Industrial Software: Business Value, Agility & Resilience

Frank J. Furrer
Dr. sc. techn. ETH-Zürich

Ringvorlesung TU Dresden WS 2014/2015
Montag, 26. Januar 2015

APPENDIX

V1.3 /18.01.2015

Appendix: Selected Topics

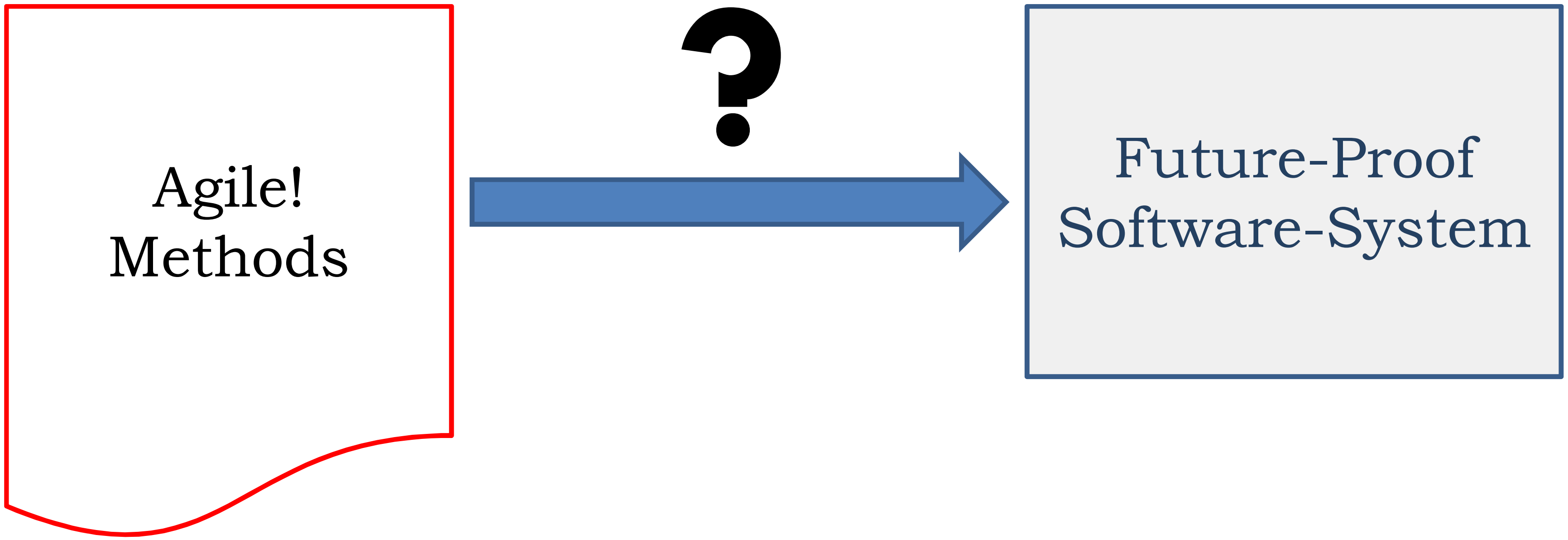
❶ Agility ≠ Agile!

❷ Architecture for Industrial Software

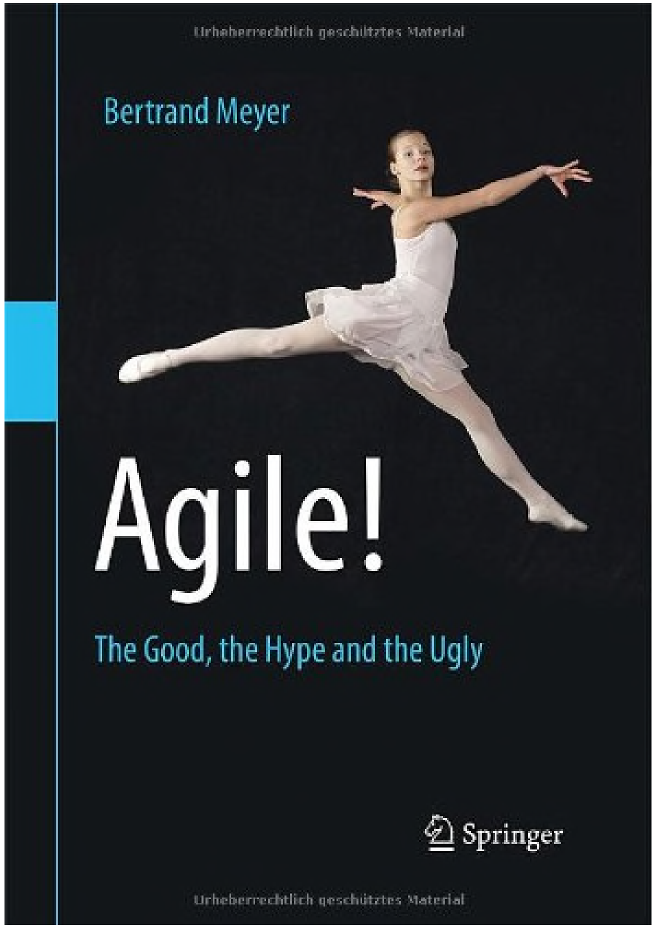
❸ Architecture Principles

Agility ≠ Agile!

http://www.acceptic.com


Agile!
Methods
Future-Proof
Software-System

Bertrand Meyer, 2014, ISBN 978-3-319-05154-3



«Agile texts defy a simple judgment:

- you may find in one paragraph a *brilliant insight*,
- in the next paragraph a *harmless platitude*,
- and in the one after some *freakish advice* guaranteed to damage your software process and products»

Bertrand Meyer, 2014, ISBN 978-3-319-05154-3

Agile Method [Bertrand Meyer, iSBN 978-3-319-05154-3]	supportive	detrimental	deadly	Remarks
Depreciation of upfront activitities			√	There is no substitute for serious requirements, considerate architecture, and careful design
Iterative development	√			Good, but not new
Continuous refactoring	√			Significant contribution
User stories instead of explicit requirements			√	Resulting systems are narrowly geared to specific user stories
Feature-based development			√	Ignores dependencies and fit into an existing system (complexity raise)
Rejection of traditional management tasks		√		May work in small development teams, but never in large projects
Embedded customer	√			Customer cooperation is good, although not as part of the development team
Test-driven development		√		Testing each function is important. However, "test-fix-refactor" on a module level is not sufficient
Depreciation of documents			√	True in some industries. Generally a very bad idea, because large SW-systems require sustainable doc's
Collective code ownership		√		Modern approach: Ownership lies with the domain owner
Short daily programmer meetings	√			Useful for small programming teams (< 25)
Short iterations	√			Good implementation practice
Closed window rule	√			Good, but not new
Continuous integration	√			Size of the functionality is critical
Time-boxing	√			Instills discipline in the programming

Conclusions:

1. The agile canon misses the whole superstructure for future-proof software systems (e.g. requirements gathering, formal modeling, **architecture** development & maintenance, system optimization)

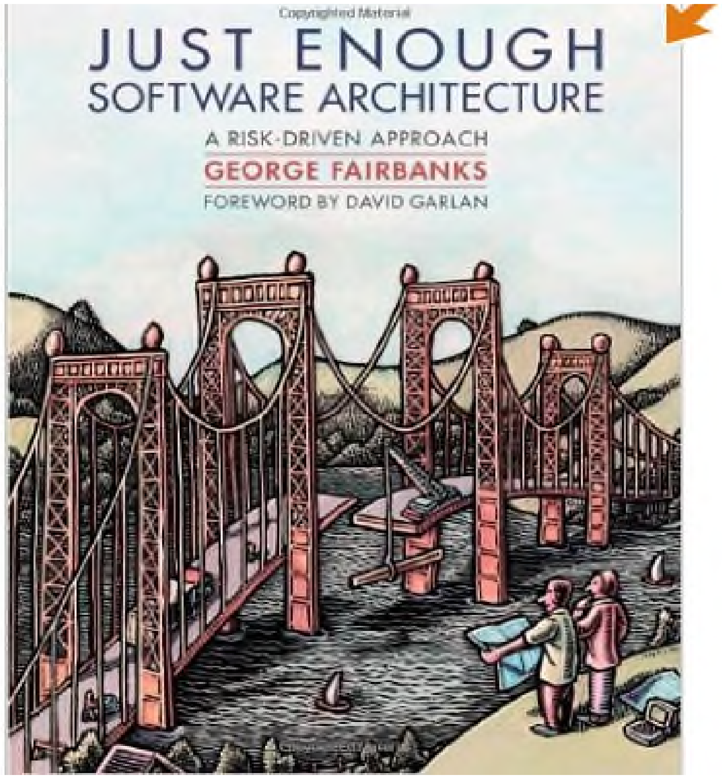
2. Agile methods bring benefits to the work of small programming teams (< 25)

3. Some agile ideas are useful in improving processes also for very large information systems

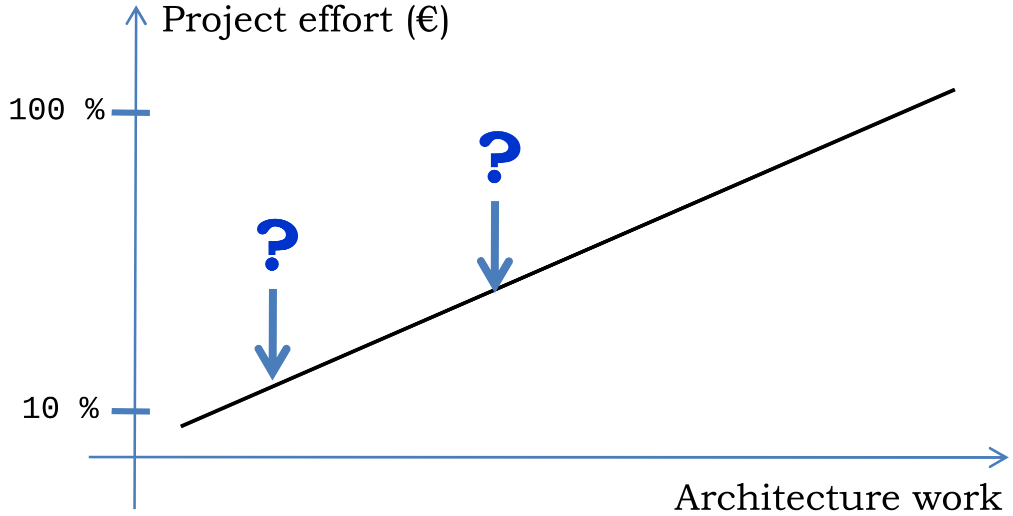
How much Architecture Work is enough?

Architecture = Front-end effort (before the productive work starts)

→ Cost, delay, constraints



G. Fairbanks / ISBN 978-0-9846181-0-1



Answer:

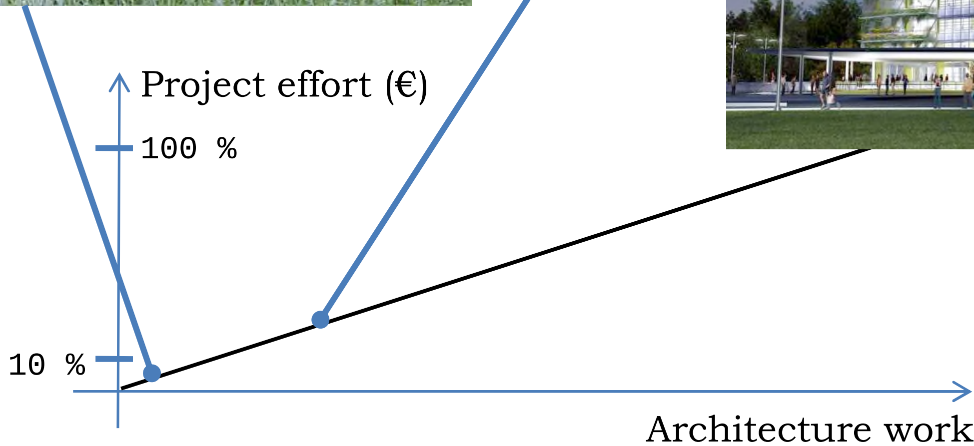
- System creation/extensions with high risk need much architecture work
- System creation/extensions with low risk need little architecture work

(George Fairbanks - ISBN 978-0-9846181-0-1, 2010)

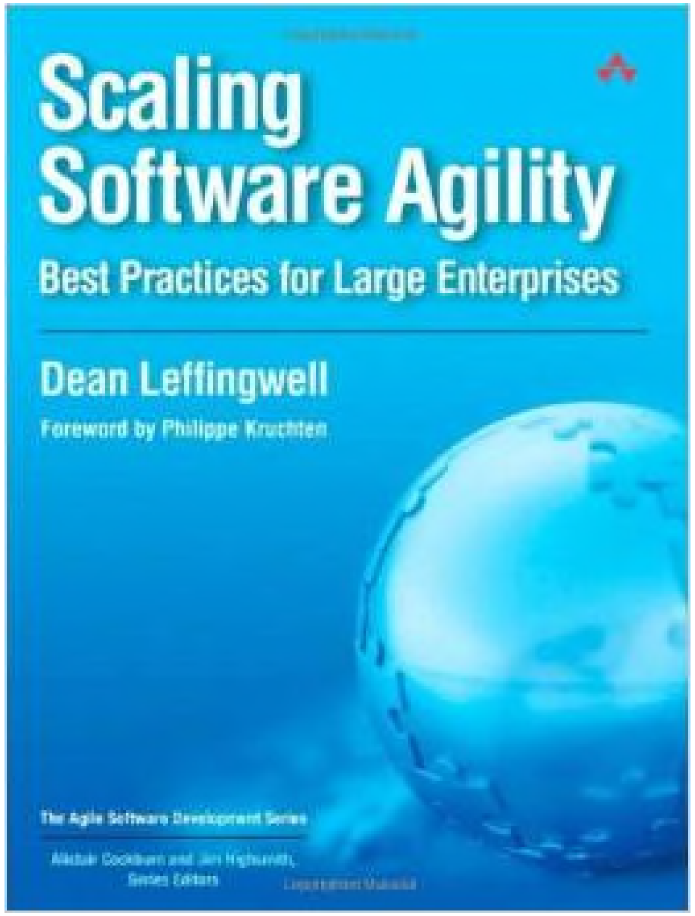
How much Architecture is enough?

http://www.dimensionsinfo.com/dimensions-of-a-dog-house

http://www.skyscrapernews.org



Is there hope for Agile in Large SW-Systems?



Dean Leffingwell: **Scaling Software Agility – *Best Practices for Large Enterprises***, Addison Wesley Publications, Inc., USA, 2007. ISBN-13: 978-0-321-4581-9

Part III:

✓ 7 agile team practices that scale

✓ 7 agile enterprise practices

… some very useful thoughts!

[Appendix]

Architecture for Industrial Software

Software must be specified, developed, maintained and evolved according to *industrial methods and processes*

⇒ **Industrial Software**

Industrial Software:

1. Clear business – IT alignment
2. Architecture-centric
3. Strategy-controlled (Evolution, Reuse, Product Lines, …)
4. Unambiguous specification of requirements (functionality and non-functional properties)
5. Powerful, accepted, enforced development process
6. Modeled (~ formal)
7. Metrics

The eternal dilemma of (industrial) software-development:

Business requirements ⇔ Architectural requirements

Business wants:

- (Very) short time to market
- Low cost
- Only essential functionality
- Newest technology

Architecture wants:

- Good fit into the existing system
- Refactoring to improve architectural quality
- Limit growth in complexity
- Use proven technologies

http://wohleranzeiger.ch/seilziehen/index.html

Example: 5th Language

Until 1995 Swiss banking IT-systems used 4 languages:

```
Deutsch: Kontostand am 31.12.2012
Französisch: Solde bancaire le 31.12.2013
Italienisch: Saldo il 31.12.2013
Englisch: Balance at 31.12.2013
```

Due to globalization, in Y2000 a new language (Spanish) had to be offered to the customers

```
Spanisch: Saldo el 31.12.2013
```

```
PROGRAMM N
…
(1;12;Kontostand am x.y.z)
(2;12;Solde bancaire le x.y.z)
(3;12;Saldo il x.y.z)
(4;12;Balance at x.y.z)
…
…
```

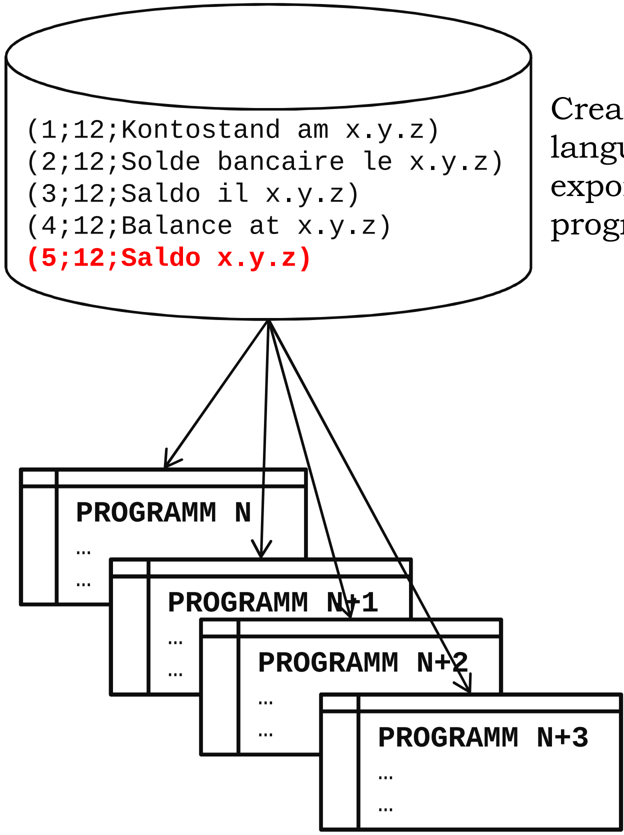
Traditionally, the texts were part of the individual programs („text-string“), identified by language code and text code

Solution 1:

```
PROGRAMM N
…
(1;12;Kontostand am x.y.z)
(2;12;Solde bancaire le x.y.z)
(3;12;Saldo il x.y.z)
(4;12;Balance at x.y.z)
(5;12;Saldo x.y.z)
…
…
```

Individually modify all the programs which need Spanish output (ca. 5'000 applications)

Solution 2:



Create a central language file and export it to all programs

X-Architectures

Vertical Architectures

Structural Architecture Layers

Horizontal Architectures

	Security Architecture (Defense)	Safety Architecture (Accidents)	Performance Architecture (Real-Time)	System Management Architecture (Control)	... etc.
Business Architecture (Business Processes)					
Applications Architecture (Functionality)					
Information (Data) Architecture (Information & Data)					
Integration Architecture (Cooperation Mechanisms)					
Technical Architecture (Technical Infrastructure)					

http://www.vehicle-lab.net

Example: Automotive Control

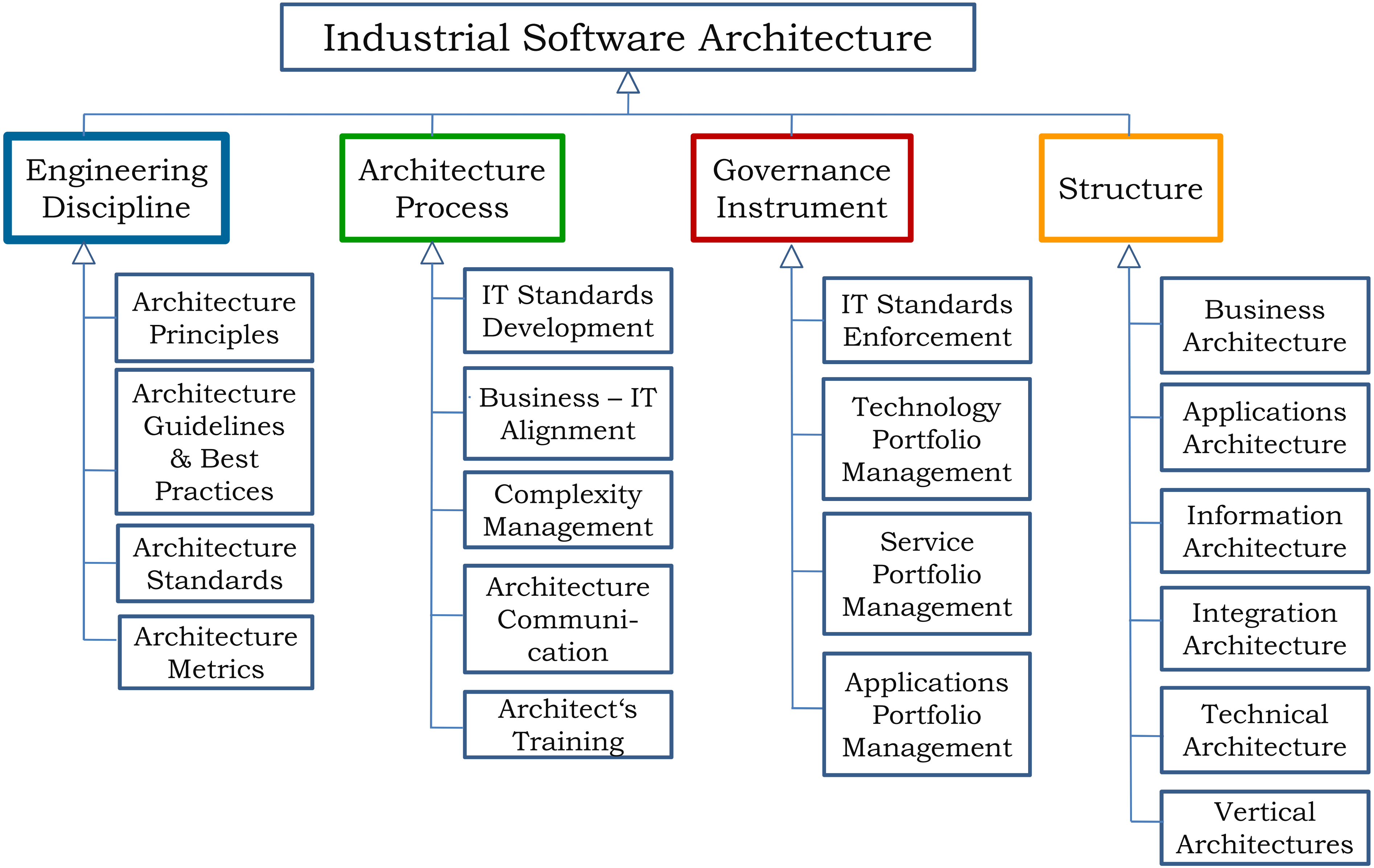
Business Architecture:
- Definition of Functionality & Interactions

Applications Architecture:
- Assignment of functionality to tasks, definition of interfaces, redundancy

Information (Data) Architecture:
- Specification of information used (car & environment) and data structures

Integration Architecture:
- Design of bus-structure(s) and interaction mechanisms & middleware

Technical Architecture:
- Number and location of the ECU's, cabling structure
- System software (RTOS)


Industrial Software Architecture
Engineering Discipline
Architecture Principles
Architecture Guidelines & Best Practices
Architecture Standards
Architecture Metrics
Architecture Process
IT Standards Development
Business – IT Alignment
Complexity Management
Architecture Communi-cation
Architect's Training
Governance Instrument
IT Standards Enforcement
Technology Portfolio Management
Service Portfolio Management
Applications Portfolio Management
Structure
Business Architecture
Applications Architecture
Information Architecture
Integration Architecture
Technical Architecture
Vertical Architectures

Architecture Principles

… the birth of architecture principles (1972):

Programming Techniques — R. Morris, Editor

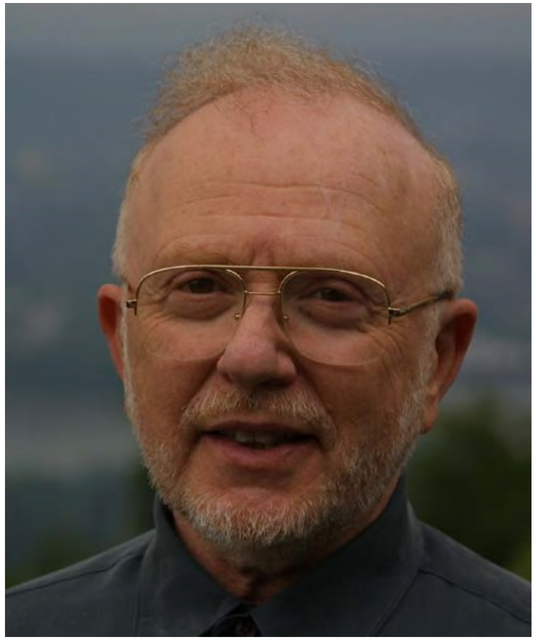
On the Criteria To Be Used in Decomposing Systems into Modules

D.L. Parnas
Carnegie-Mellon University

This paper discusses modularization as a mechanism for improving the flexibility and comprehensibility of a system while allowing the shortening of its development time. The effectiveness of a "modularization" is dependent upon the criteria used in dividing the system into modules. A system design problem is presented and

Introduction

A lucid statement of the philosophy of modular programming can be found in a 1970 textbook on the design of system programs by Gouthier and Pont [1, ¶10.23], which we quote below:[1]



http://datapeak.net/computerscientists.htm

David L. Parnas
* February 10, 1941 in Pittsburgh, USA

Communications of the ACM, Volume 15, Number 12, December 1972
Available at: http://www.cs.umd.edu/class/spring2003/cmsc838p/Design/criteria.pdf

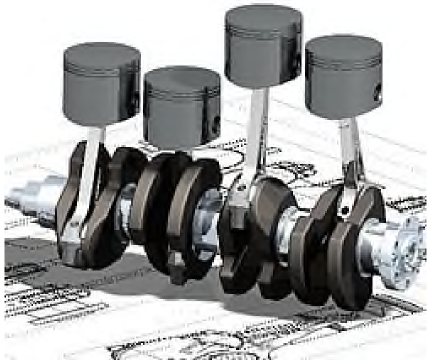
Architecture Principles:

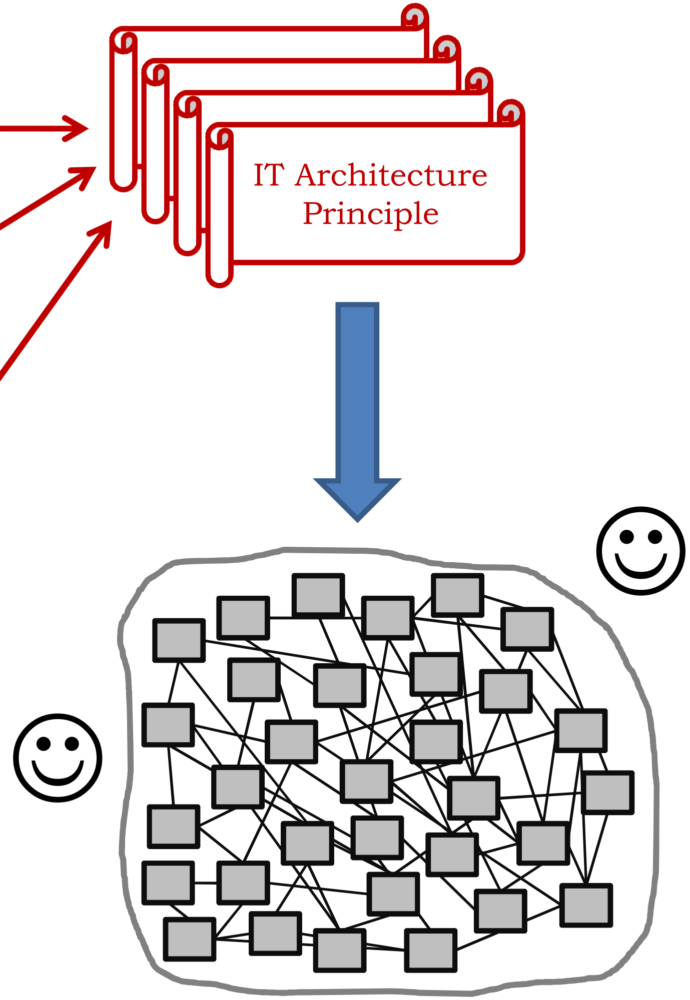
Fundamental insights – *formulated as enforcable rules* – how a good software-system should be built.

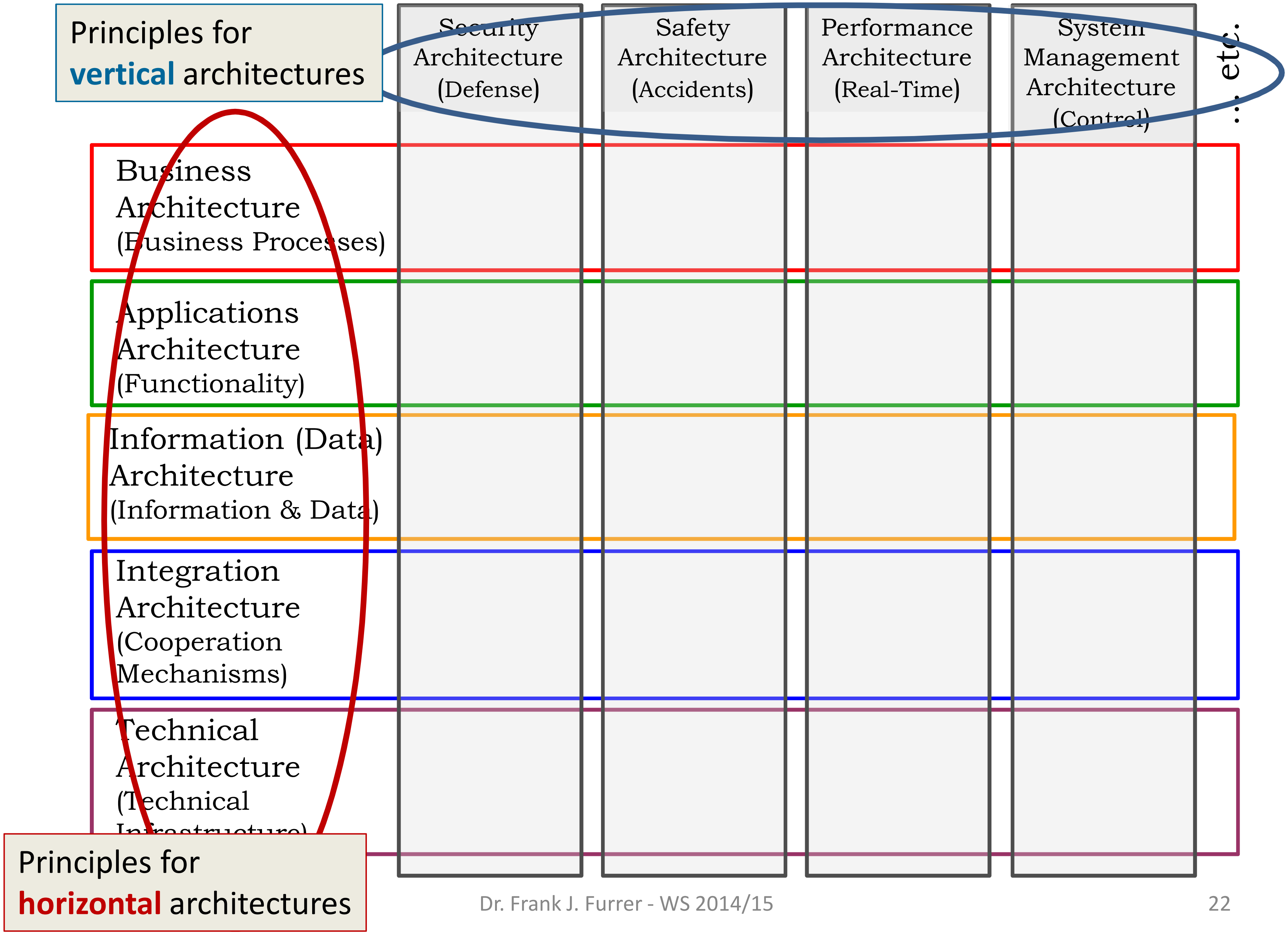
⇒ Foundation of future-proof SW-structures

Where do Architecture Principles come from? Which are the *good* ones?

Software Engineering Institute | Carnegie Mellon






Principles for
vertical architectures
Security Architecture (Defense)
Safety Architecture (Accidents)
Performance Architecture (Real-Time)
System Management Architecture (Control)
… etc.
Business Architecture (Business Processes)
Applications Architecture (Functionality)
Information (Data) Architecture (Information & Data)
Integration Architecture (Cooperation Mechanisms)
Technical Architecture (Technical Infrastructure)
Principles for
horizontal architectures

Example: Horizontal Architecture Principle

A1

Architecture Principle A1:
Architecture Layer Isolation

Isolate the architecture layers via standardized, technology-independent and product-independent mechanisms.

Never implement technical functionality in the applications.

- Business Architecture
- Applications Architecture
- Information Architecture
- Integration Architecture
- Technical Architecture

Justification: Any reliance on specific technologies or products generates dependencies which (massively) reduce agility.

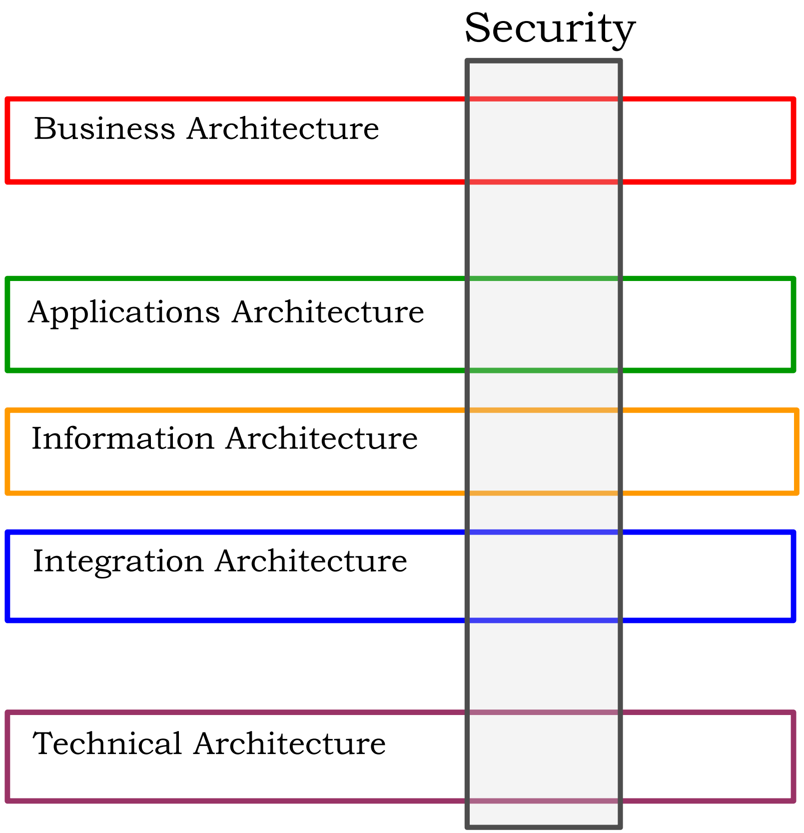
Architecture layers should be able to evolve in their own pace without impacting the other layers by force

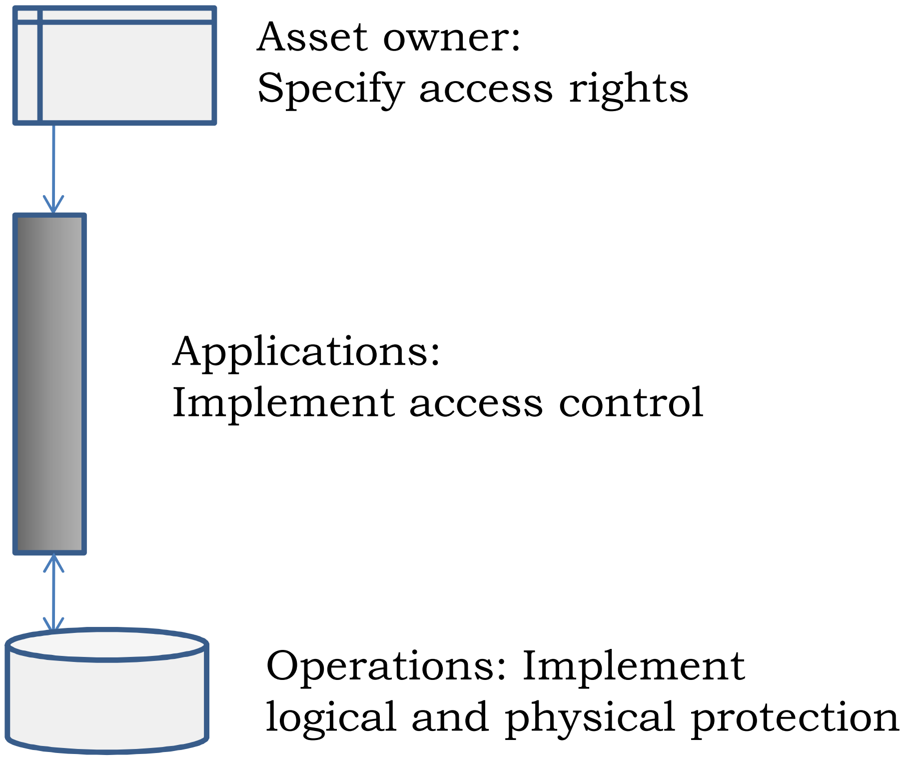
A_{S1}

Security Principle:
Least Privilege

Assign only the minimum set of access rights required for the task (for people and applications)

Example: Vertical Architecture Principle
Security Architecture Principle
«Least Privilege»





Horizontal Architecture Principles

- A1: Architecture Layer Isolation
- A2: Partitioning, Encapsulation and Coupling
- A3: Redundancy
- A4: Interoperability
- A5: Common Functions
- A6: Reference Architectures, Frameworks and Patterns
- A7: Reuse and Parametrization
- A8: Industry Standards
- A9: Information Architecture
- A10: Formal Modeling
- A11: Systems-of-Systems (SoS)
- A12: Complexity and Simplification

http://www.uni-heidelberg.de

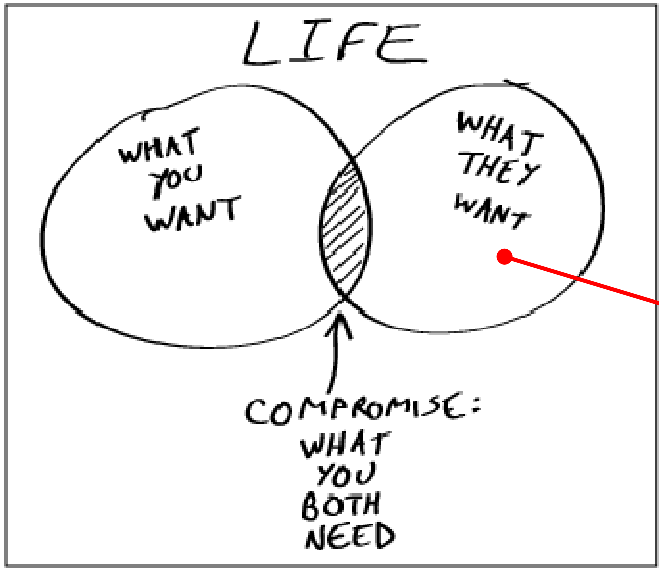
Architecture Principles **Enforcement**

consult, convince, negotiate

force, constrain, repress

Strategy of a successful IT-architect

compromise



http://www.allisonbraun.ca

But: Manage the technical debt!

https://yourdailygerman.wordpress.com

Questions please ?