Chapter 11 – Dependability

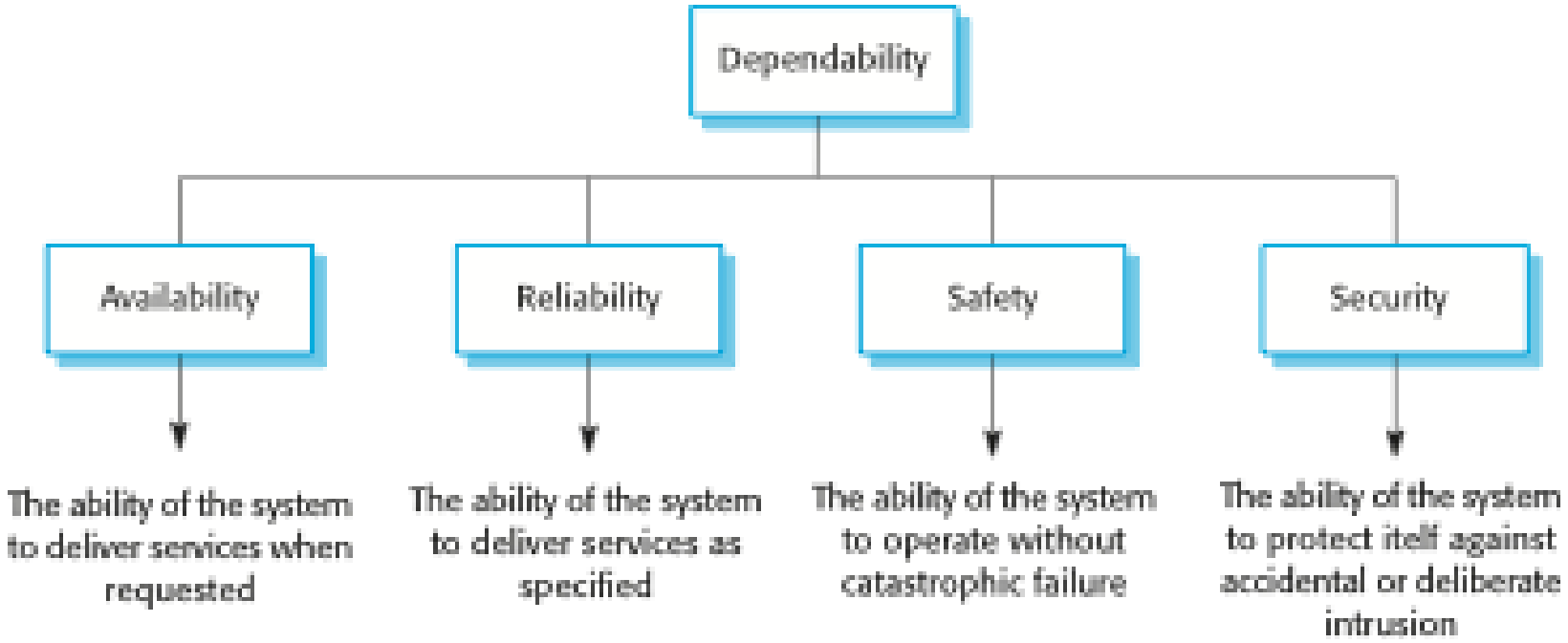
Topics covered

- Dependability properties
  - The system attributes that lead to dependability.
- Availability and reliability
  - Systems should be available to deliver service and perform as expected.
- Safety
  - Systems should not behave in an unsafe way.

System dependability

- For many computer-based systems, the most important system property is the dependability of the system.
- The dependability of a system reflects the user's degree of trust in that system. It reflects the extent of the user's confidence that it will operate as users expect and that it will not 'fail' in normal use.
- Dependability covers the related systems attributes of reliability, availability and security. These are all inter-dependent.

Principal dependability properties



Principal properties

- Availability
  - The probability that the system will be up and running and able to deliver useful services to users.
- Reliability
  - The probability that the system will correctly deliver services as expected by users.
- Safety
  - A judgment of how likely it is that the system will cause damage to people or its environment.
- Security
  - A judgment of how likely it is that the system can resist accidental or deliberate intrusions.

Availability and reliability

- Reliability
  - The probability of failure-free system operation over a specified time in a given environment for a given purpose
- Availability
  - The probability that a system, at a point in time, will be operational and able to deliver the requested services
- Both of these attributes can be expressed quantitatively e.g. availability of 0.999 means that the system is up and running for 99.9% of the time.

Availability and reliability

- It is sometimes possible to subsume system availability under system reliability
  - Obviously if a system is unavailable it is not delivering the specified system services.
- However, it is possible to have systems with low reliability that must be available.
  - So long as system failures can be repaired quickly and does not damage data, some system failures may not be a problem.
- Availability is therefore best considered as a separate attribute reflecting whether or not the system can deliver its services.
- Availability takes repair time into account, if the system has to be taken out of service to repair faults.

Perceptions of reliability

- The formal definition of reliability does not always reflect the user's perception of a system's reliability
  - The assumptions that are made about the environment where a system will be used may be incorrect
    - Usage of a system in an office environment is likely to be quite different from usage of the same system in a university environment
  - The consequences of system failures affects the perception of reliability
    - Unreliable windscreen wipers in a car may be irrelevant in a dry climate
    - Failures that have serious consequences (such as an engine breakdown in a car) are given greater weight by users than failures that are inconvenient

Availability perception

- Availability is usually expressed as a percentage of the time that the system is available to deliver services e.g. 99.95%.
- However, this does not take into account two factors:
  - The number of users affected by the service outage. Loss of service in the middle of the night is less important for many systems than loss of service during peak usage periods.
  - The length of the outage. The longer the outage, the more the disruption. Several short outages are less likely to be disruptive than 1 long outage. Long repair times are a particular problem.

Reliability terminology

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

Faults and failures

- Failures are a usually a result of system errors that are derived from faults in the system
- However, faults do not necessarily result in system errors
  - The erroneous system state resulting from the fault may be transient and ‘corrected’ before an error arises.
  - The faulty code may never be executed.
- Errors do not necessarily lead to system failures
  - The error can be corrected by built-in error detection and recovery
  - The failure can be protected against by built-in protection facilities. These may, for example, protect system resources from system errors

Reliability in use

- Removing X% of the faults in a system will not necessarily improve the reliability by X%. A study at IBM showed that removing 60% of product defects resulted in a 3% improvement in reliability.
- Program defects may be in rarely executed sections of the code so may never be encountered by users. Removing these does not affect the perceived reliability.
- Users adapt their behaviour to avoid system features that may fail for them.
- A program with known faults may therefore still be perceived as reliable by its users.

Reliability achievement

- Fault avoidance
  - Development technique are used that either minimise the possibility of mistakes or trap mistakes before they result in the introduction of system faults.
- Fault detection and removal
  - Verification and validation techniques that increase the probability of detecting and correcting errors before the system goes into service are used.
- Fault tolerance
  - Run-time techniques are used to ensure that system faults do not result in system errors and/or that system errors do not lead to system failures.

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.
- Safety requirements are often exclusive requirements i.e. they exclude undesirable situations rather than specify required system services. These generate functional safety requirements.

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 systems, which can then have safety consequences. For example, the MHC-PMS is safety-critical as failure may lead to inappropriate treatment being prescribed.

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

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 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.

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.

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.

Chapter 19 – Service-oriented Architecture

Web services

- A web service is an instance of a more general notion of a service:

  "an act or performance offered by one party to another. Although the process may be tied to a physical product, the performance is essentially intangible and does not normally result in ownership of any of the factors of production".

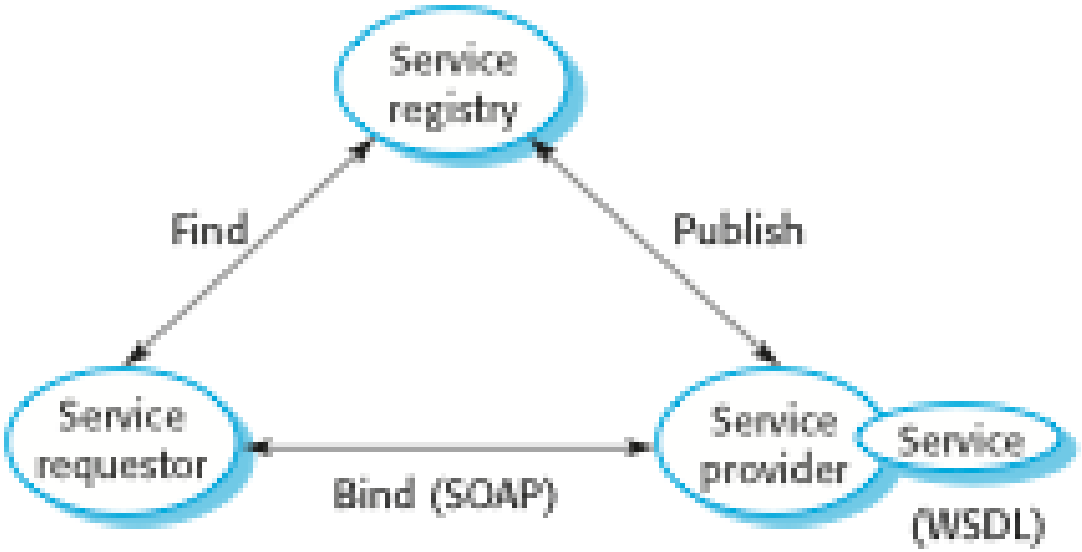
- The essence of a service, therefore, is that the provision of the service is independent of the application using the service.

- Service providers can develop specialized services and offer these to a range of service users from different organizations.

Service-oriented architectures

- A means of developing distributed systems where the components are stand-alone services
- Services may execute on different computers from different service providers
- Standard protocols have been developed to support service communication and information exchange

Service-oriented architecture

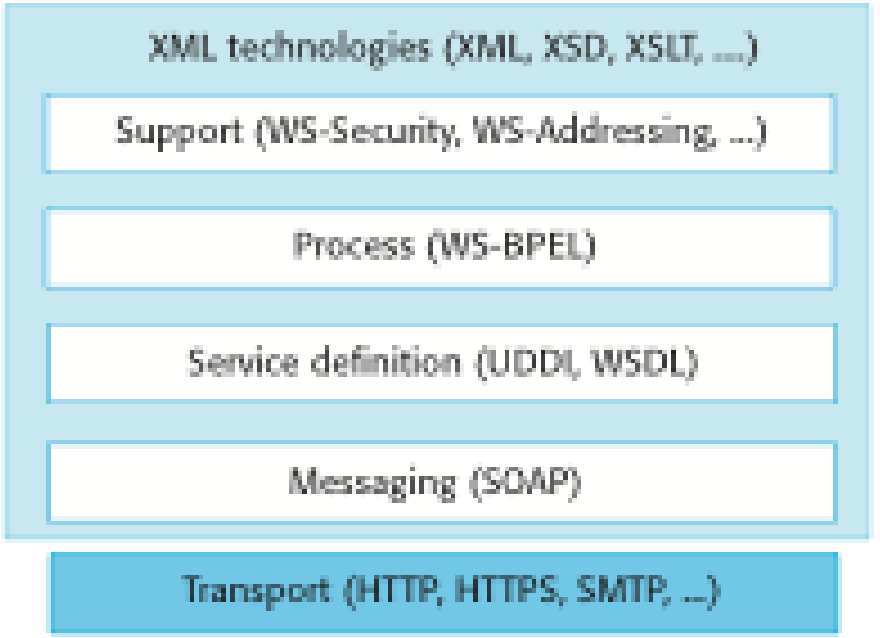


Benefits of SOA

- Services can be provided locally or outsourced to external providers
- Services are language-independent
- Investment in legacy systems can be preserved
- Inter-organisational computing is facilitated through simplified information exchange

Key standards

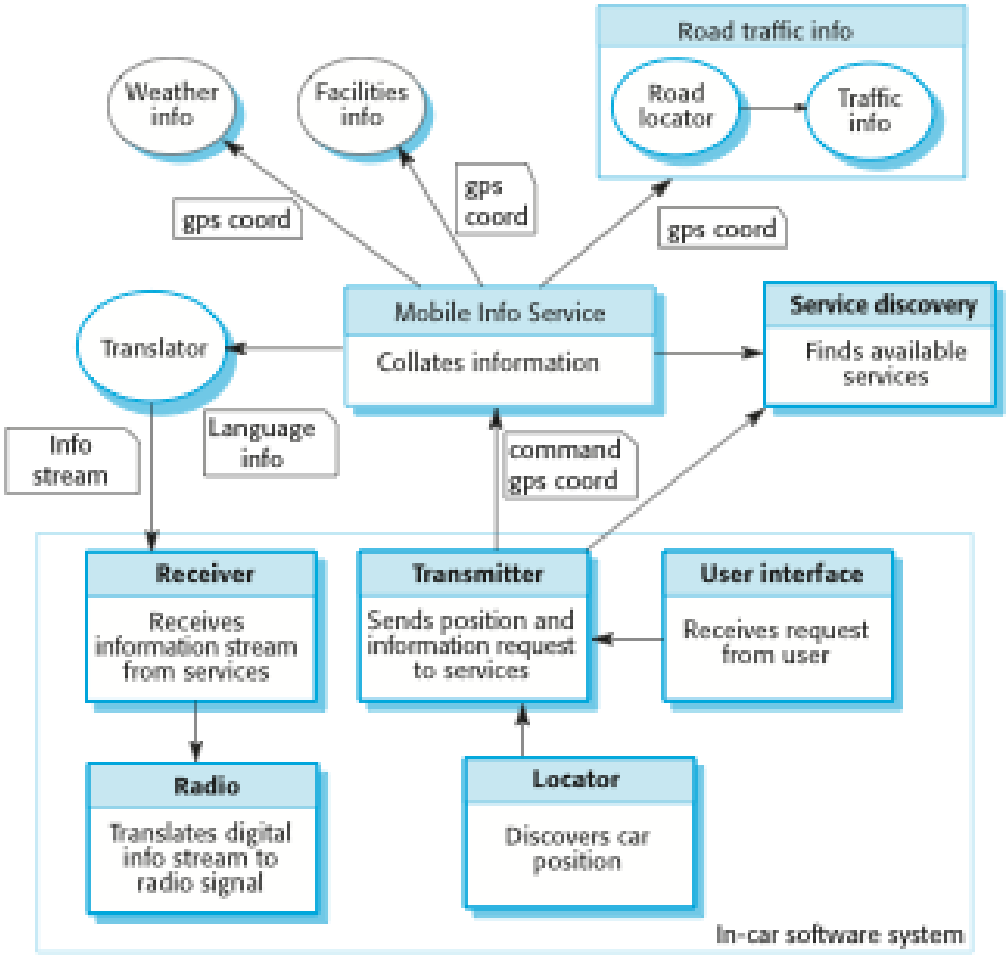
- SOAP
  - A message exchange standard that supports service communication
- WSDL (Web Service Description Language)
  - This standard allows a service interface and its bindings to be defined
- WS-BPEL
  - A standard for workflow languages used to define service composition



Services scenario

- An in-car information system provides drivers with information on weather, road traffic conditions, local information etc. This is linked to car radio so that information is delivered as a signal on a specific radio channel.
- The car is equipped with GPS receiver to discover its position and, based on that position, the system accesses a range of information services. Information may be delivered in the driver's specified language.

A service-based, in-car information system



Advantage of SOA for this application

- It is not necessary to decide when the system is programmed or deployed what service provider should be used or what specific services should be accessed.
  - As the car moves around, the in-car software uses the service discovery service to find the most appropriate information service and binds to that.
  - Because of the use of a translation service, it can move across borders and therefore make local information available to people who don't speak the local language.

Service-oriented software engineering

- Existing approaches to software engineering have to evolve to reflect the service-oriented approach to software development
  - Service engineering. The development of dependable, reusable services
    - Software development for reuse
  - Software development with services. The development of dependable software where services are the fundamental components
    - Software development with reuse

Services as reusable components

- A service can be defined as:
  - *A loosely-coupled, reusable software component that encapsulates discrete functionality which may be distributed and programmatically accessed. A web service is a service that is accessed using standard Internet and XML-based protocols*
- A critical distinction between a service and a component as defined in CBSE is that services are independent
  - Services do not have a ‘requires’ interface
  - Services rely on message-based communication with messages expressed in XML

Chapter 21 - Aspect-oriented Software Development

Aspect-oriented software development

- An approach to software development based around a relatively new type of abstraction - an aspect.
- Used in conjunction with other approaches - normally object-oriented software engineering.
- Aspects encapsulate functionality that cross-cuts and co-exists with other functionality.
- Aspects include a definition of where they should be included in a program as well as code implementing the cross-cutting concern.

The separation of concerns

- The principle of separation of concerns states that software should be organised so that each program element does one thing and one thing only.
- Each program element should therefore be understandable without reference to other elements.
- Program abstractions (subroutines, procedures, objects, etc.) support the separation of concerns.

Concerns

- Concerns are not program issues but reflect the system requirements and the priorities of the system stakeholders.
  - Examples of concerns are performance, security, specific functionality, etc.
- By reflecting the separation of concerns in a program, there is clear traceability from requirements to implementation.
- Core concerns are the functional concerns that relate to the primary purpose of a system; secondary concerns are functional concerns that reflect non-functional and QoS requirements.

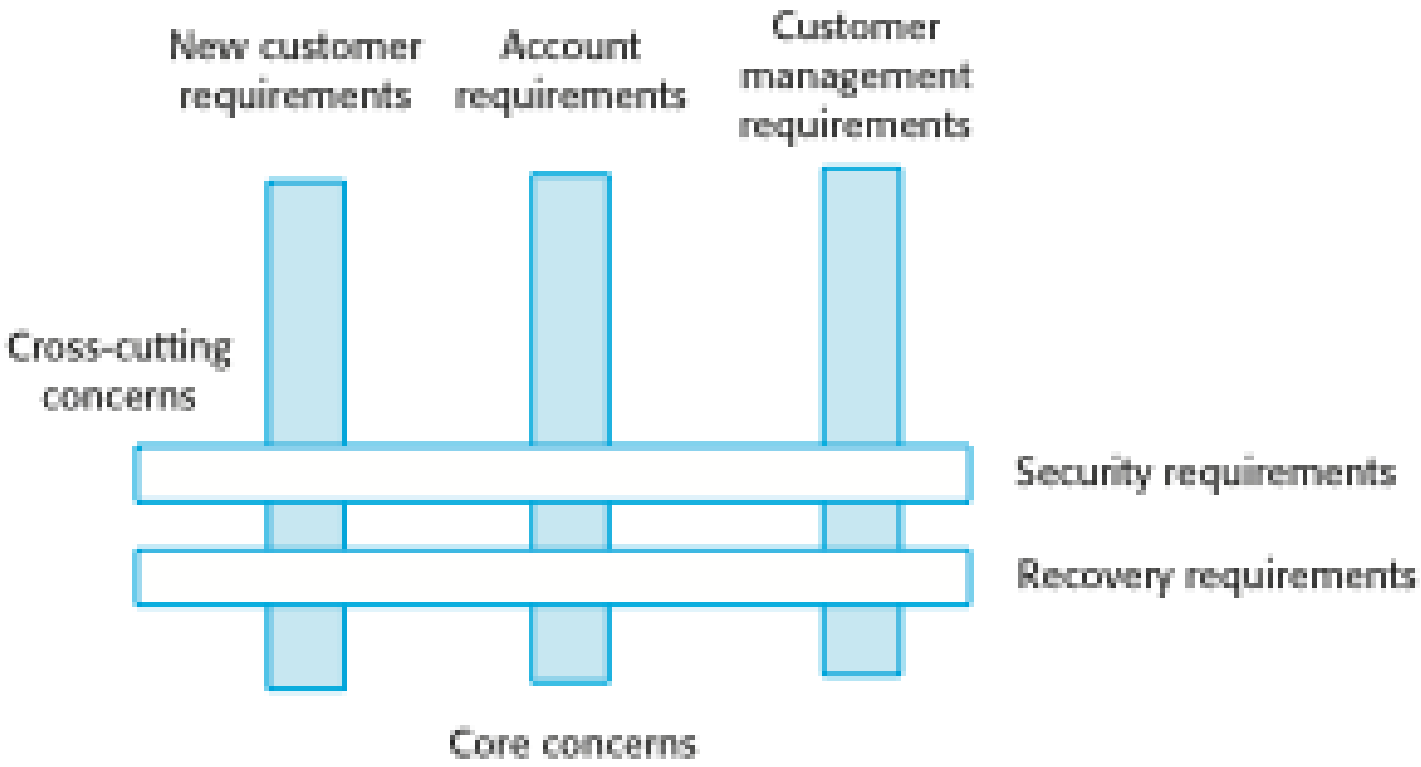
Stakeholder concerns

- Functional concerns which are related to specific functionality to be included in a system.
- Quality of service concerns which are related to the non-functional behaviour of a system.
- Policy concerns which are related to the overall policies that govern the use of the system.
- System concerns which are related to attributes of the system as a whole such as its maintainability or its configurability.
- Organisational concerns which are related to organisational goals and priorities such as producing a system within budget, making use of existing software assets or maintaining the reputation of an organisation.

Cross-cutting concerns

- Cross-cutting concerns are concerns whose implementation cuts across a number of program components.
- This results in problems when changes to the concern have to be made - the code to be changed is not localised but is in different places across the system.
- Cross cutting concerns lead to tangling and scattering.

Cross-cutting concerns



Aspects, join points and pointcuts

- An aspect is an abstraction which implements a concern. It includes information where it should be included in a program.
- A join point is a place in a program where an aspect may be included (woven).
- A pointcut defines where (at which join points) the aspect will be included in the program.

Terminology used in aspect-oriented software engineering

Term	Definition
advice	The code implementing a concern.
aspect	A program abstraction that defines a cross-cutting concern. It includes the definition of a pointcut and the advice associated with that concern.
join point	An event in an executing program where the advice associated with an aspect may be executed.
join point model	The set of events that may be referenced in a pointcut.
pointcut	A statement, included in an aspect, that defines the join points where the associated aspect advice should be executed.
weaving	The incorporation of advice code at the specified join points by an aspect weaver.

AspectJ - join point model

- ✧ Call events
  - Calls to a method or constructor
- ✧ Execution events
  - Execution of a method or constructor
- ✧ Initialisation events
  - Class or object initialisation
- ✧ Data events
  - Accessing or updating a field
- ✧ Exception events
  - The handling of an exception

Pointcuts

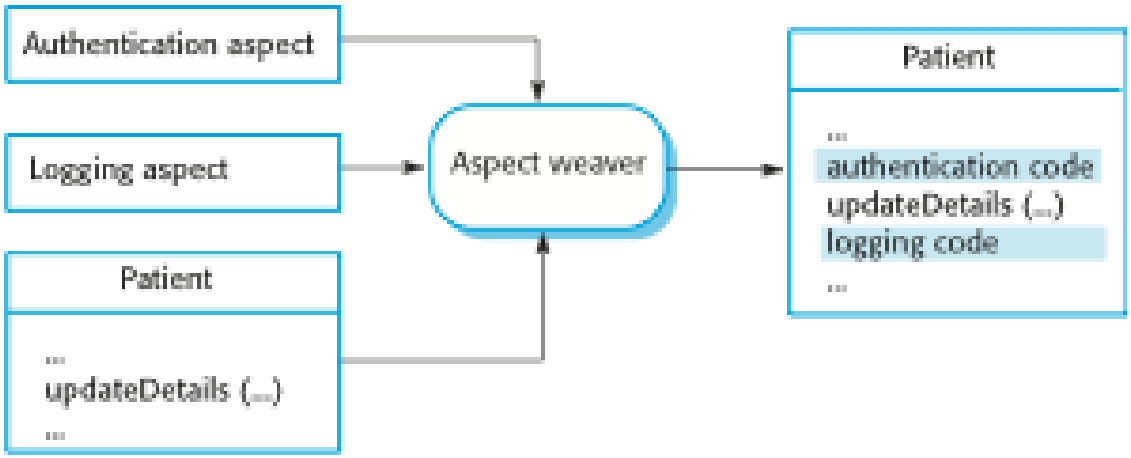
- Identifies the specific events with which advice should be associated.
- Examples of contexts where advice can be woven into a program
  - Before the execution of a specific method
  - After the normal or exceptional return from a method
  - When a field in an object is modified

An authentication aspect


```
aspect authentication
{
        before: call (public void update* (..))   // this is a pointcut
        {
                // this is the advice that should be executed when woven into
                // the executing system
                int tries = 0 ;
                string userPassword = Password.Get ( tries ) ;
                while (tries < 3 && userPassword != thisUser.password ( ) )
                {
                        // allow 3 tries to get the password right
                        tries = tries + 1 ;
                        userPassword = Password.Get ( tries ) ;
                }
                if (userPassword != thisUser.password ( )) then
                        //if password wrong, assume user has forgotten to logout
                        System.Logout (thisUser.uid) ;
        }
} // authentication
```

Aspect weaving

- Aspect weavers process source code and weave the aspects into the program at the specified pointcuts.
- Three approaches to aspect weaving
  - Source code pre-processing
  - Link-time weaving
  - Dynamic, execution-time weaving



Software engineering with aspects

- Aspects were introduced as a programming concept but, as the notion of concerns comes from requirements, an aspect oriented approach can be adopted at all stages in the system development process.
- The architecture of an aspect-oriented system is based around a core system plus extensions.
- The core system implements the primary concerns. Extensions implement secondary and cross-cutting concerns.