

Software Engineering

Lesson #09 - Lecture



Lesson #09 - Lecture

Your KBTU 202309 Software Engineering
class information is updating ...

Lesson #09 update is in progress

This will take around 2 hours to complete

Please, don't turn off your head



Part I: Introduction to Software Engineering

Part 1. Introduction to Software Engineering



System Dependability and Security

Part 2. System Dependability and Security



System Dependability and Security

```
#include<iostream>
Using namespace std;

int main()
{
    cout << "System Dependability and Security" << endl;

    return 0;
}
```

Agenda: Lesson #09 - Software Engineering - Lecture

1 Dependability properties

2 Sociotechnical systems

3 Redundancy and diversity

4 Dependable processes

5 Formal methods and dependability

System dependability

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



System dependability

System dependability

Dependability covers the related systems attributes of reliability, availability and security. These are all interdependent



System dependability

Importance of dependability

System failures may have widespread effects with large numbers of people affected by the failure

Systems that are not dependable and are unreliable, unsafe or insecure may be rejected by their users



System dependability

Importance of dependability

The costs of system failure may be very high if the failure leads to economic losses or physical damage

Undependable systems may cause information loss with a high consequent recovery cost



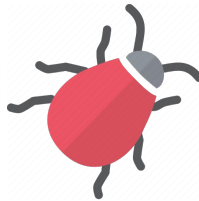
Causes of failure

Hardware failure



Hardware fails because of design and manufacturing errors or because components have reached the end of their natural life

Software failure



Software fails due to errors in its specification, design or implementation

Causes of failure

Operational failure



Human operators make mistakes. Now perhaps the largest single cause of system failures in socio-technical systems

System dependability



Agenda: Lesson #09 - Software Engineering - Lecture

1

Dependability properties

2

Sociotechnical systems

3

Redundancy and diversity

4

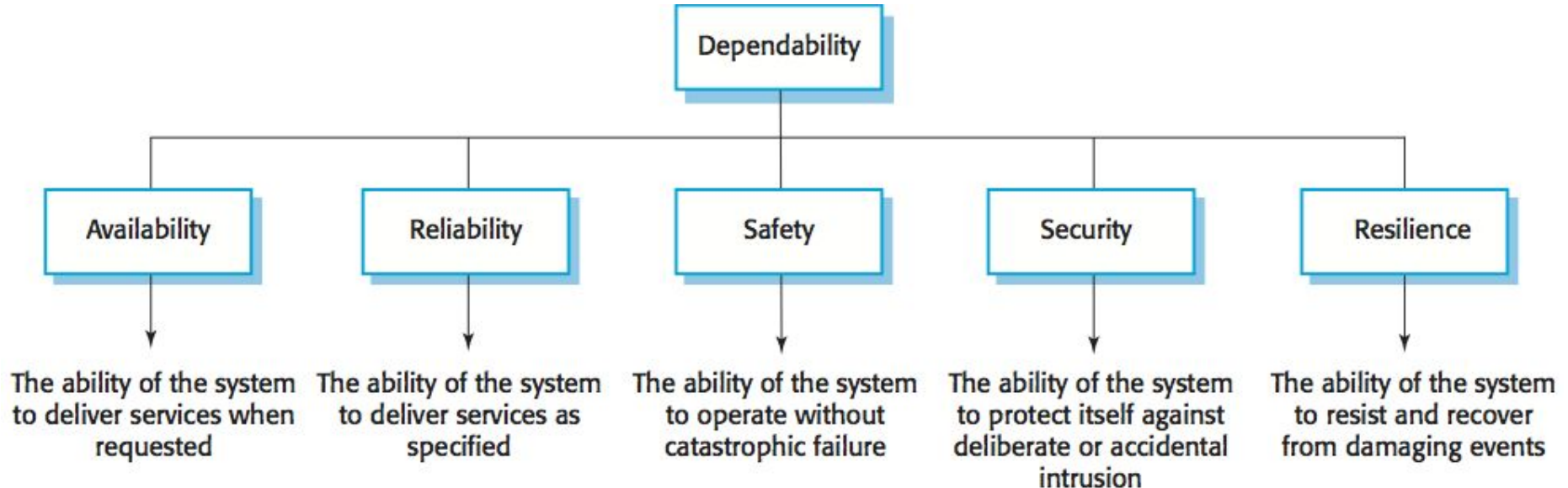
Dependable processes

5

Formal methods and dependability

Dependability properties

The principal dependability properties



Dependability properties

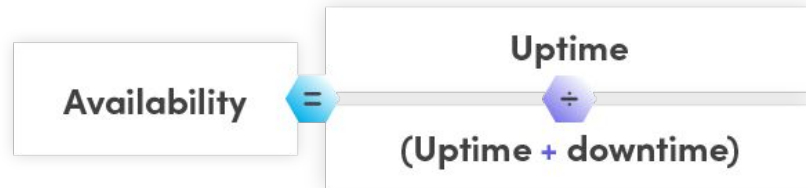
Principal properties

Availability

The probability that the system will be up and running and able to deliver useful services to users

Reliability

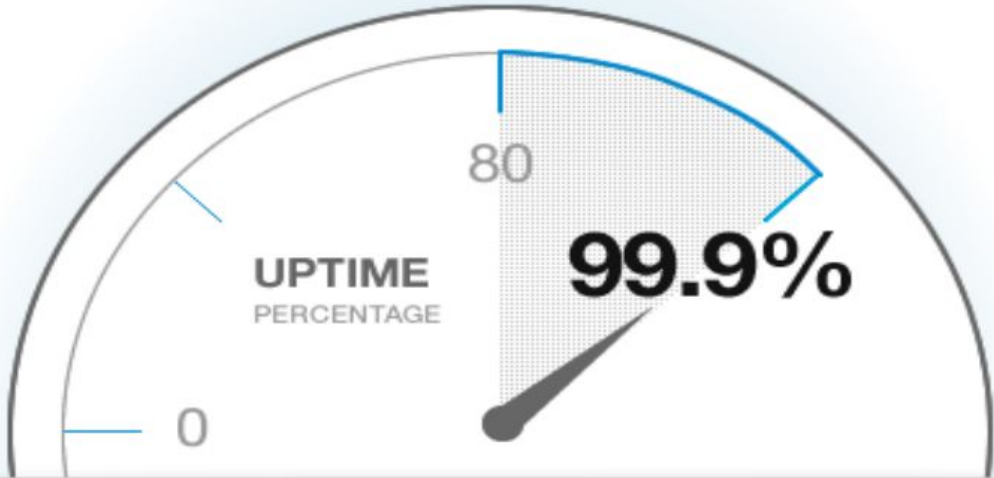
The probability that the system will correctly deliver services as expected by users



Dependability properties

Principal properties

Availability



High availability percentages of SLAs

PERCENTAGE	YEARLY DOWNTIME*
99.9	8hr 45m 57s
99.99	52m 35.7s
99.999	5m 15.6s
99.9999	31.6s
99.99999	3.2s
99.999999	0.3s
99.9999999	31.6 ms

Dependability properties

Principal properties

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



Principal properties

Resilience

A judgment of how well a system can maintain the continuity of its critical services in the presence of disruptive events such as equipment failure and cyberattacks



Dependability properties

Other dependability properties

Repairability

Reflects the extent to which the system can be repaired in the event of a failure

Maintainability

Reflects the extent to which the system can be adapted to new requirements



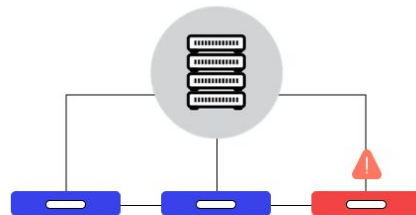
Dependability properties

Other dependability properties

Error tolerance

Reflects the extent to which user input errors can be avoided and tolerated

Fault Tolerance



Dependability properties

Dependability attribute dependencies

Safe system operation depends on the system being available and operating reliably

A system may be unreliable because its data has been corrupted by an external attack

Dependability properties

Dependability attribute dependencies

Denial of service attacks on a system are intended to make it unavailable

If a system is infected with a virus, you cannot be confident in its reliability or safety

Dependability achievement

Avoid the introduction of accidental errors when developing the system

Design V & V processes that are effective in discovering residual errors in the system

Design systems to be fault tolerant so that they can continue in operation when faults occur

Dependability properties

Dependability achievement

Design protection mechanisms that guard against external attacks

Configure the system correctly for its operating environment

Include system capabilities to recognise and resist cyberattacks

Include recovery mechanisms to help restore normal system service after a failure

Dependability properties

Dependability costs

Dependability costs tend to increase exponentially as increasing levels of dependability are required

Dependability properties

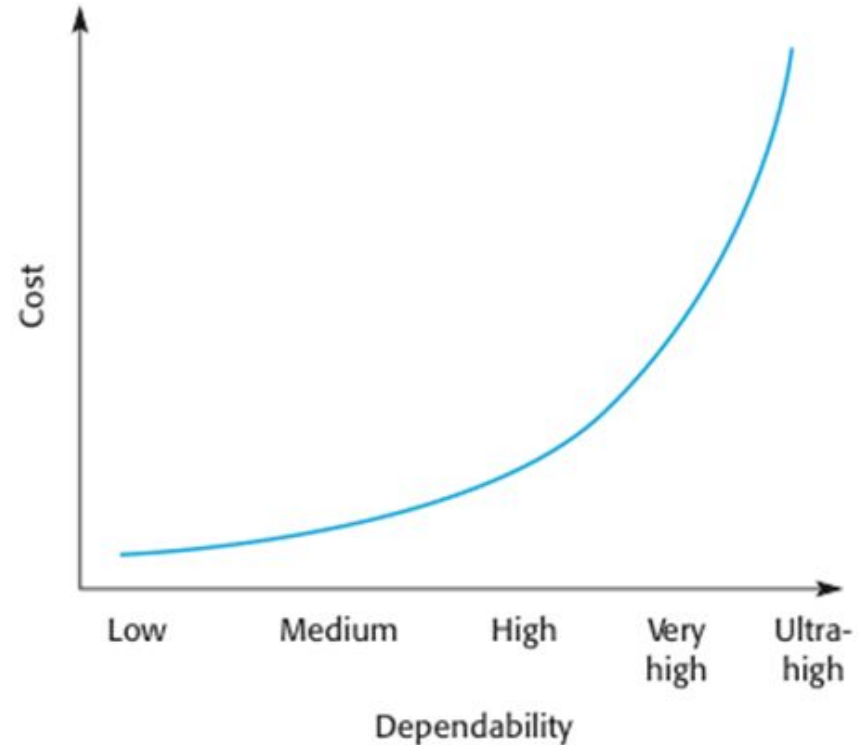
Dependability costs

There are two reasons for this

- The use of more expensive development techniques and hardware that are required to achieve the higher levels of dependability
- The increased testing and system validation that is required to convince the system client and regulators that the required levels of dependability have been achieved

Dependability properties

Cost/dependability curve



Dependability properties

Dependability economics

Because of very high costs of dependability achievement, it may be more cost effective to accept untrustworthy systems and pay for failure costs

However, this depends on social and political factors. A reputation for products that can't be trusted may lose future business

Depends on system type - for business systems in particular, modest levels of dependability may be adequate

Agenda: Lesson #09 - Software Engineering - Lecture

1

Dependability properties

2

Sociotechnical systems

3

Redundancy and diversity

4

Dependable processes

5

Formal methods and dependability

Systems and software

Software engineering is not an isolated activity but is part of a broader systems engineering process

Software systems are therefore not isolated systems but are essential components of broader systems that have a human, social or organizational purpose

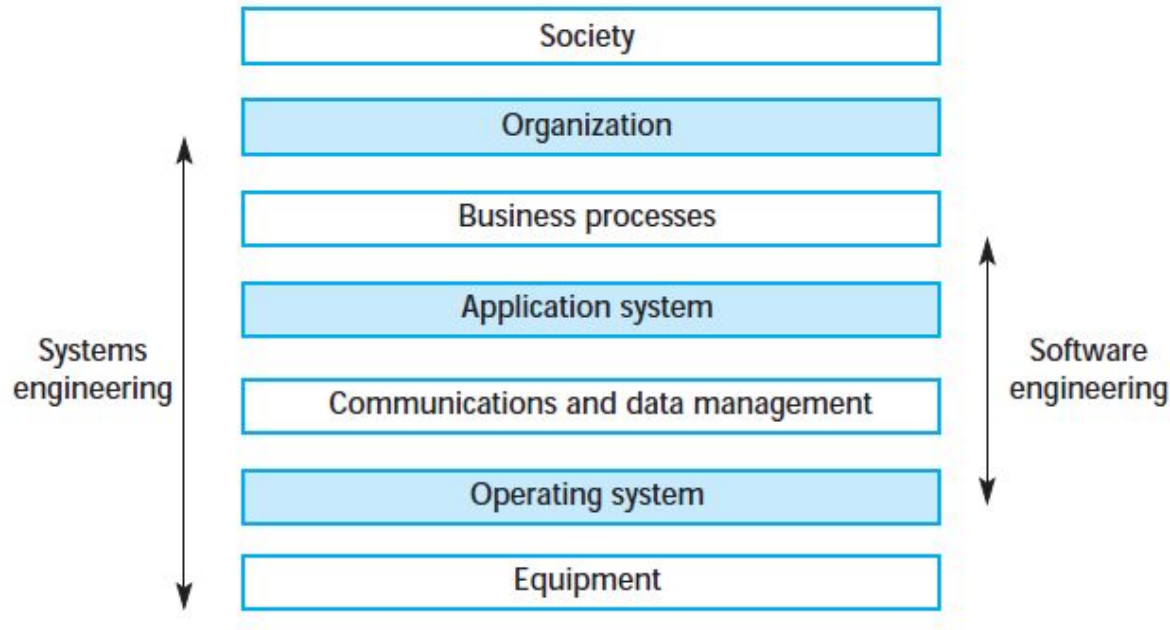
Systems and software

Example

- The wilderness weather system is part of broader weather recording and forecasting systems
- These include hardware and software, forecasting processes, system users, the organizations that depend on weather forecasts, etc.

Sociotechnical systems

The sociotechnical systems stack



Layers in the STS stack

- Equipment
 - Hardware devices, some of which may be computers. Most devices will include an embedded system of some kind
- Operating system
 - Provides a set of common facilities for higher levels in the system

Layers in the STS stack

- Communications and data management
 - Middleware that provides access to remote systems and databases
- Application systems
 - Specific functionality to meet some organization requirements

Layers in the STS stack

- Business processes
 - A set of processes involving people and computer systems that support the activities of the business
- Organizations
 - Higher level strategic business activities that affect the operation of the system

Layers in the STS stack

- Society
 - Laws, regulation and culture that affect the operation of the system

Holistic system design

There are interactions and dependencies between the layers in a system and changes at one level ripple through the other levels

- Example: Change in regulations (society) leads to changes in business processes and application software

Regulation and compliance

The general model of economic organization that is now almost universal in the world is that privately owned companies offer goods and services and make a profit on these

To ensure the safety of their citizens, most governments regulate (limit the freedom of) privately owned companies so that they must follow certain standards to ensure that their products are safe and secure

Regulated systems

Many critical systems are regulated systems, which means that their use must be approved by an external regulator before the systems go into service

- Nuclear systems
- Air traffic control systems
- Medical devices



Regulated systems

A safety and dependability case has to be approved by the regulator. Therefore, critical systems development has to create the evidence to convince a regulator that the system is dependable, safe and secure

Safety regulation

Regulation and compliance (following the rules) applies to the sociotechnical system as a whole and not simply the software element of that system

Safety-related systems may have to be certified as safe by the regulator

Safety regulation

To achieve certification, companies that are developing safety-critical systems have to produce an extensive safety case that shows that rules and regulations have been followed

It can be as expensive develop the documentation for certification as it is to develop the system itself

Agenda: Lesson #09 - Software Engineering - Lecture

1

Dependability properties

2

Sociotechnical systems

3

Redundancy and diversity

4

Dependable processes

5

Formal methods and dependability

Process diversity and redundancy

Process activities, such as validation, should not depend on a single approach, such as testing, to validate the system

Redundant and diverse process activities are important especially for verification and validation

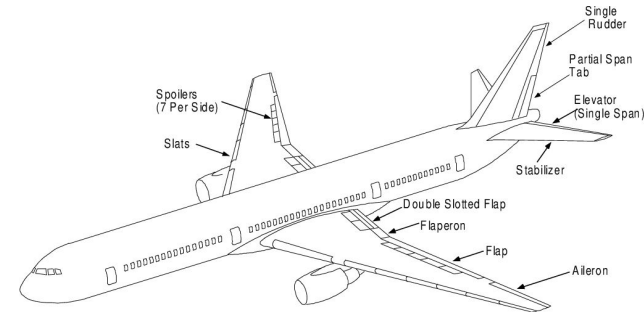
Multiple, different process activities complement each other and allow for cross-checking help to avoid process errors, which may lead to errors in the software

Redundancy and diversity

Problems with redundancy and diversity

Adding diversity and redundancy to a system increases the system complexity

This can increase the chances of error because of unanticipated interactions and dependencies between the redundant system components

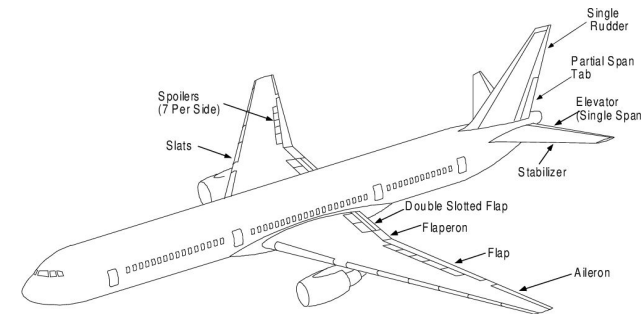


Redundancy and diversity


Problems with redundancy and diversity

Some engineers therefore advocate simplicity and extensive V & V as a more effective route to software dependability

Airbus FCS architecture is redundant/diverse;
Boeing 777 FCS architecture has no software diversity



Problems with redundancy and diversity



[About us](#)[Our portfolio](#)[Newsroom](#)

[Home](#) > [Newsroom](#) > [News and Features](#) > [Statement regarding Alert Operator Transmission \(AOT\) to A400M operators](#)

Statement regarding Alert Operator Transmission (AOT) to A400M operators

May 19, 2015 - Press release


Airbus Defence and Space has today (Tuesday 19 May) sent an Alert Operator Transmission (AOT) to all operators of the A400M informing them about specific checks to be performed on the fleet.

To avoid potential risks in any future flights, Airbus Defence and Space has informed the operators about necessary actions to take. In addition, these results have immediately been shared with the official investigation team.


The AOT requires Operators to perform one-time specific checks of the Electronic Control Units (ECU) on each of the aircraft's engines before next flight and introduces additional detailed checks to be carried out in the event of any subsequent engine or ECU replacement.

Contact us

Media corner



Photos & videos



Agenda: Lesson #09 - Software Engineering - Lecture

1 Dependability properties

2 Sociotechnical systems

3 Redundancy and diversity

4 Dependable processes

5 Formal methods and dependability

Dependable process characteristics

Explicitly defined

A process that has a defined process model that is used to drive the software production process. Data must be collected during the process that proves that the development team has followed the process as defined in the process model

Dependable process characteristics

Repeatable

A process that does not rely on individual interpretation and judgment. The process can be repeated across projects and with different team members, irrespective of who is involved in the development.

Dependable processes

Attributes of dependable processes

Process characteristic	Description
Auditable	The process should be understandable by people apart from process participants, who can check that process standards are being followed and make suggestions for process improvement.
Diverse	The process should include redundant and diverse verification and validation activities.
Documentable	The process should have a defined process model that sets out the activities in the process and the documentation that is to be produced during these activities.
Robust	The process should be able to recover from failures of individual process activities.
Standardized	A comprehensive set of software development standards covering software production and documentation should be available.

Dependable processes

Dependable process activities

Requirements reviews to check that the requirements are, as far as possible, complete and consistent

Requirements management to ensure that changes to the requirements are controlled and that the impact of proposed requirements changes is understood

Dependable processes

Dependable process activities

Formal specification, where a mathematical model of the software is created and analyzed

System modeling, where the software design is explicitly documented as a set of graphical models, and the links between the requirements and these models are documented

Dependable processes

Dependable process activities

Design and program inspections, where the different descriptions of the system are inspected and checked by different people

Static analysis, where automated checks are carried out on the source code of the program

Dependable process activities

Test planning and management, where a comprehensive set of system tests is designed

- The testing process has to be carefully managed to demonstrate that these tests provide coverage of the system requirements and have been correctly applied in the testing process

Dependable processes and agility

Dependable software often requires certification so both process and product documentation has to be produced

Up-front requirements analysis is also essential to discover requirements and requirements conflicts that may compromise the safety and security of the system

Dependable processes and agility

These conflict with the general approach in agile development of co-development of the requirements and the system and minimizing documentation

An agile process may be defined that incorporates techniques such as iterative development, test-first development and user involvement in the development team

Dependable processes and agility

So long as the team follows that process and documents their actions, agile methods can be used

However, additional documentation and planning is essential so 'pure agile' is impractical for dependable systems engineering

Agenda: Lesson #09 - Software Engineering - Lecture

1

Dependability properties

2

Sociotechnical systems

3

Redundancy and diversity

4

Dependable processes

5

Formal methods and dependability

Formal specification

Formal methods are approaches to software development that are based on mathematical representation and analysis of software

Formal methods include

- Formal specification;
- Specification analysis and proof;
- Transformational development;
- Program verification.

Formal specification

Formal methods significantly reduce some types of programming errors and can be cost-effective for dependable systems engineering

Formal approaches

Verification - based approaches

- Different representations of a software system such as a specification and a program implementing that specification are proved to be equivalent
- This demonstrates the absence of implementation errors

Formal approaches

Refinement - based approaches

- A representation of a system is systematically transformed into another, lower-level representation e.g. a specification is transformed automatically into an implementation
- This means that, if the transformation is correct, the representations are equivalent

Use of formal methods

The principal benefits of formal methods are in reducing the number of faults in systems

Consequently, their main area of applicability is in dependable systems engineering. There have been several successful projects where formal methods have been used in this area

In this area, the use of formal methods is most likely to be cost-effective because high system failure costs must be avoided

Classes of error

Specification and design errors and omissions

- Developing and analysing a formal model of the software may reveal errors and omissions in the software requirements. If the model is generated automatically or systematically from source code, analysis using model checking can find undesirable states that may occur such as deadlock in a concurrent system

Classes of error

Inconsistencies between a specification and a program

- If a refinement method is used, mistakes made by developers that make the software inconsistent with the specification are avoided. Program proving discovers inconsistencies between a program and its specification

Benefits of formal specification

Developing a formal specification requires the system requirements to be analyzed in detail. This helps to detect problems, inconsistencies and incompleteness in the requirements

As the specification is expressed in a formal language, it can be automatically analyzed to discover inconsistencies and incompleteness

Benefits of formal specification

If you use a formal method such as the B method, you can transform the formal specification into a 'correct' program

Program testing costs may be reduced if the program is formally verified against its specification

Acceptance of formal methods

Formal methods have had limited impact on practical software development:

- Problem owners cannot understand a formal specification and so cannot assess if it is an accurate representation of their requirements
- It is easy to assess the costs of developing a formal specification but harder to assess the benefits. Managers may therefore be unwilling to invest in formal methods

Acceptance of formal methods

Formal methods have had limited impact on practical software development:

- Software engineers are unfamiliar with this approach and are therefore reluctant to propose the use of FM
- Formal methods are still hard to scale up to large systems. Formal specification is not really compatible with agile development methods

Agenda: Lesson #09 - Software Engineering - Lecture

Q & A