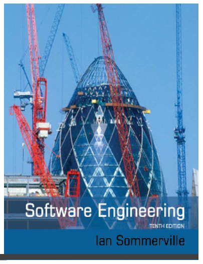
- ✧ Risk assessment while the system is being developed and after it has been deployed
- ✧ More information is available - system platform, middleware and the system architecture and data organisation.
- ✧ Vulnerabilities that arise from design choices may therefore be identified.

Design and risk assessment



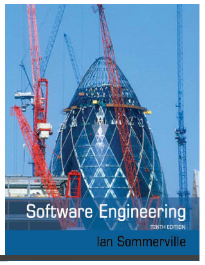


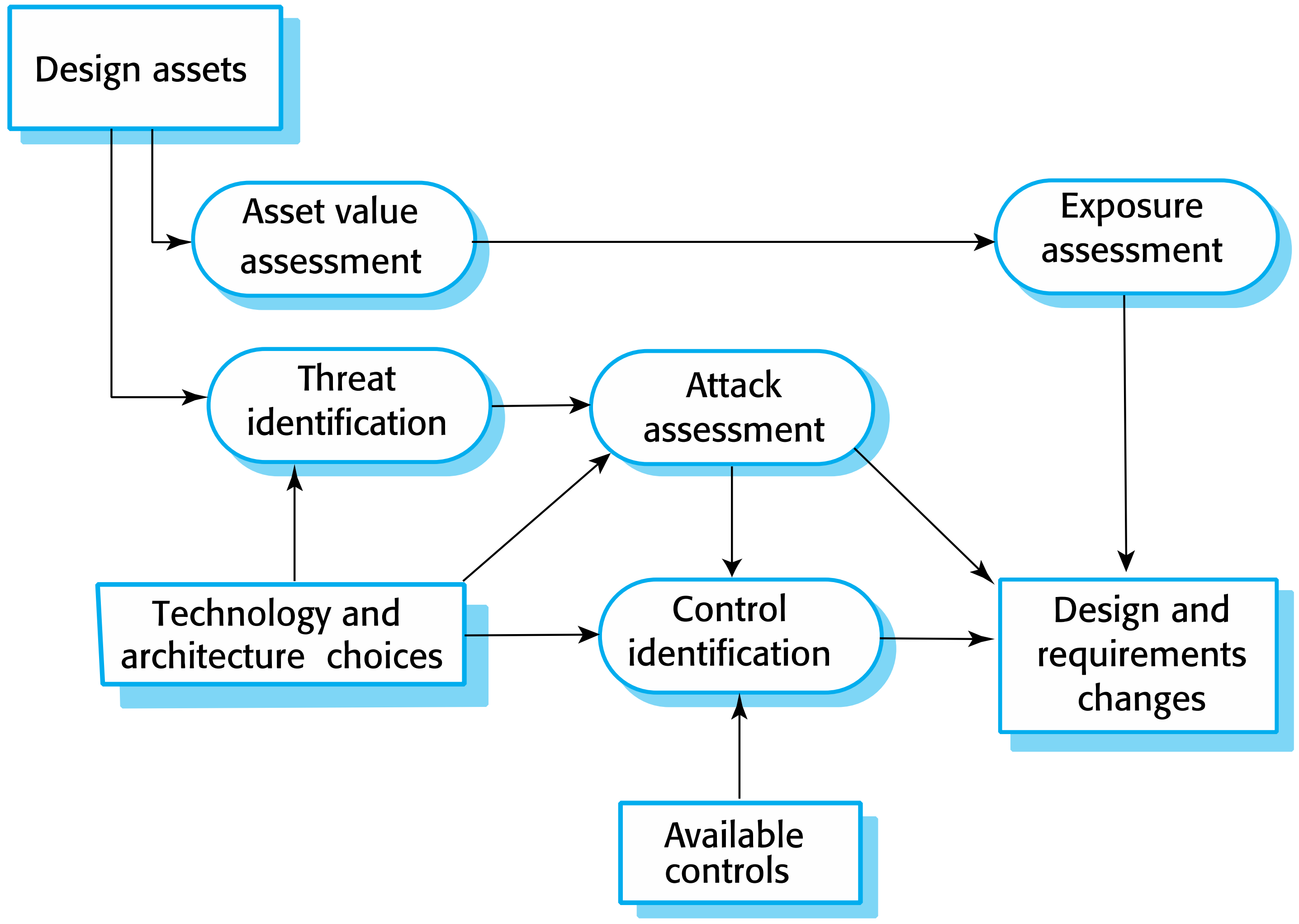
Protection requirements



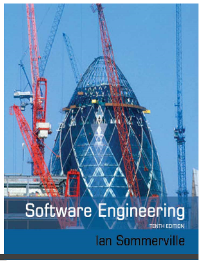
- ✧ Protection requirements may be generated when knowledge of information representation and system distribution
- ✧ Separating patient and treatment information limits the amount of information (personal patient data) that needs to be protected
- ✧ Maintaining copies of records on a local client protects against denial of service attacks on the server
  - But these may need to be encrypted

Design risk assessment



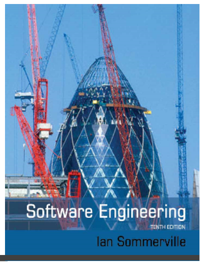


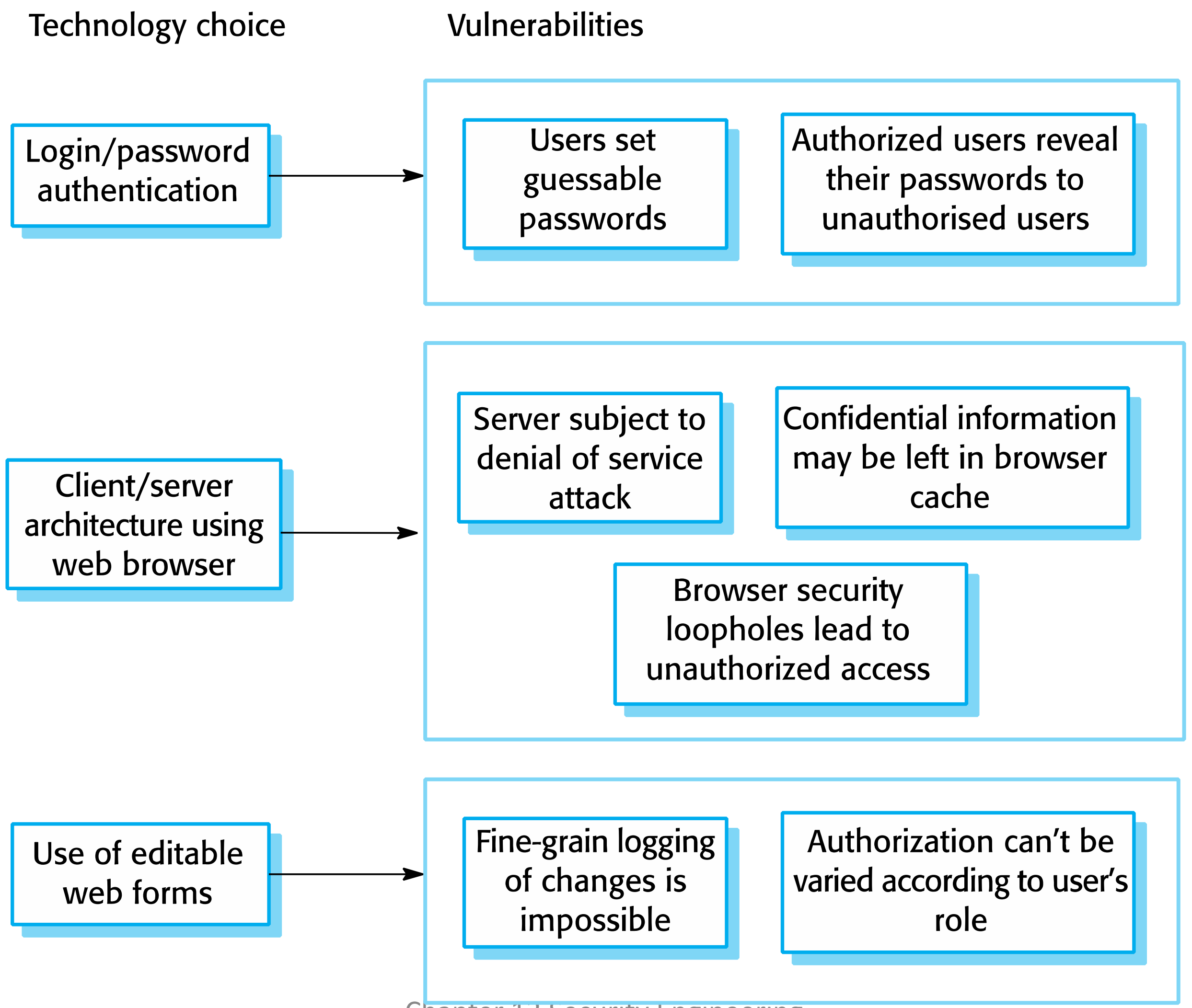
Design decisions from use of COTS



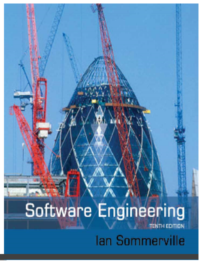
- ✧ System users authenticated using a name/password combination.
- ✧ The system architecture is client-server with clients accessing the system through a standard web browser.
- ✧ Information is presented as an editable web form.

Vulnerabilities associated with technology choices



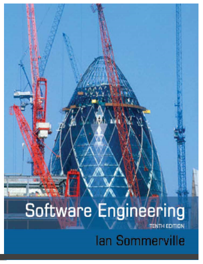

Technology choice
Vulnerabilities
Login/password authentication
Users set guessable passwords
Authorized users reveal their passwords to unauthorised users
Client/server architecture using web browser
Server subject to denial of service attack
Confidential information may be left in browser cache
Browser security loopholes lead to unauthorized access
Use of editable web forms
Fine-grain logging of changes is impossible
Authorization can't be varied according to user's role

Security requirements



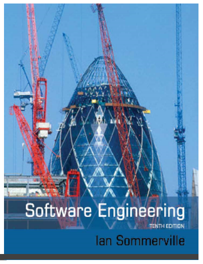
- ✧ A password checker shall be made available and shall be run daily. Weak passwords shall be reported to system administrators.
- ✧ Access to the system shall only be allowed by approved client computers.
- ✧ All client computers shall have a single, approved web browser installed by system administrators.

Architectural design



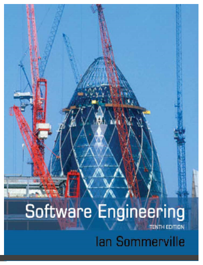
- Two fundamental issues have to be considered when designing an architecture for security.
  - Protection
    - How should the system be organised so that critical assets can be protected against external attack?
  - Distribution
    - How should system assets be distributed so that the effects of a successful attack are minimized?
- These are potentially conflicting
  - If assets are distributed, then they are more expensive to protect. If assets are protected, then usability and performance requirements may be compromised.

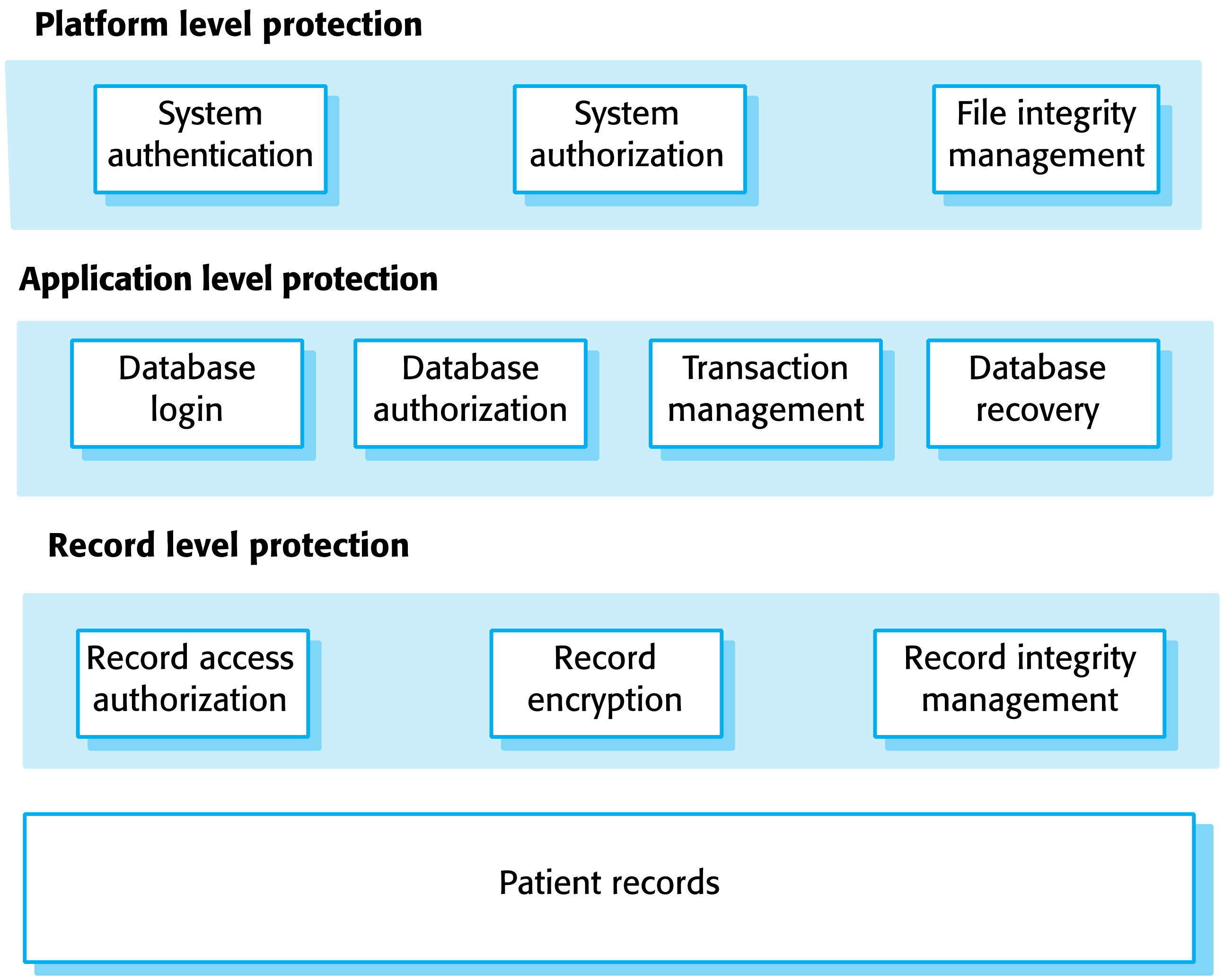
Protection



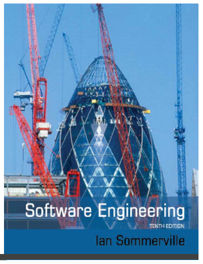
- ✧ Platform-level protection
  - Top-level controls on the platform on which a system runs.
- ✧ Application-level protection
  - Specific protection mechanisms built into the application itself e.g. additional password protection.
- ✧ Record-level protection
  - Protection that is invoked when access to specific information is requested
- ✧ These lead to a layered protection architecture

A layered protection architecture

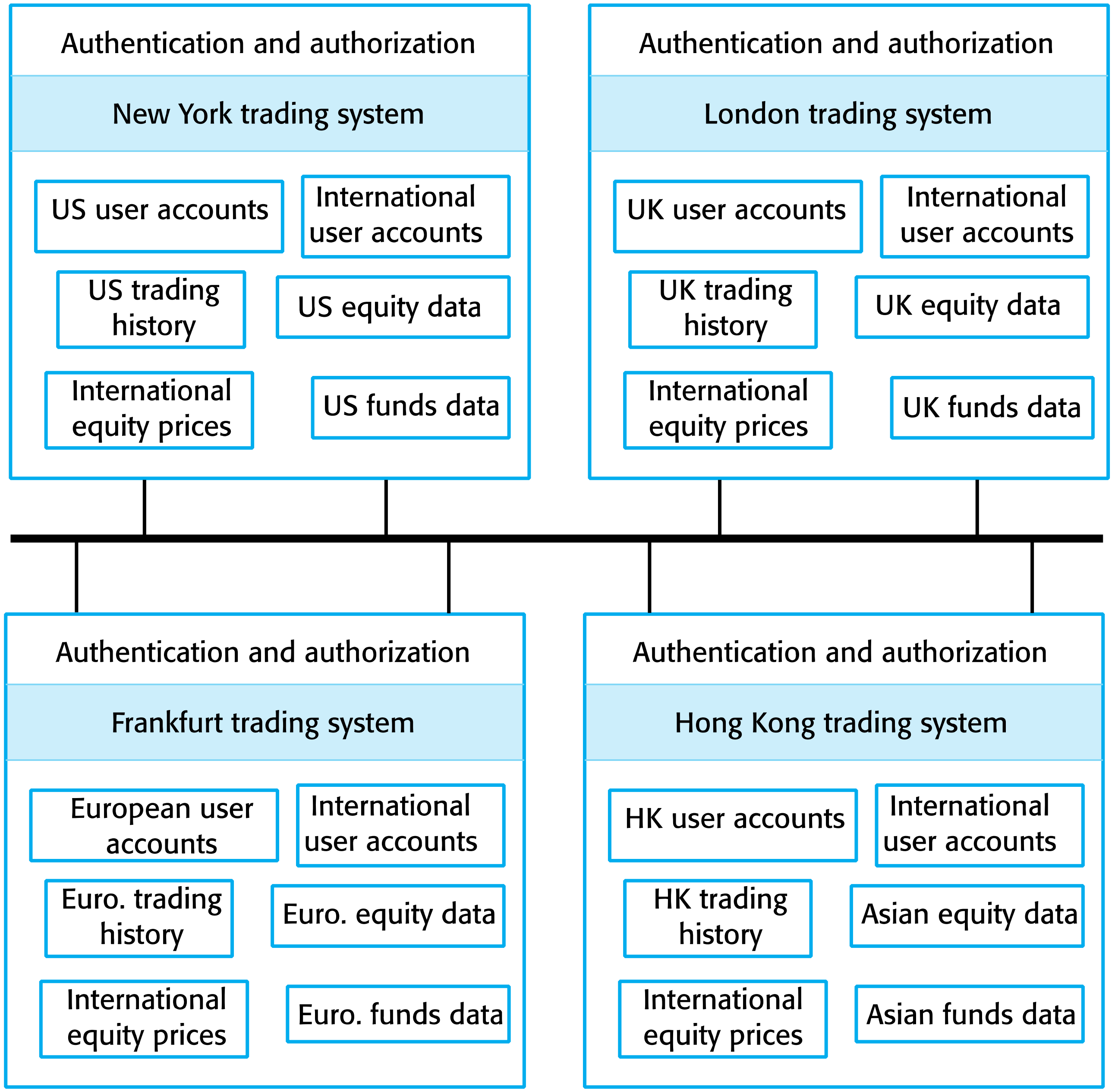


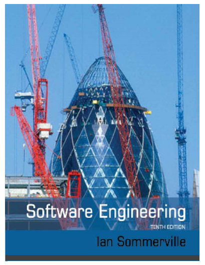


Distribution



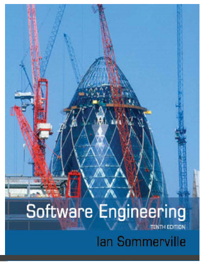
- ✧ Distributing assets means that attacks on one system do not necessarily lead to complete loss of system service
- ✧ Each platform has separate protection features and may be different from other platforms so that they do not share a common vulnerability
- ✧ Distribution is particularly important if the risk of denial of service attacks is high





Distributed assets in an equity trading system

Design guidelines for security engineering



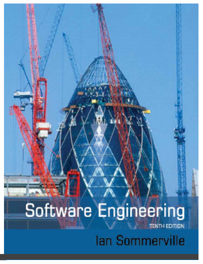
✧ Design guidelines encapsulate good practice in secure systems design

✧ Design guidelines serve two purposes:

- They raise awareness of security issues in a software engineering team. Security is considered when design decisions are made.
- They can be used as the basis of a review checklist that is applied during the system validation process.

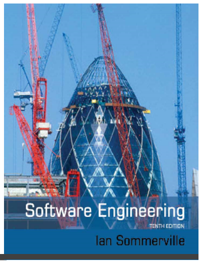
✧ Design guidelines here are applicable during software specification and design

Design guidelines for secure systems engineering



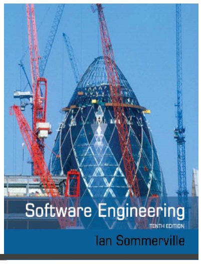
Security guidelines	
Base security decisions on an explicit security policy	
Avoid a single point of failure	
Fail securely	
Balance security and usability	
Log user actions	
Use redundancy and diversity to reduce risk	
Specify the format of all system inputs	
Compartmentalize your assets	
Design for deployment	
Design for recoverability	

Design guidelines 1-3



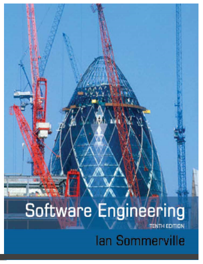
- ✧ Base decisions on an explicit security policy
  - Define a security policy for the organization that sets out the fundamental security requirements that should apply to all organizational systems.
- ✧ Avoid a single point of failure
  - Ensure that a security failure can only result when there is more than one failure in security procedures. For example, have password and question-based authentication.
- ✧ Fail securely
  - When systems fail, for whatever reason, ensure that sensitive information cannot be accessed by unauthorized users even although normal security procedures are unavailable.

Design guidelines 4-6



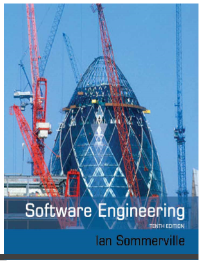
- ✧ Balance security and usability
  - Try to avoid security procedures that make the system difficult to use. Sometimes you have to accept weaker security to make the system more usable.
- ✧ Log user actions
  - Maintain a log of user actions that can be analyzed to discover who did what. If users know about such a log, they are less likely to behave in an irresponsible way.
- ✧ Use redundancy and diversity to reduce risk
  - Keep multiple copies of data and use diverse infrastructure so that an infrastructure vulnerability cannot be the single point of failure.

Design guidelines 7-10

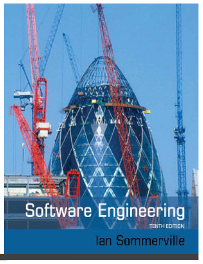


- ✧ Specify the format of all system inputs
  - If input formats are known then you can check that all inputs are within range so that unexpected inputs don't cause problems.
- ✧ Compartmentalize your assets
  - Organize the system so that assets are in separate areas and users only have access to the information that they need rather than all system information.
- ✧ Design for deployment
  - Design the system to avoid deployment problems
- ✧ Design for recoverability
  - Design the system to simplify recoverability after a successful attack.

Secure systems programming

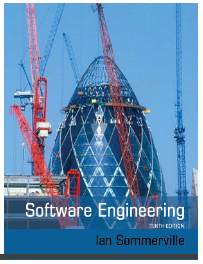


Aspects of secure systems programming



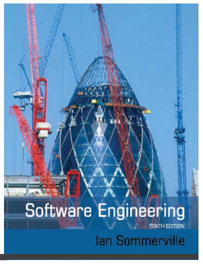
- ✧ Vulnerabilities are often language-specific.
  - Array bound checking is automatic in languages like Java so this is not a vulnerability that can be exploited in Java programs.
  - However, millions of programs are written in C and C++ as these allow for the development of more efficient software so simply avoiding the use of these languages is not a realistic option.
- ✧ Security vulnerabilities are closely related to program reliability.
  - Programs without array bound checking can crash so actions taken to improve program reliability can also improve system security.

Dependable programming guidelines



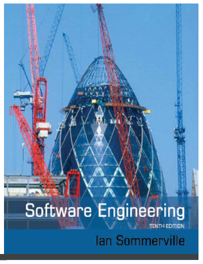
Dependable programming guidelines

1. Limit the visibility of information in a program
2. Check all inputs for validity
3. Provide a handler for all exceptions
4. Minimize the use of error-prone constructs
5. Provide restart capabilities
6. Check array bounds
7. Include timeouts when calling external components
8. Name all constants that represent real-world values



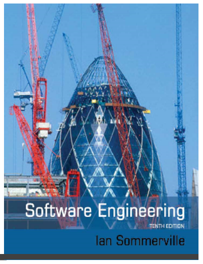
Security testing and assurance

Security testing



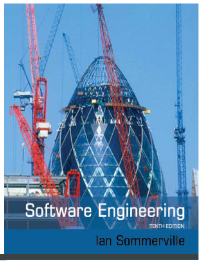
- Testing the extent to which the system can protect itself from external attacks.
- Problems with security testing
  - Security requirements are ‘shall not’ requirements i.e. they specify what should not happen. It is not usually possible to define security requirements as simple constraints that can be checked by the system.
  - The people attacking a system are intelligent and look for vulnerabilities. They can experiment to discover weaknesses and loopholes in the system.

Security validation



- ✧ Experience-based testing
  - The system is reviewed and analysed against the types of attack that are known to the validation team.
- ✧ Penetration testing
  - A team is established whose goal is to breach the security of the system by simulating attacks on the system.
- ✧ Tool-based analysis
  - Various security tools such as password checkers are used to analyse the system in operation.
- ✧ Formal verification
  - The system is verified against a formal security specification.

Examples of entries in a security checklist



Security checklist

1. Do all files that are created in the application have appropriate access permissions? The wrong access permissions may lead to these files being accessed by unauthorized users.

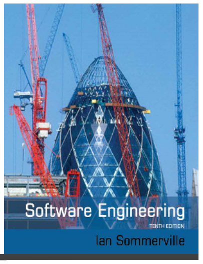
2. Does the system automatically terminate user sessions after a period of inactivity? Sessions that are left active may allow unauthorized access through an unattended computer.

3. If the system is written in a programming language without array bound checking, are there situations where buffer overflow may be exploited? Buffer overflow may allow attackers to send code strings to the system and then execute them.

4. If passwords are set, does the system check that passwords are 'strong'? Strong passwords consist of mixed letters, numbers, and punctuation, and are not normal dictionary entries. They are more difficult to break than simple passwords.

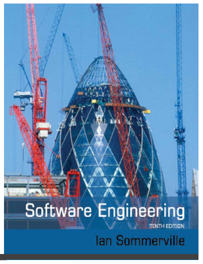
5. Are inputs from the system's environment always checked against an input specification? Incorrect processing of badly formed inputs is a common cause of security vulnerabilities.

Key points



- ✧ Security engineering is concerned with how to develop systems that can resist malicious attacks
- ✧ Security threats can be threats to confidentiality, integrity or availability of a system or its data
- ✧ Security risk management is concerned with assessing possible losses from attacks and deriving security requirements to minimise losses
- ✧ To specify security requirements, you should identify the assets that are to be protected and define how security techniques and technology should be used to protect these assets.

Key points



✧ Key issues when designing a secure systems architecture include organizing the system structure to protect key assets and distributing the system assets to minimize the losses from a successful attack.

✧ Security design guidelines sensitize system designers to security issues that they may not have considered. They provide a basis for creating security review checklists.

✧ Security validation is difficult because security requirements state what should not happen in a system, rather than what should. Furthermore, system attackers are intelligent and may have more time to probe for weaknesses than is available for security testing.