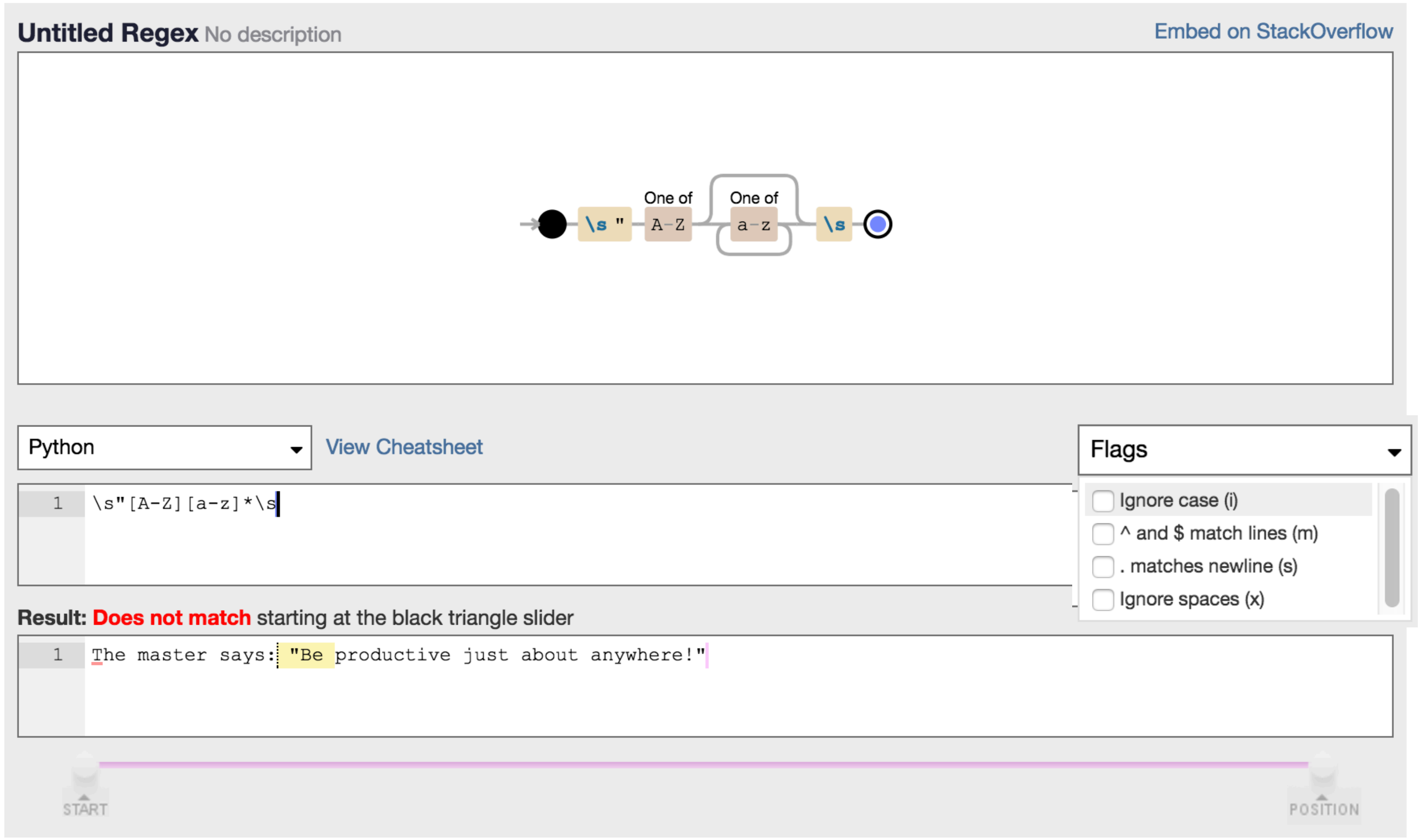
SSW-540: Fundamentals of Software Engineering

Software Safety, Security and Resilience

Dr. Richard Ens
School of Systems and Enterprises

More on Regular Expressions

- Regular expressions can get quite *gnarly*. This web site is really helpful in debugging regular expressions:
- https://www.debuggex.com/?flavor=python



```
\s"[A-Z][a-z]*\s
```

- [ ] Ignore case (i)
- [ ] ^ and $ match lines (m)
- [ ] . matches newline (s)
- [ ] Ignore spaces (x)

Result: **Does not match** starting at the black triangle slider

```
The master says: "Be productive just about anywhere!"
```

Safety

- Safety is a property of a system that reflects the system’s ability to operate, normally or abnormally, <u>without danger of causing human injury or death and without damage to the system’s environment</u>.
- Safety-critical systems are systems whose failure can lead to injuries or deaths.
- Software is of concern for safety-critical systems as most now incorporate software-based control systems.
    - Software control may be implemented so the software behavior is directly related to the overall safety of the system.
    - Software is extensively used for checking and monitoring safety-critical components.
        - Example: software monitoring aircraft engine components for early indications of component failure. If the software fails, component failure may 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
  - 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.
  - If the system specification has errors, then the system can behave as specified but still cause an accident.
  - Hardware failures can generate spurious inputs that are 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 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.
Hazard	A condition with the potential for causing or contributing to an accident.
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.
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).
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.

Safety achievement strategies

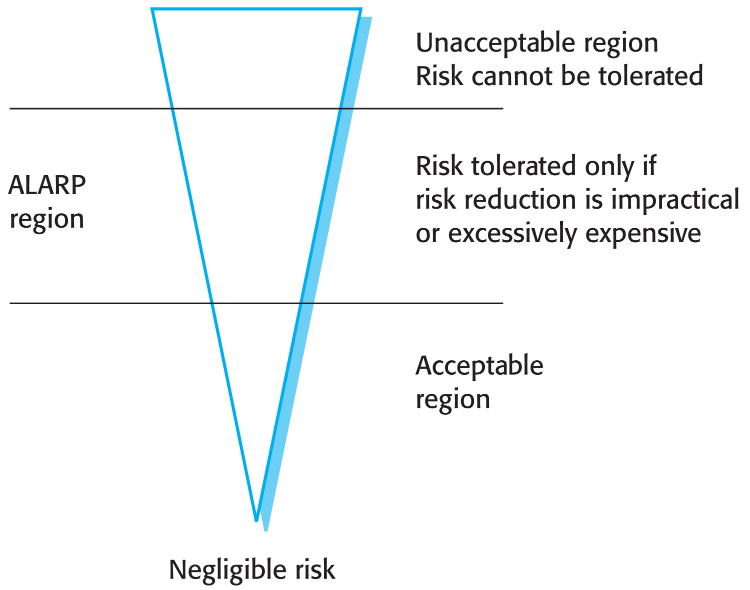
- Hazard avoidance
  - The system is designed so that some classes of hazard simply cannot arise.
- Hazard detection and removal
  - The system is designed so that hazards are detected and removed before they result in an accident.
- Damage limitation
  - The system includes protection features that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

Safety requirements

- 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.
    - A hazard-driven approach is used, looking for hazard root-cause.
    - Requirements to avoid or recover from these hazards are then written.
- 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
- Non-functional safety requirements specify how the software should implemented to, say, avoid single points of failure
- Safety requirements may be 'shall not' requirements i.e. they define situations and events that should never occur.

Safety engineering processes

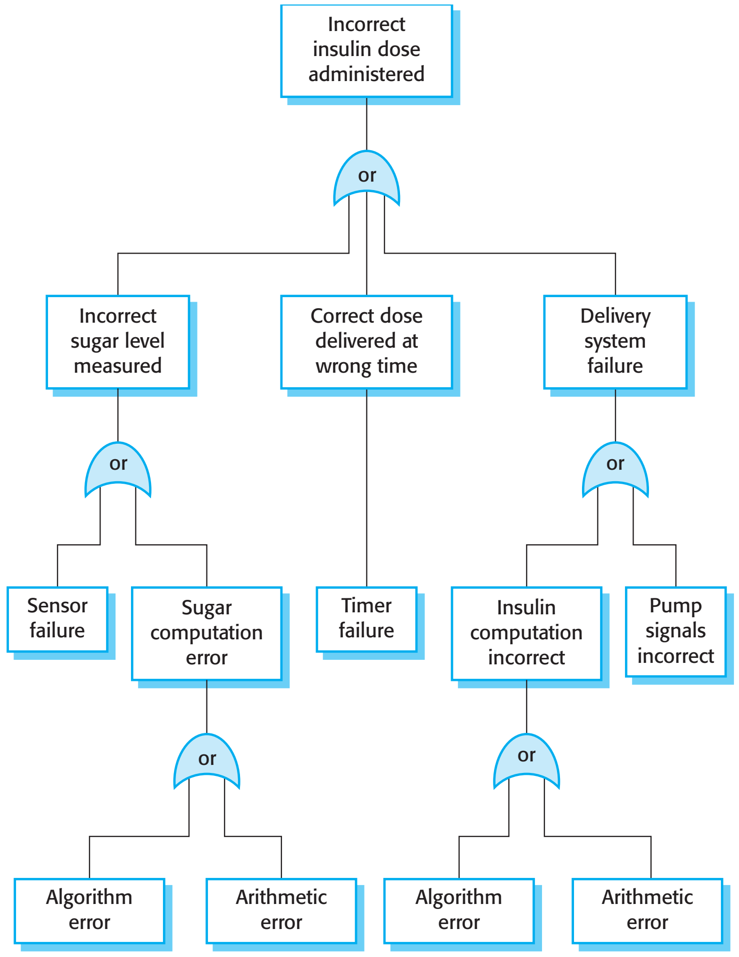
- A well-defined, certified process for software development of safety-critical system is needed.
- The process should include the identification and monitoring of potential hazards.
- Hazard assessment is concerned with probability and consequences: risk



- Risk assessment is subjective.
- The risk triangle at left points out that 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 - society becomes less willing to accept risk.

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) is often used to minimize the number of single causes of failure.



Software fault tree for the insulin pump

Safety engineering processes

- 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
- Regulators may require evidence that safety engineering processes have been used in system development
- Some agile techniques such as test-driven development may be used, but agile methods are not usually used for safety-critical systems engineering
  - Extensive process and product documentation, needed for system regulation, contradicts the focus in agile methods on the software itself.
  - A detailed safety analysis of a complete system specification contradicts agile interleaved development of a system specification and program.

Processes for safety assurance

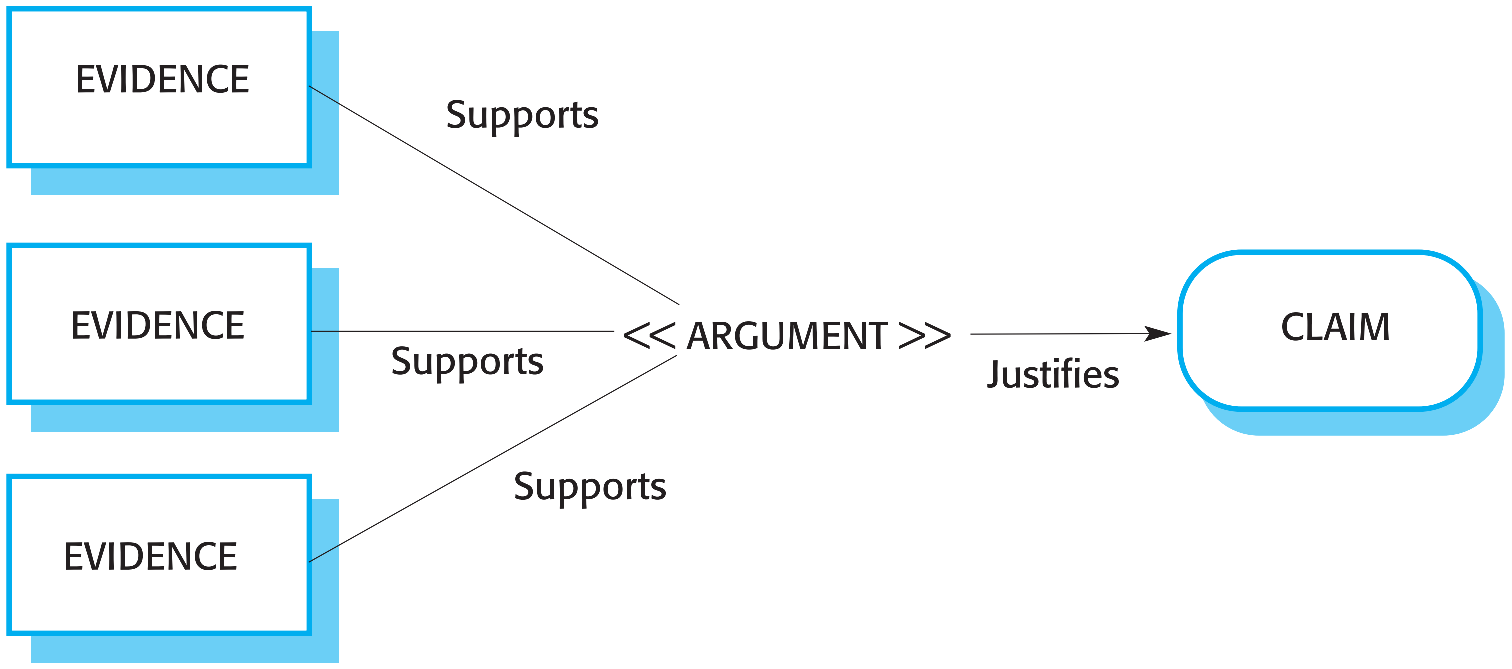
- Process assurance is important for safety-critical systems development:
  - Accidents are rare events so testing may not find all problems;
  - Safety requirements are sometimes ‘shall not’ requirements so cannot be demonstrated through testing.
- Requirement, architecture, design and code reviews will help
- Safety assurance activities that record the analyses that have been carried out and the people responsible for these should be included.
  - Personal responsibility is important as system failures may lead to subsequent legal actions.
- Formal methods, the ultimate static verification technique, can be used when a mathematical specification of the system is produced.
- Model checking, exploring all possible paths, can occur when an extended finite state model of the system has been produced.

Safety 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.
- 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.

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.



Software safety arguments

- Safety arguments are intended to show that the system cannot reach an 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.
- To construct 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.

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 an essential prerequisite for availability, reliability and safety.
  - 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.

“The three golden rules to ensure computer security are:

Do not own a computer; Do not power it on; And do not use it.”

---Robert Morris, Bell Labs and NSA

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.

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.
- E.g., the theft of SSNs

Integrity

- Information in a system may be damaged or corrupted making it unusual or unreliable.
- E.g., database hacks

Availability

- Access to a system or its data that is normally available may not be possible.
- E.g., denial of service attacks

Threat types

- Interception threats that allow an attacker to gain access to an asset.
  - E.g., an attacker gains access to the records of a celebrity patient.
- Interruption threats that allow an attacker to make part of the system unavailable.
  - E.g., a denial of service attack on a system database server so that database connections become impossible.
- Modification threats that allow an attacker to tamper with a system asset.
  - E.g., an attacker alters or destroys a database record.
- Fabrication threats that allow an attacker to insert false information into a system.
  - E.g., in a banking system, where false transactions might be added to the system that transfer money to the perpetrator's bank account.

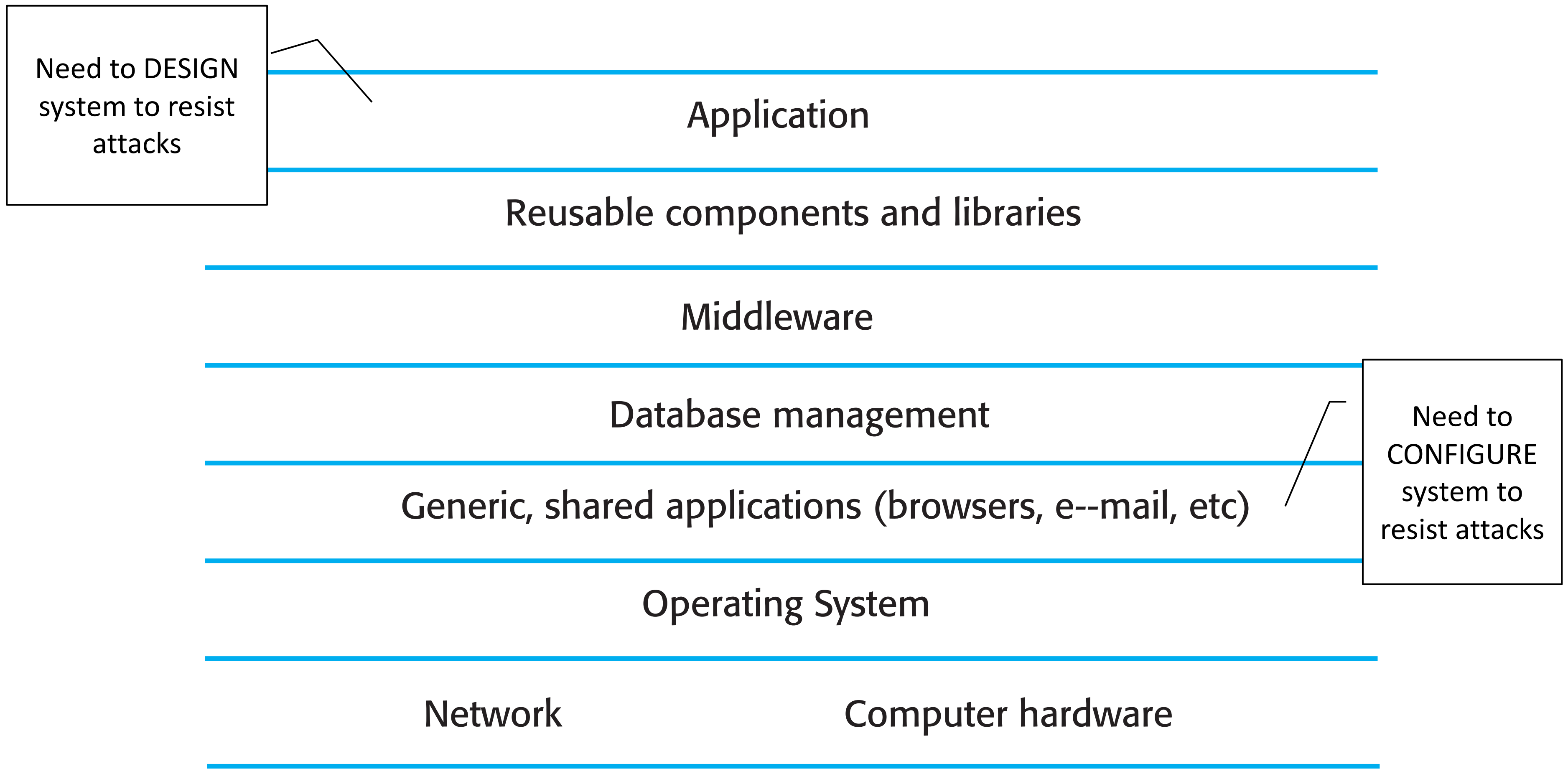
Security assurance

- 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

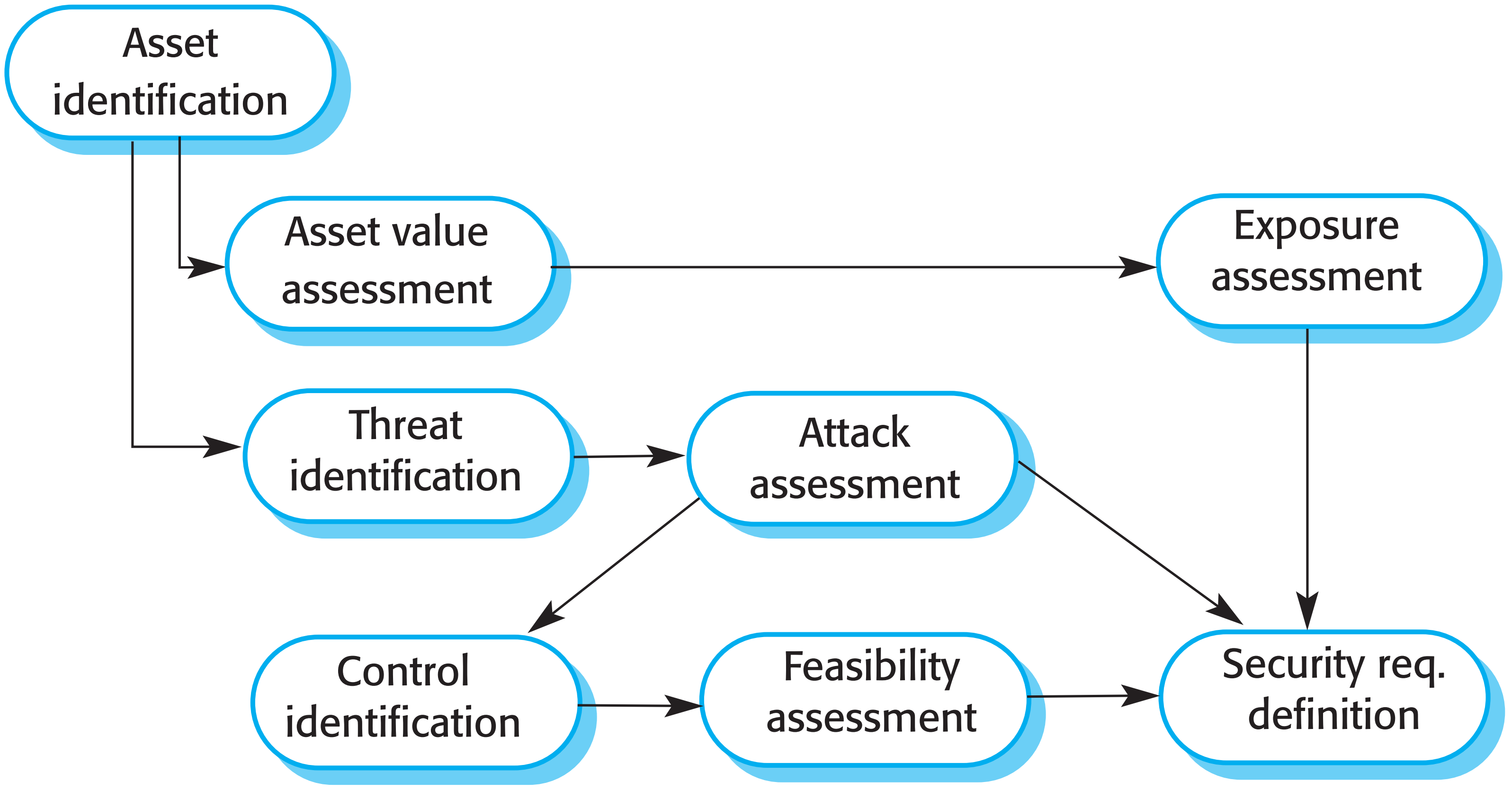
Security engineering

- Engineering for security is harder than engineering for safety!
  - 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.
- Security and safety are related: safety measures can be attacked.

System layers where security may be compromised


Need to DESIGN system to resist attacks
Application
Reusable components and libraries
Middleware
Database management
Generic, shared applications (browsers, e--mail, etc)
Need to CONFIGURE system to resist attacks
Operating System
Network
Computer hardware

Preliminary risk assessment process for security requirements

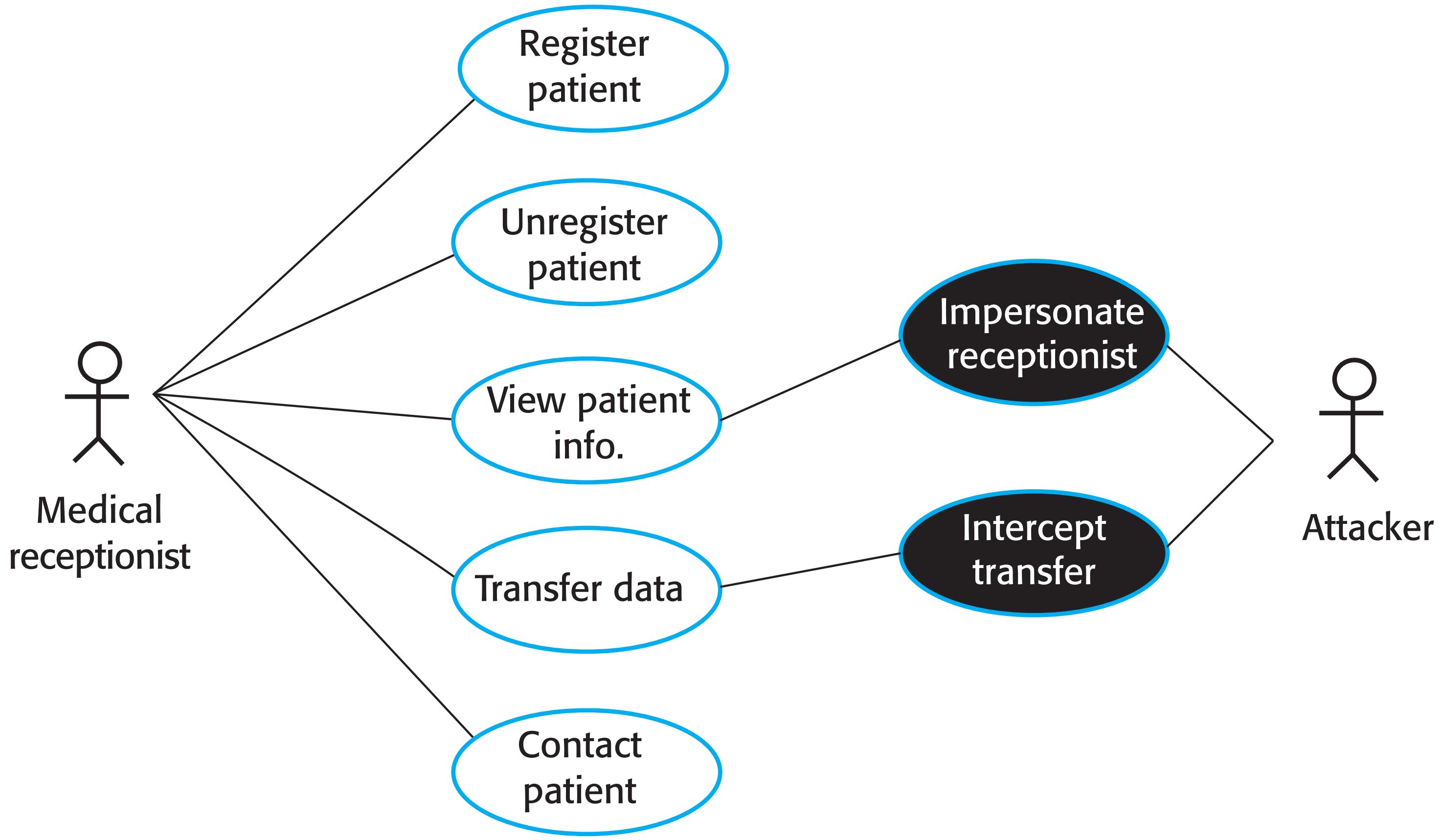


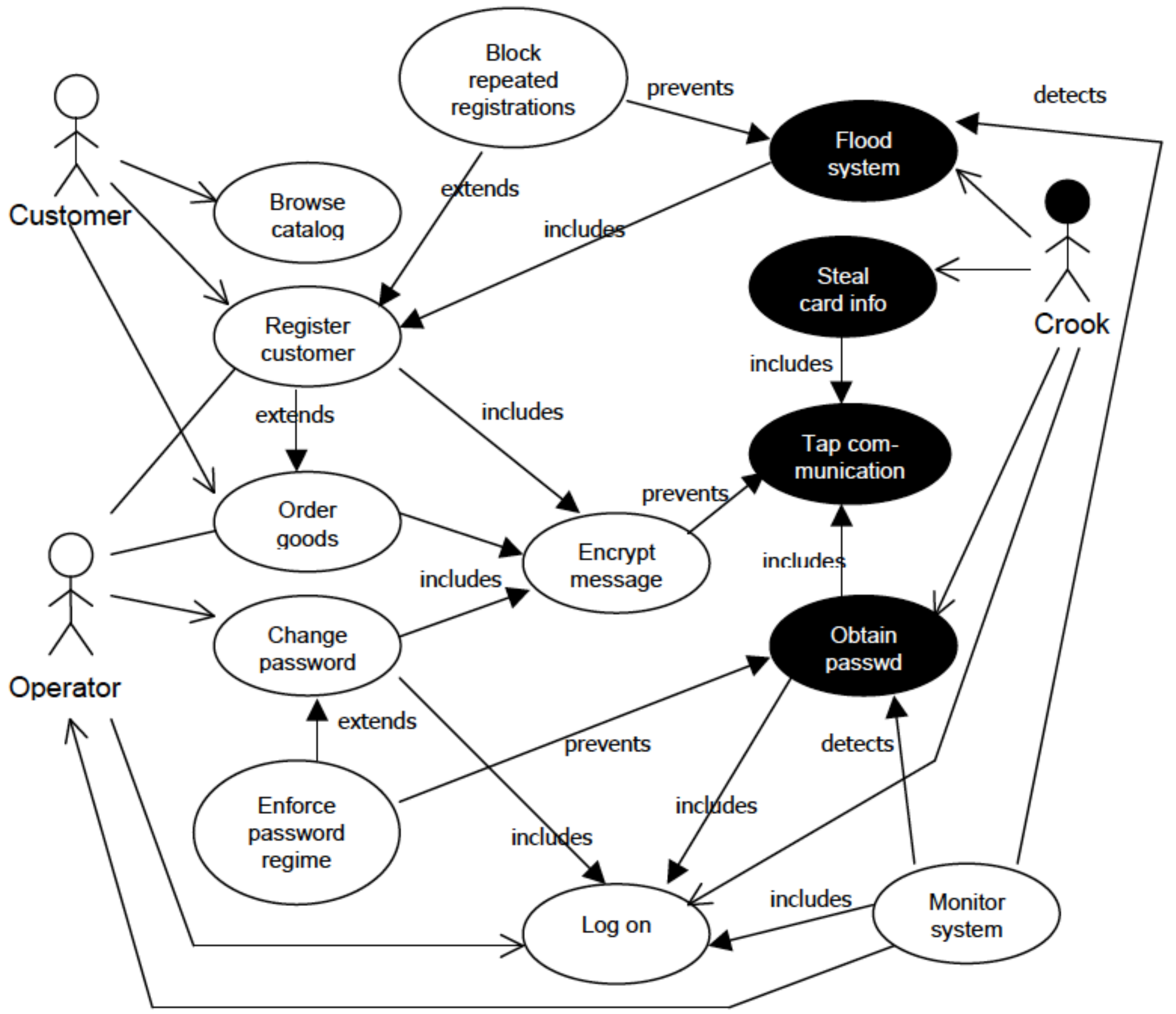
Many types of security requirements

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

Misuse cases

- Misuse cases are instances of threats to a system.




Customer
Operator
Crook
Browse catalog
Register customer
Order goods
Change password
Enforce password regime
Log on
Block repeated registrations
Encrypt message
Monitor system
Flood system
Steal card info
Tap com-munication
Obtain passwd
prevents
detects
extends
includes
includes
extends
includes
prevents
includes
includes
includes
extends
prevents
detects
includes
includes
includes

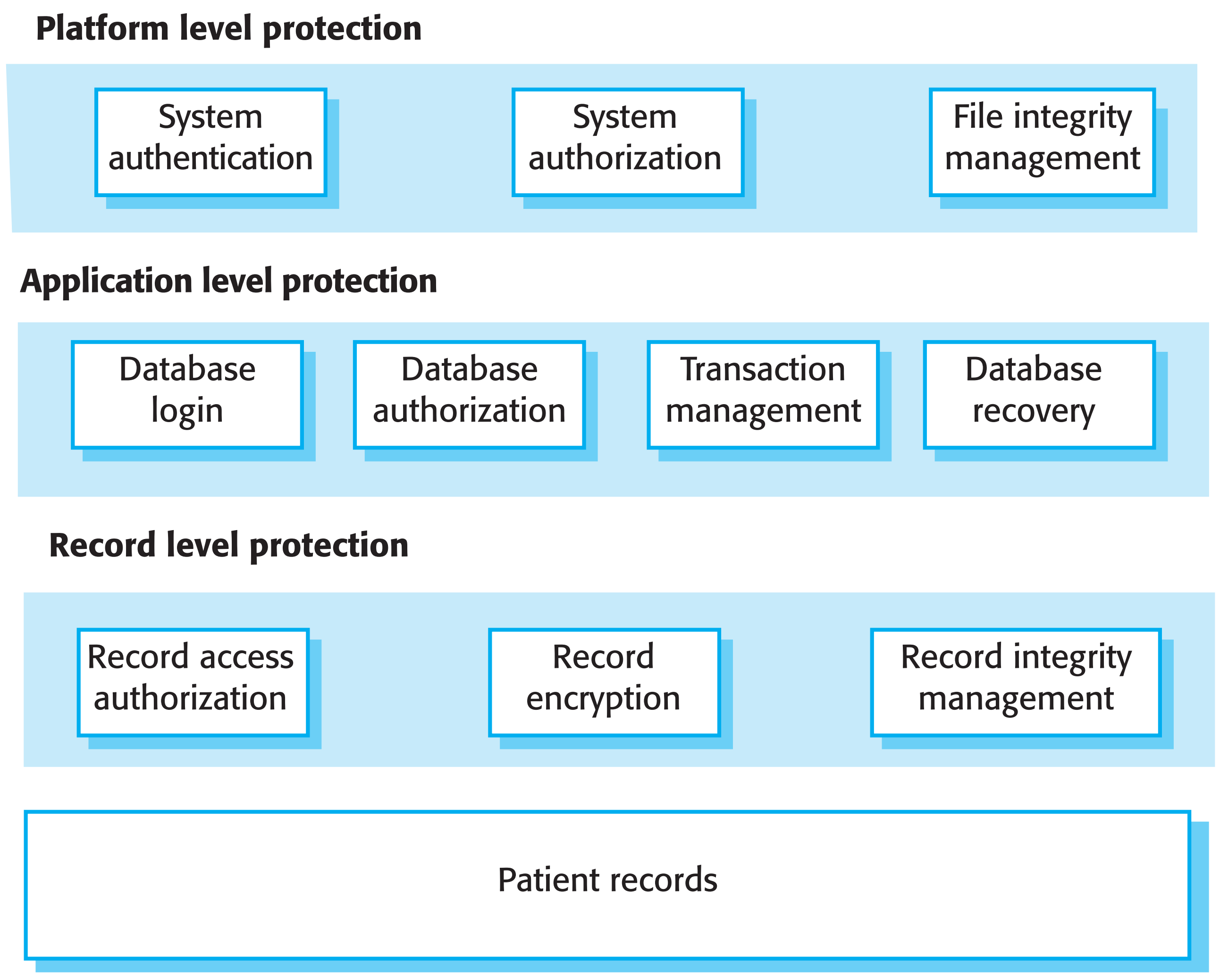
Template for a Misuse Case

- Actors – Attacker(s) plus the actors of the use case
- Description – Describing the use case and the misuse that occurs
- Data (assets) – What is threatened by the misuse
- Attacks – How the attack(s) is(are) carried out
- Mitigations – What can be done to keep the attack from happening or reduce its impact
- Requirements – The requirements that the mitigations call for

Secure systems design

- Security has to be designed into a product—add-on features usually are not effective.
- Security features are not free; they require compromises with other qualities, e.g.,
  - secure login procedures impact usability (or even security when users set guessable passwords)
  - additional layers of security impact performance
- Protection can be offered through distributed copies of encrypted vital data; but if assets are distributed, they are more expensive to protect.
- Technology choices matter (see Sommerville's example of COTS technologies)
- Layering protection (platform, application, record-level) to good design.

A layered protection architecture



Design guidelines for security engineering

- The guidelines raise awareness of security engineering among developers and can be used as a checklist during system validation:
  - Base security decisions on an explicit security policy
  - Avoid a single point of security failure
  - Fail securely
  - Balance security and usability
  - Log user actions
  - Use redundancy (e.g., data) and diversity (in infrastructure) to reduce risk
  - Specify the format of all system inputs ("bound the inputs")
  - Compartmentalize your assets
  - Design for deployment
  - Design for recoverability

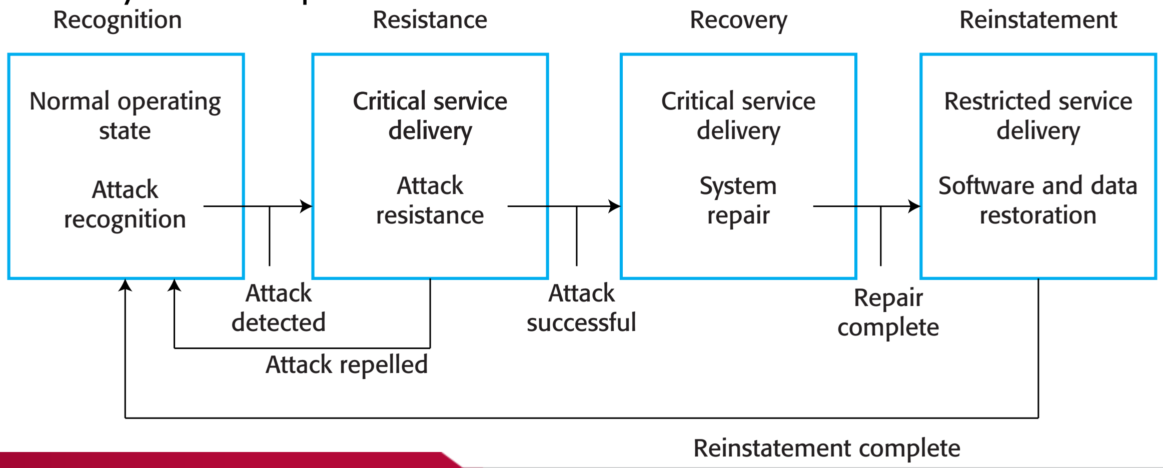
Engineering Software Resilience

The resilience of a system is a judgment of how well that system can maintain the continuity of its critical services in the presence of disruptive events, such as equipment failure and cyberattacks.

- Cyberattacks by malicious outsiders are perhaps the most serious threat faced by networked systems but resilience is also intended to cope with system failures and other disruptive events.
  - Resilience engineering assumes that it is impossible to avoid system failures and so is concerned with limiting the costs of these failures and recovering from them.
  - Resilience engineering assumes that good reliability engineering practices have been used to minimize the number of technical faults in a system.
  - It therefore places more emphasis on limiting the number of system failures that arise from external events such as operator errors or cyberattacks.

Primary resilience activities

- *Recognition* - The system or its operators should recognize early indications of system failure.
- *Resistance* - If the symptoms of a problem or cyberattack are detected early, then resistance strategies may be used to reduce the probability that the system will fail.
- *Recovery* - If a failure occurs, the recovery activity ensures that critical system services are restored quickly so that system users are not badly affected by failure.
- *Reinstatement* - In this final activity, all of the system services are restored and normal system operation can continue.

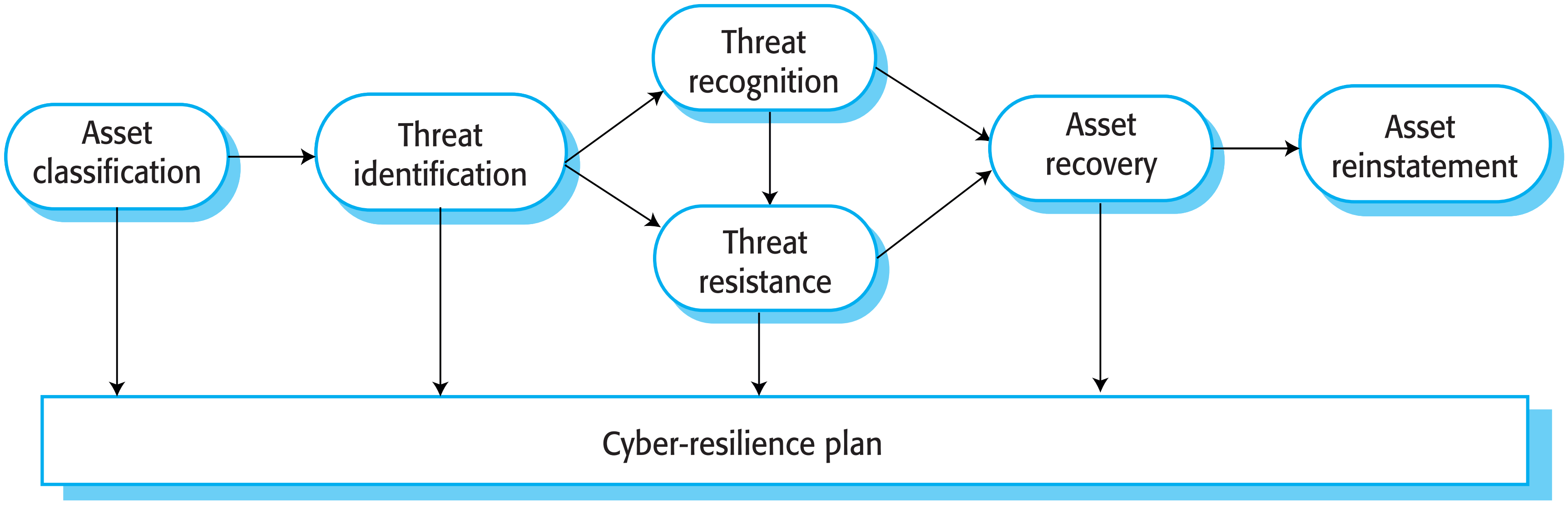


Cybersecurity

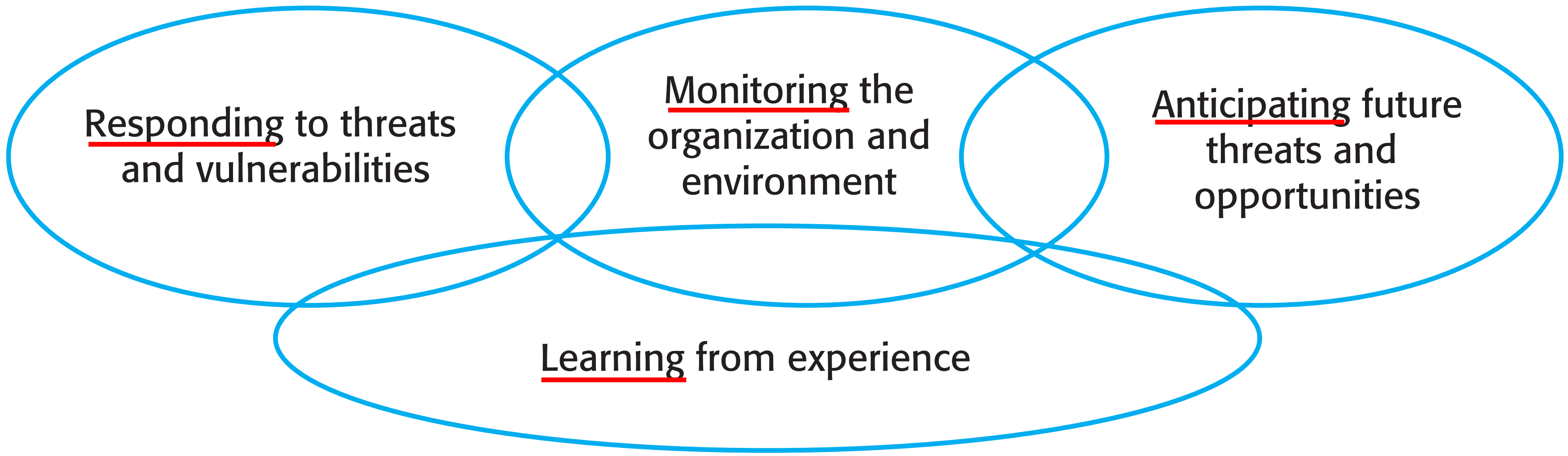
- Cybercrime is the illegal use of networked systems and is one of the most serious problems facing our society.
- Cybersecurity is a broader topic than system security engineering
  - Safety and dependability are threatened too
- Factors contributing to cybersecurity failure
  - organizational ignorance of the seriousness of the problem,
  - poor design and lax application of security procedures,
  - human carelessness,
  - inappropriate trade-offs between usability and security.

Cybersecurity threats

- *Threats to the confidentiality of assets* - Data is not damaged but it is made available to people who should not have access to it.
- *Threats to the integrity of assets* - These are threats where systems or data are damaged in some way by a cyberattack.
- *Threats to the availability of assets* - These are threats that aim to deny use of assets by authorized users.



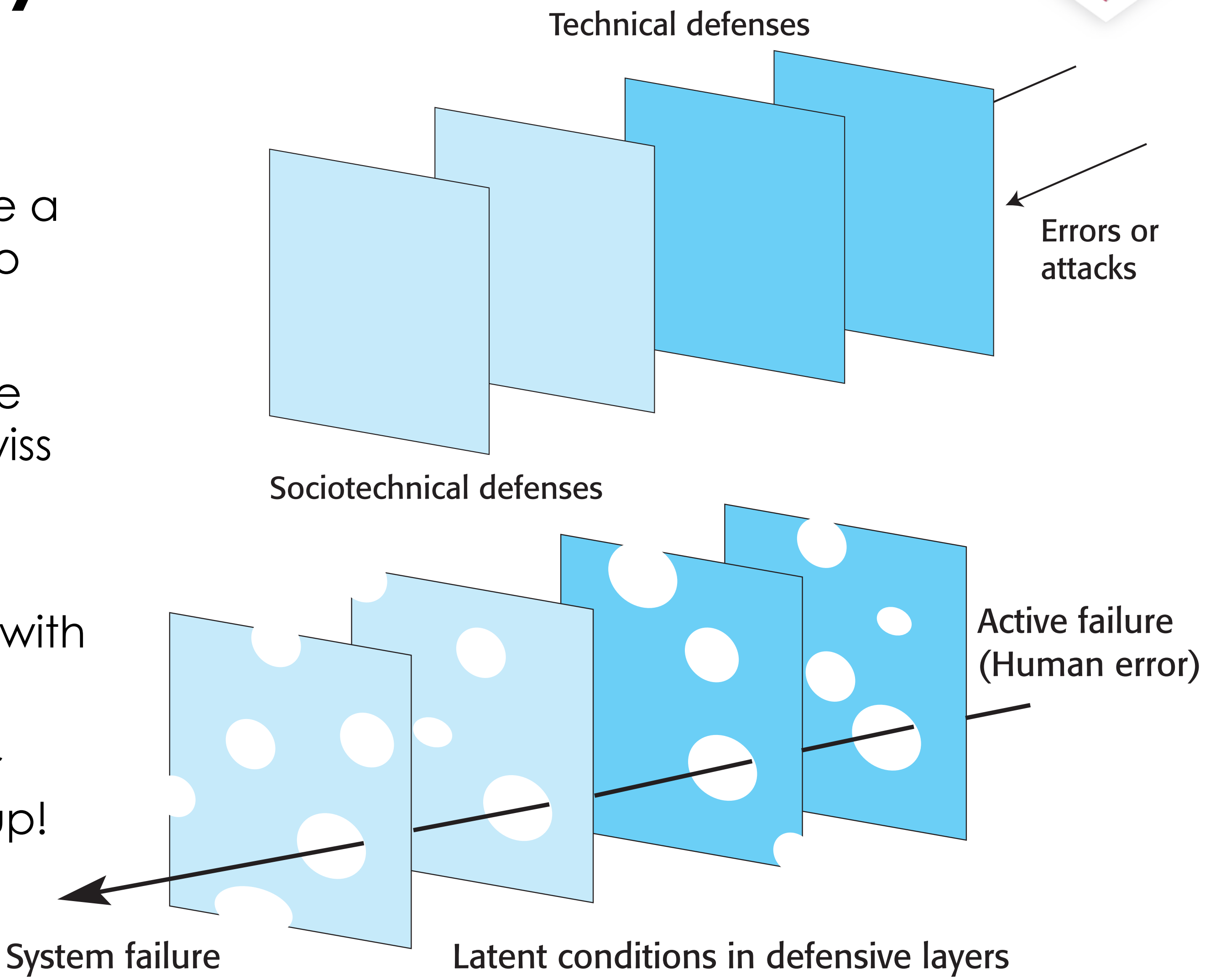
Resilience extends beyond software


Responding to threats and vulnerabilities
Monitoring the organization and environment
Anticipating future threats and opportunities
Learning from experience

Resilient organizations respond, monitor, anticipate and learn!

Defensive layers

- Each layer should use a different approach to avoid failure.
- Defensive layers have vulnerabilities (like Swiss cheese with holes)
- Vulnerabilities are dynamic—they vary with operating conditions
- System failures occur when the holes line up!

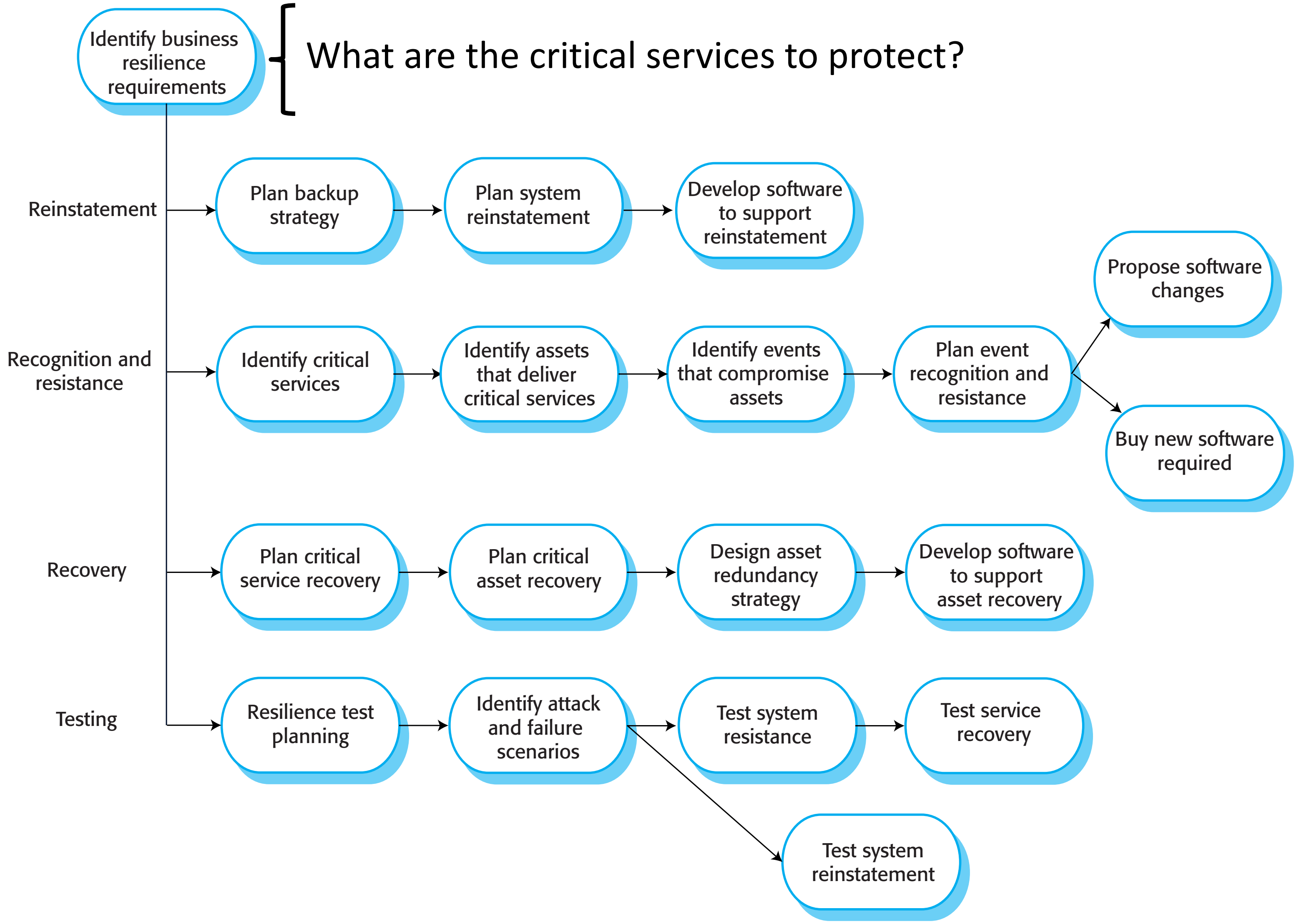


Increasing system resilience

- Reduce the probability of the occurrence of an external event that might trigger system failures.
- Increase the number of defensive layers.
  - The more layers that you have in a system, the less likely it is that the holes will line up and a system failure occurs.
- Design a system so that diverse types of barriers are included.
  - The ‘holes’ will probably be in different places and so there is less chance of the holes lining up and failing to trap an error.
- Minimize the number of latent conditions in a system.
  - This means reducing the number and size of system ‘holes’.
- Flexible operational and management processes are important defense mechanisms and, in designing a process, you need to find a balance between efficient operation and problem management.

Resilience engineering

What are the critical services to protect?


Identify business resilience requirements
Reinstatement
Plan backup strategy
Plan system reinstatement
Develop software to support reinstatement
Propose software changes
Recognition and resistance
Identify critical services
Identify assets that deliver critical services
Identify events that compromise assets
Plan event recognition and resistance
Buy new software required
Recovery
Plan critical service recovery
Plan critical asset recovery
Design asset redundancy strategy
Develop software to support asset recovery
Testing
Resilience test planning
Identify attack and failure scenarios
Test system resistance
Test service recovery
Test system reinstatement

Classroom Activity – within your EMSS project teams

Activity:

Document the dependability requirements of the EMSS associated with safety and security, including resilience. Identify at least 2 “misuse” cases that your safety and security requirements will address.

Submission to complete the Canvas Assignment EMSS Week 11:

Submit the misuse case diagram, your dependability requirements for safety and security and your plans for fulfilling them.