# Summary of Lecture 15.11.2017


http://englishskills.se

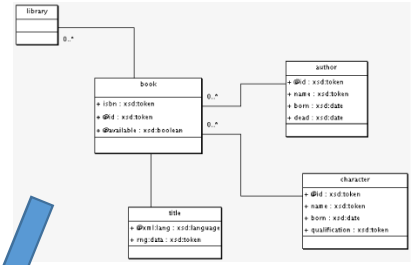
# Future-Proof Software-Systems: Summary 15.11.2017

Functionality

Documentation

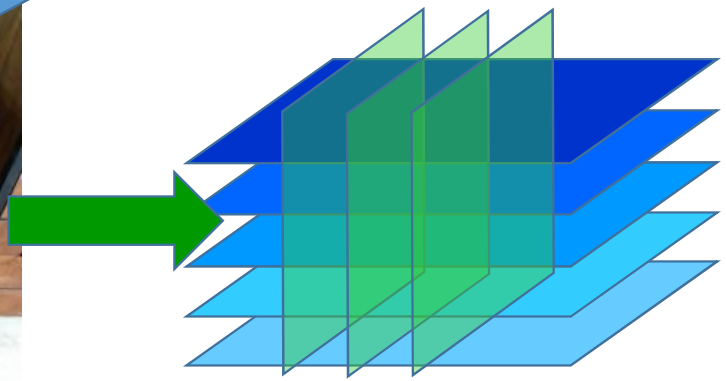
Models

Data/Information

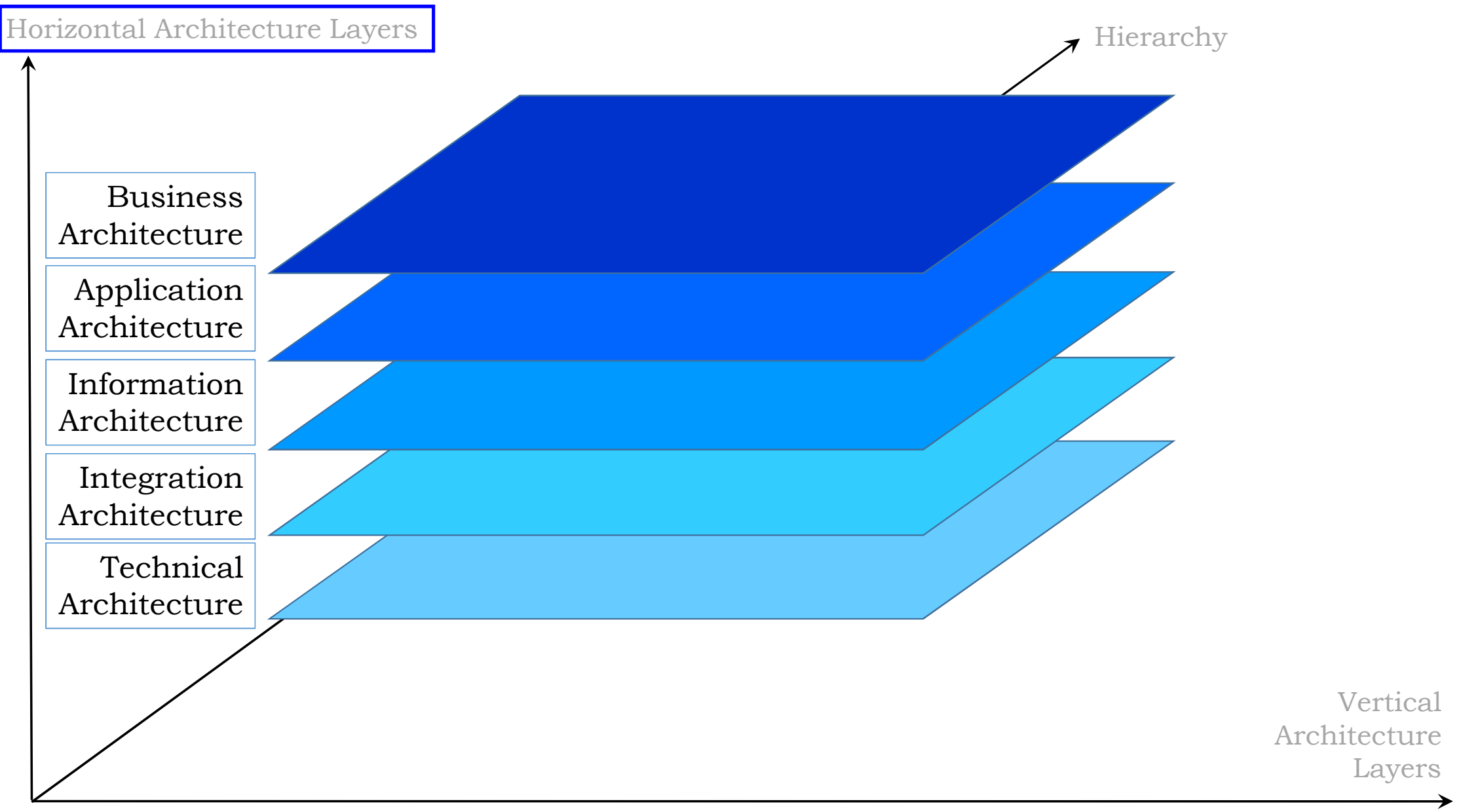


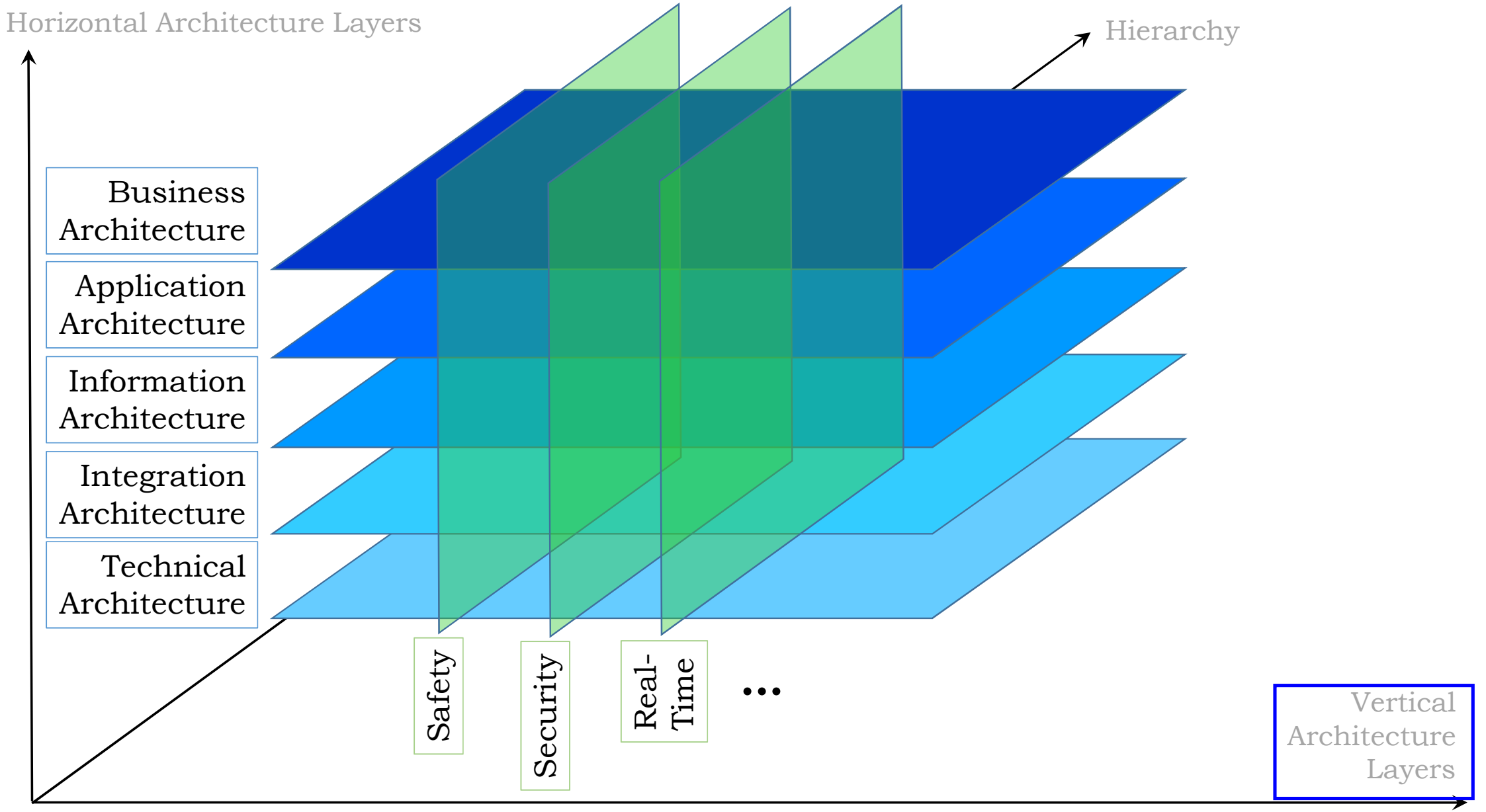

Categorization

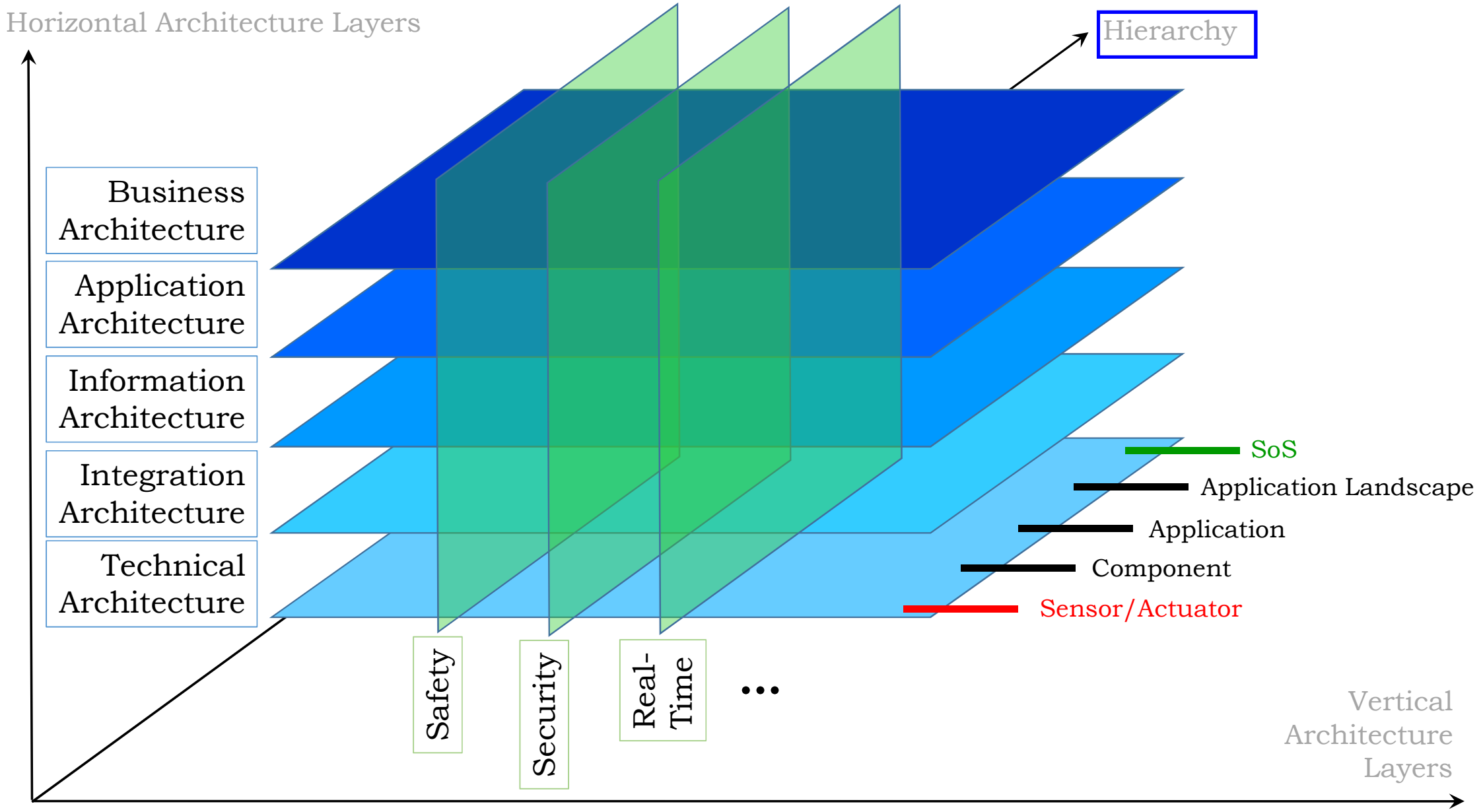
Categorization

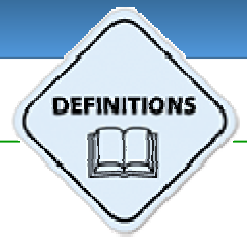


Architecture Framework


Horizontal Architecture Layers
Hierarchy
Business Architecture
Application Architecture
Information Architecture
Integration Architecture
Technical Architecture
Vertical Architecture Layers


Horizontal Architecture Layers
Hierarchy
Business Architecture
Application Architecture
Information Architecture
Integration Architecture
Technical Architecture
Safety
Security
Real-Time
...
Vertical Architecture Layers


Horizontal Architecture Layers
Hierarchy
Business Architecture
Application Architecture
Information Architecture
Integration Architecture
Technical Architecture
SoS
Application Landscape
Application
Component
Sensor/Actuator
Safety
Security
Real-Time
...
Vertical Architecture Layers




**Architecture Principles:**

Fundamental insights – formulated as *enforcable rules* – how a good software-system should be built [⇐ «Eternal Truths»]


Architecture principles are *not* directly applicable to construct an architectural solution.

They need the *future-proof software-systems engineer* to implement and enforce them.

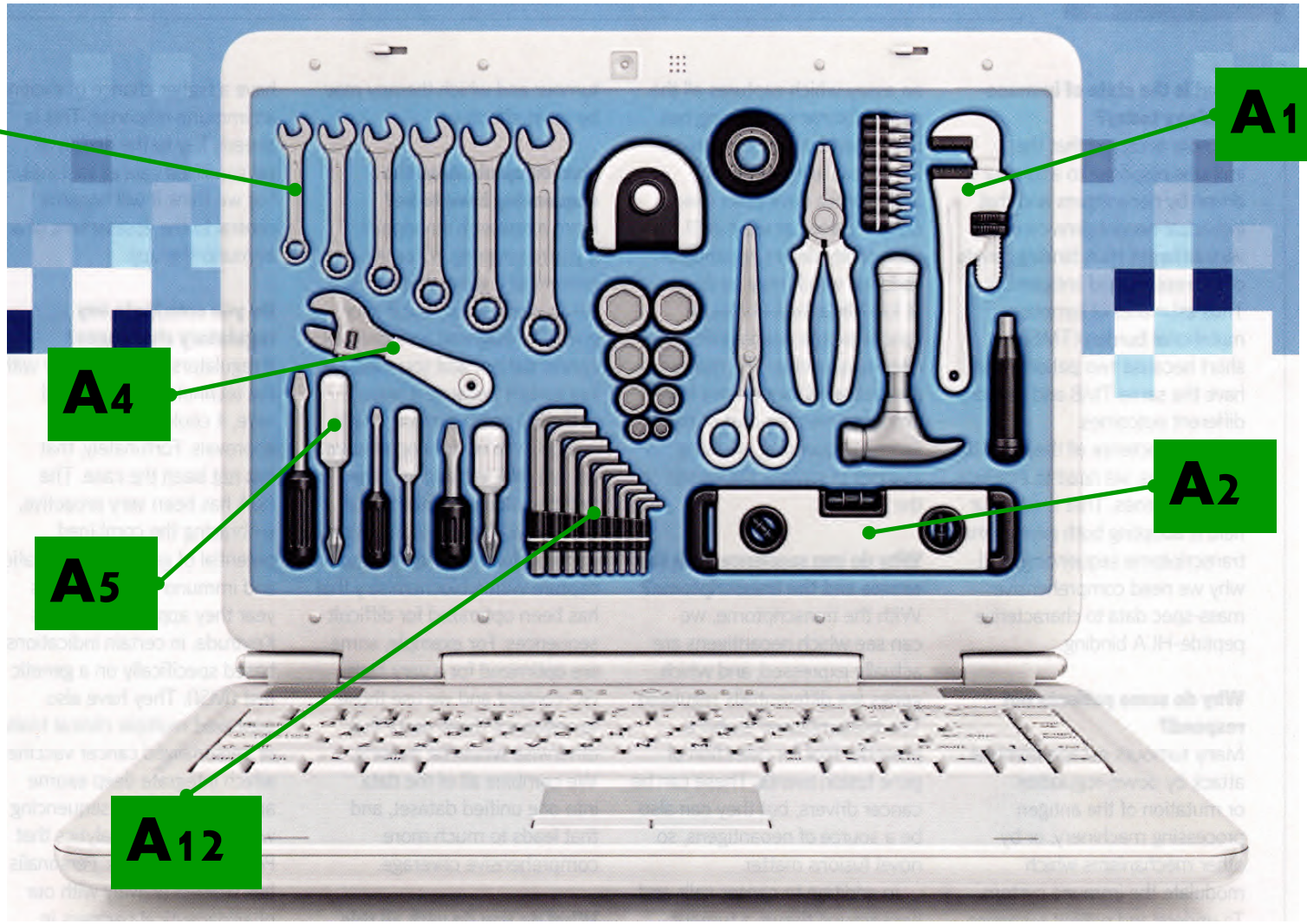
**Architecture Principles**

=

**Knowledge Toolbox**

of the

**Systems Architect**




© NATURE

**A1**

Architecture Principle A1:
**Architecture Layer Isolation**

[1] Always use standardized, technology-independent, and product-independent mechanisms for transfer of data and control between layers

[2] Never implement functionality from vertical layers in the horizontal layers (especially no technical functionality in the applications)

**Justification**: Any reliance on specific technologies or product features generates dependencies which (massively) reduce changeability.

Architecture layers should be able to evolve in their own pace without impacting the other layers by force.

Vertical functionality should not be implemented in the applications (but accessed via services), otherwise changes impact the application landscape.

**A1**

Business Architecture **Layer**
(Business Processes)

Isolation

Applications Architecture **Layer**
(Functionality)

Isolation

Information (Data) Architecture **Layer**
(Information & Data)

Isolation

Integration Architecture **Layer**
(Cooperation Mechanisms)

Isolation

Technical Architecture **Layer**
(Technical Infrastructure)

## Breaking Layers

Direct access – **bypassing** the standardized, technology-independent mechanisms

**Result:**
- Technology dependence
- Vendor lock-in
- No standards-compliance

http://media.istockphoto.com

**A2**

Architecture Principle A2:

**Partitioning, Encapsulation & Coupling**

1. Partition the functionality and data into encapsulation units according to their cohesion (thus minimizing dependencies)
2. Isolate the encapsulation units by strictly hiding any internal details. Allow access to functionality and data only through stable, well specified interfaces governed by contracts
3. Minimize the impact of dependencies between the encapsulation units by using adequate coupling mechanisms

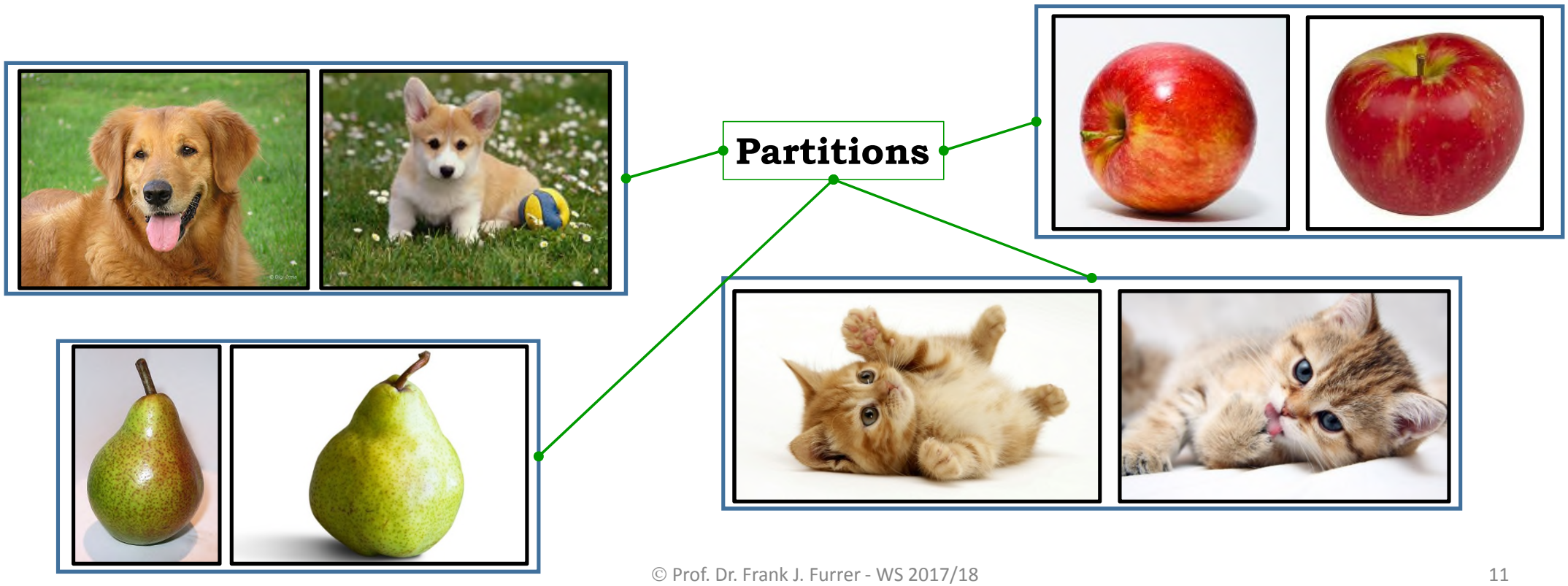
**Justification**: These 3 rules minimize the number and the impact of dependencies. The resulting system therefore offers the least resistance to change, because any change affects the smallest possible number of system elements. A low resistance to change corresponds to high **changeability**.

# A2

Decision criteria for good partitioning

**Primary Rule:**
- Respect *cohesion*
- Avoid *redundancy*

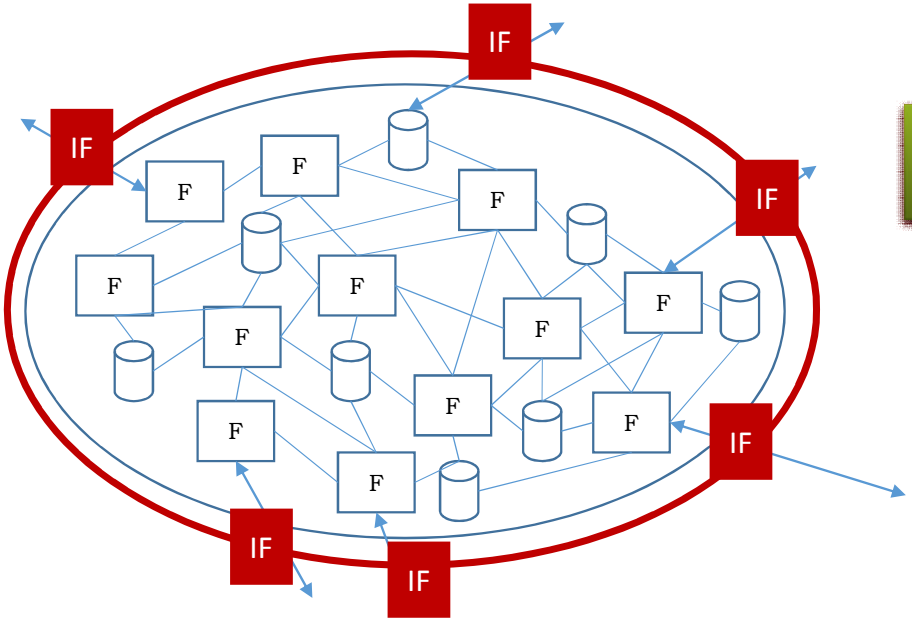
GOLDEN RULE



**A2**

**Encapsulation:**

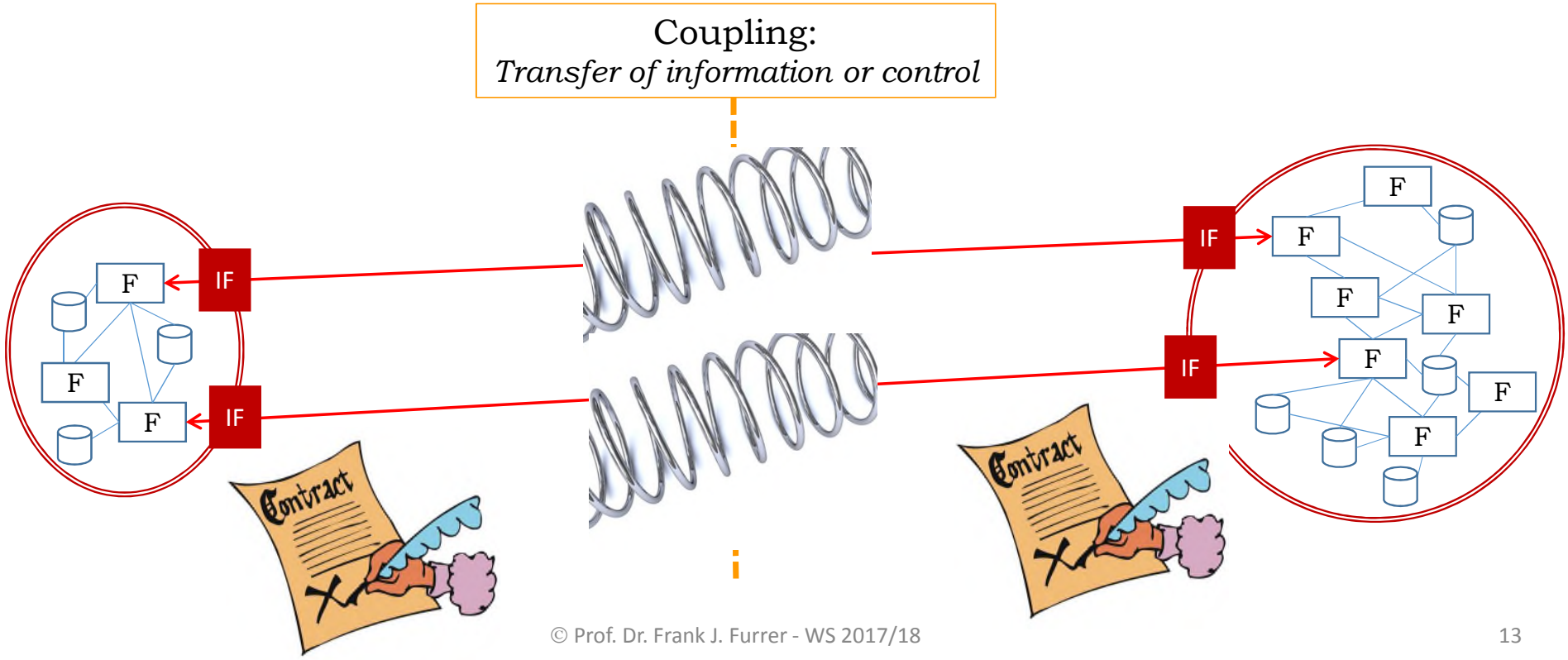
- The inner workings of the encapsulation units are hidden from the outside
- All accesses are only allowed through well-specified (formally defined) interfaces


**A2**

**Coupling**:

- Avoid all dependencies which are not essential
- Couple as loosely as possible
- Govern by interfaces/contracts



**A3**

Architecture Principle A3:

**Conceptual Integrity**

1. Define all the concepts, the full terminology and models (including their relationships and relevant properties) precisely (whenever possible formally)
2. Draw the boundary of the system in which the definitions apply
3. Consistently and consequently use the definitions in all areas of the system
4. Strictly enforce the correct use of the definitions
5. When cooperating with systems outside the boundary, match the concepts and the terminology between all systems and interfaces

**Justification**: Misunderstandings between stakeholders lead to unsatisfactory IT-systems with divergence in many areas. Misunderstandings of all sorts must therefore be eliminated in all phases of systems engineering

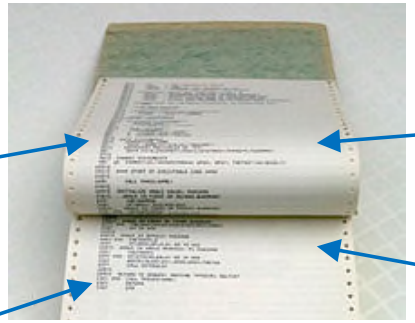
**A3**

Wheel rotation sensor

Book

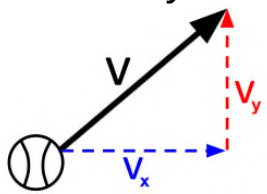
Stock price

Car

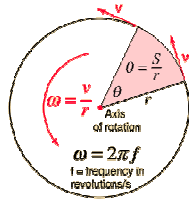


Software-System

Velocity




Rotation rate

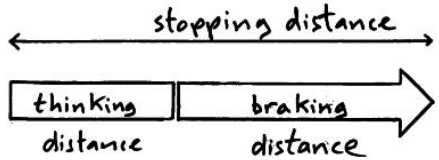



Gravitation constant

$$F_g = G\frac{m_1 m_2}{r^2}$$

**Software-Systems must be based on well-defined, universally accepted concepts**

Stopping distance



Systems developpers

Users


Legal system

Partners

Creating, maintaining, and enforcing *conceptual integrity* is mandatory in IT systems

How can we ensure conceptual integrity?

**... with a solid model foundation**

- Taxonomy
- Ontology
- Domain model
- Business object model

**A3**

**A4**

Architecture Principle A4:

**Redundancy**

1. There is only exactly *one source* for the functionality and for the data (both during development time and during run-time)
2. All redundant copies must be content-wise and time-wise synchronized (thus avoiding divergence)
3. The creation of *unmanaged* redundancy is not allowed under any circumstances. Existing unmanaged redundancy must be identified and eliminated in due course
4. Managed redundancy is allowed if there is a good (documented) reason

**Justification**: Any unmanaged redundancy may cause divergence and thus severely impact quality properties of the system's output. Any unmanaged redundancy will negatively impact the maintenance and evolution of the system

**A4**

**Redundancy** in an IT-system is – in most cases – poison for the structure and for many quality properties of an IT-system

| | Managed redundancy | **Un**managed redundancy |
|---|---|---|
| Known and wanted | **Yes** (if valid reason) | **NO!** |
| **Un**known or **un**wanted | **?** | **NO!** |


**The redundancy-ghost**

- **You don't hear it**
- **You don't see it**
- **But you feel the damage**

Allow only **managed redundancy**
(= known, justified, controlled, synchronized redundancy)
in functionality and data

**A5**

Architecture Principle A5:

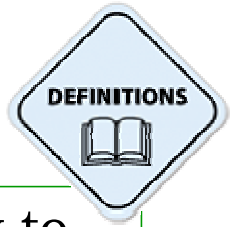
**Interoperability**

1. Precisely (formally) specify syntax and semantics in all interoperations
2. Whenever possible use formal contracts for the definition of interfaces
3. Whenever possible adopt and enforce accepted interoperability industry standards

**Justification**: Successful, unambigous interoperability is a key factor in today‘s distributed systems. Interoperabilty failures have severe consequences and are difficult to pinpoint. Formal contracts isolate the parts of the system.

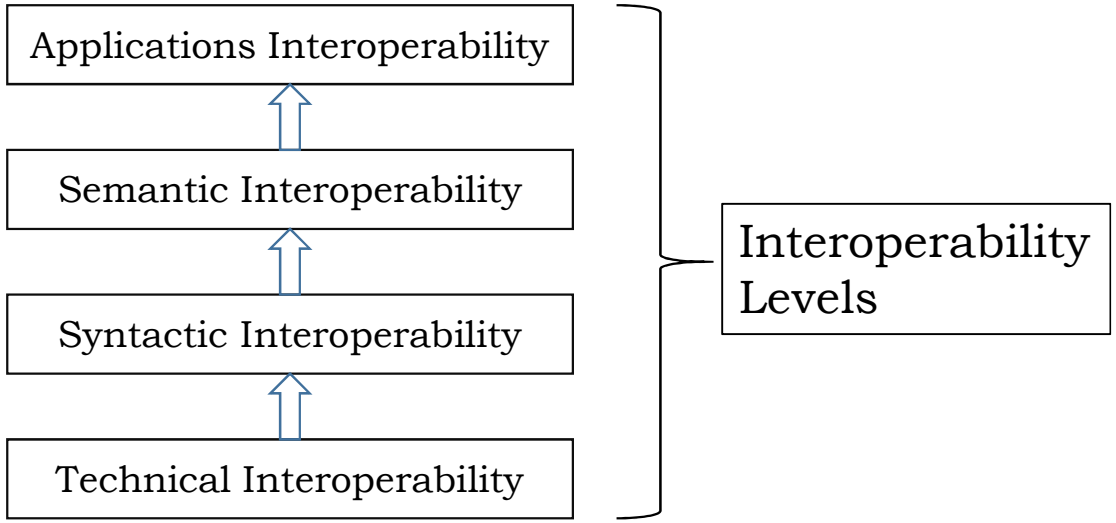
System Part **A** ⟷ System Part **B**

One of the most essential capabilities of collaborating systems is *interoperability*




Definition: **Interoperability** is the capability to exchange and make use of information and control

Interoperability must be assured on 4 levels:




A5

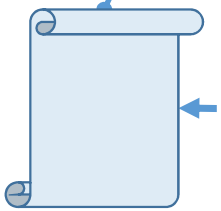
**A5**

How can we assure interoperability?

System Part **A** | Interface ⟷ Interface | System Part **B**

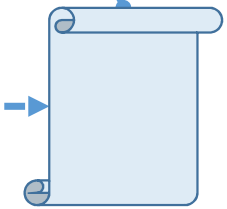
… by specifying interfaces

… and formally describe their behaviour, properties, attributes etc. in **contracts**



Interface Contract
Service Contract

- Technology alignment
- Syntax alignment
- Semantic alignment
- Model alignment



Interface Contract
Service Contract