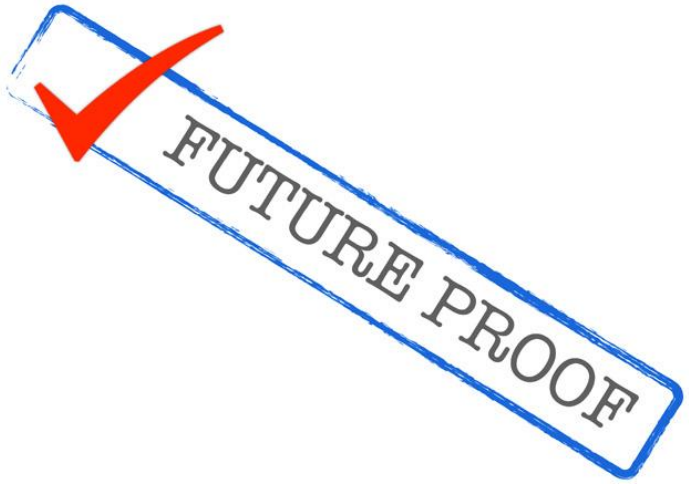


Future-Proof Software-Systems (FPSS)

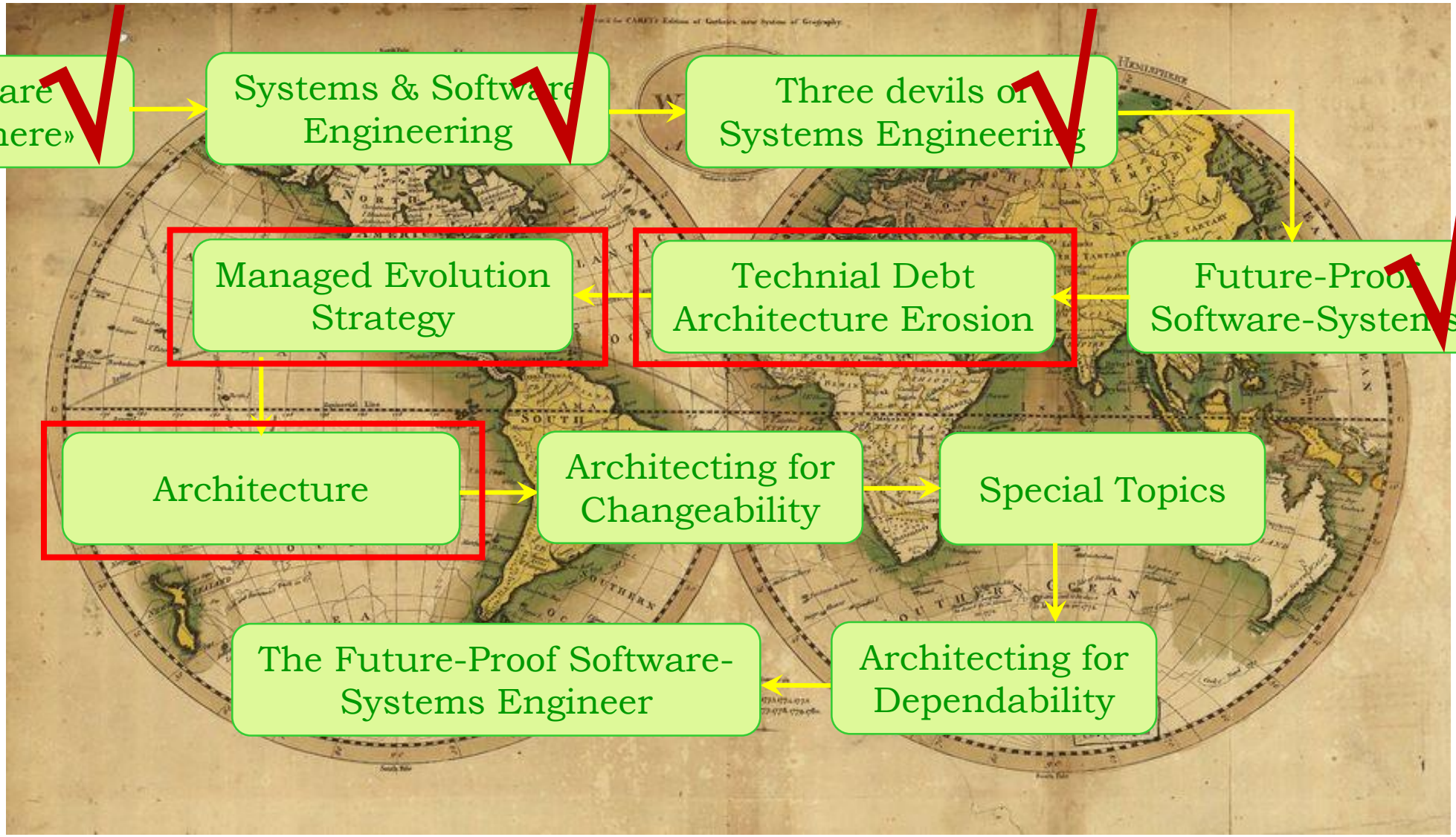
Part 2

Lecture WS 2019/20: Prof. Dr. Frank J. Furrer

Version 0.1/06.08.2019

Future-Proof Software-Systems [Part 1]

Our journey:



Content [Part 2]:

- Software Lifecycle
- Technical Debt
- Architecture Erosion
- Managed Evolution
- The Importance of Architecture
- Industrial Architecture Framework
- Architecture Principles and their Use

Software Lifecycle

Software Lifecycle

https://openclipart.org

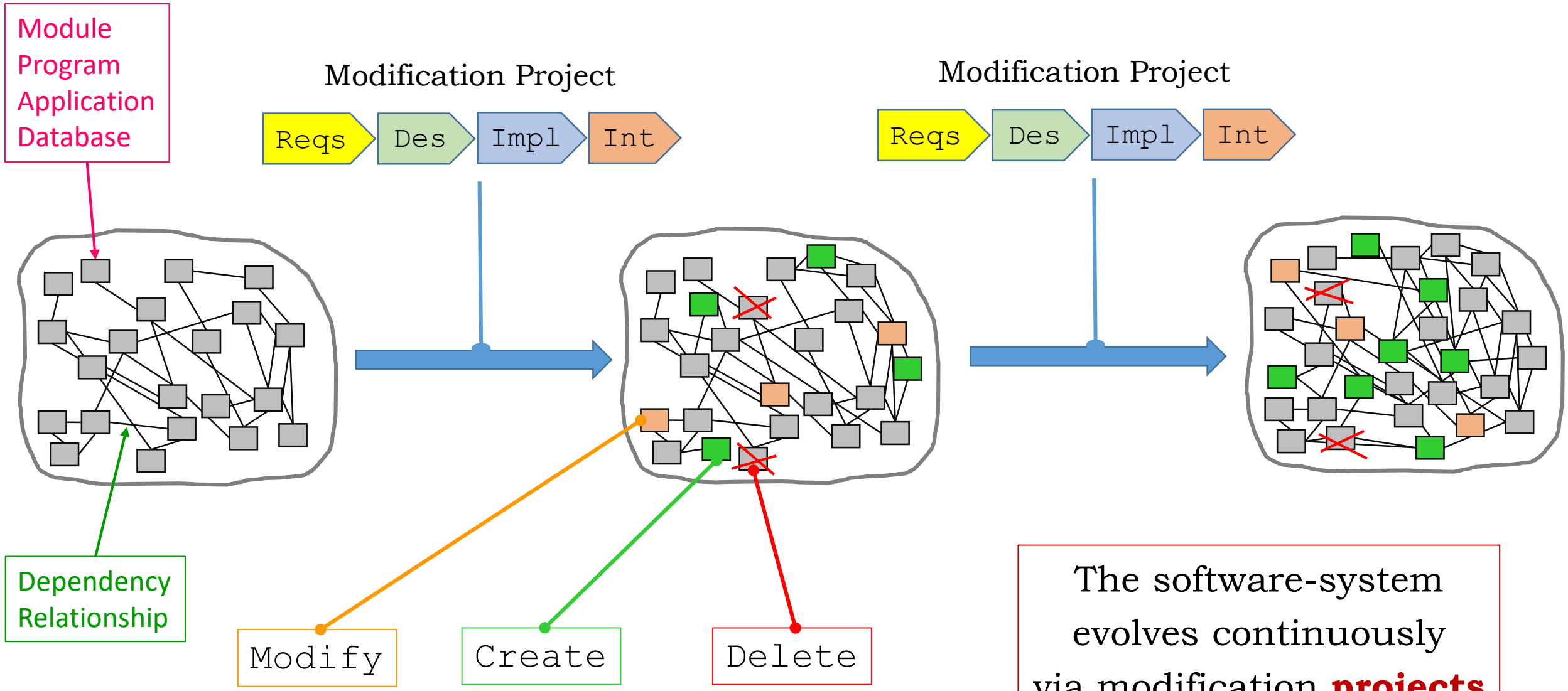
Idea or Market Need

Productive Life of the Sofware

Evolution

End of Usefulness

Future-Proof Software-Systems [Part 2]

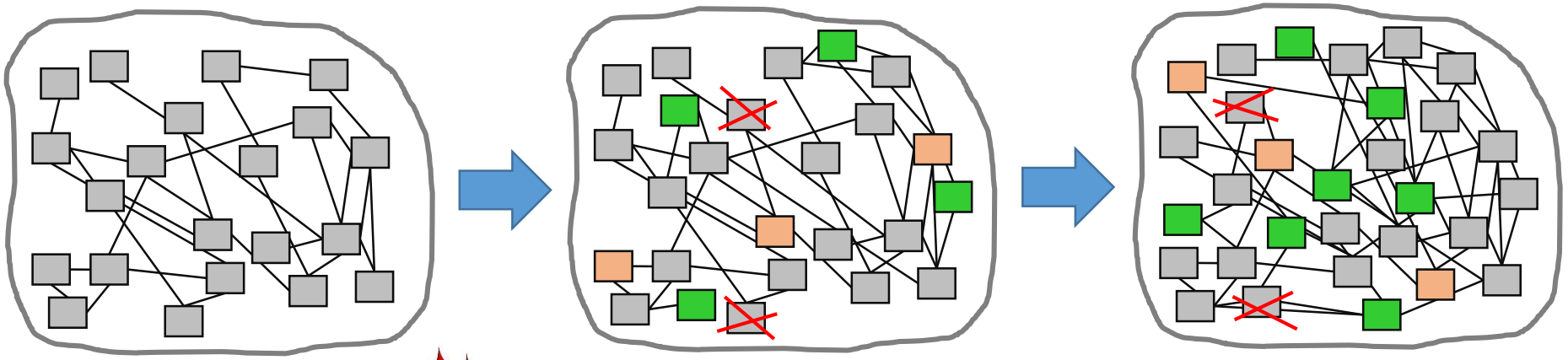


The software-system evolves continuously via modification **projects**

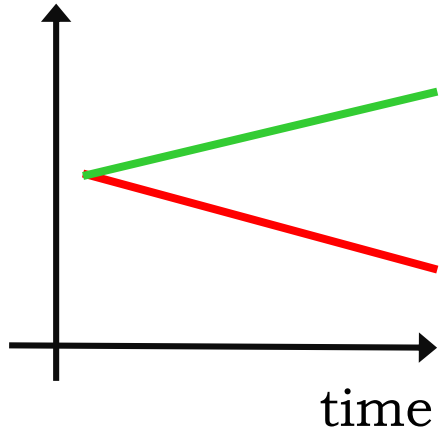
Future-Proof Software-Systems [Part 2]

Architects

Development Team



«Software Quality»



Complexity

Change

Uncertainty

Technical Debt

Architecture Erosion

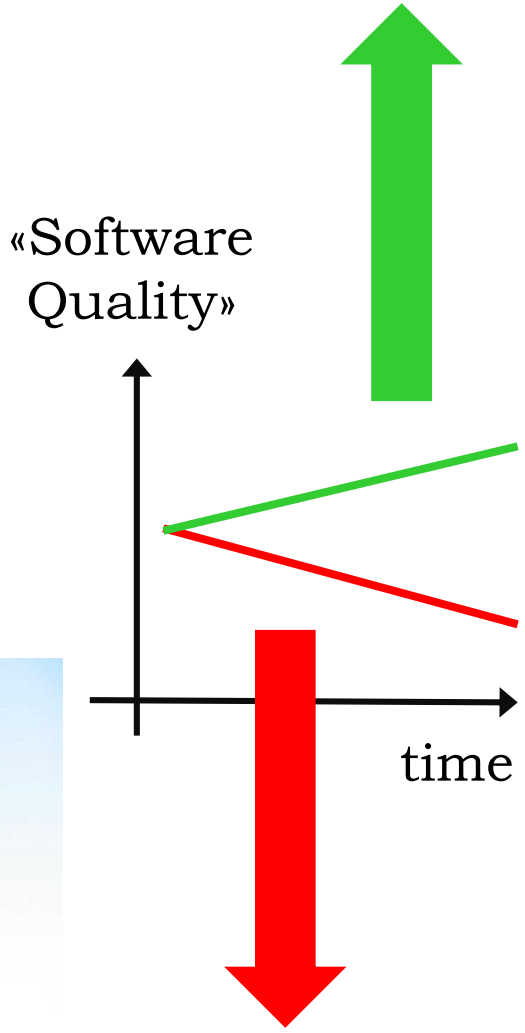
http://thoreau.colonial.net

Future-Proof Software-Systems [Part 2]

Architects

Development Teams

Managed Evolution



Evolution: Add, change, delete

Complexity

Change

Uncertainty

Technical Debt

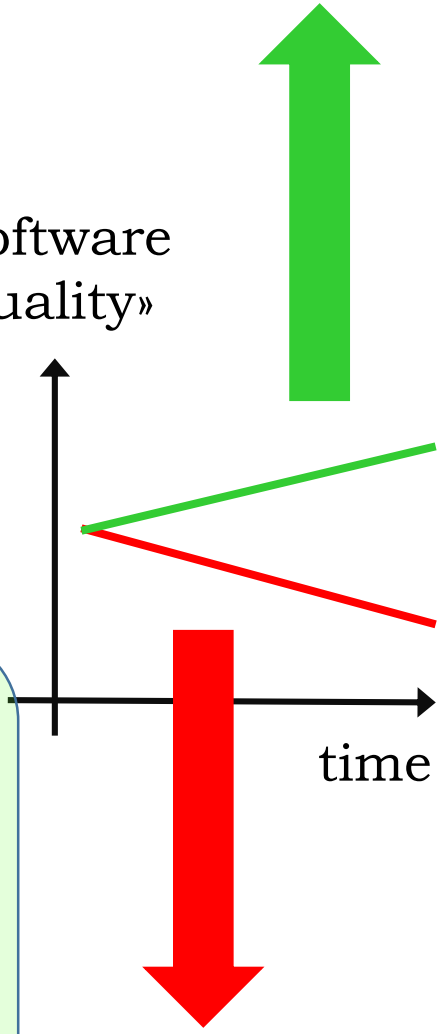
Architecture Erosion

Future-Proof Software-Systems [Part 2]

Architects

Development Teams

Managed Evolution



Evolution: Add, change, delete

Complexity

Change

Uncertainty

Technical Debt

Architecture Erosion

Technical Debt

Can Software deteriorate (erode) over time?

The next code will be directly imported from a file:

```
function X = BitXorMatrix(A,B)
%function to compute the sum without charge of two vectors

        %convert elements into usigned integers
        A = uint8(A);
        B = uint8(B);

        m1 = length(A);
        m2 = length(B);
        X = uint8(zeros(m1, m2));
        for n1=1:m1
                for n2=1:m2
                        X(n1, n2) = bitxor(A(n1), B(n2));
                end
        end
```

http://www.sueddeutsche.de

Can Software deteriorate (erode) over time?

YES

http://www.lacdp.org

The next code will be directly imported from a file:

```
function X = BitXorMatrix(A,B)
%function to compute the sum without charge of two vectors

    %convert elements into usigned integers
    A = uint8(A);
    B = uint8(B);

    m1 = length(A);
    m2 = length(B);
    X = uint8(zeros(m1, m2));
    for n1=1:m1
        for n2=1:m2
            X(n1, n2) = bitxor(A(n1), B(n2));
        end
    end
```

http://www.sueddeutsche.de

Causes:

- Technical Debt
- Architecture Erosion

Technical Debt

Definition:

Technical debt in an IT-system is the result of all those necessary things that you choose *not to do now*, but will impede future evolution if left undone

Ward Cunningham, 2007

Technical Debt:
is generated (mostly)
by ***internal*** factors

Causes of Technical Debt:

- Architecture Erosion
- Disruptive technology
- **Accumulation of mistakes + shortcuts (e.g. breaking partitions)**
- Dead code (missed explementations)
- **Bad (or ignored) programming best practices & guidelines**
- **Violation of Architecture Principles, e.g. unmanaged redundancy**
- Deferred refactoring
- Progress in software-engineering (e.g. programming languages)
- Careless or skipped upgrades
- **Missing or bad documentation**

… and some more

http://www.spyderonlines.com

Technical debt sneaks into the system

– some time seen, some time unseen

The continuous accumulation of technical debt
is many times justified by the statement:
«*we know we should do it differently – but there is no time now – we will fix it later*» *(… and forget about)*
is a massive danger for any IT-system

http://worldartsme.com

http://credit-collections.ca

The first kind of technical debt is the kind that is incurred ***unintentionally***

For example, a design approach just turns out to be error-prone or a junior programmer just writes bad code. This technical debt is the result *of doing a poor job*.

Steve McConnell, 2007

This includes decisions like "We don't have time to reconcile these two databases, so we'll write some glue code that keeps them synchronized for now and reconcile them after we ship"

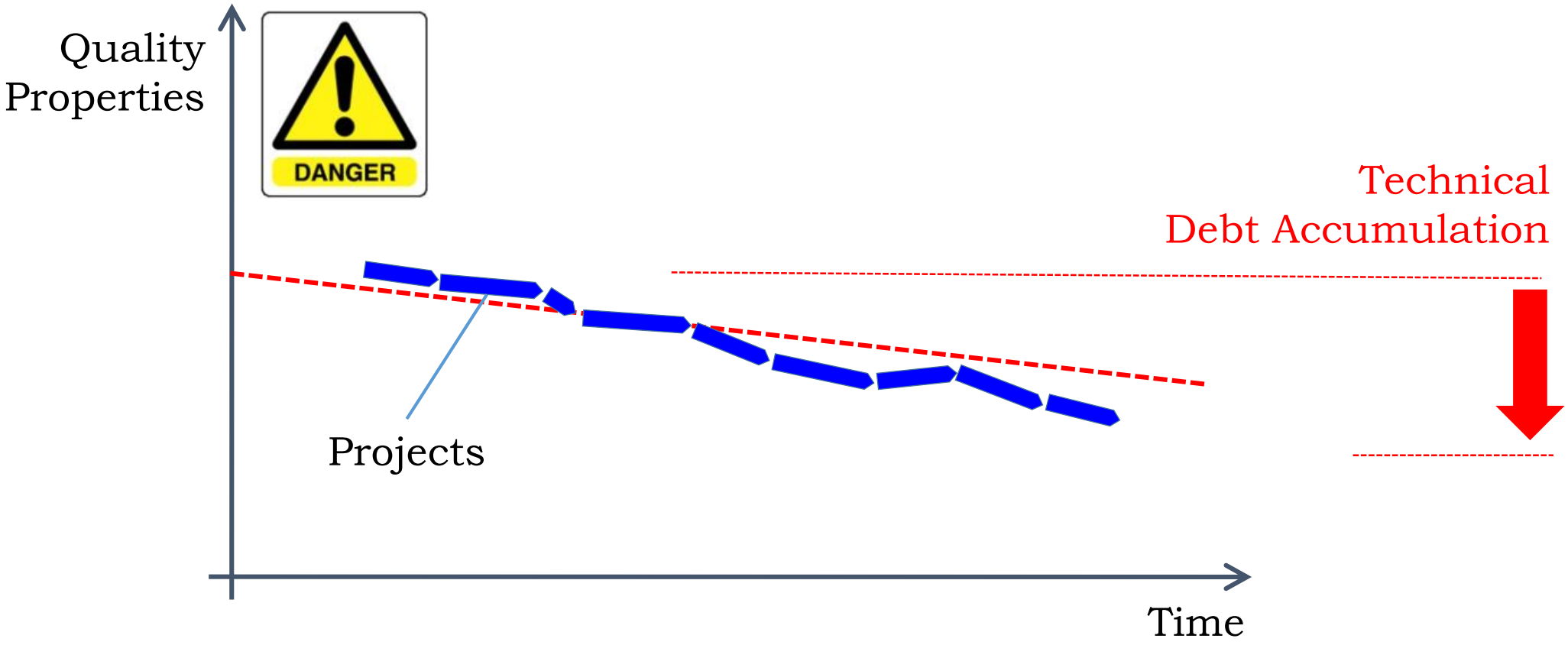
http://credit-collections.ca

The second kind of technical debt is the kind that is incurred ***intentionally***

This occurs when an organization makes a conscious decision to *optimize for the present rather* than for the future

Steve McConnell, 2007


DEBT


Quality Properties
DANGER
Technical Debt Accumulation
Projects
Time

… worrying research:

http://dandev91.wordpress.com/

Cost of one source line of embedded systems code:

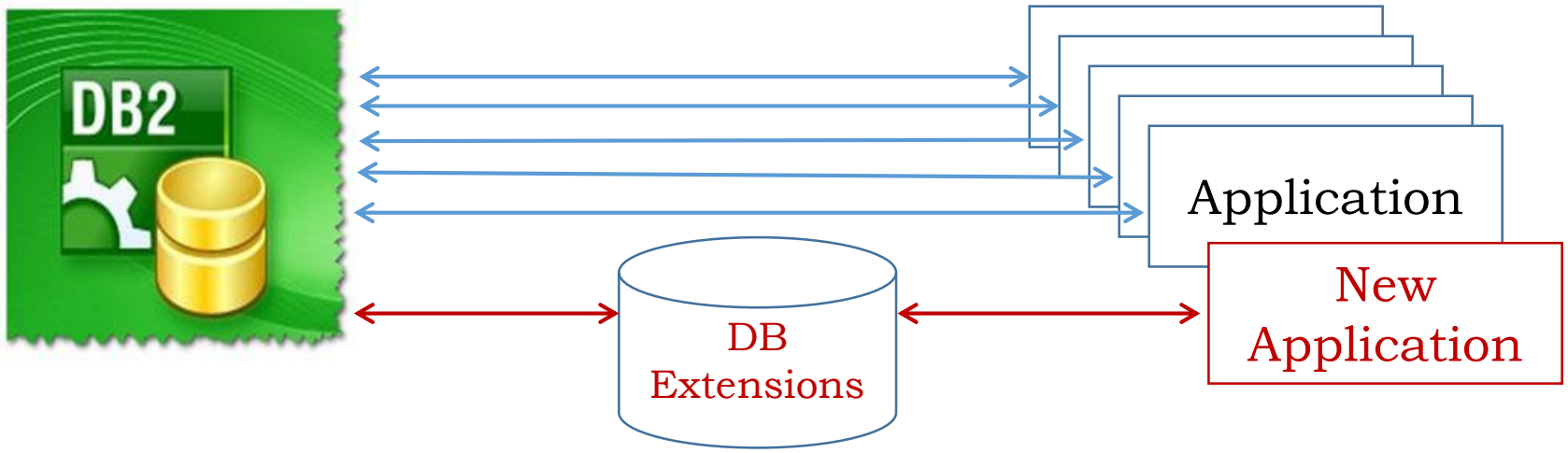
€ 15.00 … € 40.00

Average Technical Debt in each source line of embedded systems code:

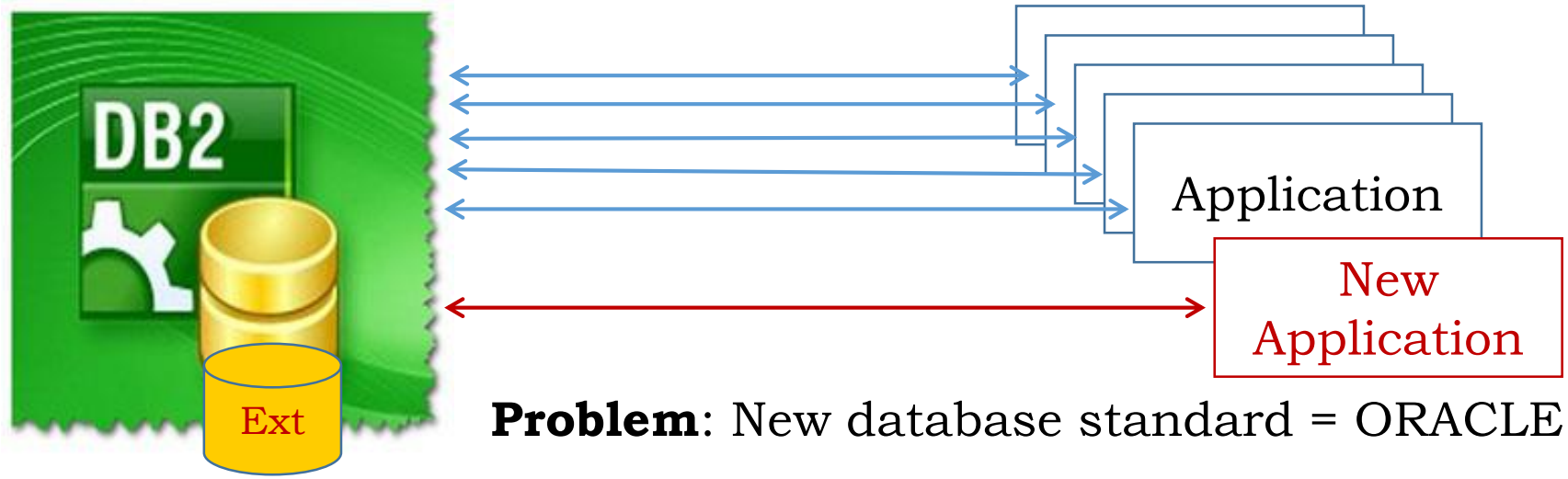
€ 2.70

[Deloitte Consulting LLP: Tech Trends 2014]

Example: Database Extension (1/3)



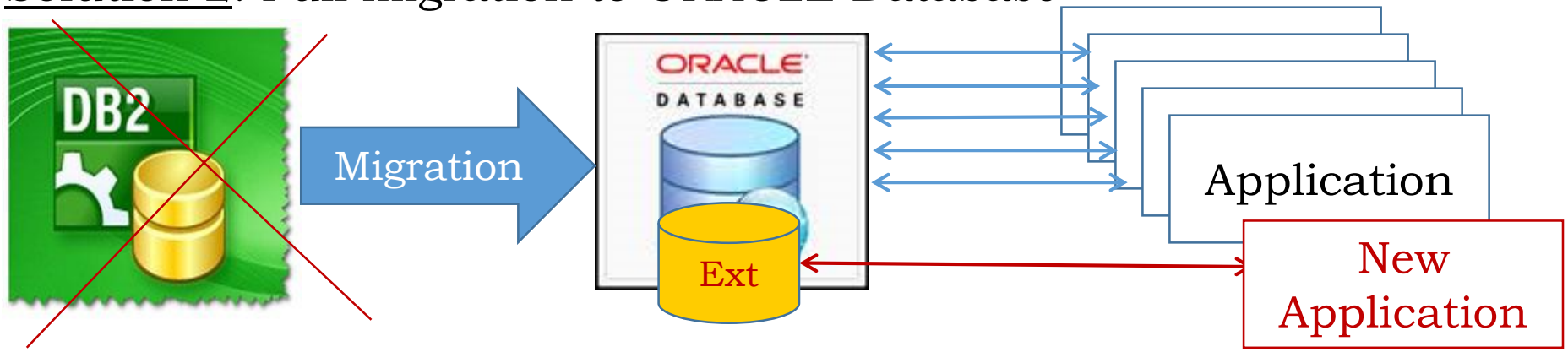
Solution 1: Extend **DB2** Database



Problem: New database standard = ORACLE

Example: Database Extension (2/3)

Solution 2: Full migration to ORACLE Database



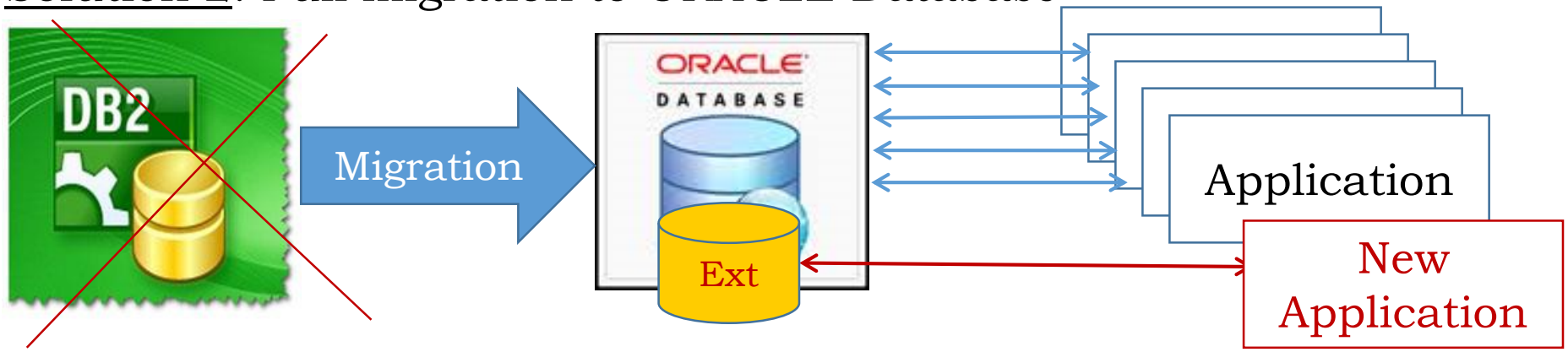
Solution 3: Bridging

Bridging:
- Synchronization
- Alignment

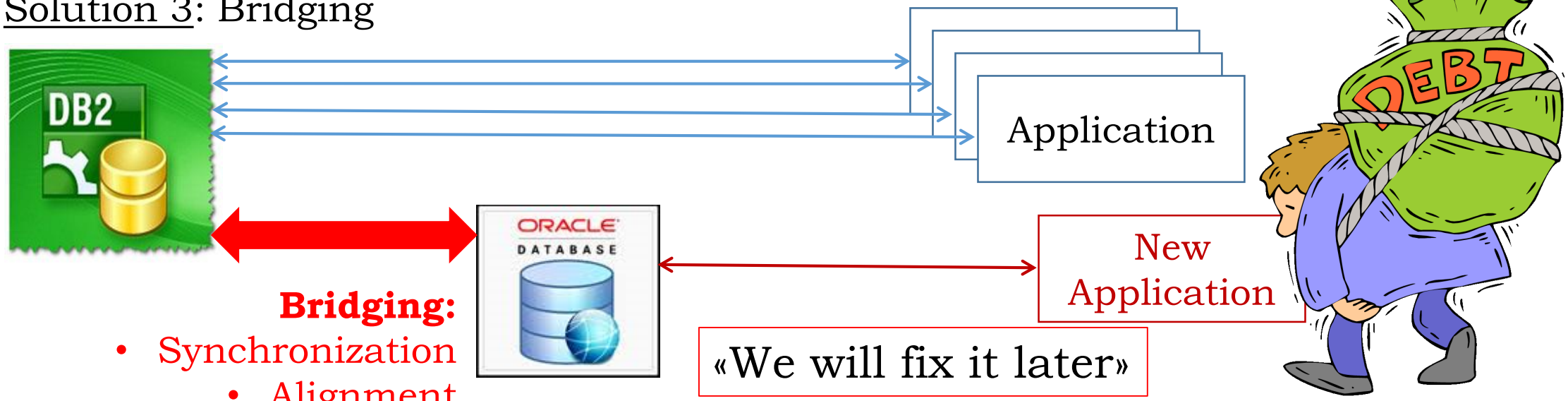
«We will fix it later»

Example: Database Extension (3/3)

Solution 2: Full migration to ORACLE Database



Solution 3: Bridging



Bridging:
- Synchronization
- Alignment

«We will fix it later»

Example: 5th Language (1/3)

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 **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 identified by **language code** and **text code**

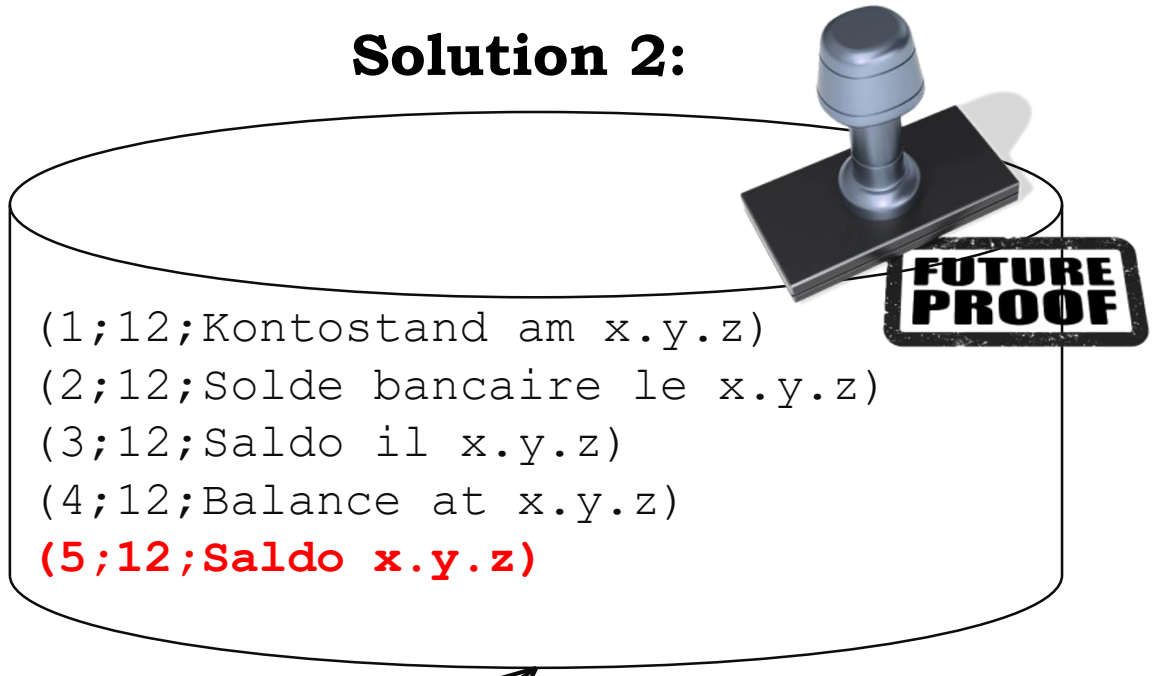
Example: 5th Language (2/3)

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)

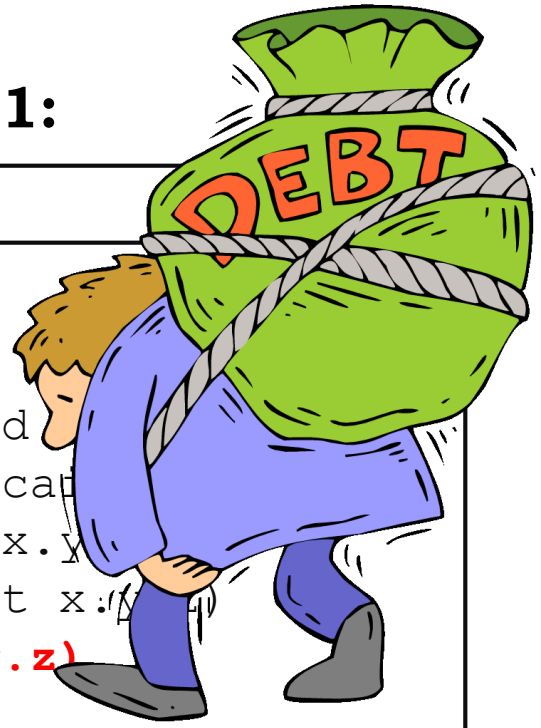
Solution 2:



Create a **central language file** and export it to all programs

Example: 5th Language (3/3)

Solution 1:

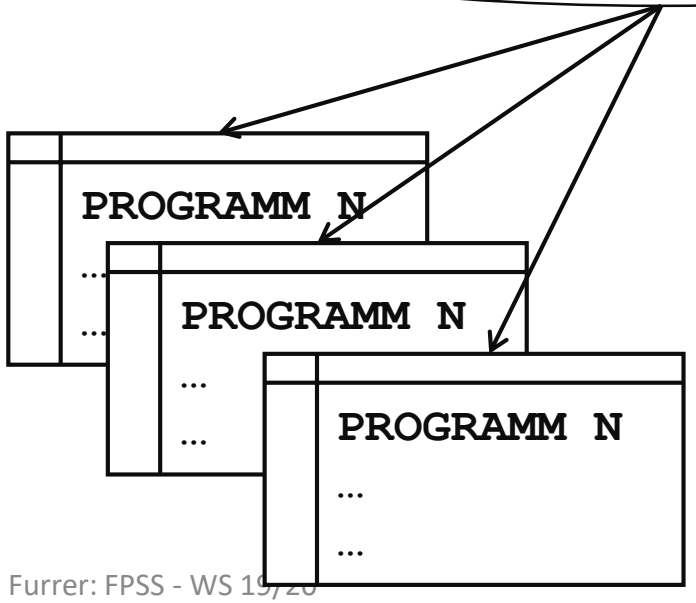



```
PROGRAMM N
…
(1;12;Kontostand
(2;12;Solde banca
(3;12;Saldo il x.y
(4;12;Balance at x.y.z)
(5;12;Saldo x.y.z)
…
```

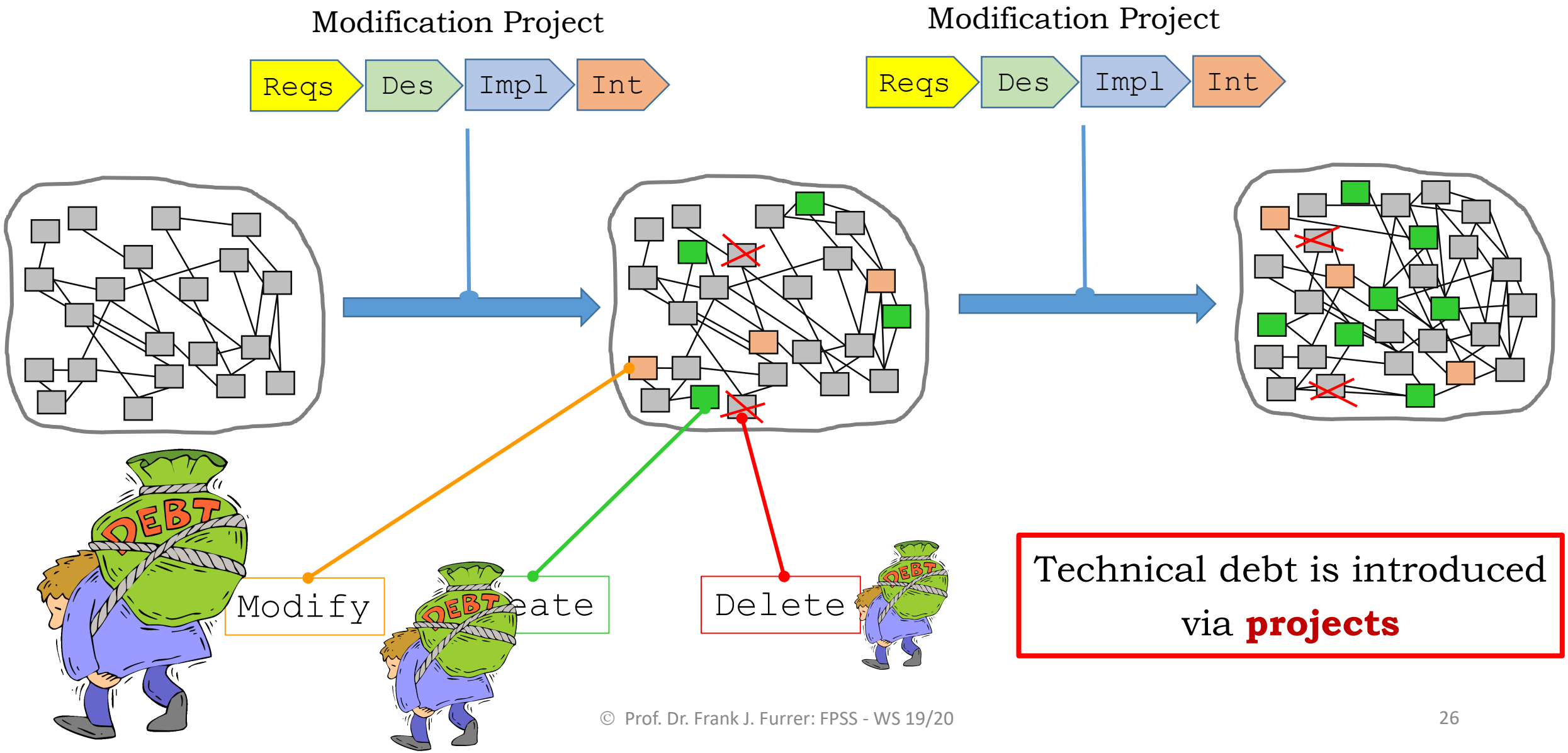
Individually modify **all** the programs which need Spanish output (ca. 5‘000 applications)

Solution 2:


```
(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)
```



Create a **central language file** and export it to all programs



Technical debt is introduced via **projects**

What can we do against the accumulation of technical debt?

Quality Properties

Fight **Technical Debt Accumulation**

Allow additional:
- **Money (Δ€, Δ$)**
- **Time (ΔT_{Pn})**

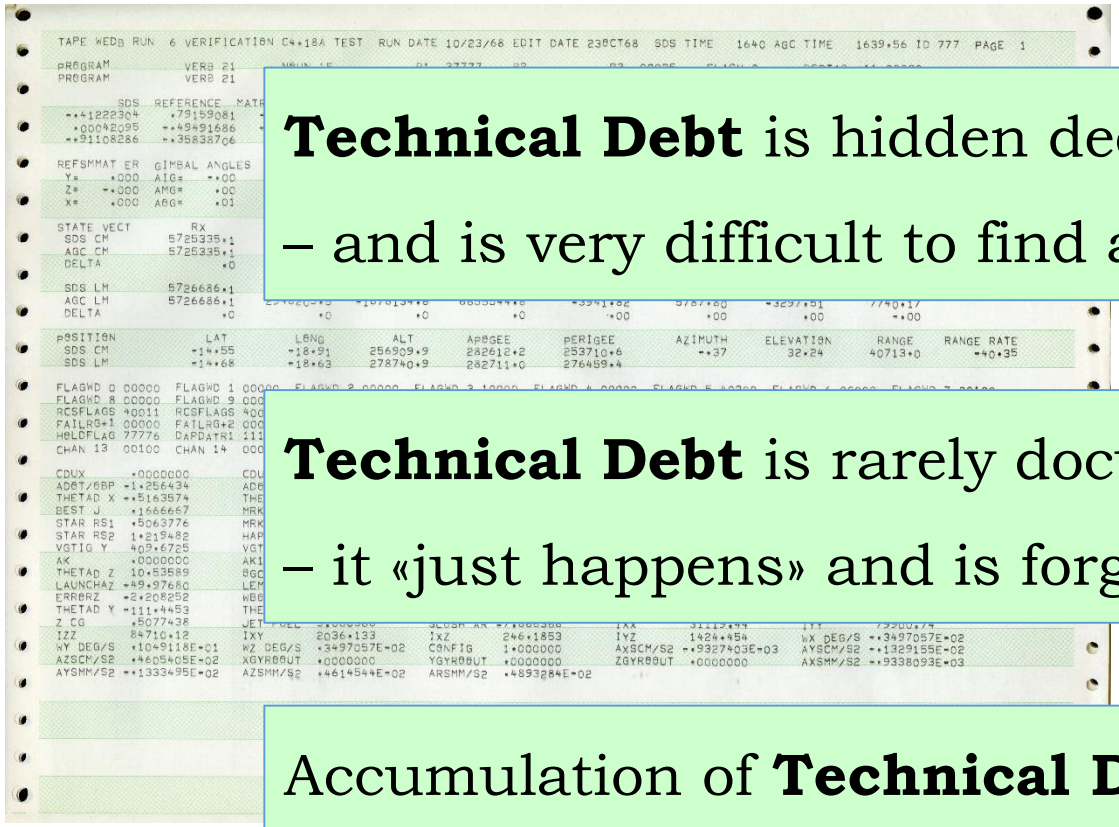
in each project

Project P_n

Project P_n

Business Value

Gain of Business Value



Technical Debt is hidden deep in the program code

– and is very difficult to find and to eliminate

Technical Debt is rarely documented

– it «just happens» and is forgotten

Accumulation of **Technical Debt** in a software system is a very strong long-time risk

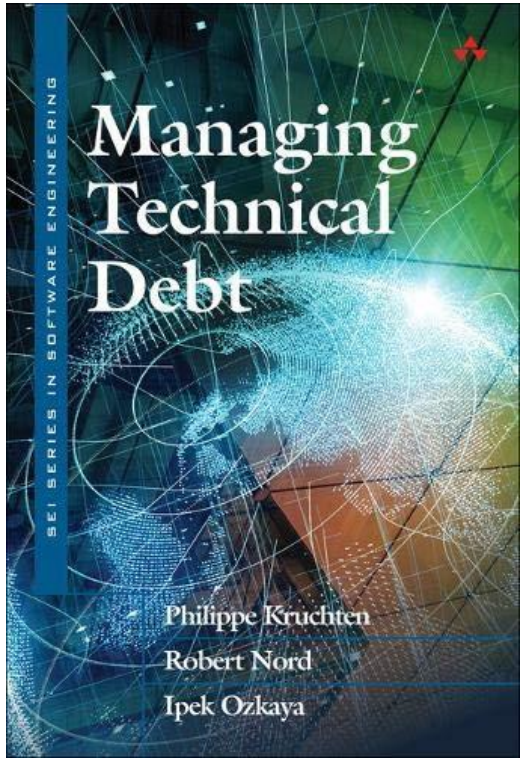
- and its *elimination* is costly and difficult

«we will fix it later»

https://www.behance.net

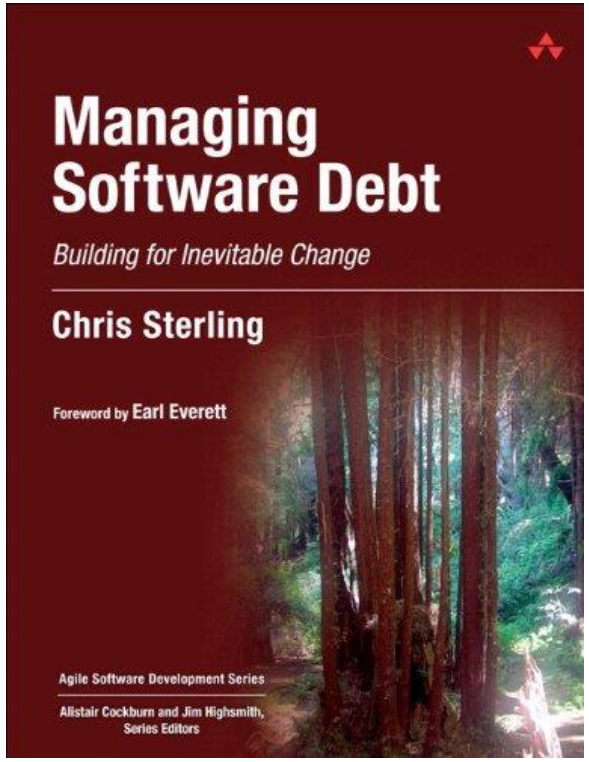
… is the direct way to (software) hell

Textbook



Philippe Kruchten, Robert Nord, Ipek Ozkaya:
Managing Technical Debt – *Reducing Friction in Software Development*
Pearson Education, USA, 2019. ISBN 978-0-135-64593-2

Textbook



Chris Sterling:
Managing Software Debt – *Building for Inevitable Change*
Pearson Education, USA, 2013. ISBN 978-0-321-94861-8

Architecture Erosion

«Architecture»

https://www.theurbandeveloper.com

«Architecture»: We know the term from building, e.g. **towns**

… it means defining, planning, and drawing:

- the **buildings**
- the connecting **infrastructure** (roads, water supply, electricity, …)
- the **services** (emergency services, public transport, …)

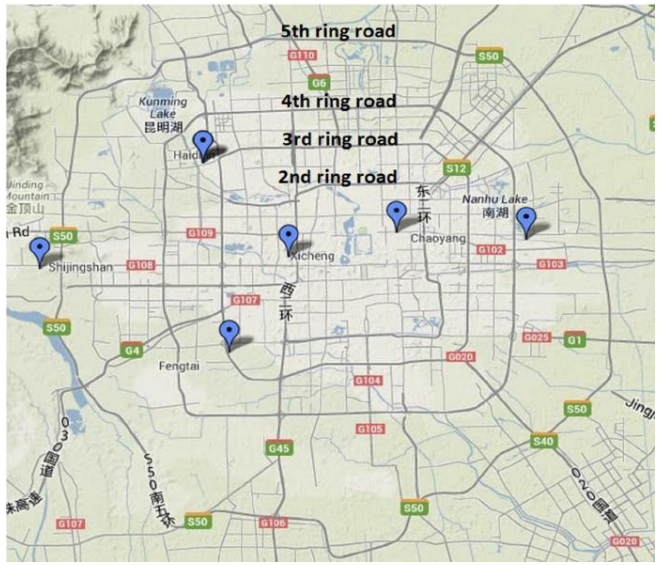
⇒ so that the town functions in a satisfactory way for the inhabitants and visitors

«Architecture»

https://www.theurbandeveloper.com

Layout of roads, rails, bridges, buildings, …

= Structure



https://www.researchgate.net

Quality Poperties:

Emergency Services

Public Security

etc.

What does «Architecture» mean in software-engineering?

Structure

IT Architecture Definition:

"The fundamental *organization* of a system embodied in its *parts*, their *relationships* to each other and to the environment, and the *principles* guiding its design and evolution"

[adapted from IEEE00]

Elements

Principles

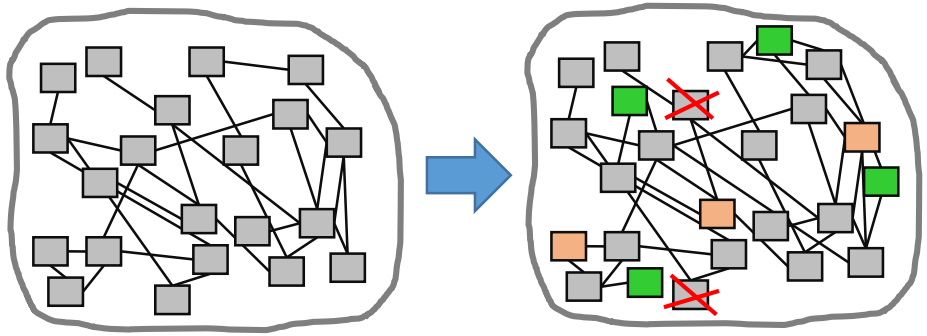
Future-Proof Software-Systems [Part 2]

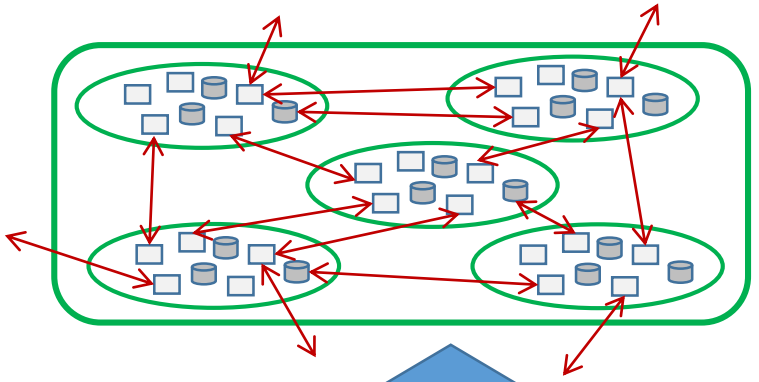
Parts

Relationships

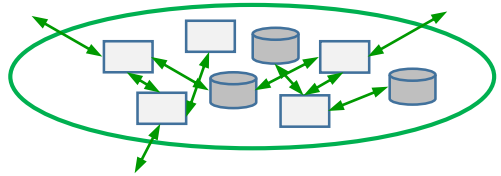
Organization

Principles

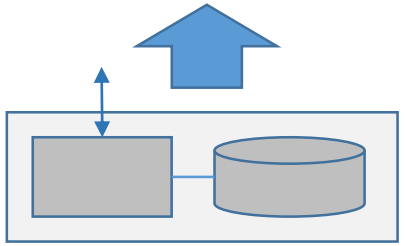




Application Landscape



Application



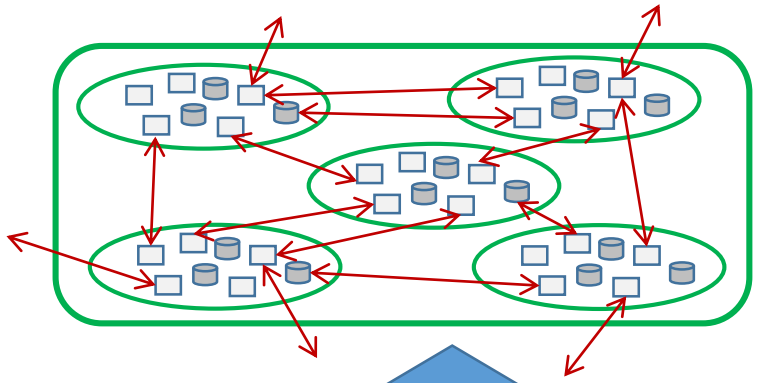
Component

```
The next code will be directly imported from a file:
function X = BitXorMatrix(A,B)
%function to compute the sum without charge of two vectors

    %convert elements into usigned integers
    A = uint8(A);
    B = uint8(B);

    m1 = length(A);
    m2 = length(B);
    X = uint8(zeros(m1, m2));
    for n1=1:m1
        for n2=1:m2
            X(n1, n2) = bitxor(A(n1), B(n2));
        end
    end
```

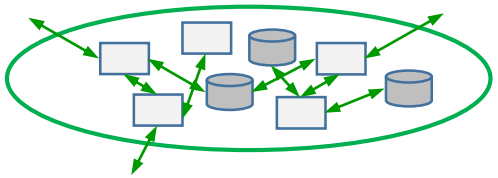
Program, Module

Future-Proof Software-Systems [Part 2]

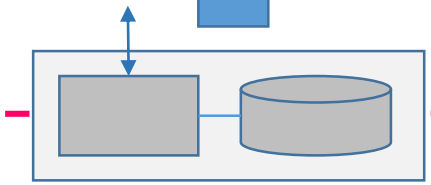
Technology-**in**dependent



Application Landscape architecture



Application **architecture**



Component **architecture**

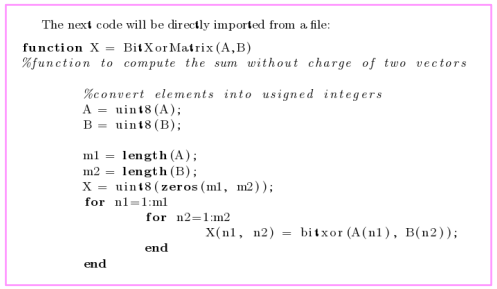
Component **design**

Architecture

Design

Technology-dependent

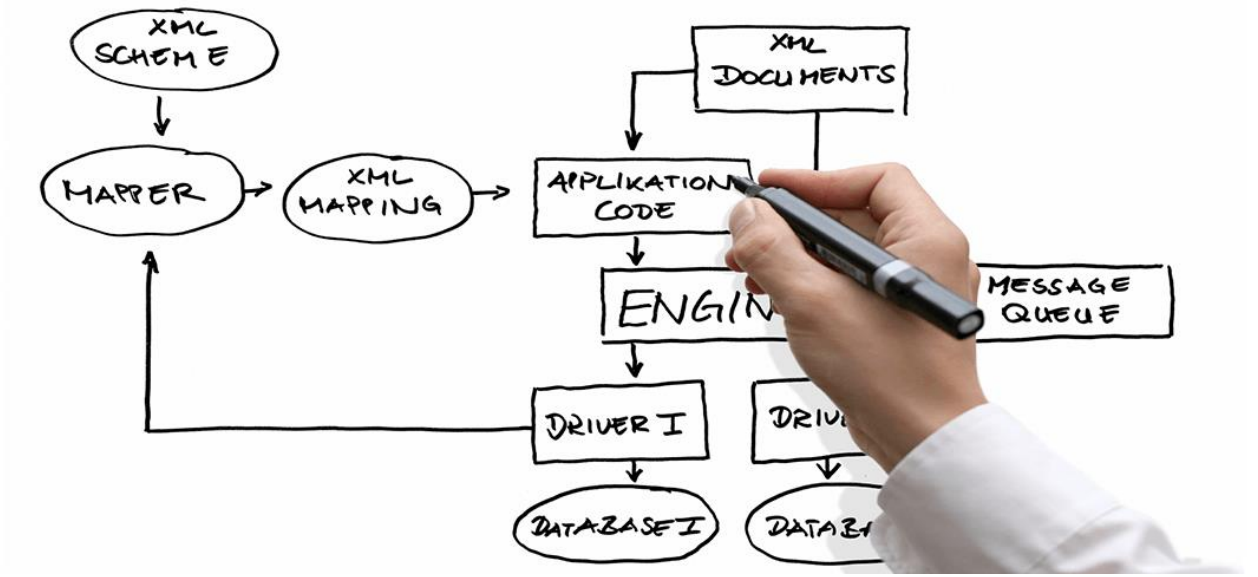
- Operating System
- Protocols
- Progamming Language
- DB-System
- …



```
The next code will be directly imported from a file:
function X = BitXorMatrix(A,B)
%function to compute the sum without charge of two vectors

        %convert elements into usigned integers
        A = uint8(A);
        B = uint8(B);

        m1 = length(A);
        m2 = length(B);
        X = uint8(zeros(m1, m2));
        for n1=1:m1
                for n2=1:m2
                        X(n1, n2) = bitxor(A(n1), B(n2));
                end
        end
```

Program, Module **design**

Lookahead: Importance of Software-Architecture



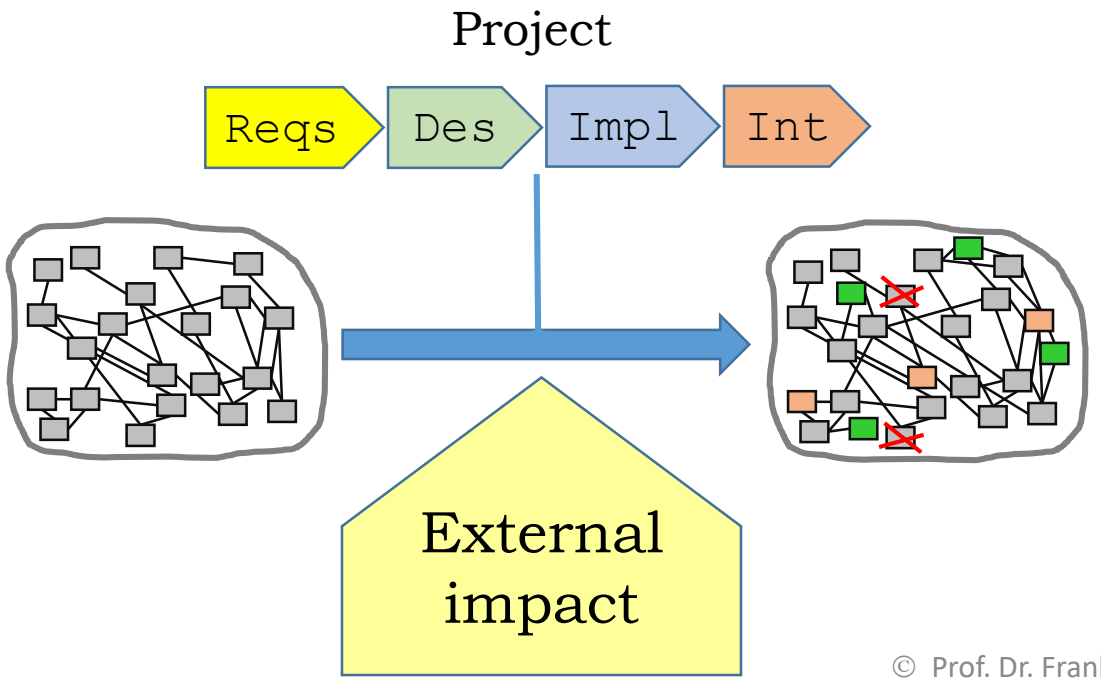
Later in the lecture we will demonstrate:

***Software-Architecture* is the single most important factor**

for future-proof software-systems

Every system has an architecture

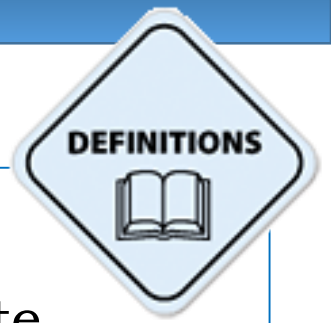
- If the architecture is **implicit** (not engineered, not visible, not documented) - *we can hardly influence it*
- If the architecture is **explicit** (continuously engineered, well documented, respected by all stakeholders) – *we can explain it, reason about it, improve it*



Architecture erosion

Architecture deterioration

Architecture improvement



Definition: Architecture Erosion

Architecture erosion is the process where an initially well-designed, adequate architecture of a software-system is gradually destroyed by the activities of evolution and maintenance of the software-system.

http://www.spyderonlines.com

Architecture erosion sneaks into the system

– some times seen, some times unseen

… but it gradually destroys the system!

Architecture Erosion

https://www.linkedin.com

Software systems are under constant pressure to adapt to changing requirements, new technologies and to the environment.

Often, **modifications** made to the software system over a period of time **damage its structural integrity** and violate its design principles – *the initial, good architecture continuously erodes!*

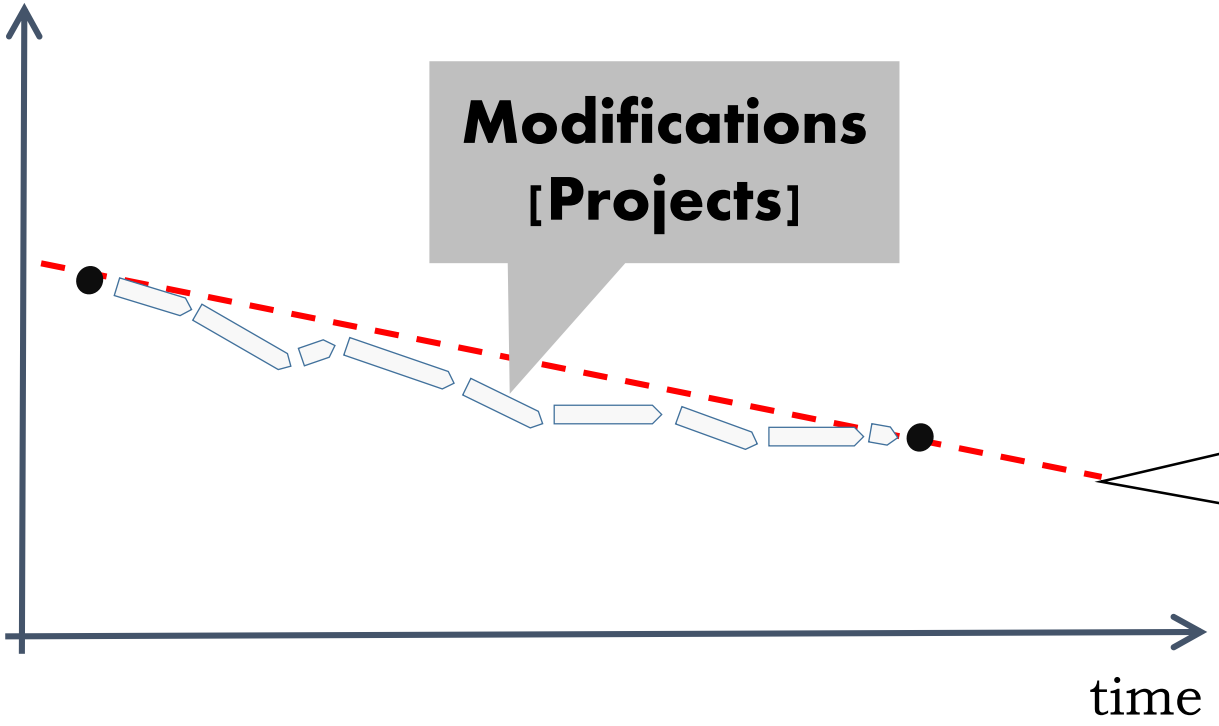
Architecture Erosion

https://www.linkedin.com

What is the impact of architecture erosion?

- Merciless degradation of quality *properties*
- More and more difficult/expensive/slow to *modify*
- Hard/costly to *maintain*

http://thoreau.colonial.net/Students/EricksonHoyt/erosion



The force of **erosion** continuously **reduces** architecture quality (and other quality properties)

http://thoreau.colonial.net/Students/EricksonHoyt/erosion

Architecture Erosion:

Any IT-architecture is continuously *degenerating* due to many factors:

- **Accumulation of technical debt**
- SW Paradigm changes (e.g. SOA)
- New laws & regulations
- New standards (e.g. interoperability standards)
- New technology platforms (e.g. Web Services)
- Introduction of new architecture principles
- Complexity increase
- New malicious activities

… and some more

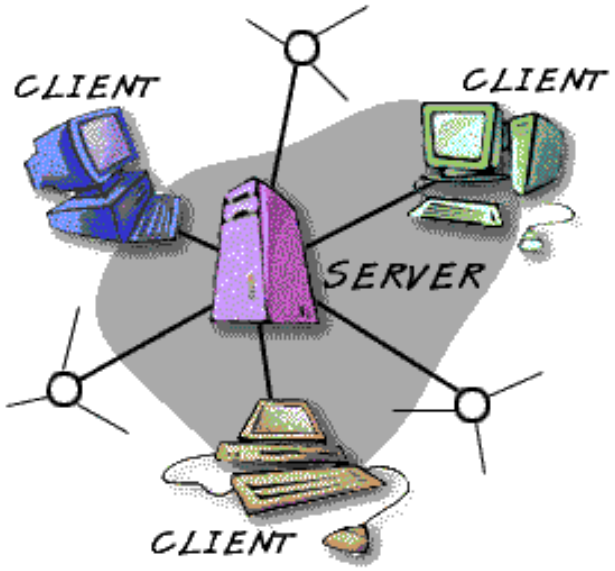
Architecture Erosion is generated by *internal & external* factors

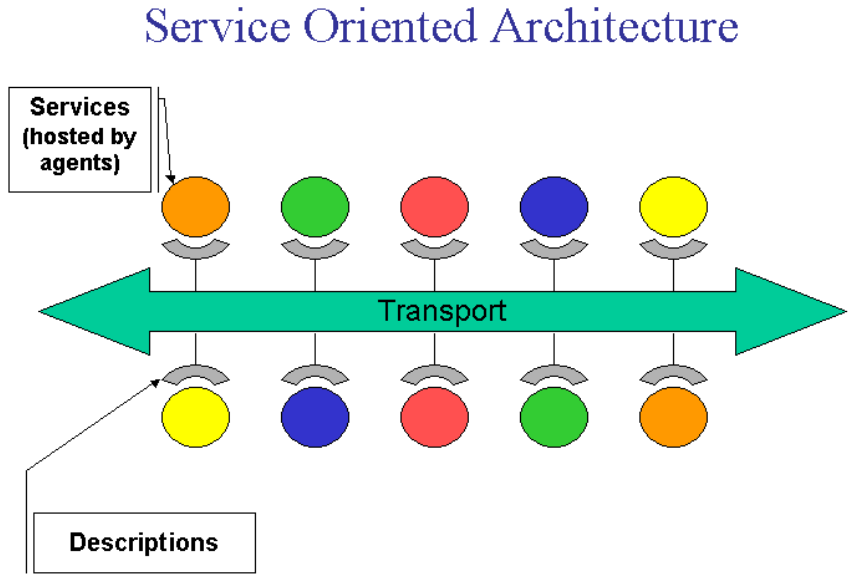
External causes of architecture erosion:

In the history of software engineering there were many disruptive *paradigm changes* which led to massive **architecture erosion**, e.g.:

- ✓ Procedural programming → Object-Orientation
- ✓ Local processing → distributed processing
- ✓ Monoliths → Client-Server architecture
- ✓ Client-Server architecture → Service-oriented architecture
- ✓ Remote procedure calls/CORBA → Web services
- ✓ Microservices
- ✓ Programming by people → Model-based code generation
- ✓ … and more to come

Example: Client-Server architecture → Service-oriented architecture (1/2)

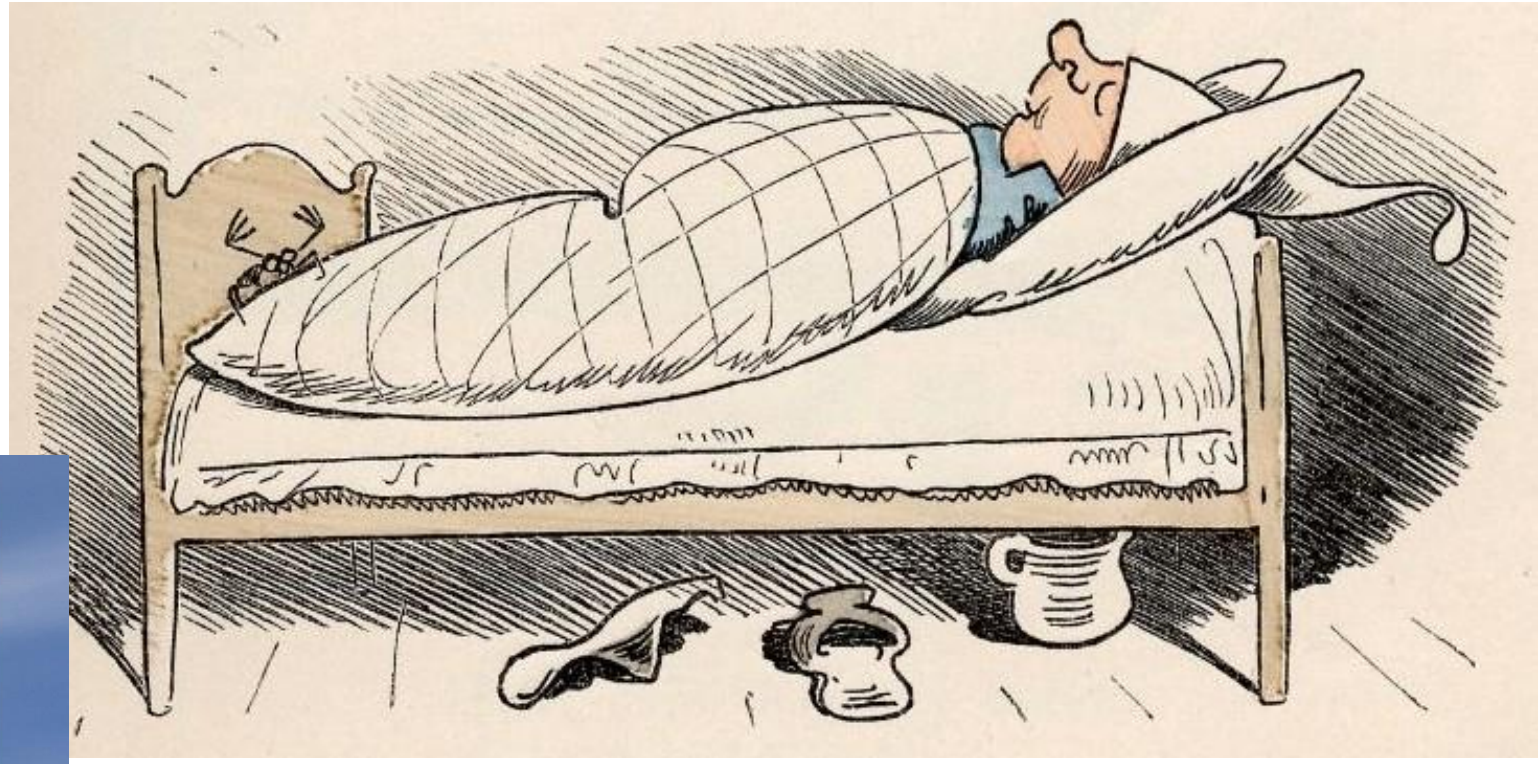




Gain of Changeability: ≈ 30%

[Estimate for large commercial systems]

Example: Client-Server architecture → Service-oriented architecture (2/2)



«Overnight» large numbers of applications became technologically outdated, i.e. architecturally **eroded**!

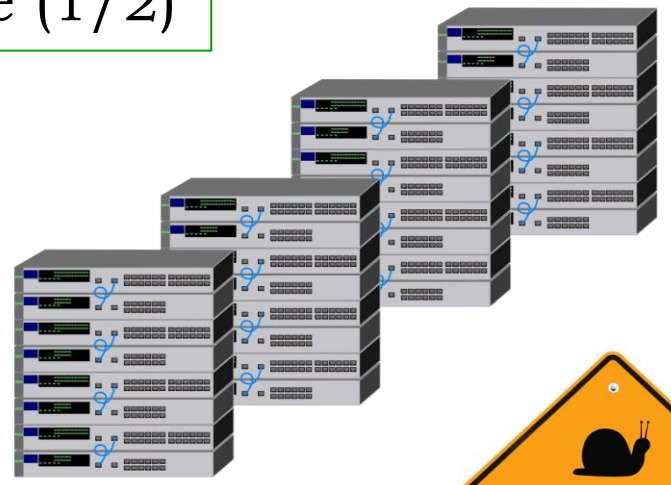
http://www.hellenicaworld.com

Example: Mainframe ⇒ Server architecture (1/2)

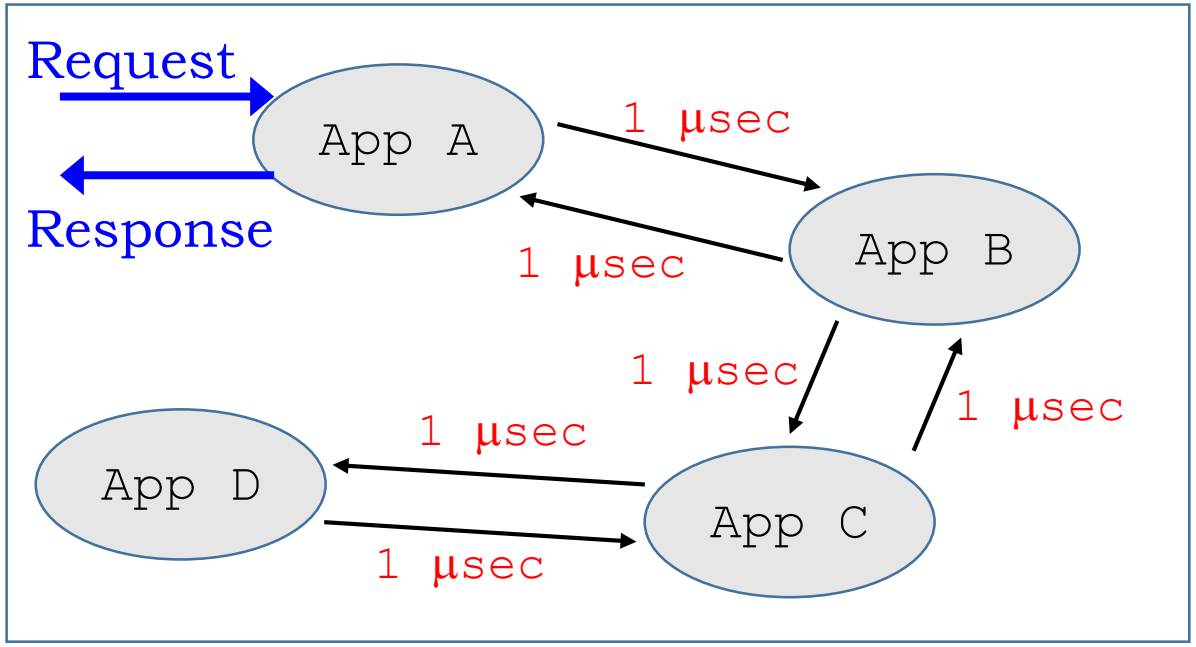
https://cdn.mos.cms.futurecdn.net

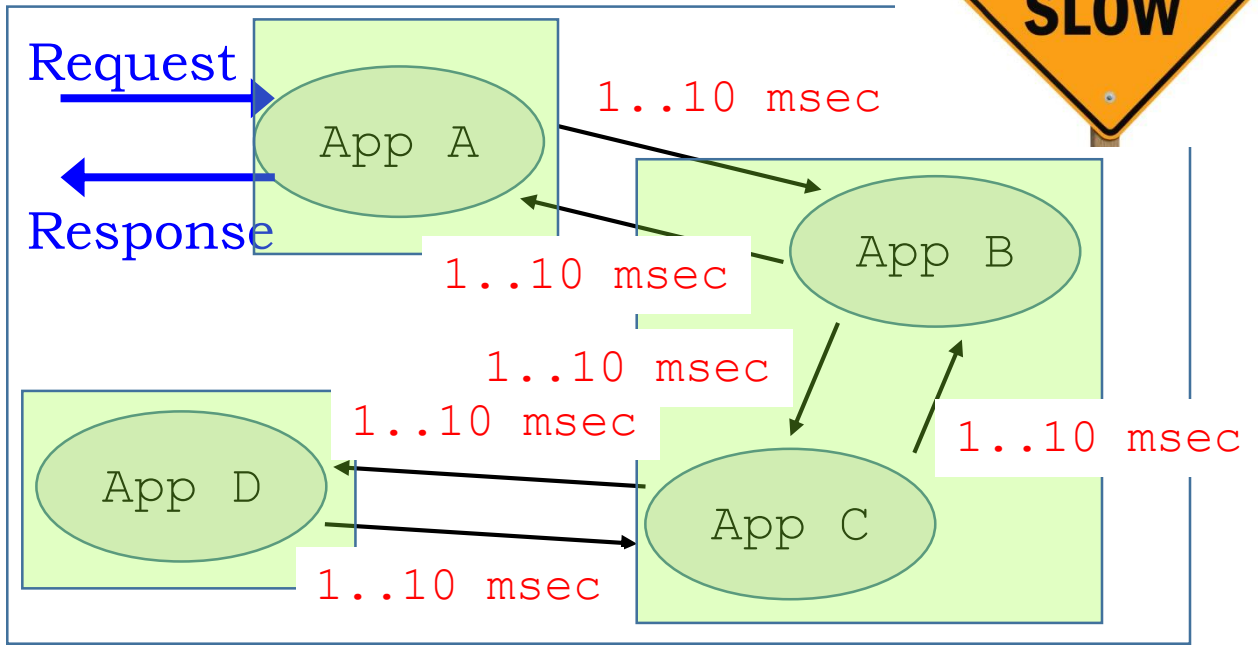
IBM Mainframe Architecture

Distributed Server Architecture



http://weclipart.c





Example: Mainframe ⇒ Server architecture (2/2)

Communications Latency

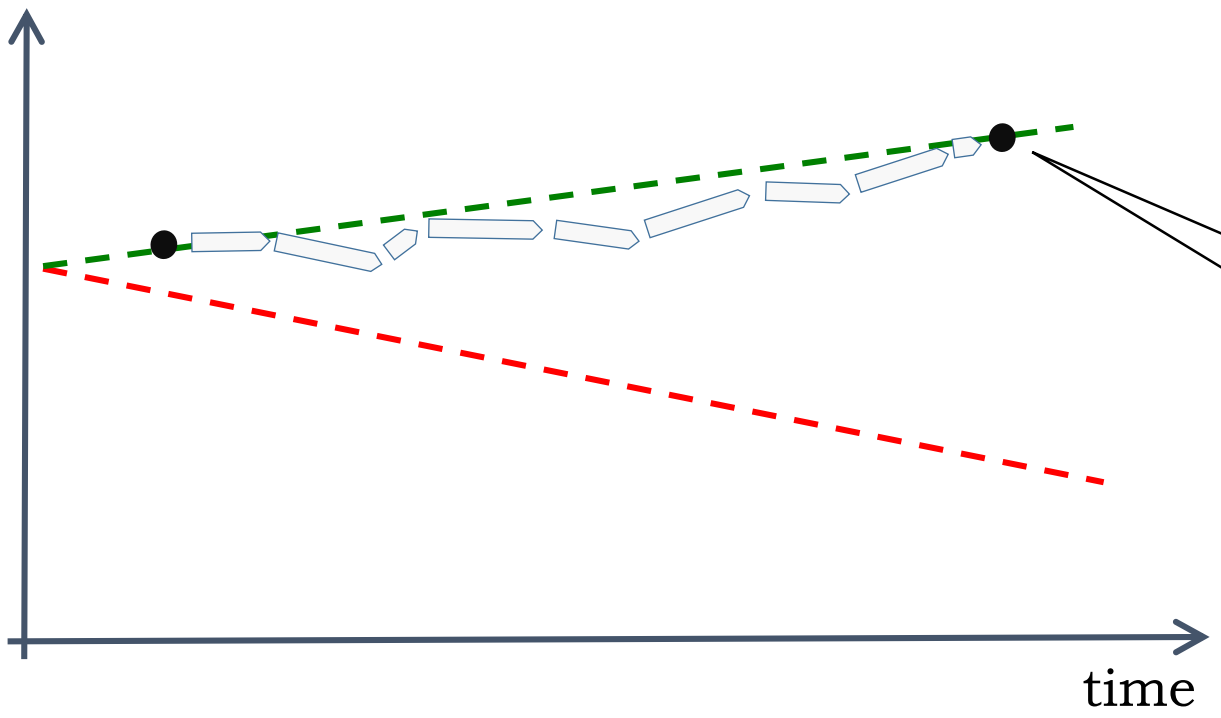
= Time lost during transmission

Mainframe Infrastructure: *Communications Latency* is **NOT** a design criterium

Server Infrastructure: *Communications Latency* is a **HEAVY** design criterium

⇒ MASSIVE architecture consequences

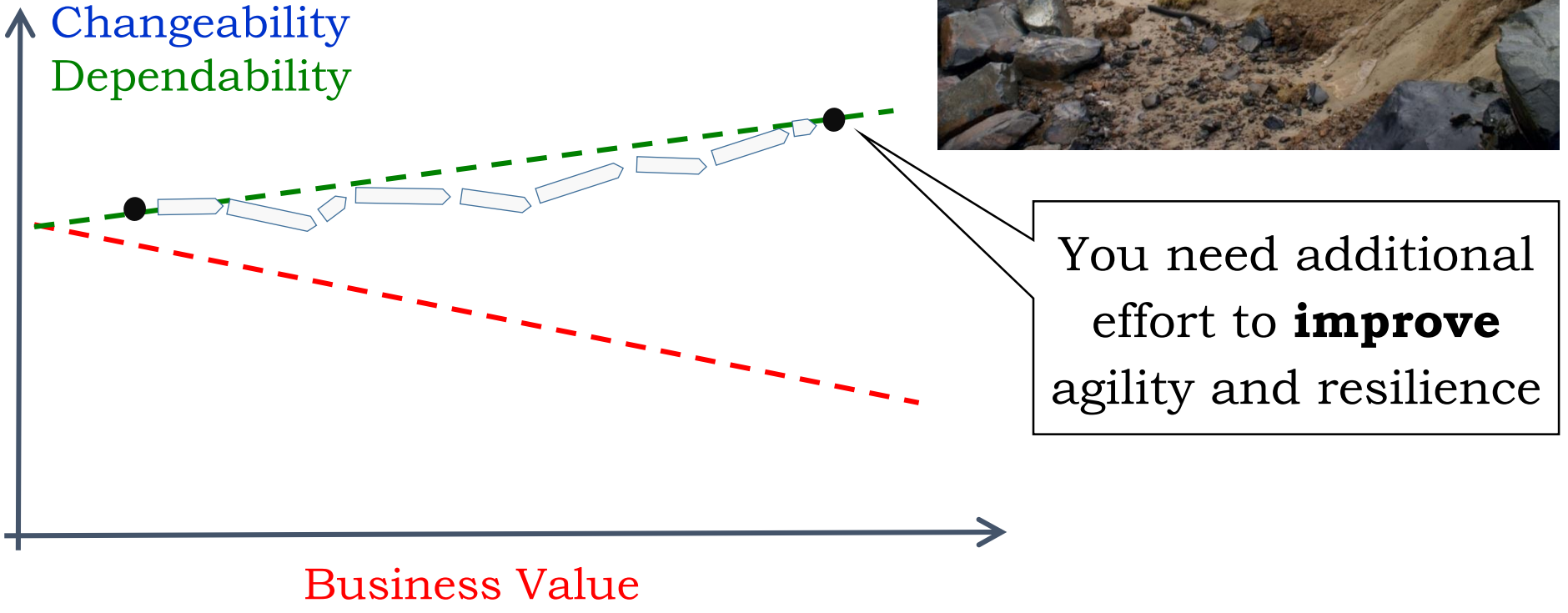
http://thoreau.colonial.net/Students/EricksonHoyt/erosion



We need *additional* effort to **improve** architectural quality

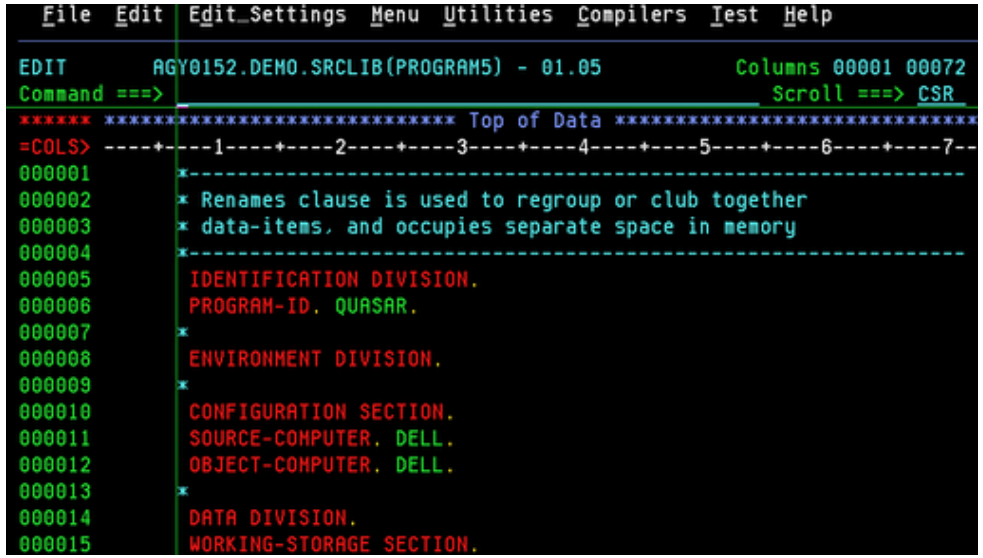
Architecture Erosion

http://thoreau.colonial.net/Students/EricksonHoyt/erosion



You need additional effort to **improve** agility and resilience

Example: COBOL Programming



COBOL (*CO*mmon *B*usiness-*O*riented *L*anguage, 1959) is a compiled programming language designed for business, finance, and administrative systems use.

Since 15 years COBOL is not fit for new applications

Technical Erosion: **Replacement**?

Gartner 1997:

Around *200 billion lines* of COBOL code are in live operation

75% of the world's business data, and 90% of financial transactions, are processed in COBOL

http://en.wikipedia.org/wiki/COBOL

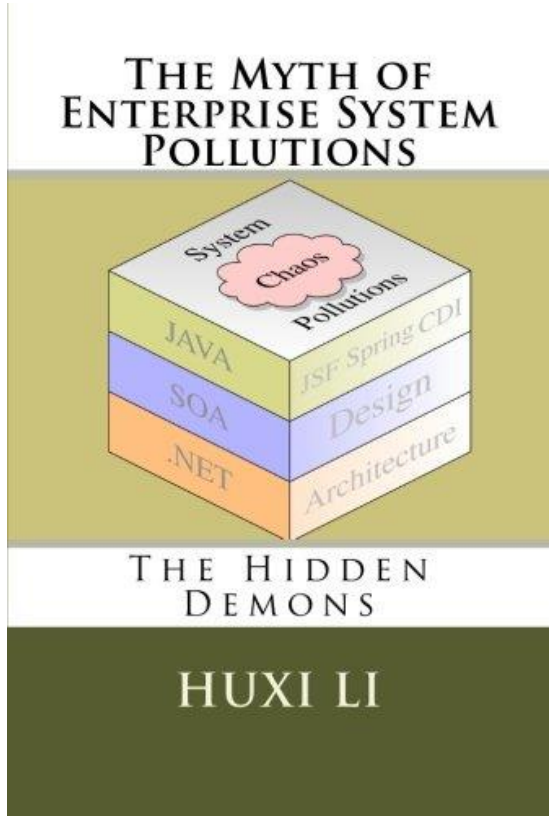
Is there a **medicine** against architecture erosion and the accumulation of technical debt?

http://www.theparentreport.com

⇒ **Managed Evolution**: Management & Funding

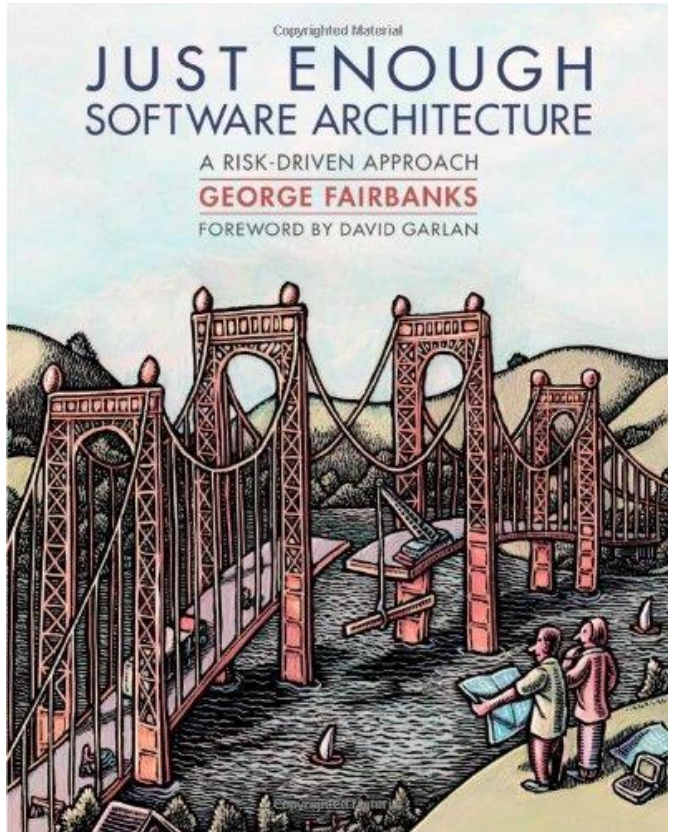
⇒ **IT-Architecture**: Technical Integrity & Principles

Textbook



Huxi Li:
The Myth of Enterprise System Pollutions – *The Hidden Demons*
CreateSpace Independent Publishing Platform, 2013. ISBN 978-1-4812-8050-1

Textbook



George Fairbanks:
Just Enough Software Architecture – *A Risk-Driven Approach*
Marshall & Brainerd, Boulder CO, USA, 2010.
ISBN 978-0-9846181-0-1

Managed Evolution

Some Definitions

Software Properties: Functional and Non-Functional (= Quality Properties)

http://creepypasta.wikia.com

http://efdreams.com

Functionality:
- Fly the plane autonomously

http://www.slate.com

Non-functional properties:
- Handle errors & malfunctions
  ⇒ **safety**
  - etc.

Software Properties: **Functional** and Non-Functional (= Quality Properties)

http://efdreams.com

Functionality:
- Fly the plane autonomously

http://creepypasta.wikia.com

- Understand Flight Conditions
- Adhere to Flight Plan
- Operate Engines, Rudder, Flaps etc.
- Autonomously fly long distance
- Support or autonomously land the plane
- …

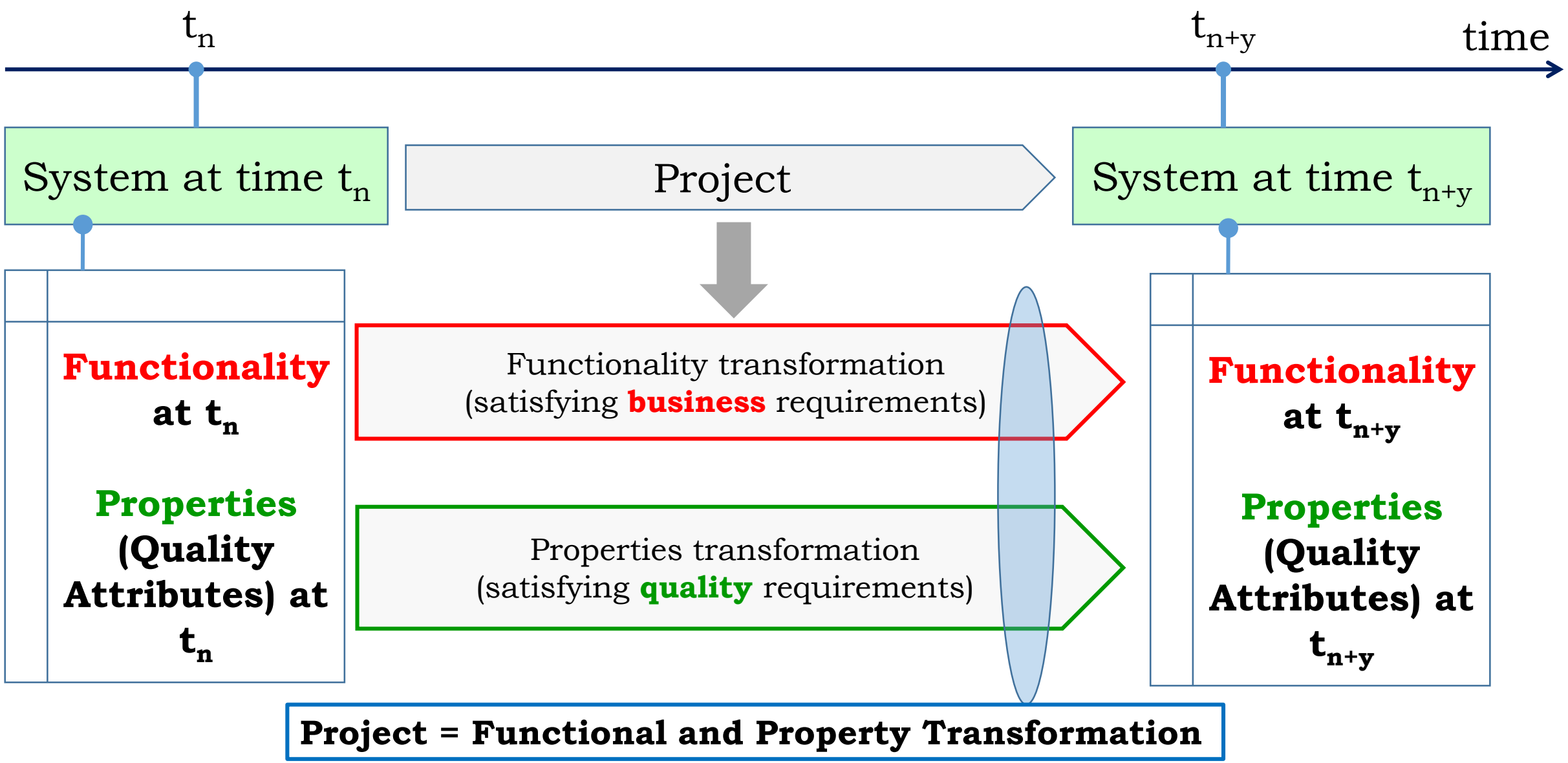
Software Properties: Functional and **Non-Functional** (= Quality Properties)

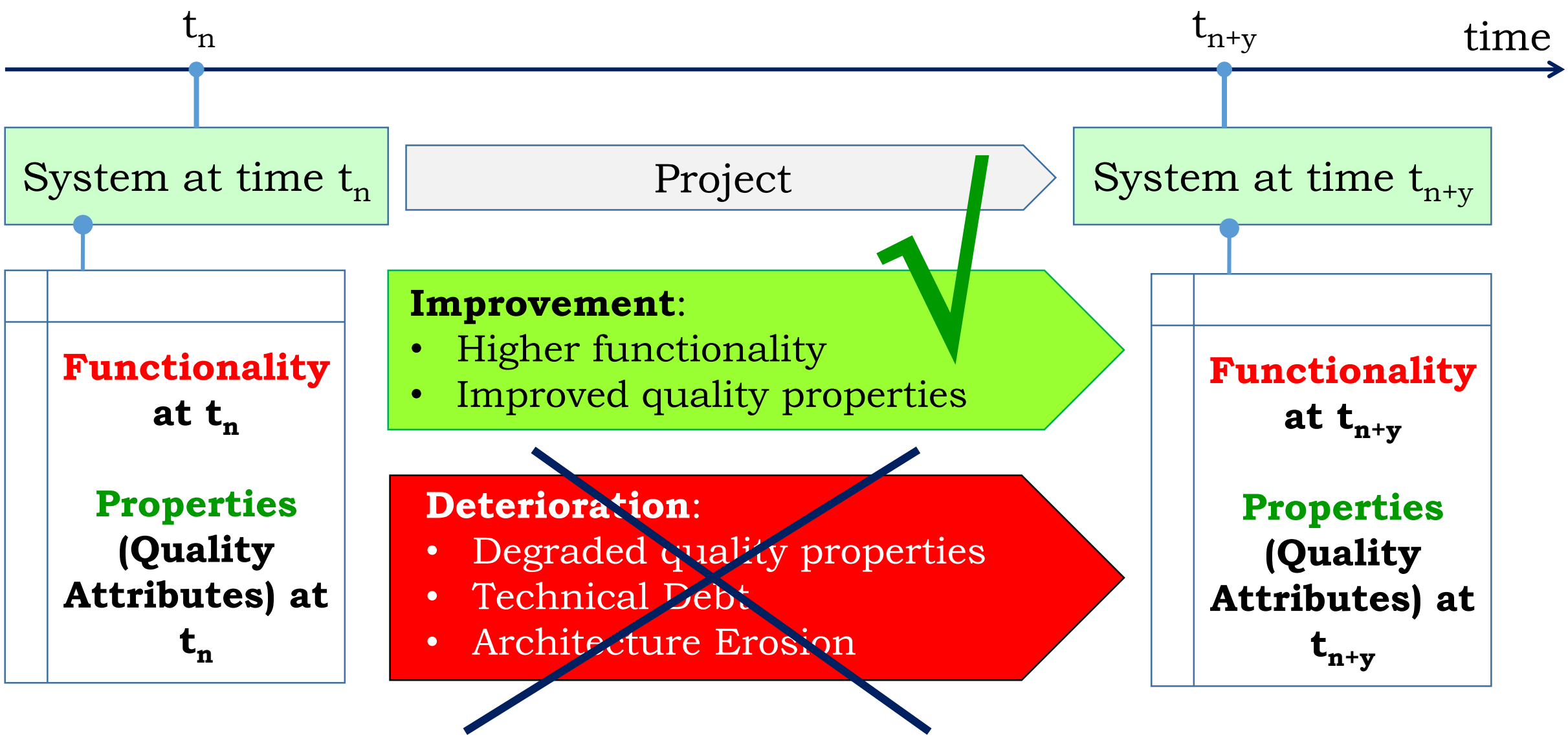
http://creepypasta.wikia.com

Non-functional properties [= Quality Attributes]

- Handle errors, malfunctions & unexpected situations
- Defend against attacks and failures (hacking)
- Cope with resources
- Comply with regulations & laws
- Adhere to industry standards
- Record malfunctions and errors
- Support pilots (e.g. stall warning)
- …

http://www.slate.com


t_n
t_{n+y}
time
System at time t_n
Project
System at time t_{n+y}
Functionality at t_n
Properties (Quality Attributes) at t_n
Functionality transformation (satisfying **business** requirements)
Properties transformation (satisfying **quality** requirements)
Functionality at t_{n+y}
Properties (Quality Attributes) at t_{n+y}
Project = Functional and Property Transformation


t_n
t_{n+y}
time
System at time t_n
Project
System at time t_{n+y}
Functionality at t_n
Properties (Quality Attributes) at t_n
Improvement:
• Higher functionality
• Improved quality properties
Deterioration:
• Degraded quality properties
• Technical Debt
• Architecture Erosion
Functionality at t_{n+y}
Properties (Quality Attributes) at t_{n+y}

Future-Proof Software-Systems

Because of …

complexity

change

uncertainty

Disruptive environment

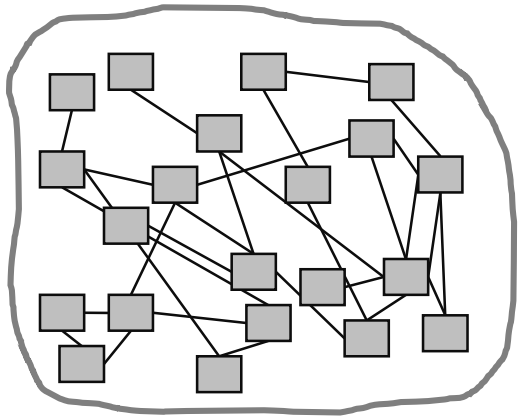
technical debt

Architecture erosion

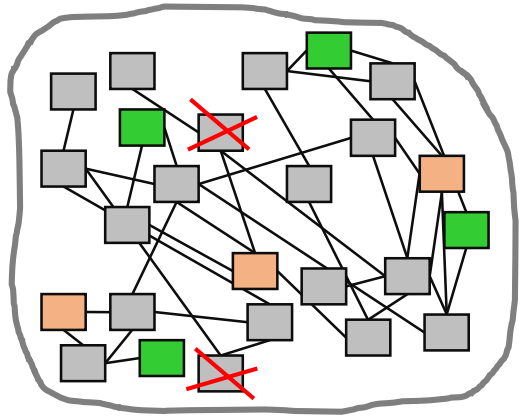
… our projects must continuously improve our software

Future-Proof Software-Systems

… our projects must continuously improve our software



Evolution: Software Life-Cycle



Continuous improvement: We need **three positive powers**

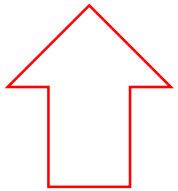
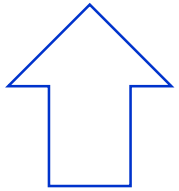
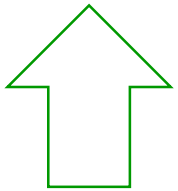
Good architects

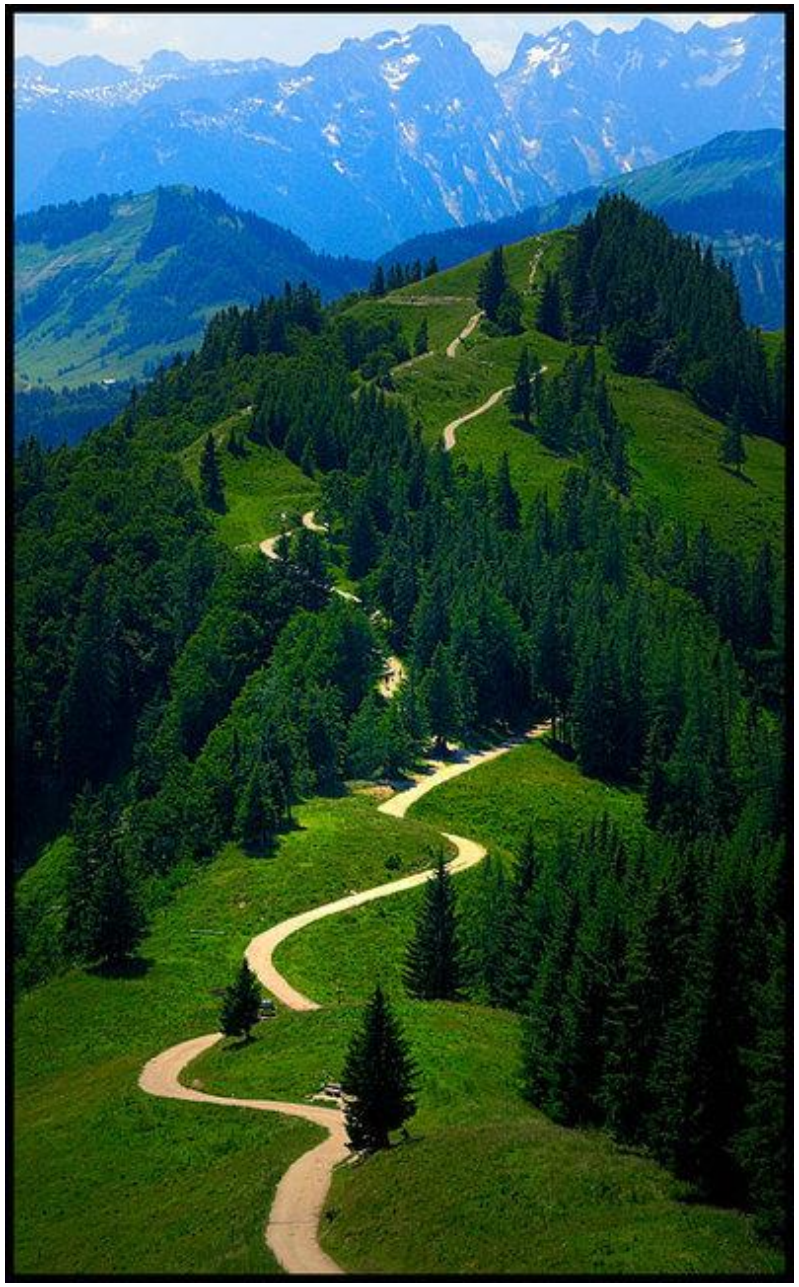
Managed Evolution

Managed Evolution Strategy for Software-Systems

Future-Proof Software-Systems

Primary Properties:

Business Value	Changeability	Dependability
		
Is the **business** reason for building and operating the software system	Is the key factor for **success** in today's competitive markets	Is the base for **survival** in today's dangerous environment



https://img00.deviantart.net

Definition: **Strategy**

1. A method or plan chosen to bring about a desired future, such as achievement of a goal or solution to a problem (⇐ *in our case:* **building future-proof software-systems**)

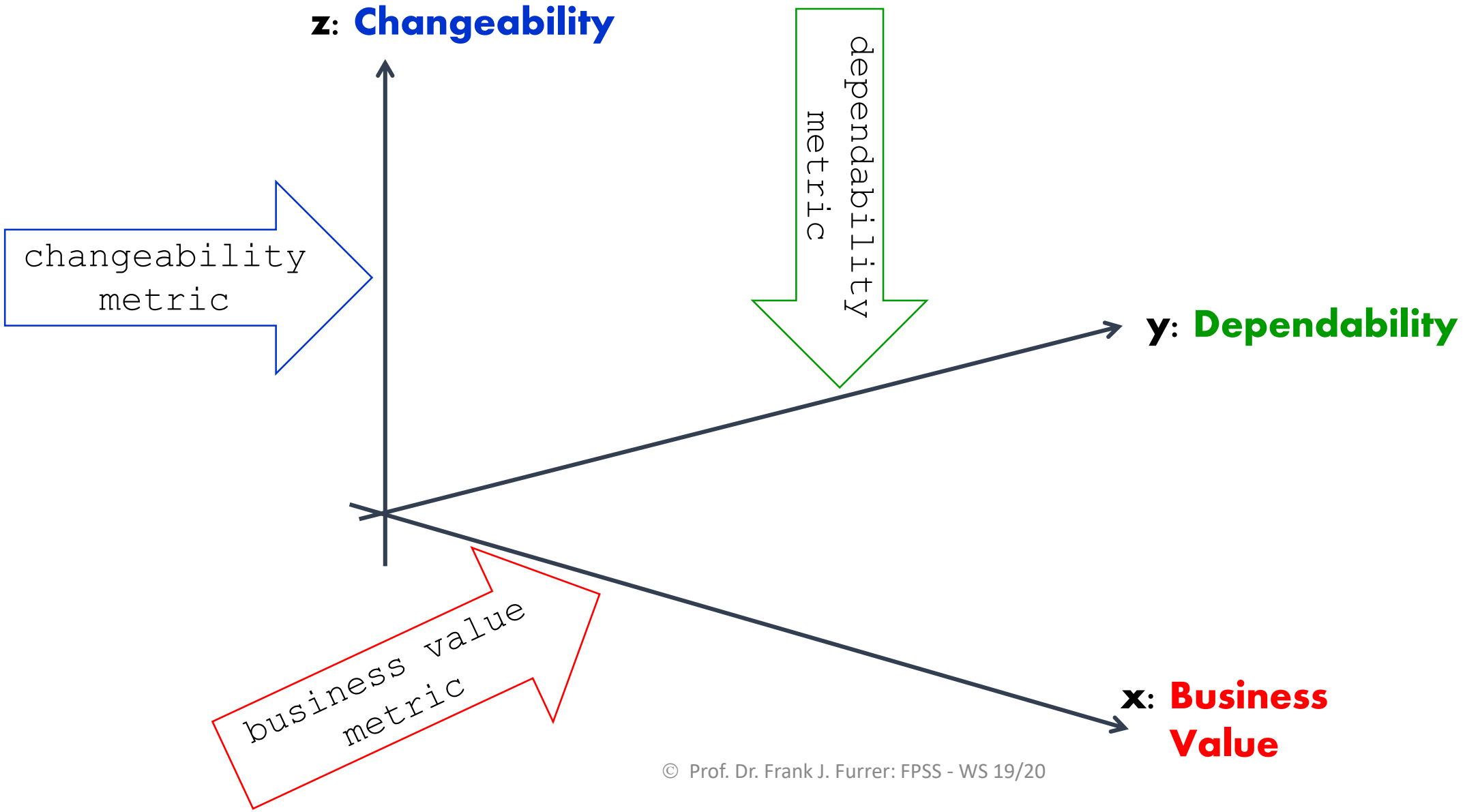
2. The art and science of planning and managing resources for their most efficient and effective use

http://www.businessdictionary.com/definition/strategy.html

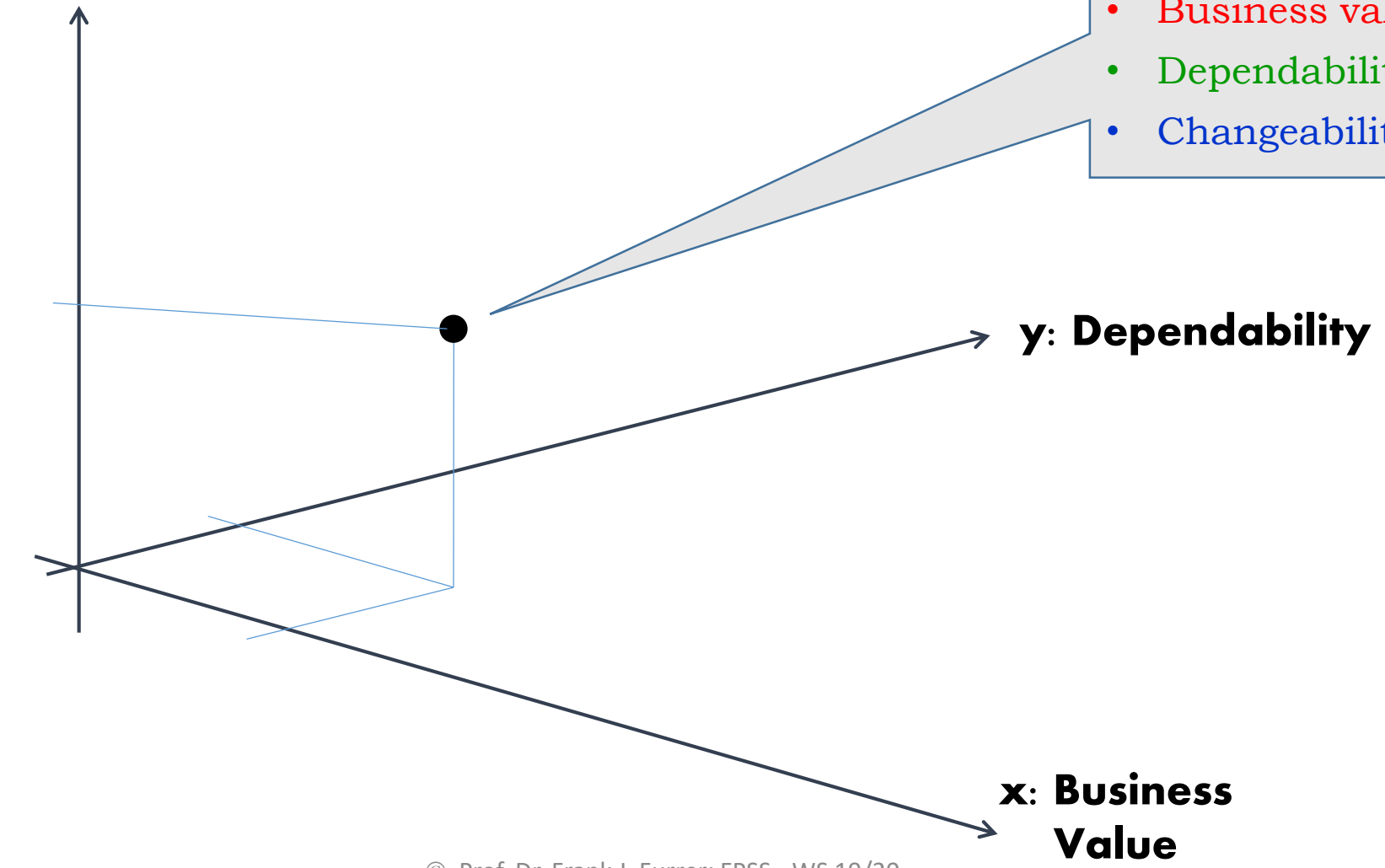
https://spin.atomicobject.com

1. The strategy must be understood and accepted by all
2. The strategy must be monitored, measured and enforced
3. The strategy must measurably lead to the **desired goals**

Managed Evolution **Coordinate System**

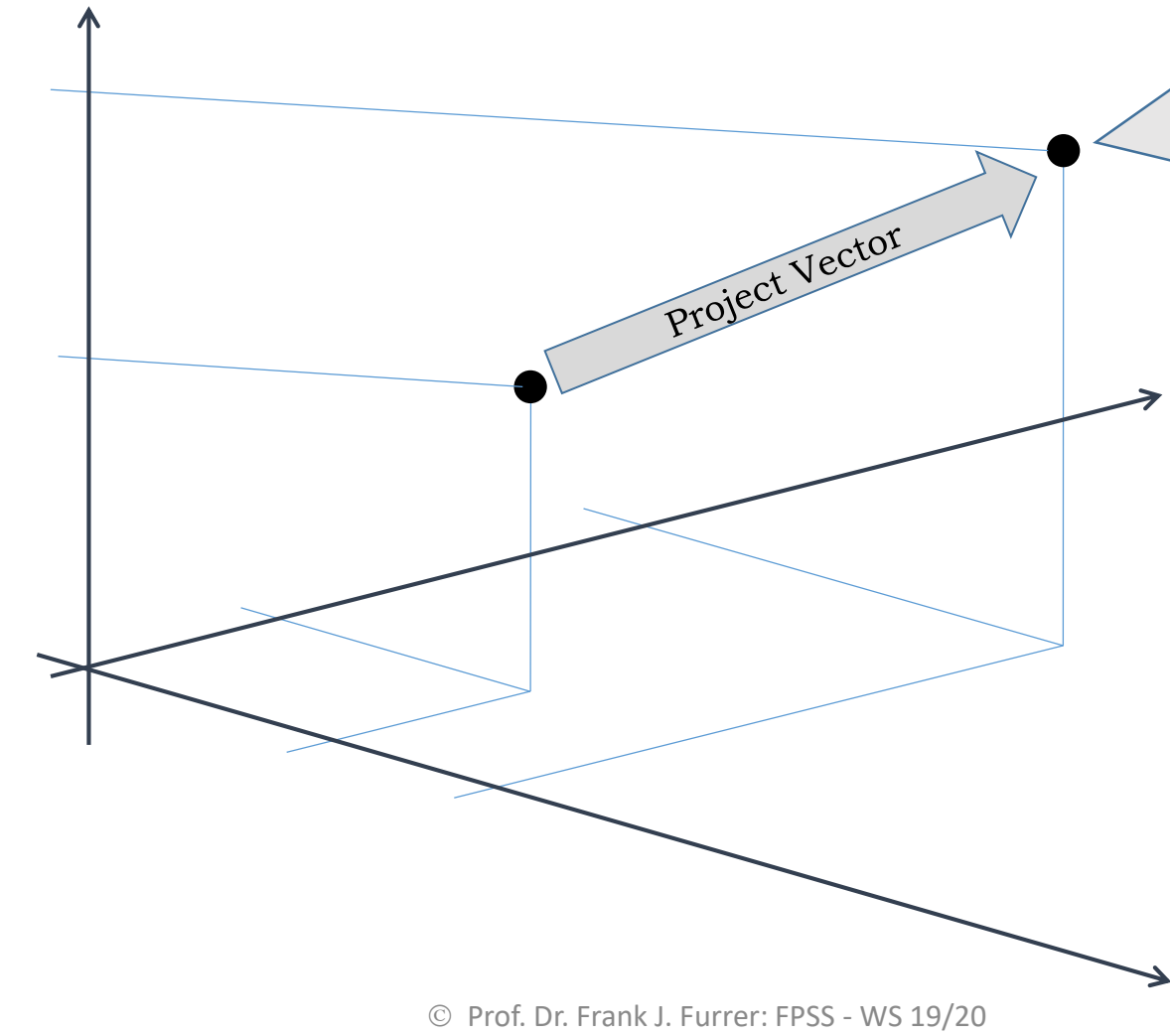


Managed Evolution Coordinate System



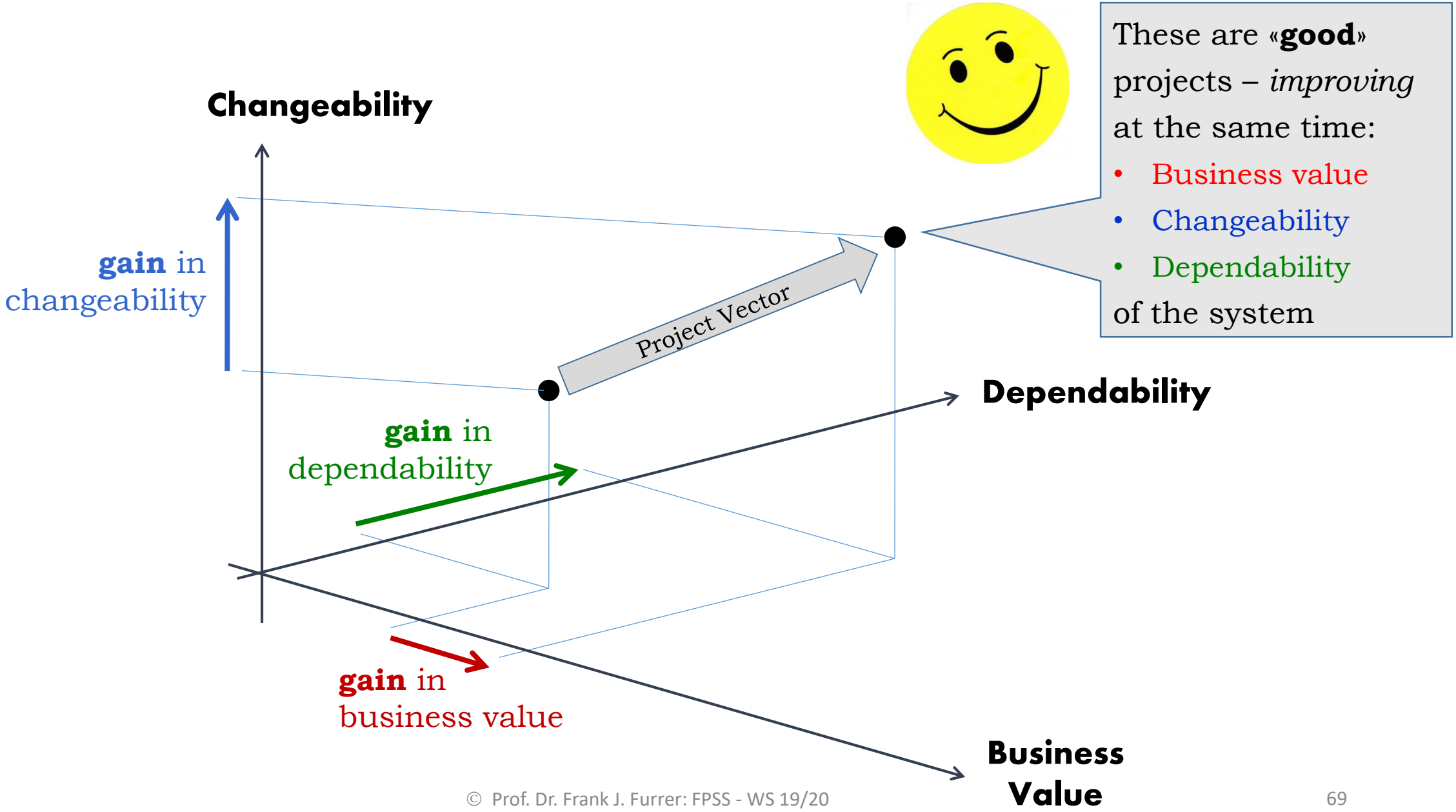
At any point in time t_n
a **system** has a quantified:

- Business value
- Dependability
- Changeability



Every project transforms the system, i.e. it *improves* or *deteriorates* the:

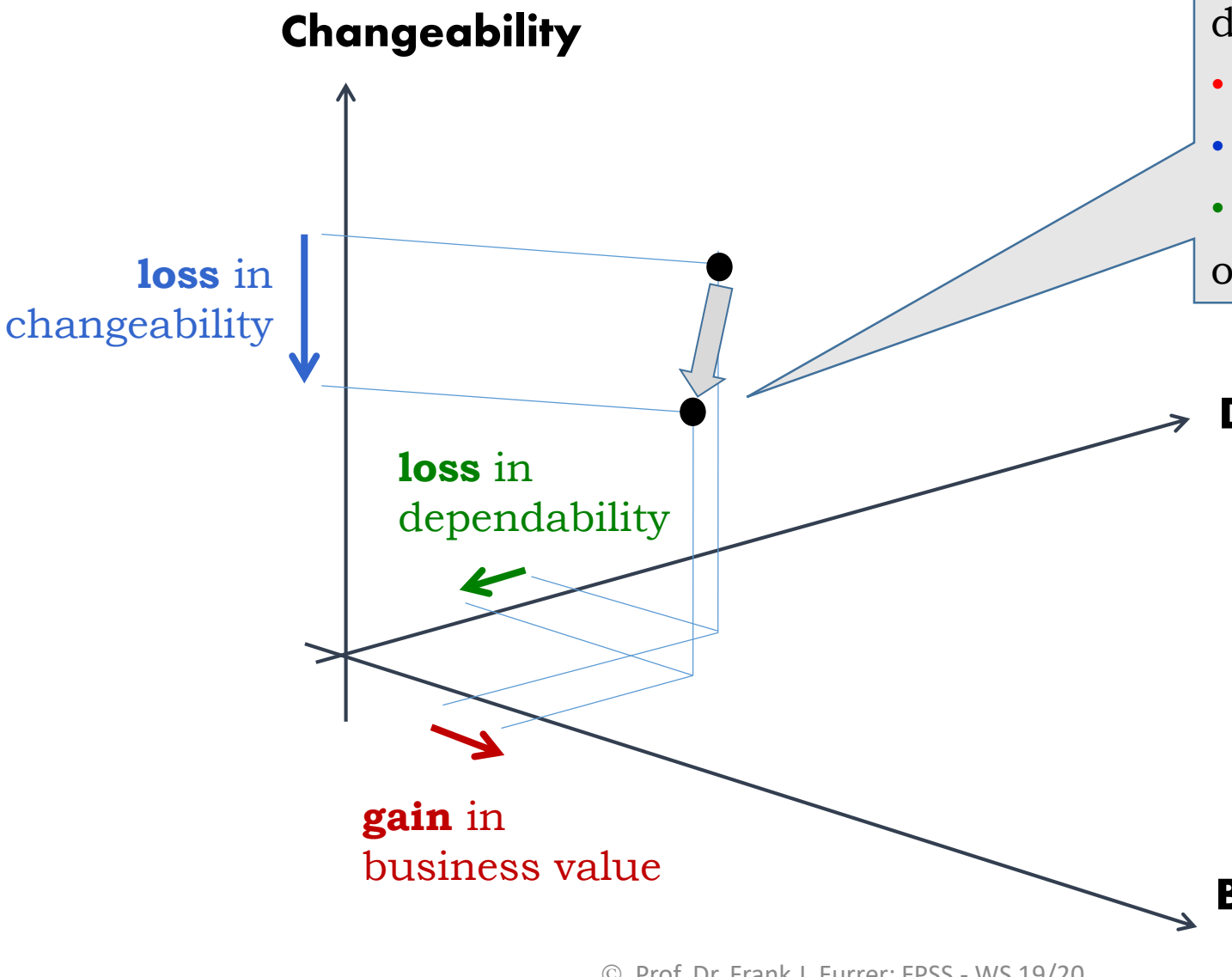
- Business value
- Dependability
- Changeability



These are «**good**» projects – *improving* at the same time:

- Business value
- Changeability
- Dependability

of the system



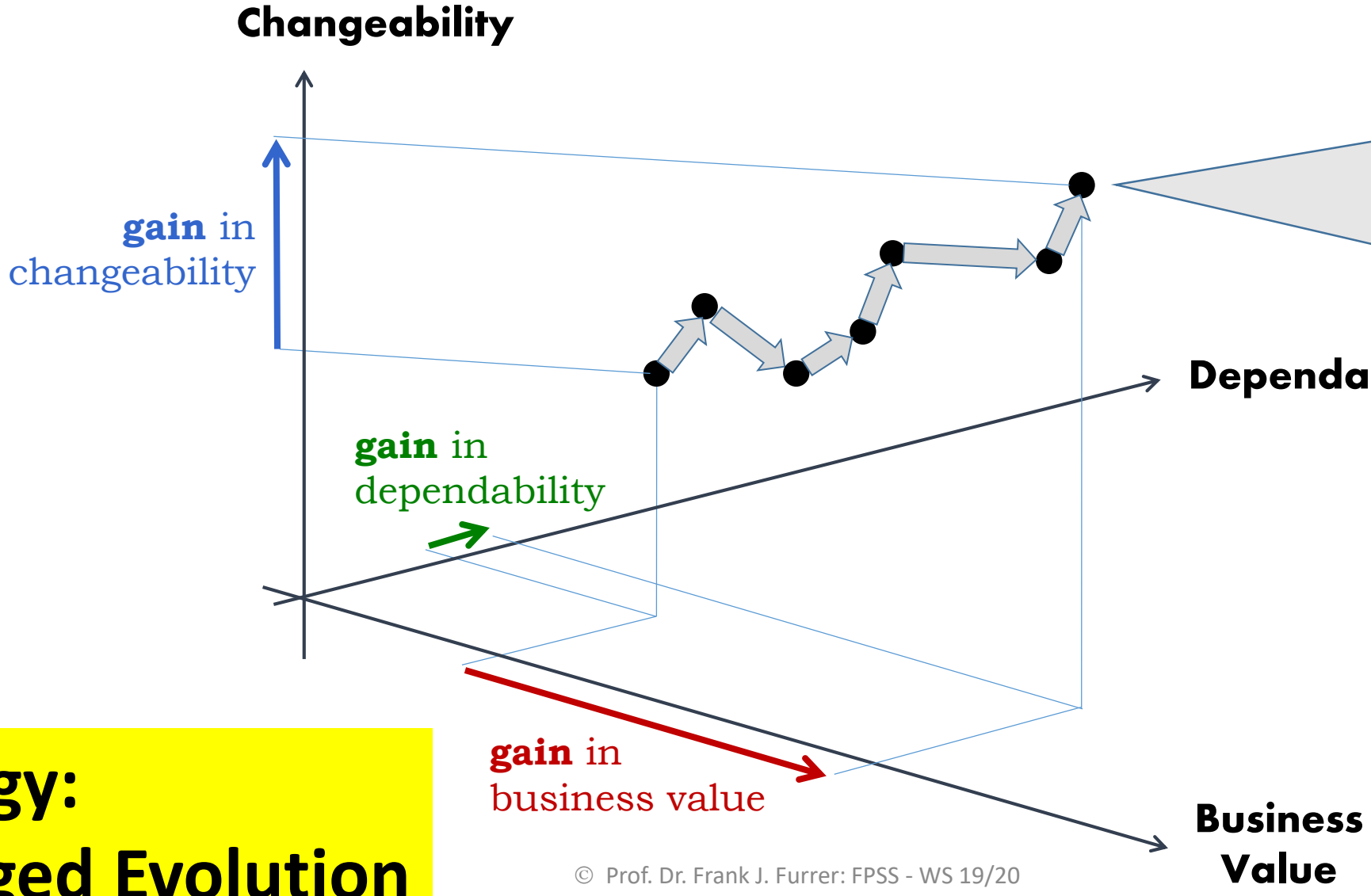
These are «**bad**» projects – deteriorating at least one of the:

- Business value
- Changeability
- Dependability

of the system

Evolution Trajectory = Sequence of Projects

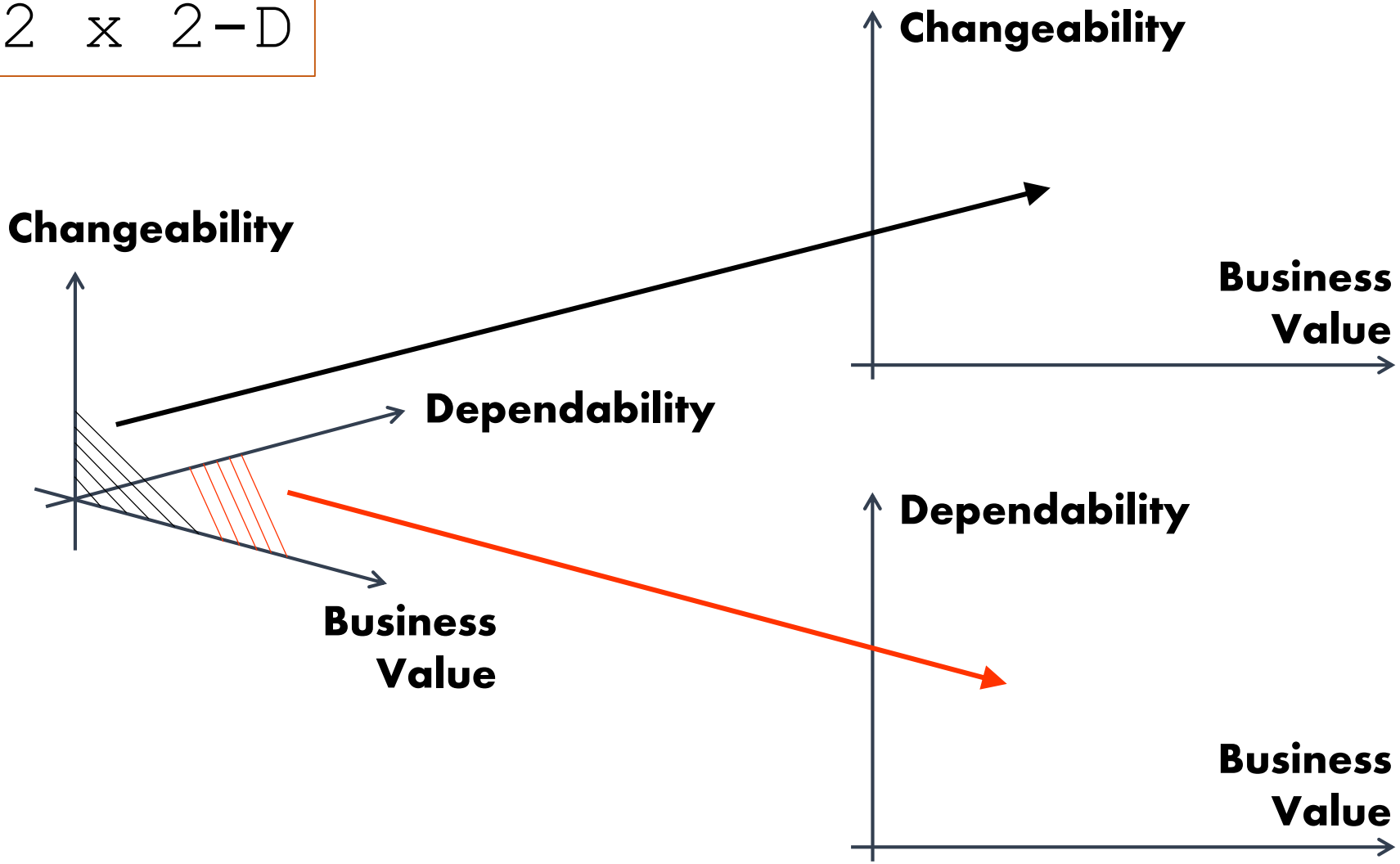
A sequence of projects builds a transformation **trajectory** of the IT system (= the evolution trajectory)



Strategy: Managed Evolution

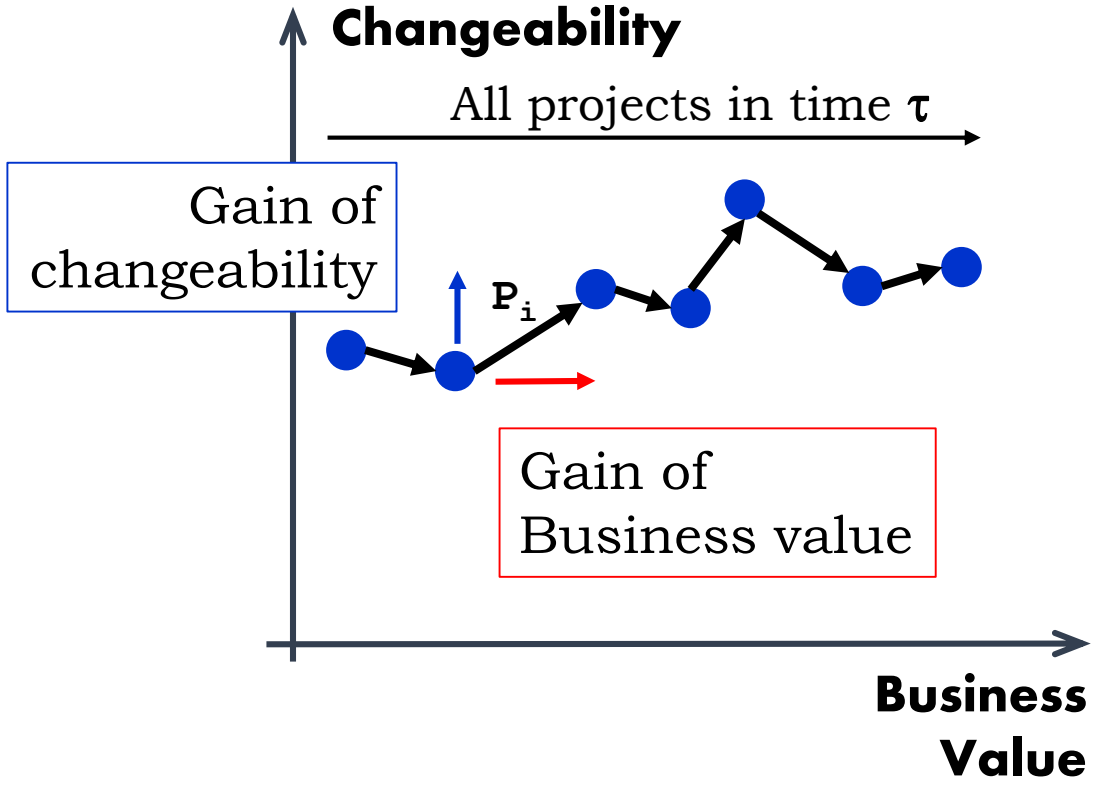
Representation Simplification:

3-D ⇒ 2 x 2-D

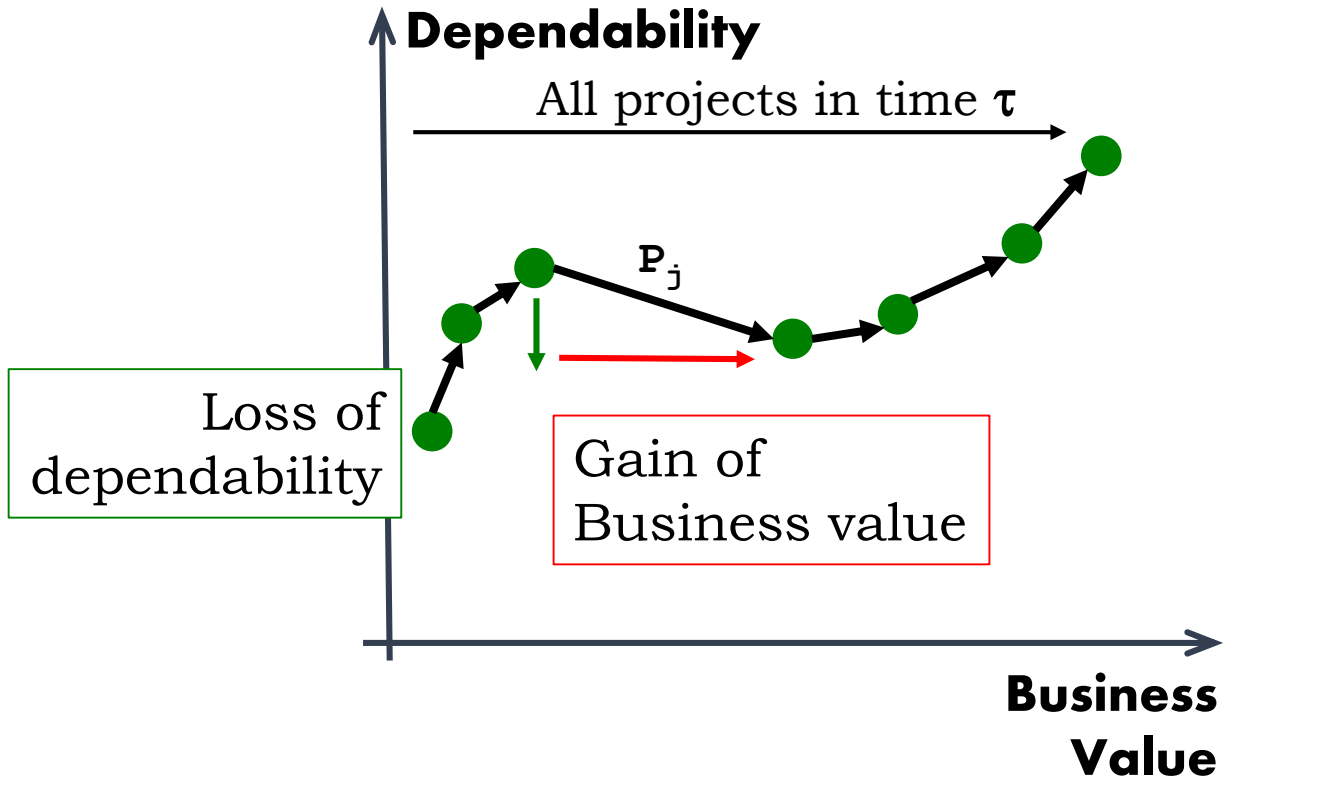


Future-Proof Software: **2 Managed Evolution** Coordinate Systems

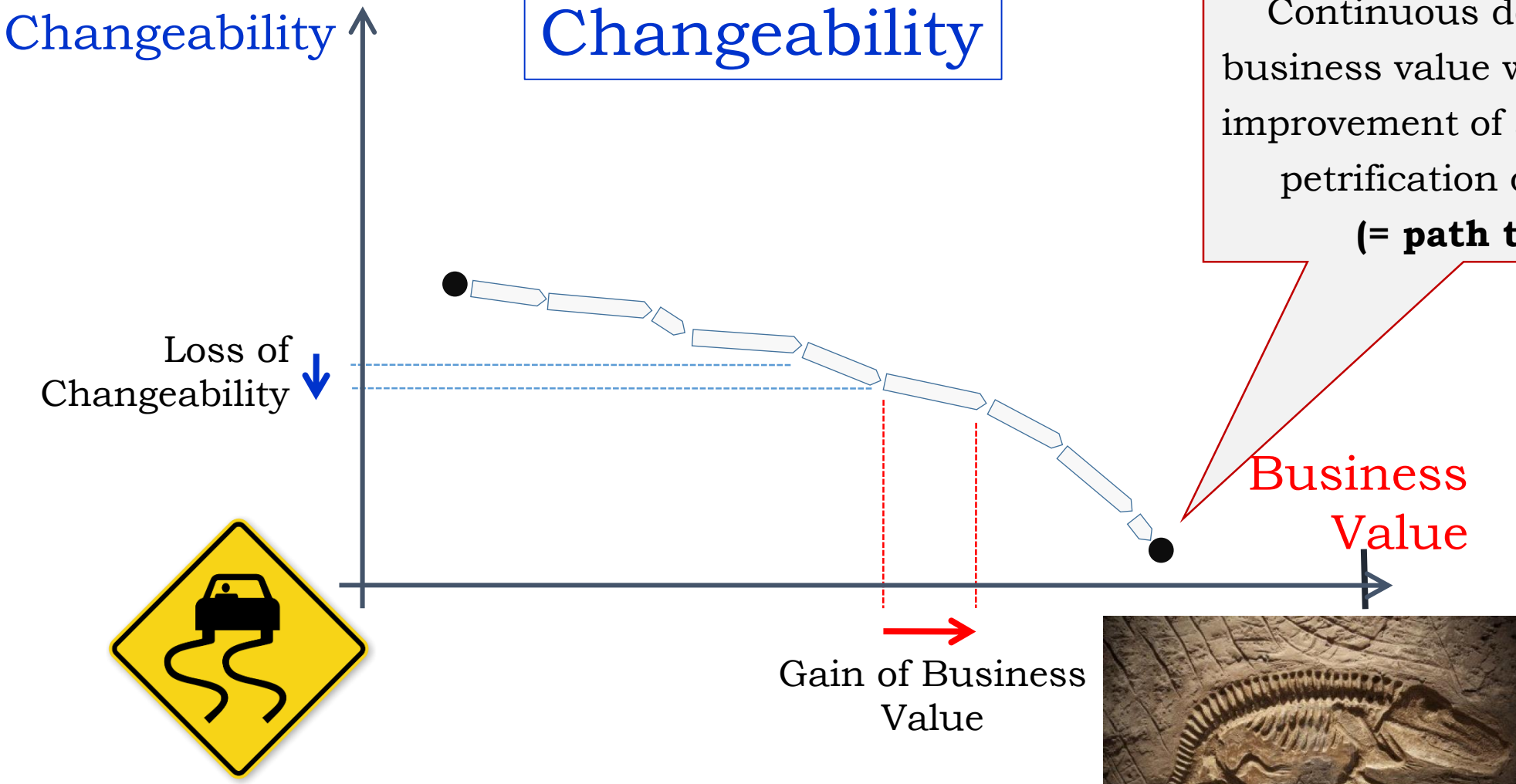
Changeability Evolution Trajectory



Dependability Evolution Trajectory

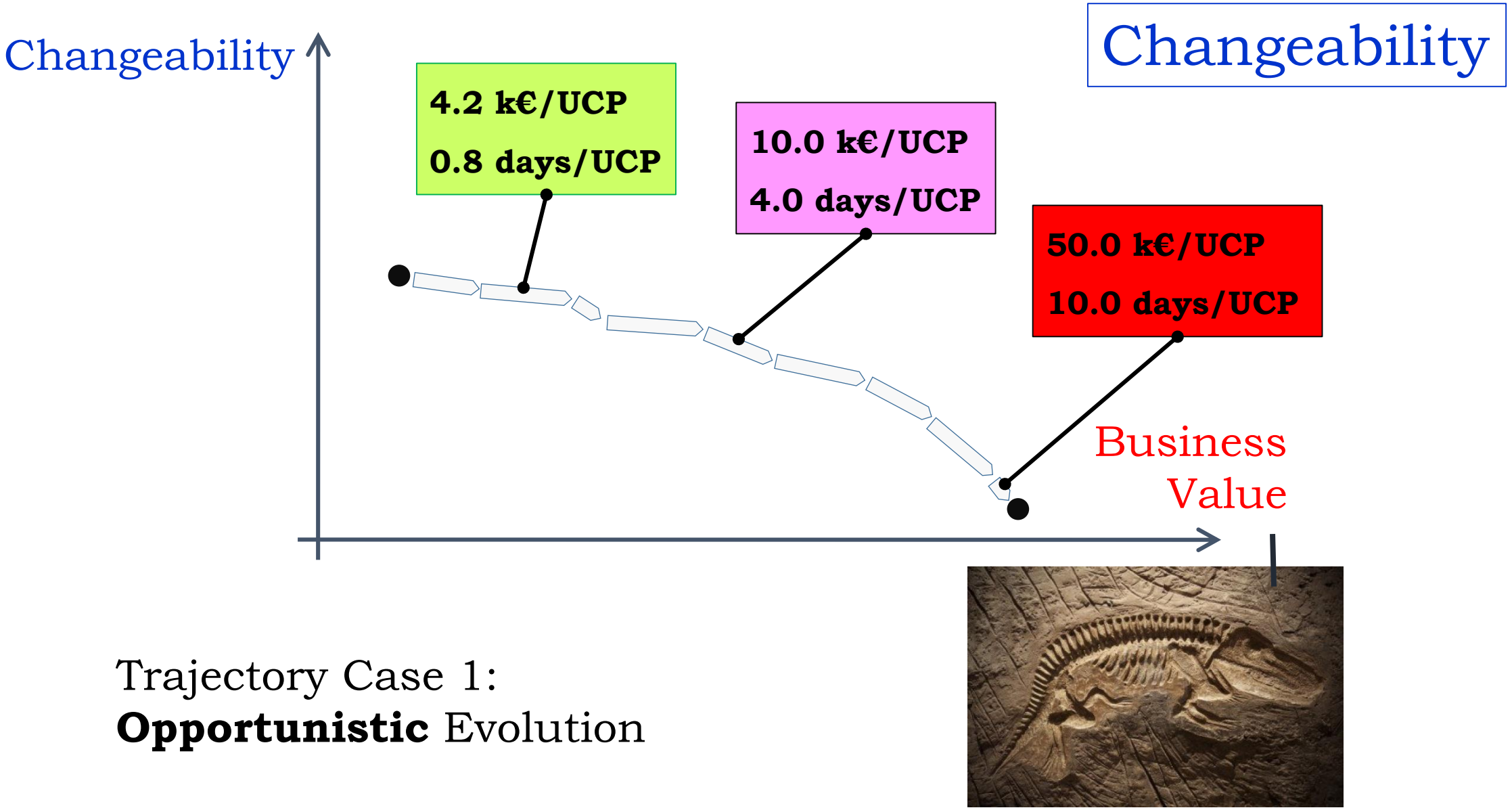


Changeability



Continuous development of business value while **neglecting** improvement of agility leads to a petrification of the system **(= path to death)**

Trajectory Case 1:
Opportunistic Evolution



Trajectory Case 1:
Opportunistic Evolution

Changeability

high

low

Example Project:
Size: 250 UCP

4.2 k€/UCP
0.8 days/UCP

→ Project Cost: 1'050 k€
Time to Market: 200 days

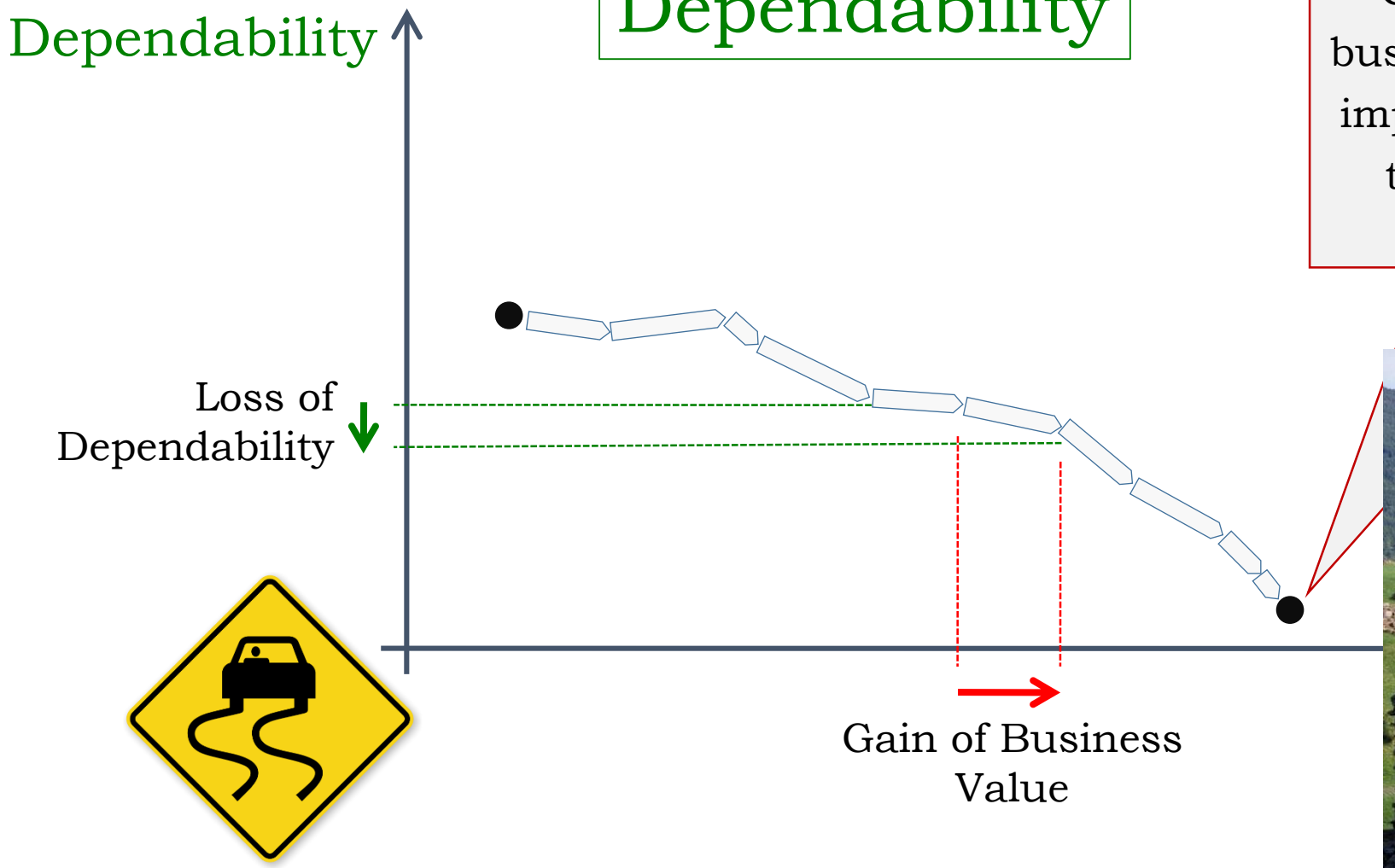
10.0 k€/UCP
4.0 days/UCP

→ Project Cost: 2'500 k€
Time to Market: 1'000 days

50.0 k€/UCP
10.0 days/UCP

→ Project Cost: 12'500 k€
Time to Market: 2'500 days

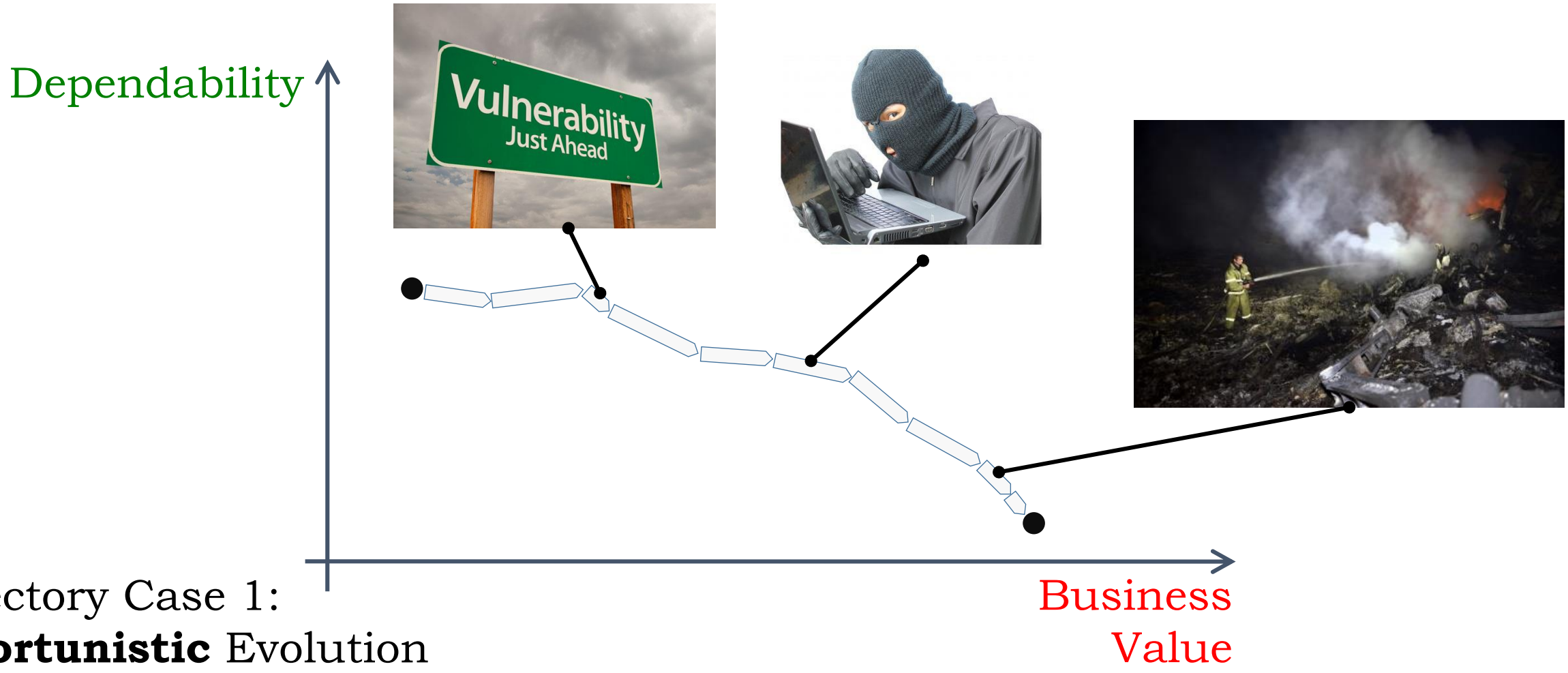
Dependability



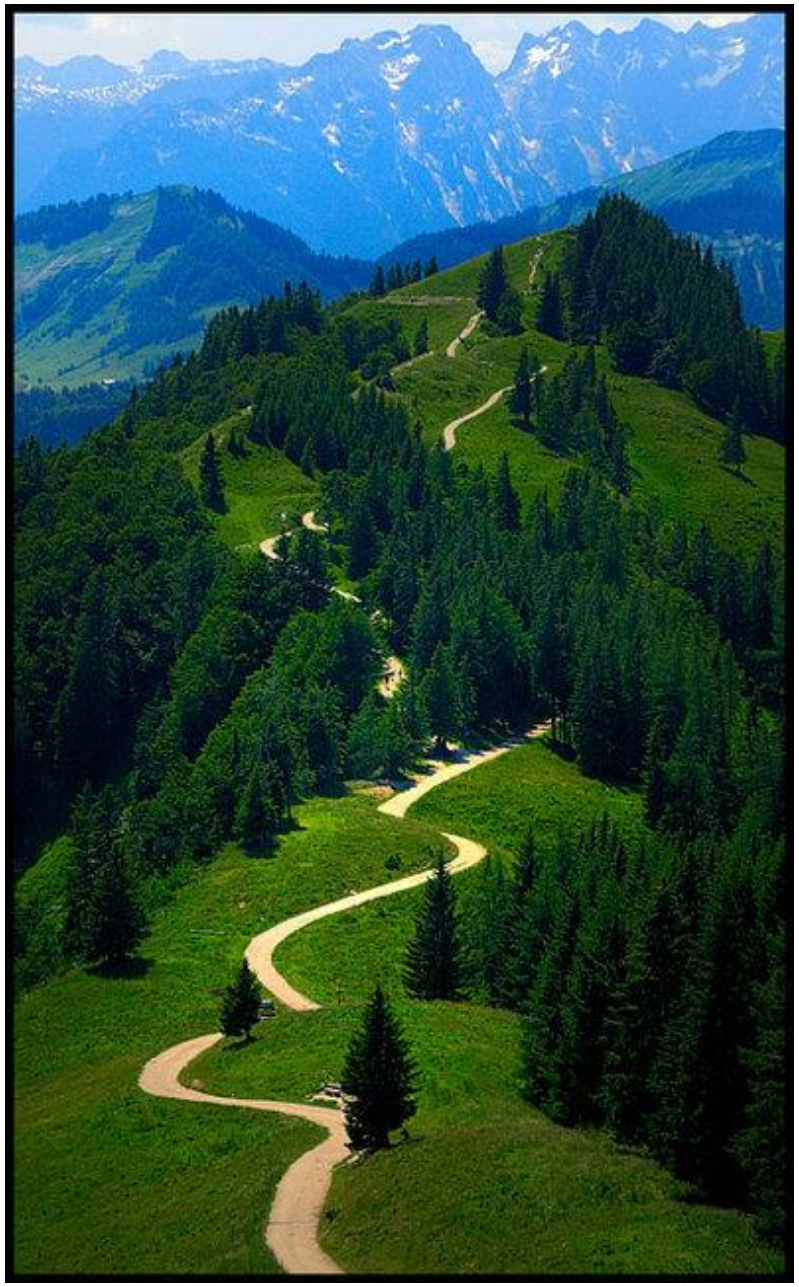
Continuous development of business value while **neglecting** improvement of resilience leads to an undefendable system **(= path to death)**

Trajectory Case 1:
Opportunistic Evolution

Dependability Evolution Trajectory: What does it mean?



Trajectory Case 1:
Opportunistic Evolution



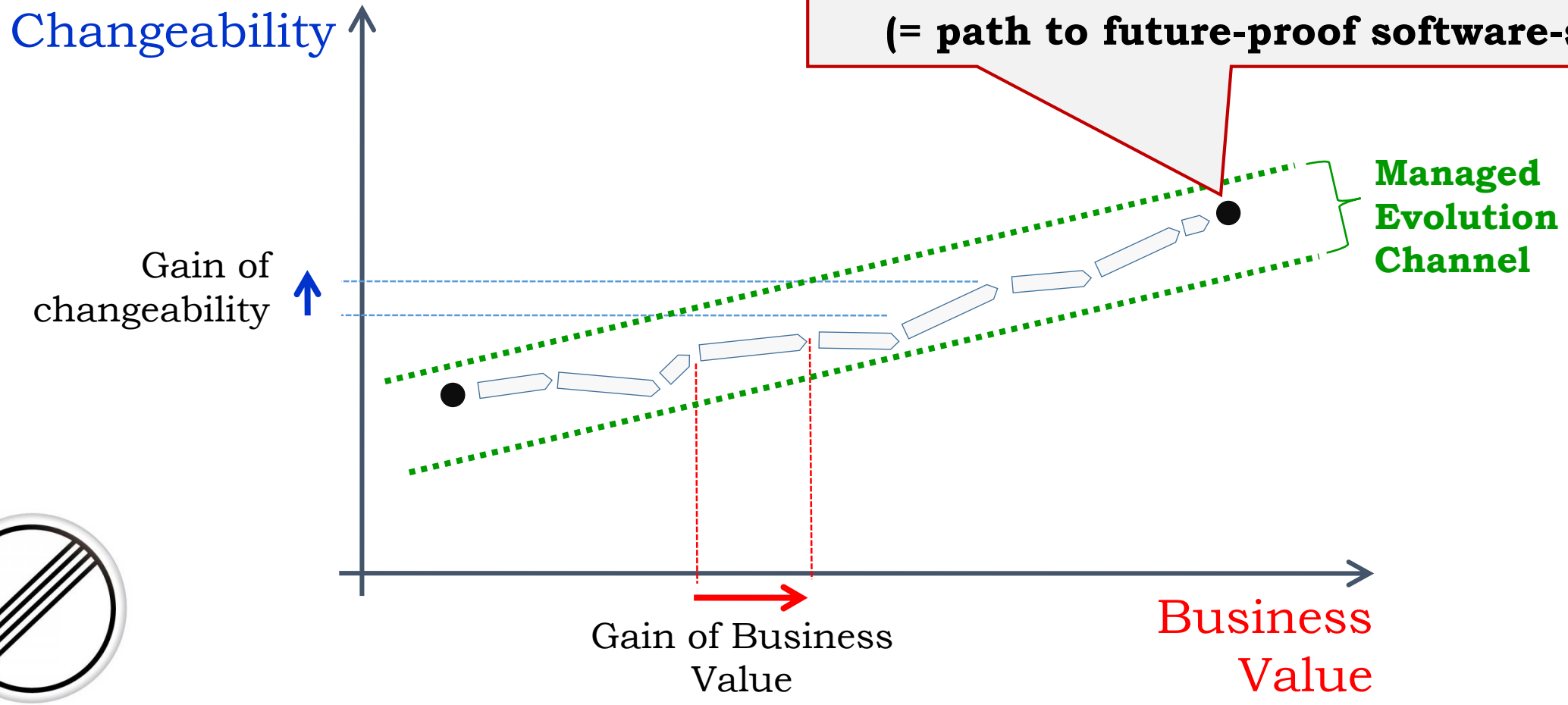
https://img00.deviantart.net

Which is the successful **strategy** for Future-Proof Software-Systems ?

Answer:

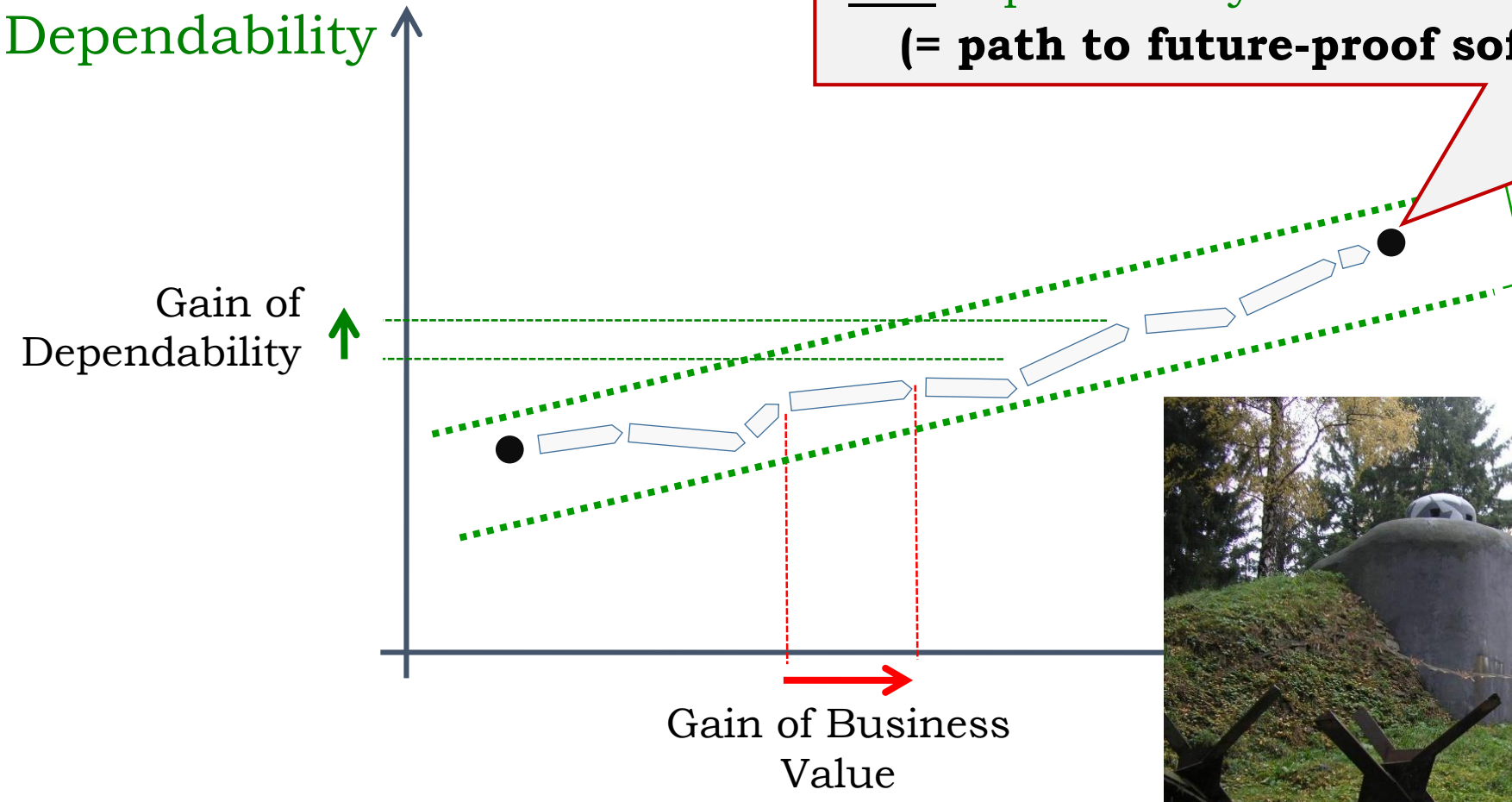
Managed Evolution

Continuous development of both business value **and** changeability leads to a sustainable system **(= path to future-proof software-systems)**



Trajectory Case 2:
Managed Evolution

Continuous development of both business value **and** dependability leads to a sustainable system **(= path to future-proof software-systems)**



http://en.wikipedia.org/wiki/Czechoslovak_border_fortifications

Manifesto of Managed Evolution

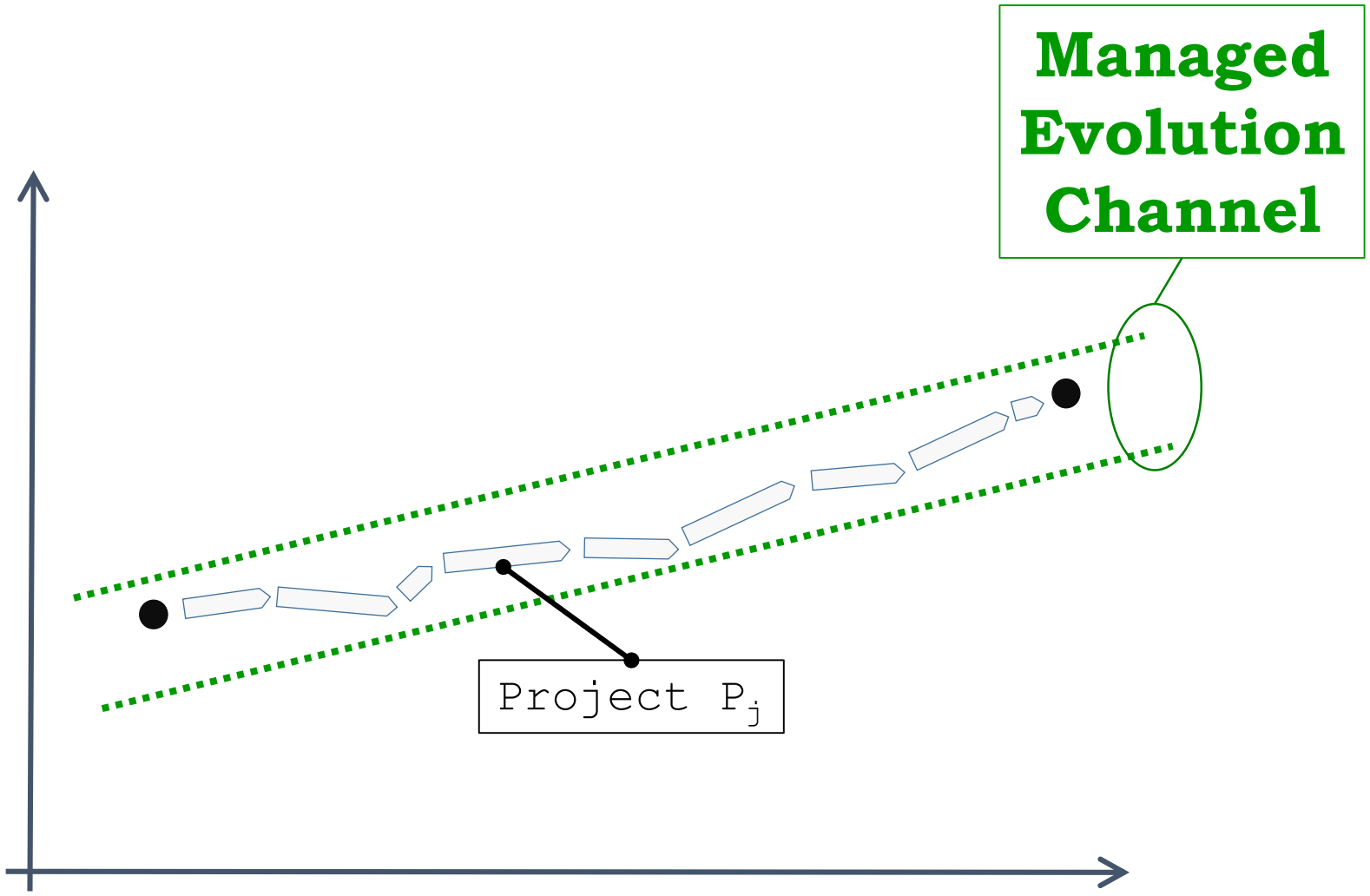
1. Business value, changeability and dependability are continuously improved,

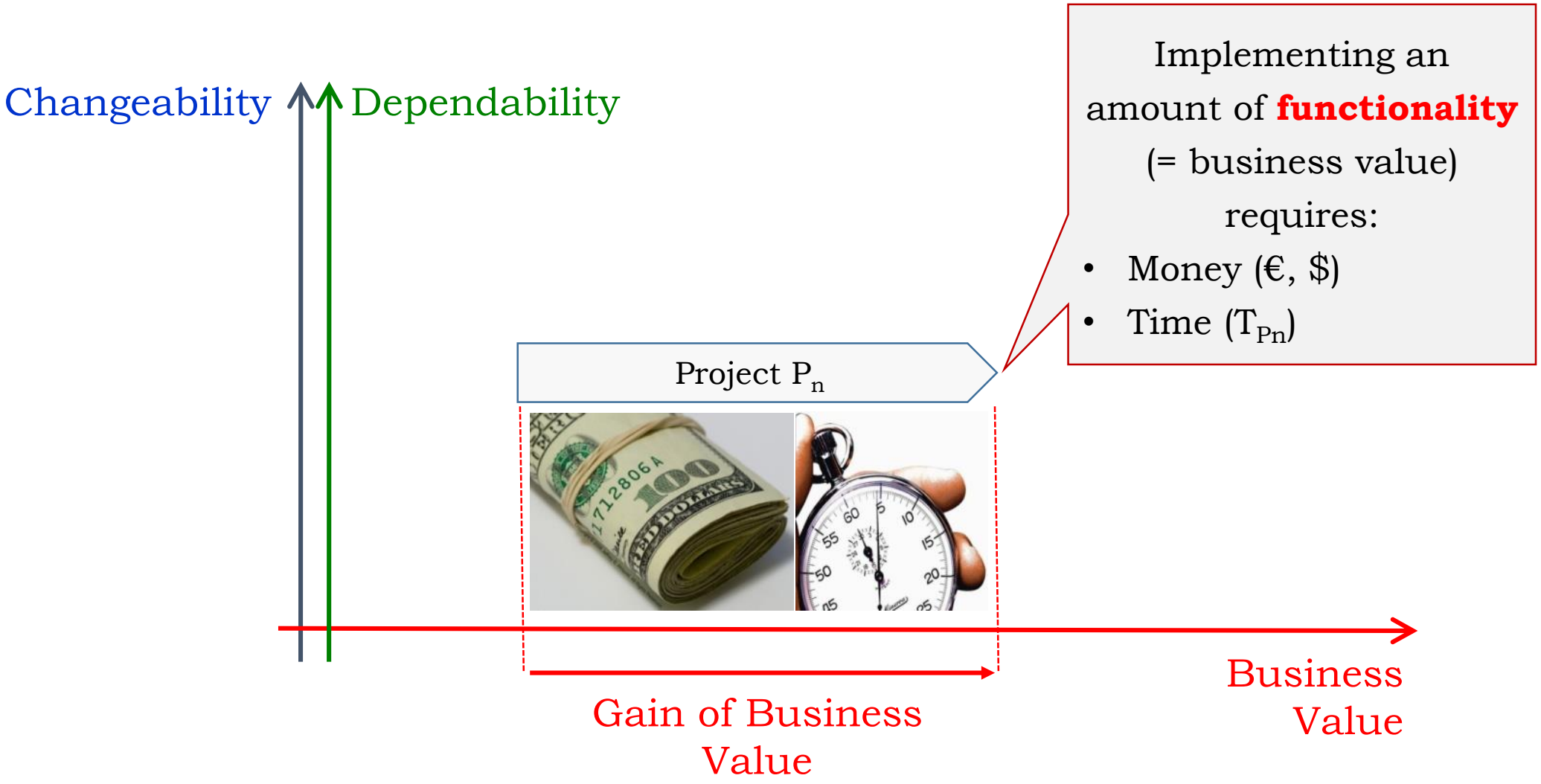
2. Business value, changeability and dependability are expressed and tracked by reliable metrics,

3. All (other) quality attributes are as good as necessary,

4. The system evolves in manageable, risk-controlled steps

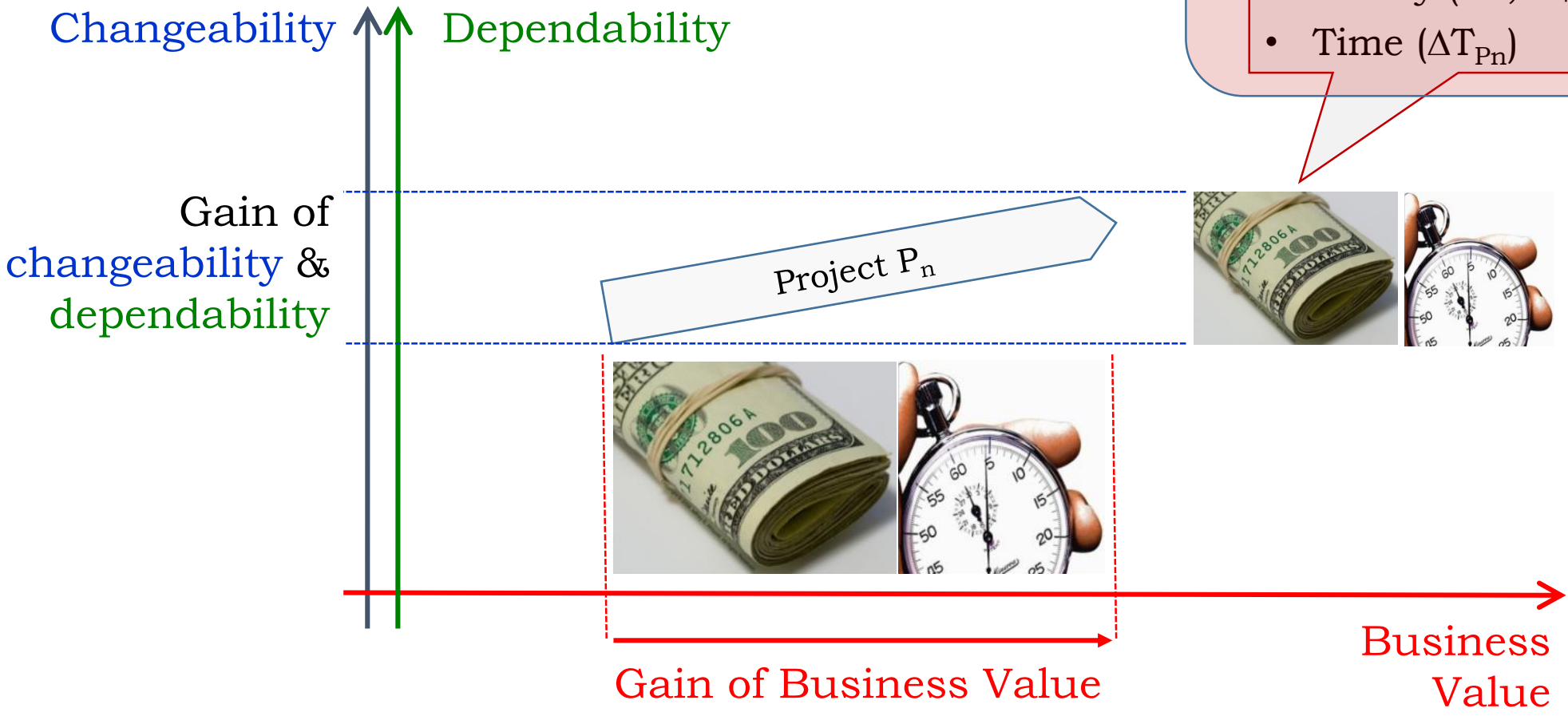
Changeability
Dependability

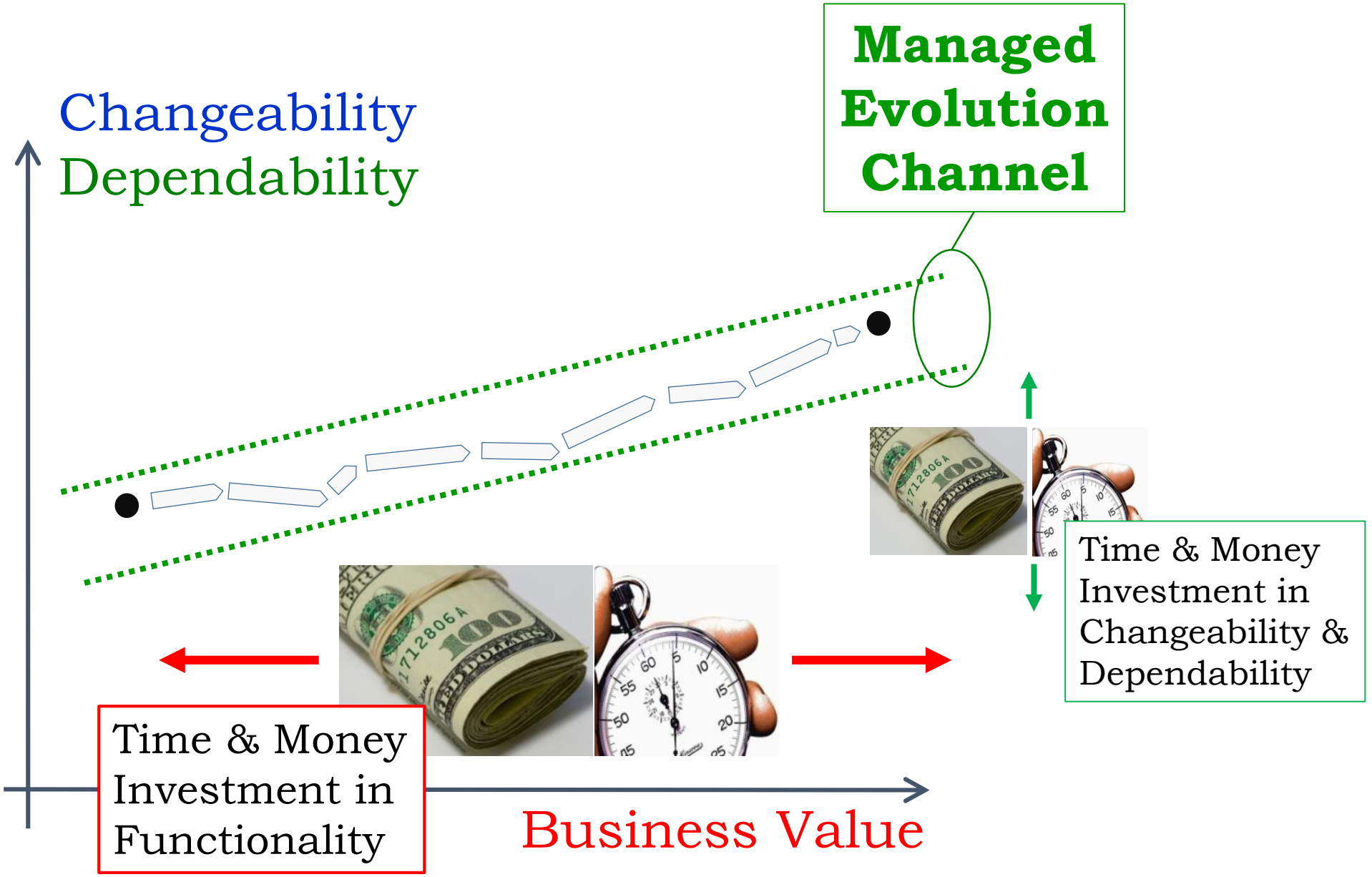



Changeability
Dependability
Implementing an amount of **functionality** (= business value) requires:
• Money (€, $)
• Time (T_{Pn})
Project P_n
Gain of Business Value
Business Value

Improving **changeability** and **dependability** requires additional:

- Money (Δ€, Δ$)
- Time (ΔT_{Pn})




Managed Evolution Channel
Changeability
Dependability
Time & Money Investment in Changeability & Dependability
Time & Money Investment in Functionality
Business Value

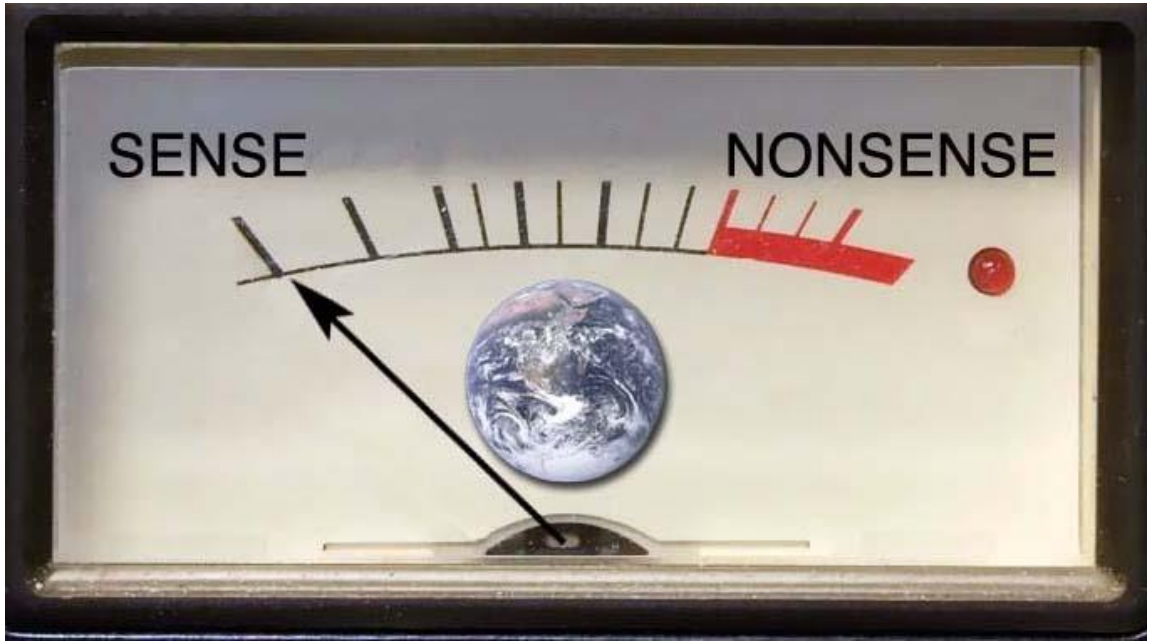
https://spin.atomicobject.com

The execution of the *managed evolution strategy* assures:

1. The optimum generation of **business value**
2. The continuous improvement of **changeability**
3. The reliable increase in **dependability**
4. The guarantee of the other **quality attributes**

⇒ <u>therefore</u>: The sustainable increase of the **value** of the software

… sounds good, but …



http://www.blacksheepproductions.com

Is there an obstacle to managed evolution?

Is there an obstacle to managed evolution?

☑ YES

☐ NO

Business People

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

CIO & IT-Architects

Business wants:

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

CIO & Architecture want:

- Improving Changeability
- Improving Dependability
- Limit growth in complexity
- No technical debt & architecture erosion

Is there an obstacle to managed evolution?

☑ YES

☐ NO

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

Business People

CIO & IT-Architects

Conflict of Interests: Time-to-Market, Development Cost vs. Clean implementation

Business wants:

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

CIO & Architecture want:

- Improving Changeability
- Improving Dependability
- Limit growth in complexity
- No technical debt & architecture erosion

Is there a significant obstacle to managed evolution?

Business People

CIO & IT-Architects

Management & Governance Issue

Necessary:

- Good IT-business alignment
- Trust and respect between business and IT department
- Adequate architecture process

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

Is there an obstacle to managed evolution?

Changeability ↑ Dependability ↑

Gain of changeability & dependability

Project P_n

Improving **changeability** and **dependability** requires additional:

- Money (Δ€, Δ$)
- Time (ΔT_{Pn})

Are your business people prepared to pay?

For every project?

http://sgs-uae.com

Example: Boeing 787
(787 Dreamliner Grounding)

https://www.scientificamerican.com/article/how-lithium-ion-batteries-grounded-the-dreamliner

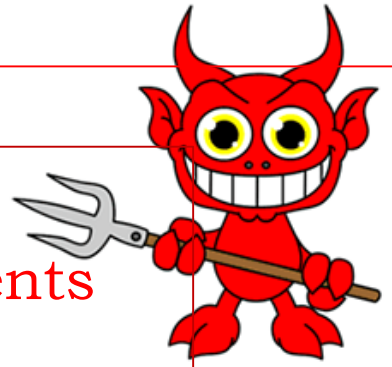
At 10:21 on Jan. 7, 2013, about a minute after all 183 passengers and 11 crew members from Japan Airlines Flight 008 disembarked at Boston's Logan International Airport, a member of the cleaning crew spotted smoke in the aft cabin of the Boeing 787-8.

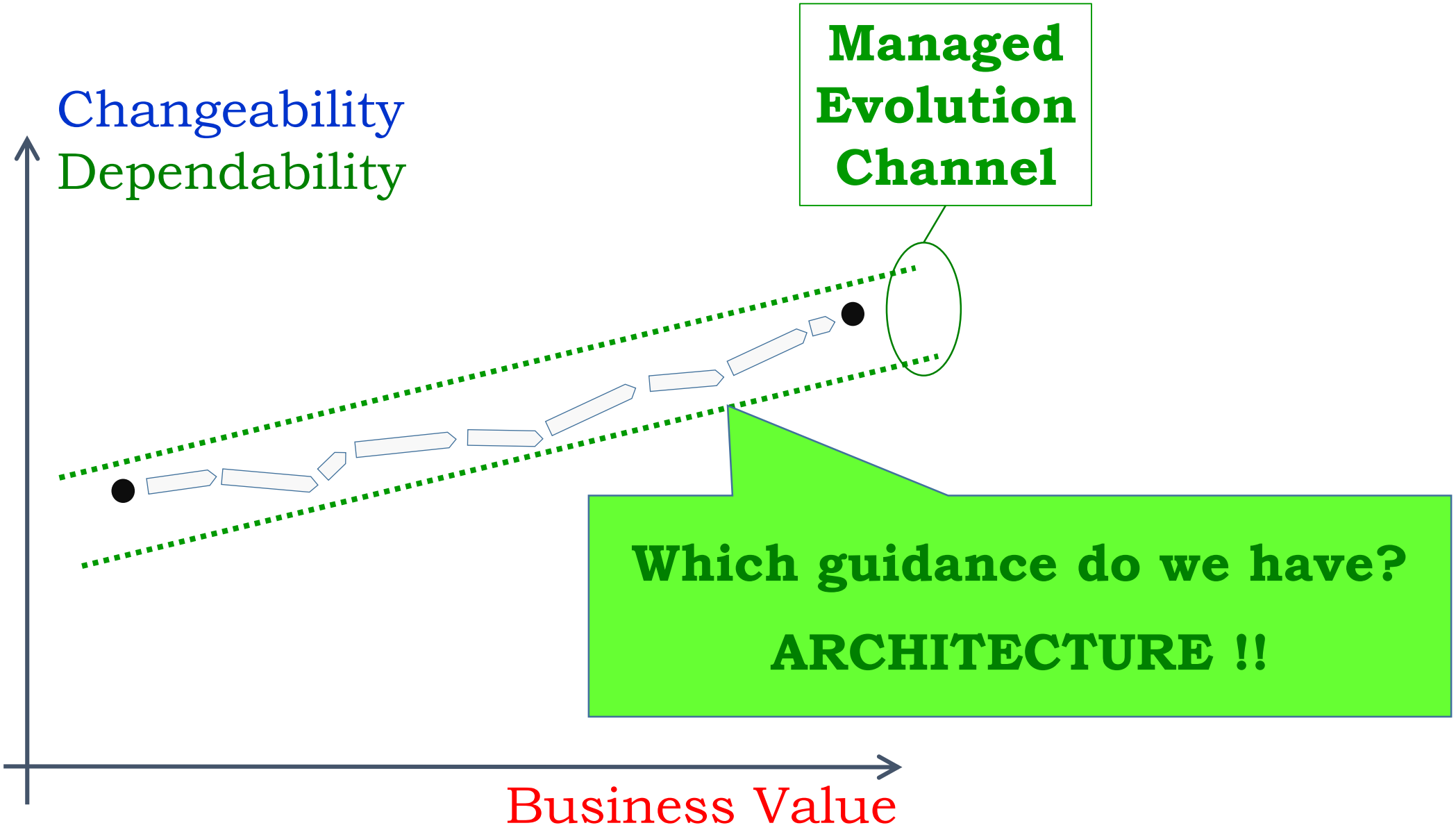
The reason was a **fire** in the lithium-ion battery.

As a consequence, the U.S. Federal Aviation Administration **grounded** the entire 787 fleet

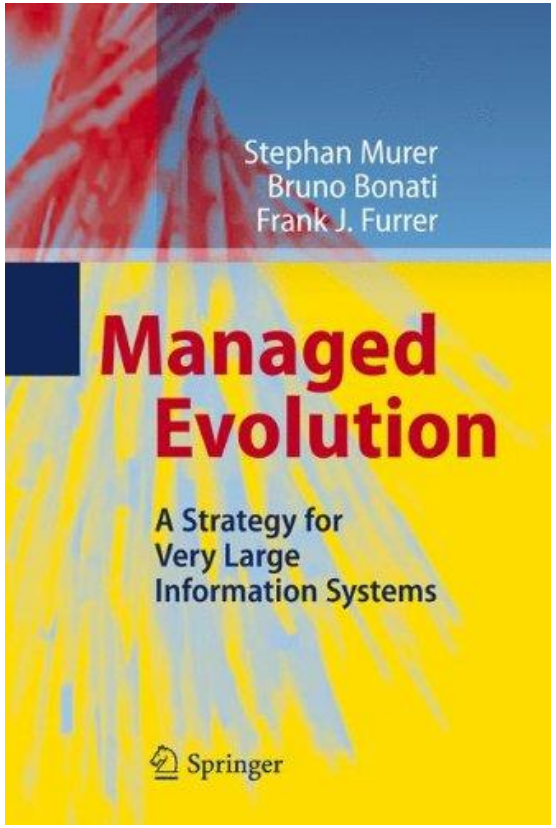
Highly dangerous:

Business requirements massively overruled engineering requirements



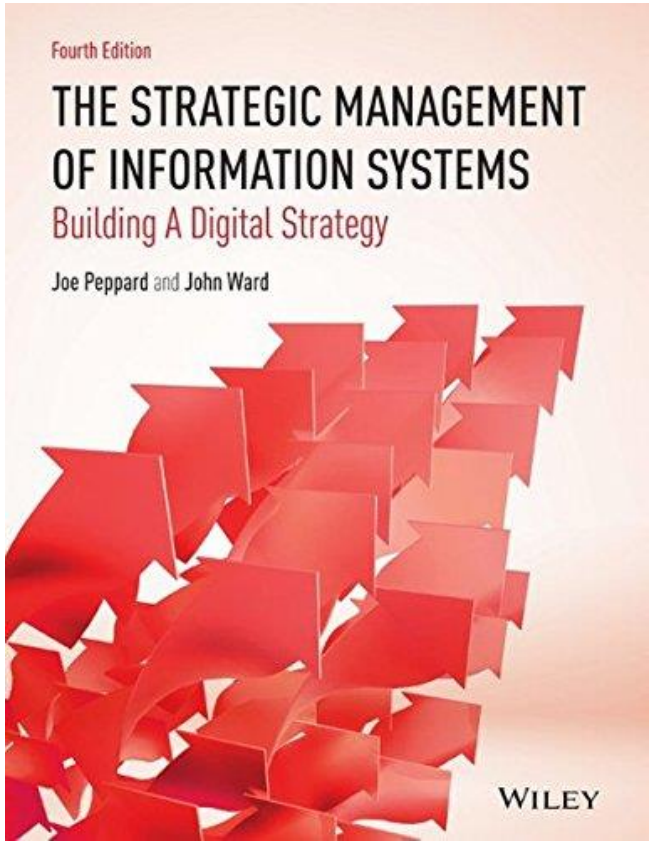

Managed Evolution Channel
Changeability
Dependability
Business Value
Which guidance do we have?
ARCHITECTURE !!

Textbook



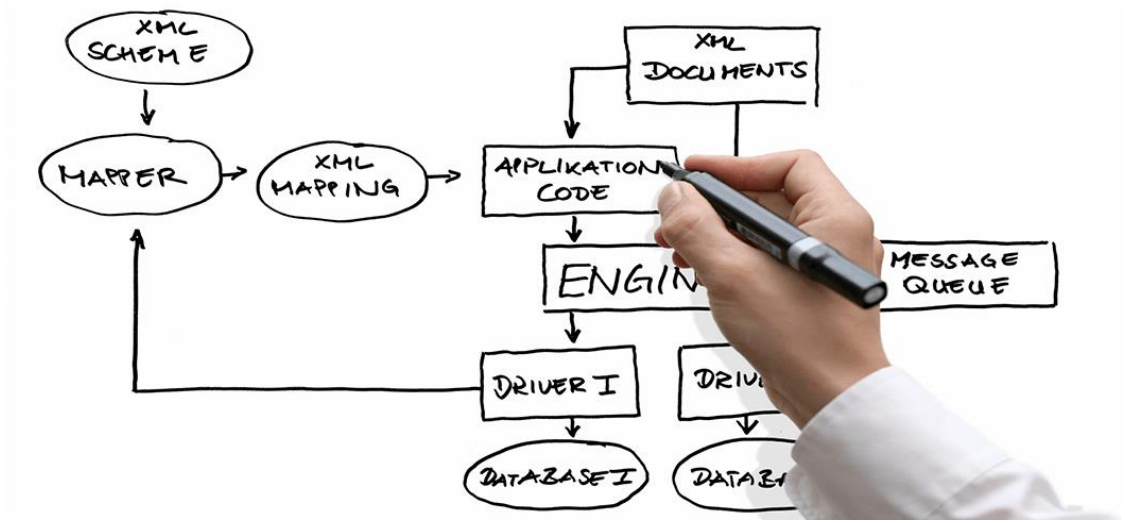
Stephan Murer, Bruno Bonati, Frank J. Furrer:
Managed Evolution – *A Strategy for Very Large Information Systems*
Springer-Verlag, Germany, 2011. ISBN 978-3-642-01632-5

Textbook



Joe Peppard, John Ward:
The Strategic Management of Information Systems – *Building a Digital Strategy*
John Wiley & Sons, USA, 2016. ISBN 978-0-470-03467-5

The Importance of Architecture



***Software-Architecture* is the single most important**

factor for future-proof software-systems

Functionality
[Business Value]

Changeability
Dependability

Other
Quality Attributes

Parts of the system
and their relationsships
→ **Architecture**

Definition:

A future-proof software-system is a **structure**

that enables the management

of complexity, change and uncertainty

with the least effort, with acceptable risk and with specified quality properties

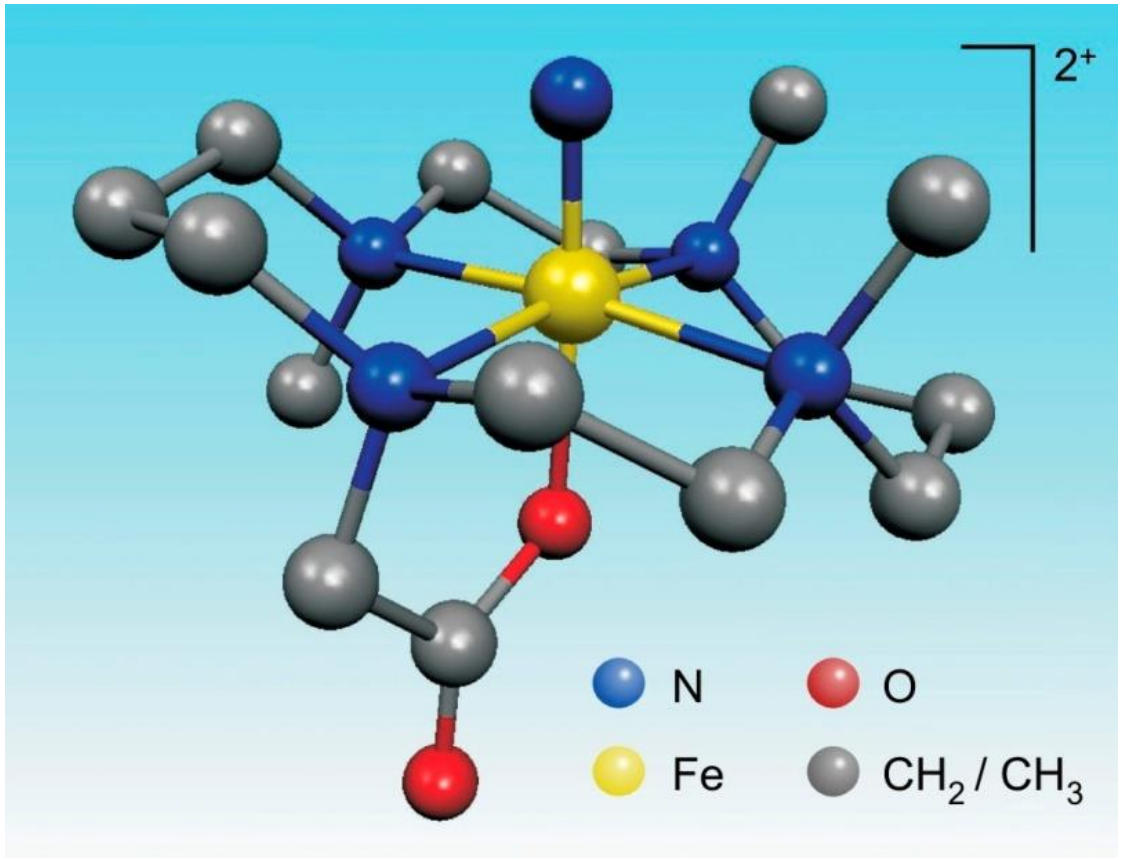
Analogy: Town Architecture

http://www.0lll.com/architecture-exhibitions/?gal=24

http://www.asisbiz.com/index.html

Which structure is easier to expand and evolve?
Which structure has the better properties, e.g. quality of life?
Which structure is future-proof (expandable)?

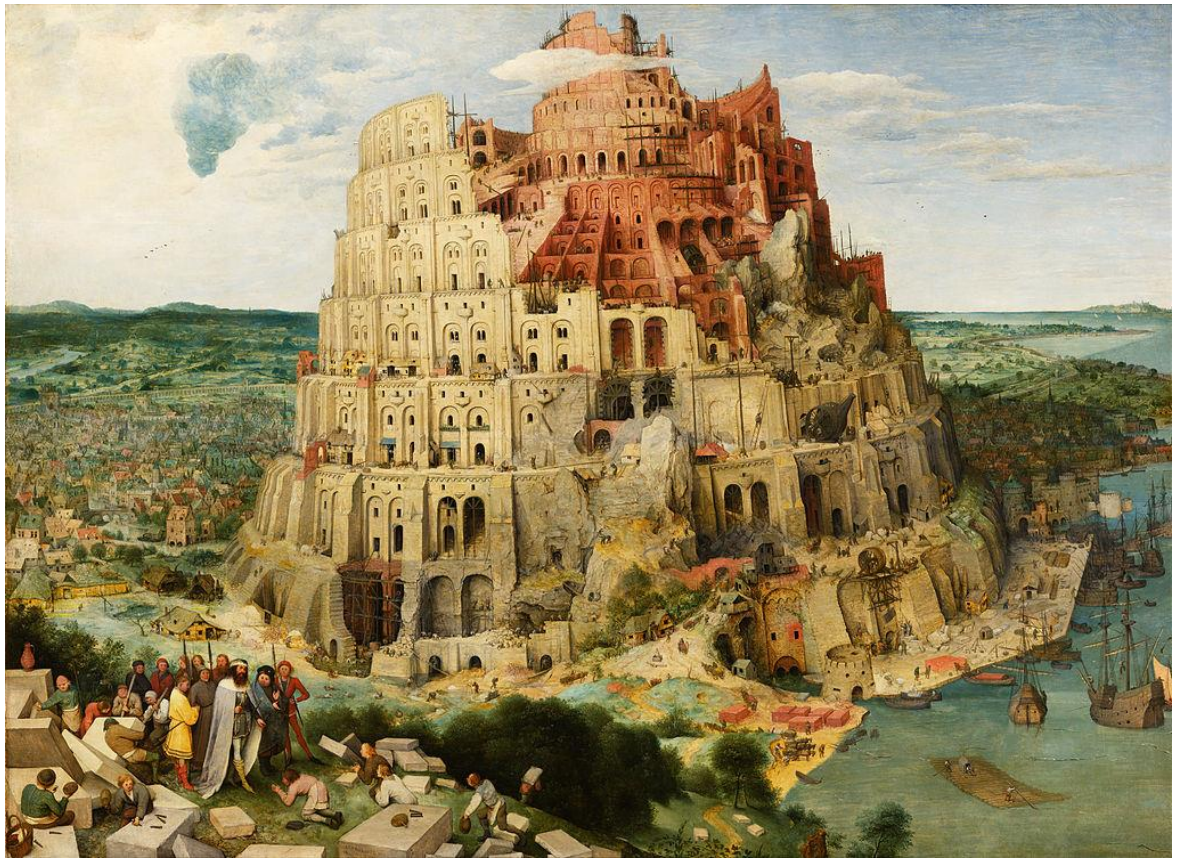
Why is structure important?



http://www.news.wisc.edu/newsphotos/ironVI.html

Structure is the foundation for ordered, managed evolution

What determines structure?



The tower of babel by Pieter Bruegel the Elder (1563)

Architecture! Architecture! Architecture!

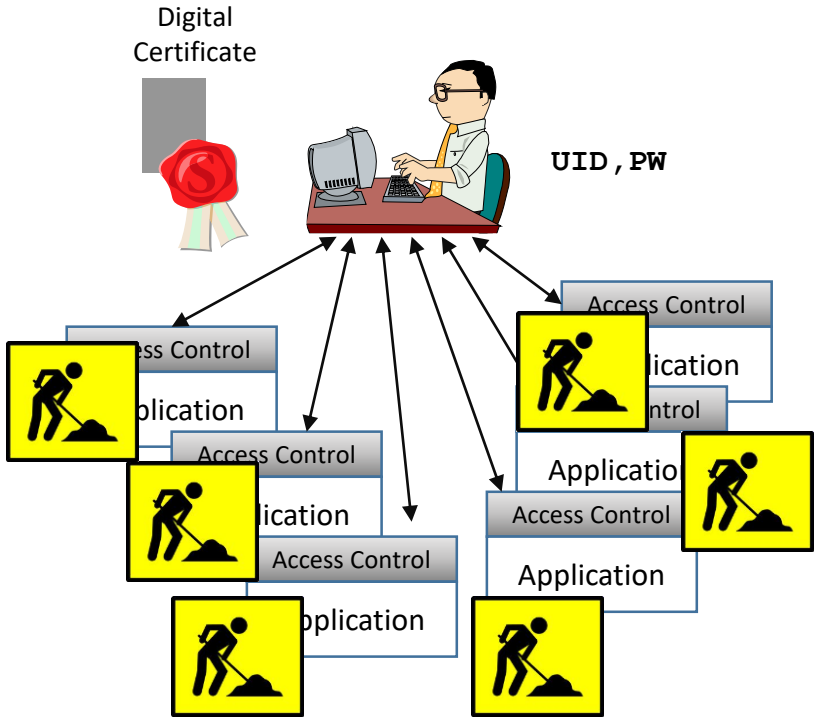
Digital Certificate

New Requirement: Authentication by **`Digital Certificate`**

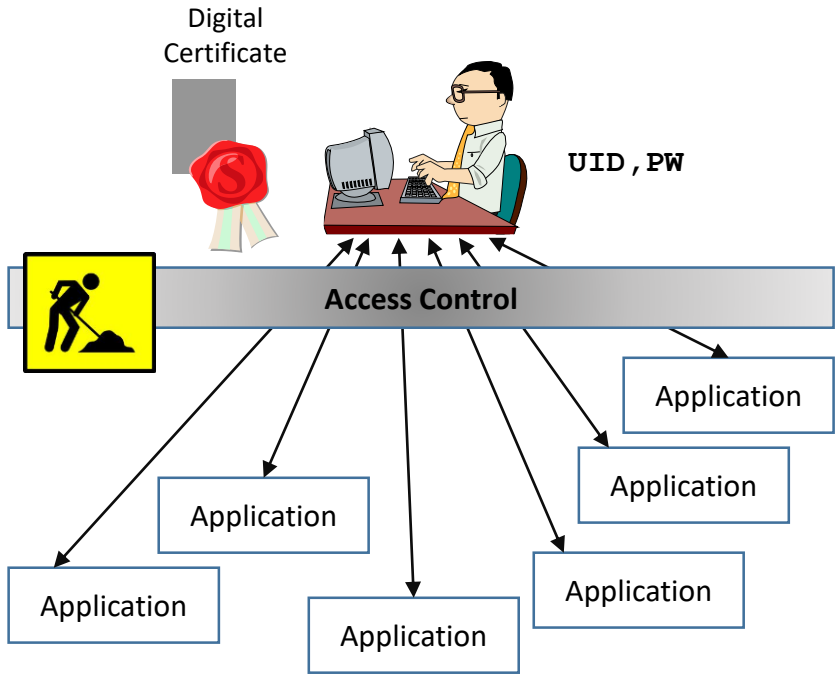
Example: Access Control
(Applications Security)

Impact of a change: 5'000 privacy-critical banking applications

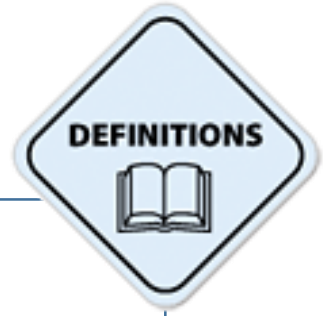
Structure 1: Distributed Access Control



Structure 2: Central Access Control



Definition: **IT Architecture**

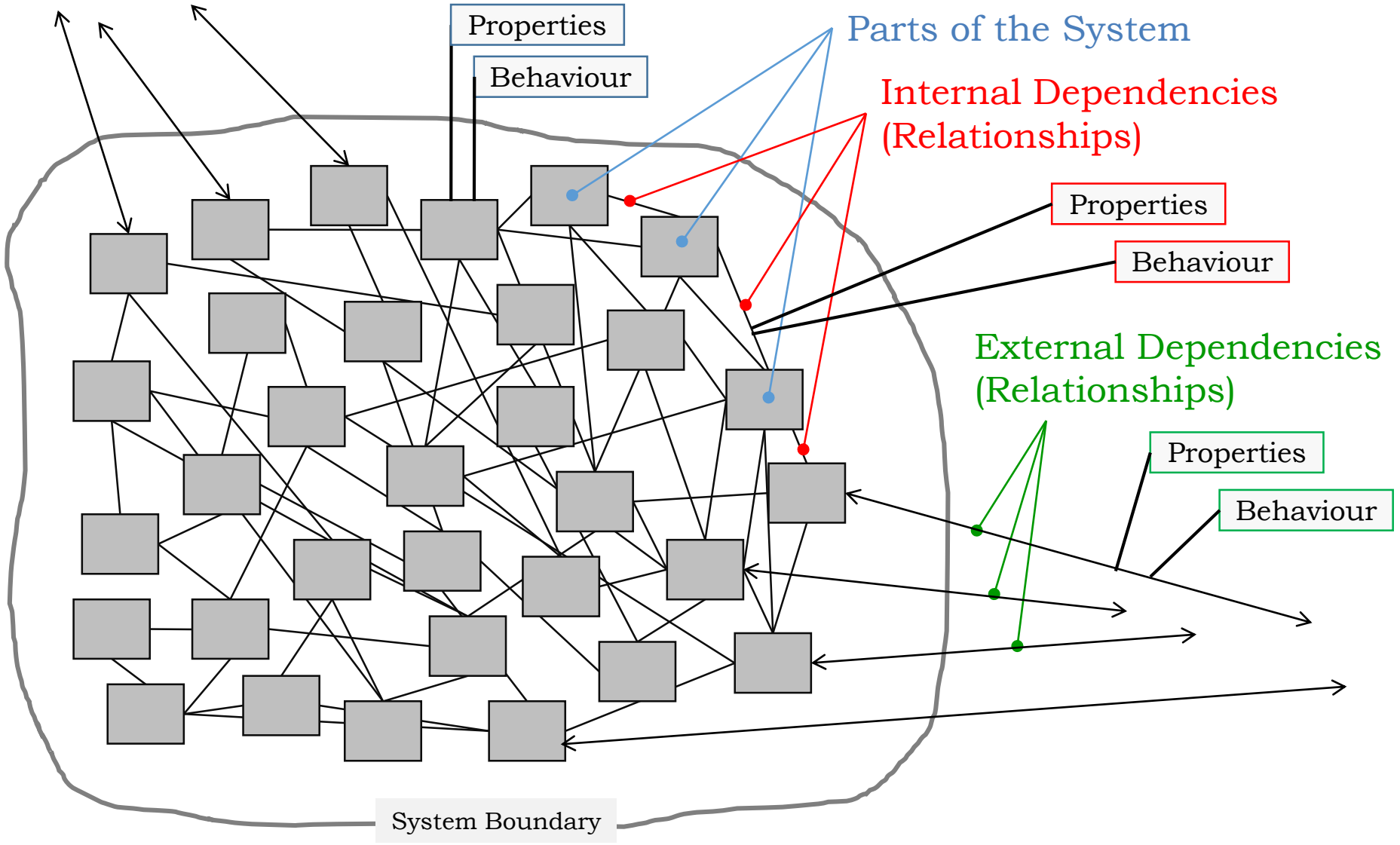


IT Architecture Definition:

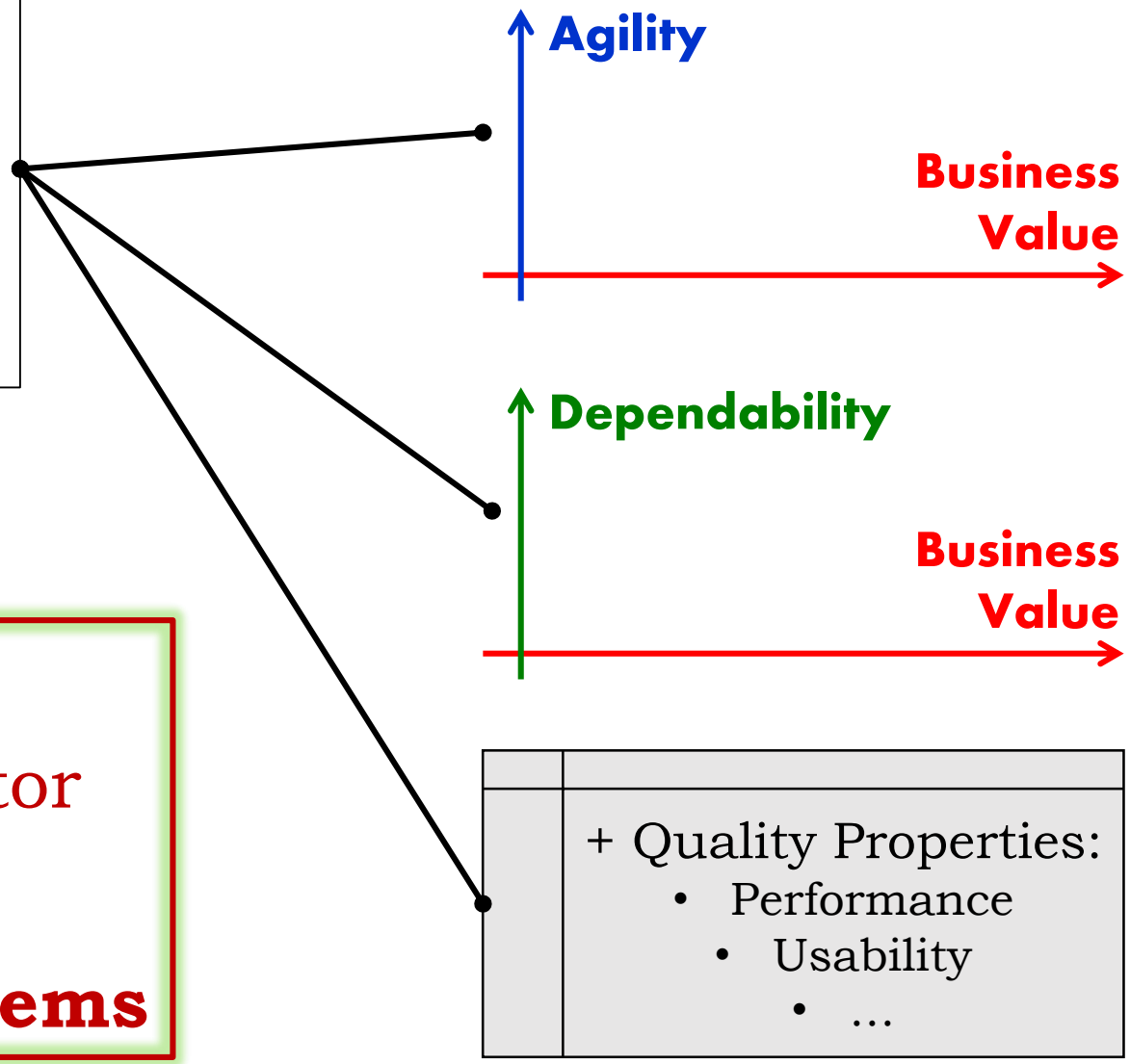
"The fundamental *organization* of a system embodied in its *parts*, their *relationships* to each other and to the environment, and the *principles* guiding its design and evolution"

[adapted from IEEE00]

Definition:
IT Architecture



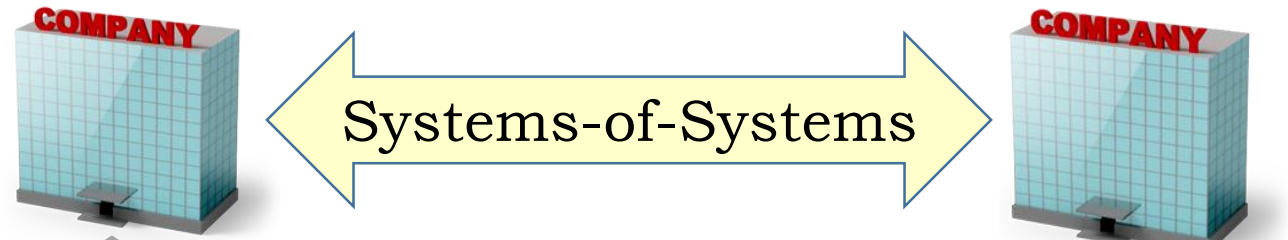
The structure of the system – i.e. its **architecture** – determines to a large extent the properties of the system



Architecture
is the most important factor
for
future-proof software-systems

Future-Proof Software-Systems [Part 2]

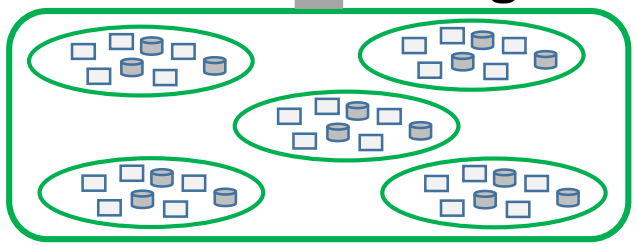
Architecture Levels



Enterprise-Architecture

SoS-Architecture

Integration

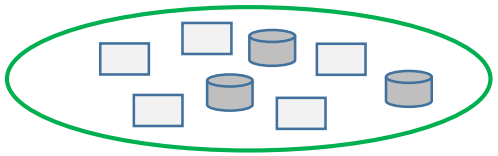


Application Landscape

Application Landscape Architecture

Integration

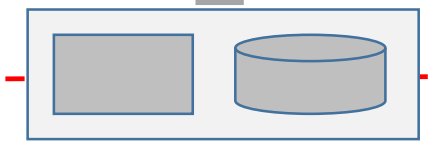
Technology-independence



Application

Application-Architecture

Integration



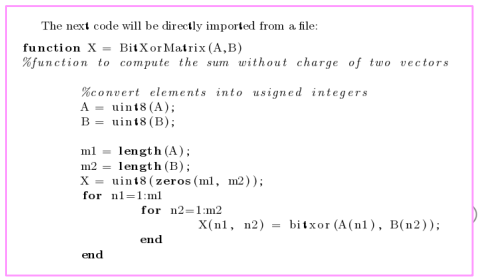
Component

Component-Architecture

Component-Design

Integration

Technology-dependence



```
The next code will be directly imported from a file:
function X = BitXorMatrix(A,B)
%function to compute the sum without charge of two vectors

        %convert elements into usigned integers
        A = uint8(A);
        B = uint8(B);

        m1 = length(A);
        m2 = length(B);
        X = uint8(zeros(m1, m2));
        for n1=1:m1
                for n2=1:m2
                        X(n1, n2) = bitxor(A(n1), B(n2));
                end
        end
```

Program Module

Program-/Module-Design

Project Types:

a) «Greenfield»:

https://c1.staticflickr.com

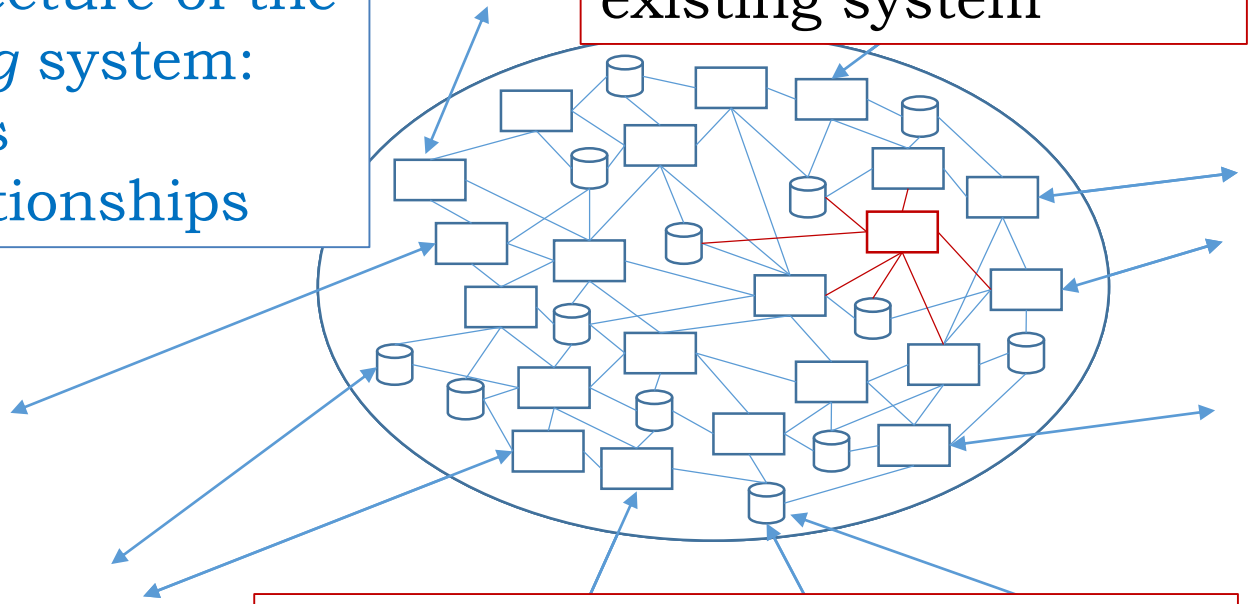
The system is new and can be built from scratch

⇒ Rare case!

b) «Integration»:

The new parts must be integrated into an existing system

Architecture of the *existing* system:
- Parts
- Relationships



Architecture of the *new* element:
- Parts
- Relationships

System/Software Engineering/Development Process

http://www.ovocreatives.com

http://stevendwood.com

Sum of all decisions

IT Architect

System/Software Engineering/Development Process

optimum fit into ***existing*** system

Architecture development is a *front activity*, i.e. it must be done (mostly) before the actual software development starts

Architecture Development

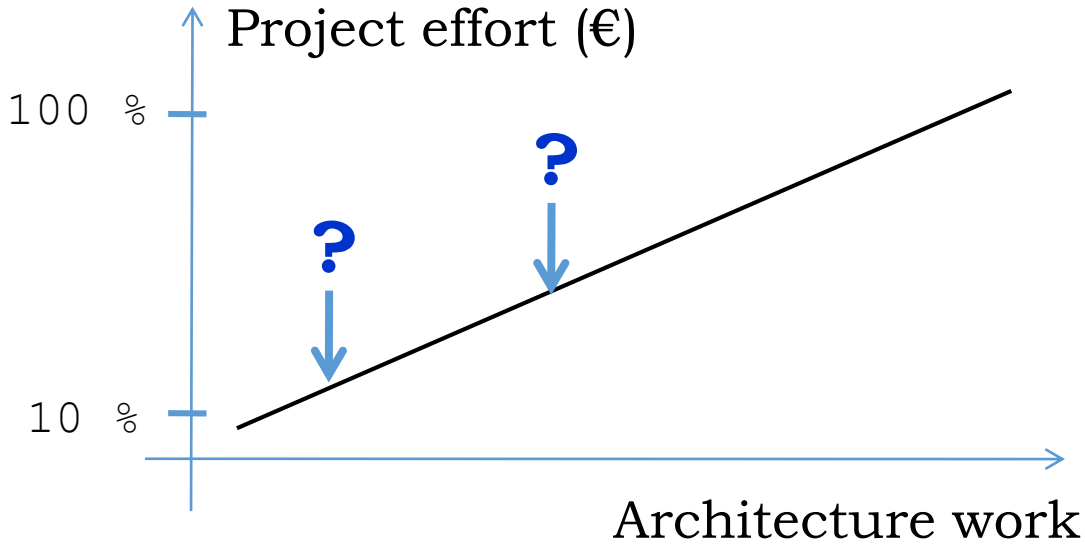
Design, Implementation, Deployment

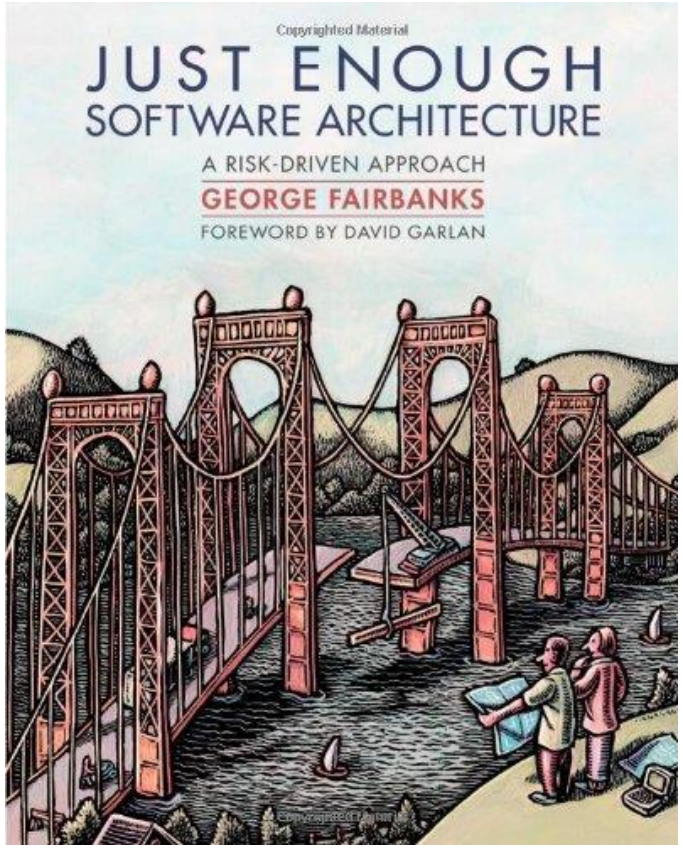
adequate architecture of ***new*** parts & relationships

How much shall we invest into architecture devlopment?

- **Money (5%, 12%, 27%, ...) ?**
- **Time (3%, 11%, 21%) ?**

How much Architecture is enough?





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

Answer:

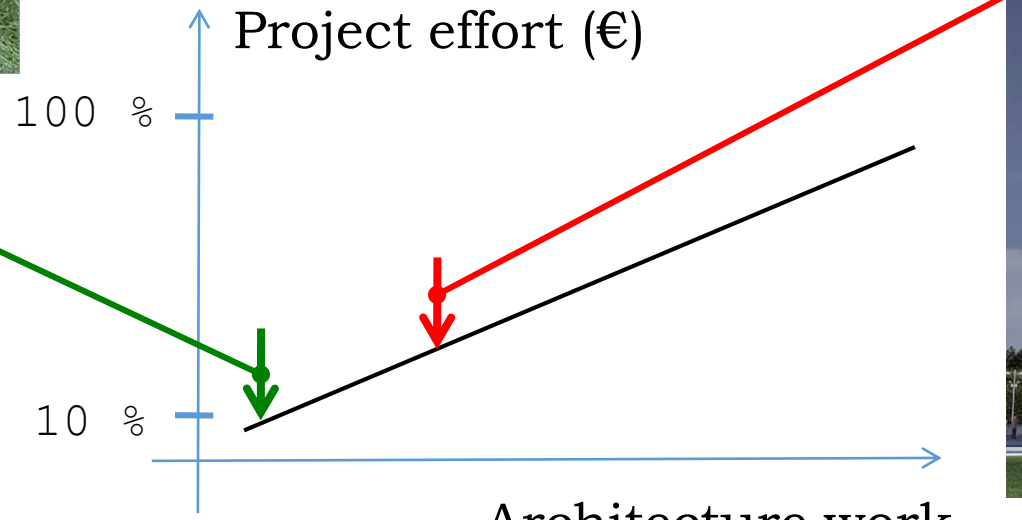
- System creation/extensions with **high risk** need much architecture work
- System creation/extensions with **low risk** need little architecture work

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

How much Architecture is enough?

Low Risk

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



High Risk

http://www.skyscrapernews.org

How much Architecture is enough?

Architecture Evaluation

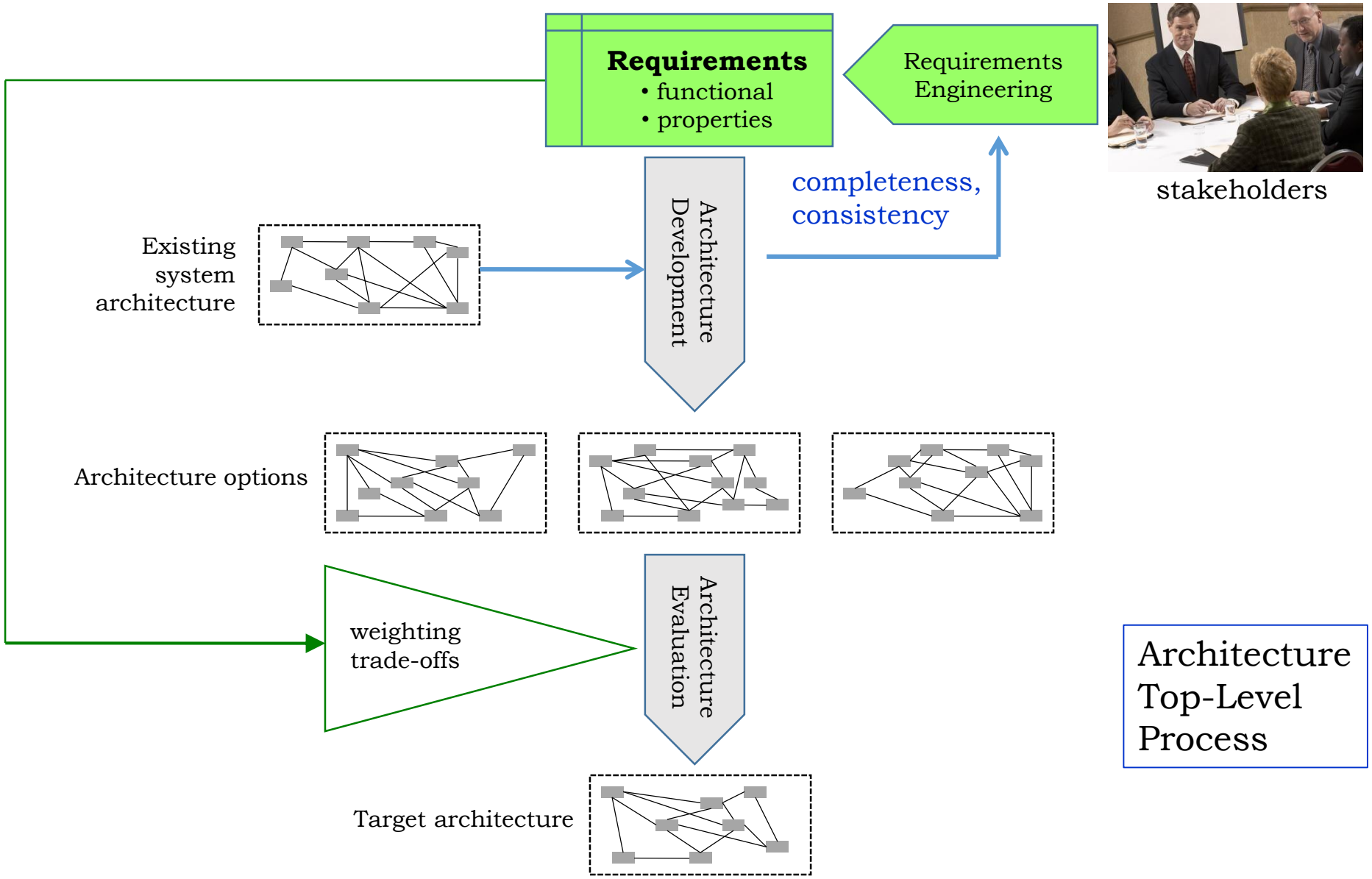
When have we done enough architecture work?

How do we know that we have a good architecture?

http://2.bp.blogspot.com

Architecture Principles

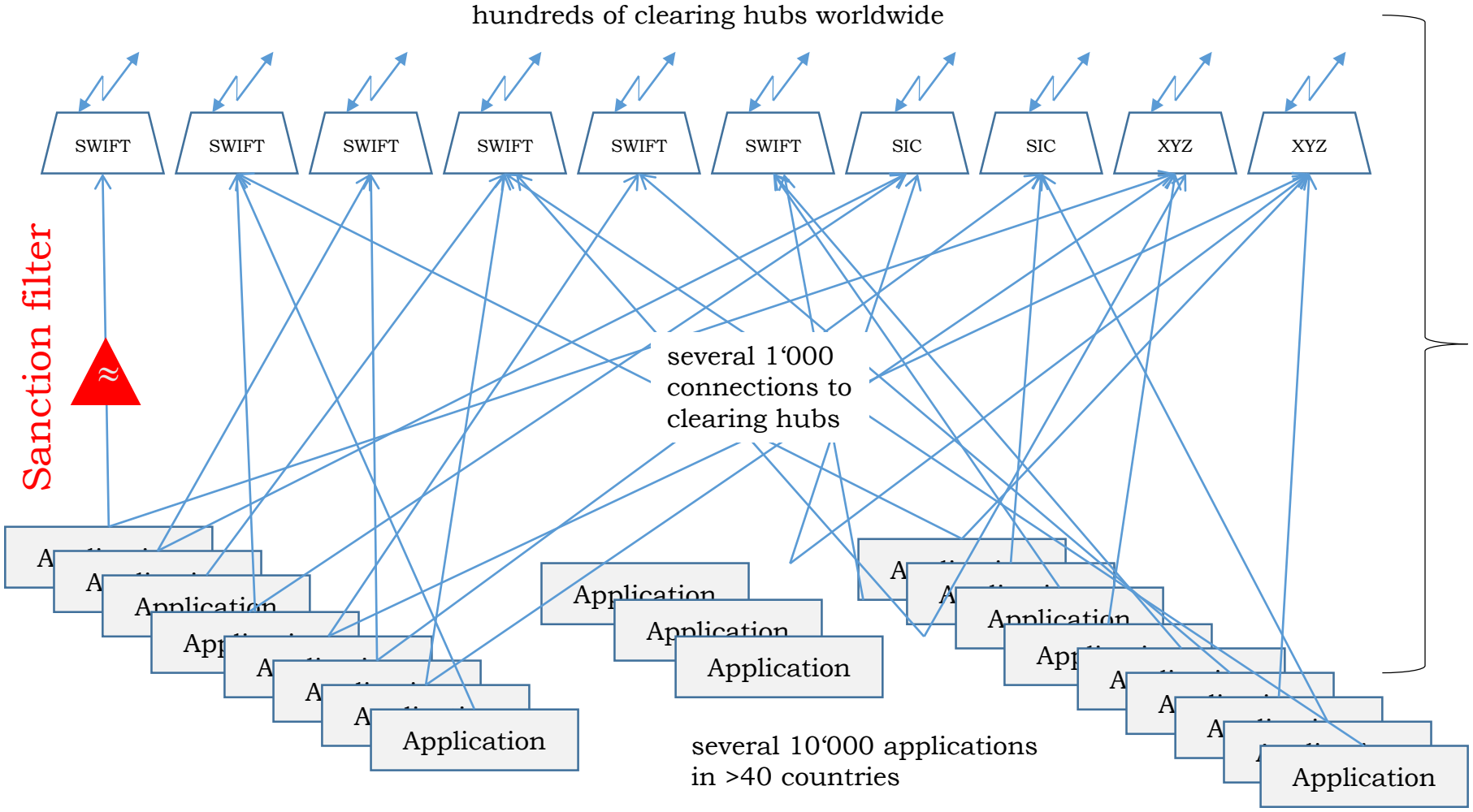
Future-Proof Software-Systems [Part 2]



Architecture Top-Level Process

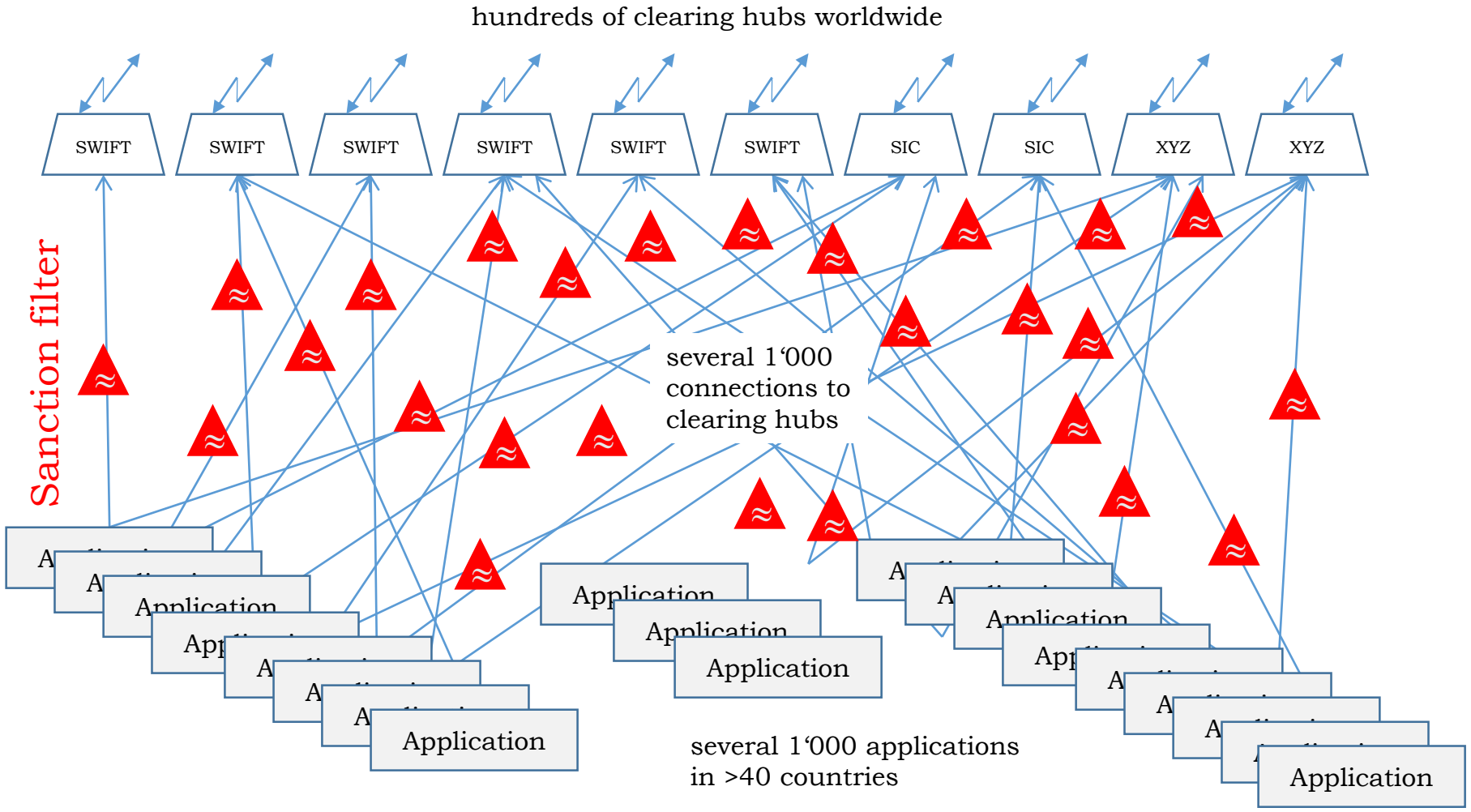
New legal requirement:
„Strictly enforce embargo lists worldwide“

Example: Sanction Filter
(Financial embargo enforcement)



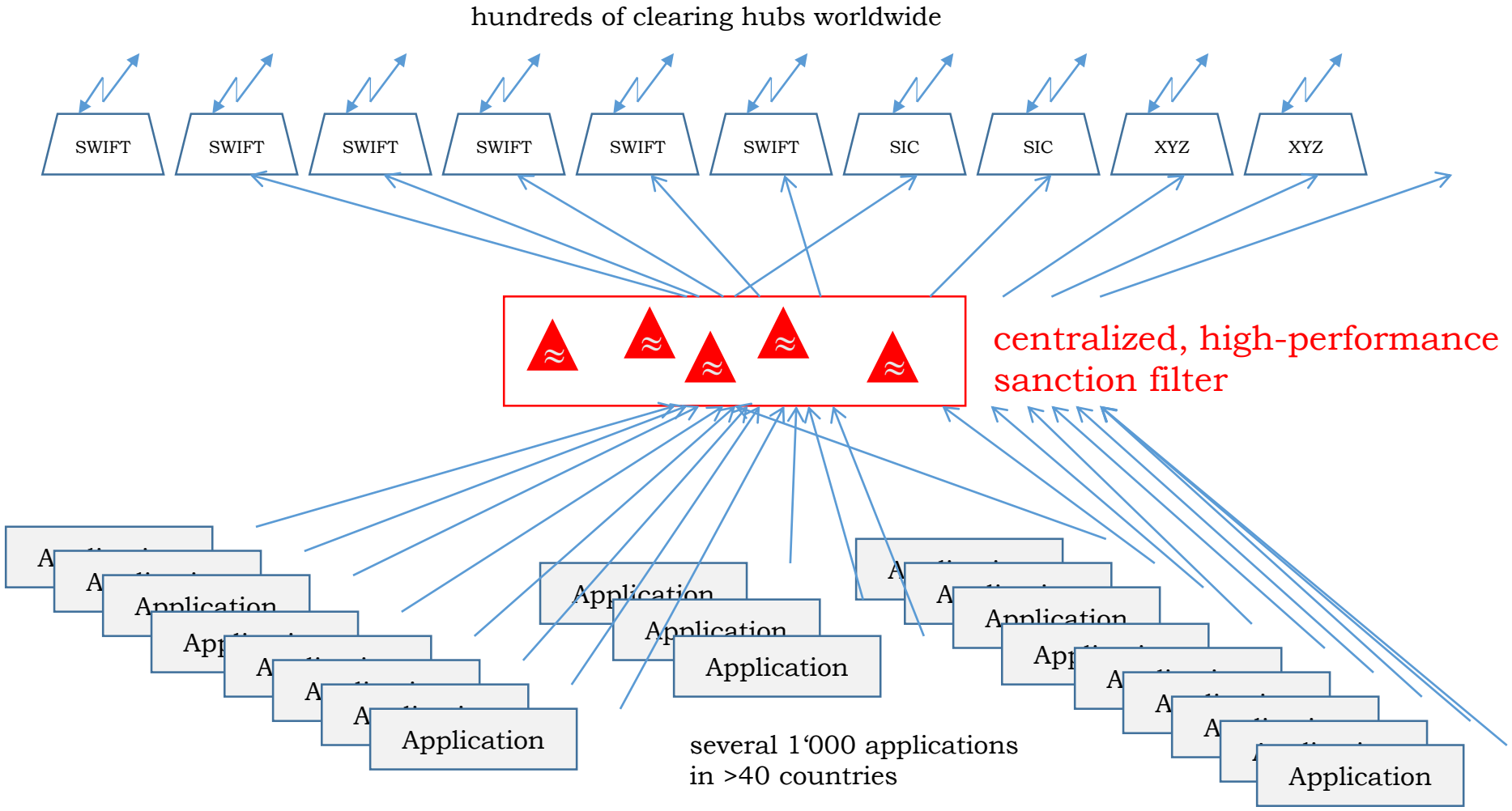
Architecture option 1:
Fully decentralized installation

Example: Sanction Filter
(Financial embargo enforcement)



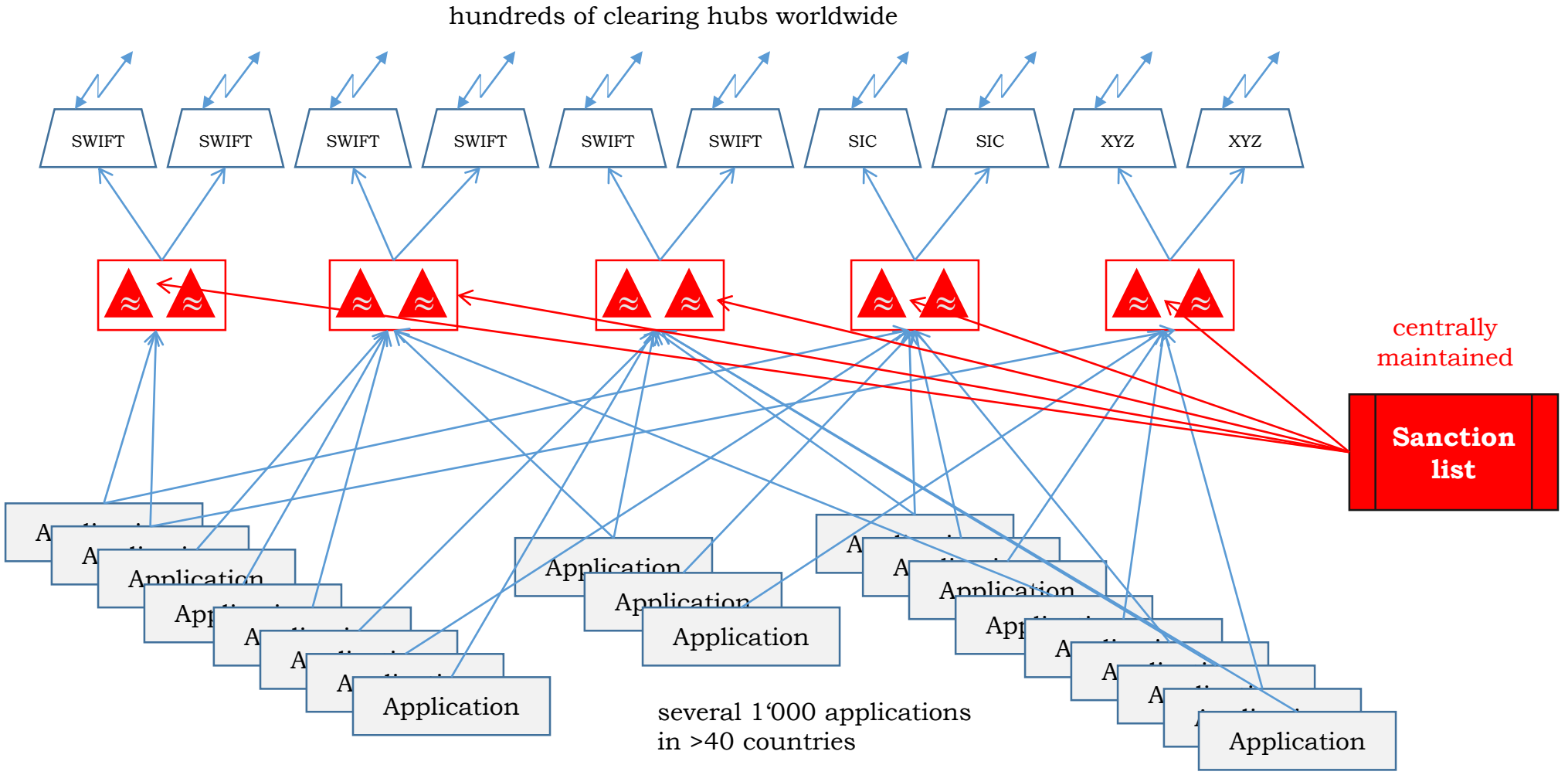
Architecture option 2:
Fully centralized installation

Example: Sanction Filter
(Financial embargo enforcement)

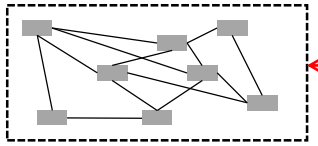


Architecture option 3:
Sub-clustering

Example: Sanction Filter
(Financial embargo enforcement)



Target architecture



Example: Sanction Filter

(Financial embargo enforcement)

Criteria	Option 1: fully decentralized	Option 2: fully centralized	Option 3: Sub-clustering
Performance	3	1	2
Security	1	3	2
Maintainability	1	3	3
Dependability	3	1	2
Implementation cost	1	2	3
Operational cost	1	3	2
Match with organizational structure	1	1	3
Governance	1	1	3
Legal & compliance conformance	2	3	3
Archiving	1	3	2
Assessment	15	21	25

1 = low
2 = average
3 = good

Future-Proof Software-Systems [Part 2]

Good Architecture:

- Manages essential **complexity**
- Minimizes accidental **complexity**
- Provides optimal **changeability** (= minimum resistance to change, DevC, TtM)
- Enables **dependability** and other quality properties
- Reduces the impact of **uncertainty**
- «Fun to work»

Bad Architecture:

- Difficult to **understand**, maintain and evolve
- Messy **dependencies** («far effects»)
- **Erosion**: «Path to Death»
- Entangled quality properties (Orthogonality)
- Demotivating, «overpriced» work

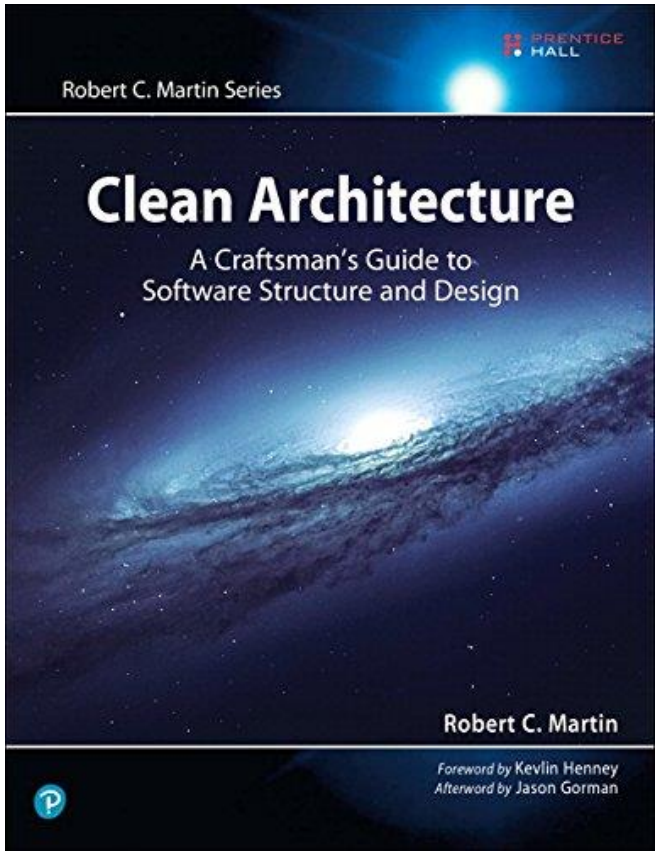
https://www.linkedin.com

The *structure* of a system is defined by its **architecture**.

The architecture must be adequate and follow proven **architecture principles**.

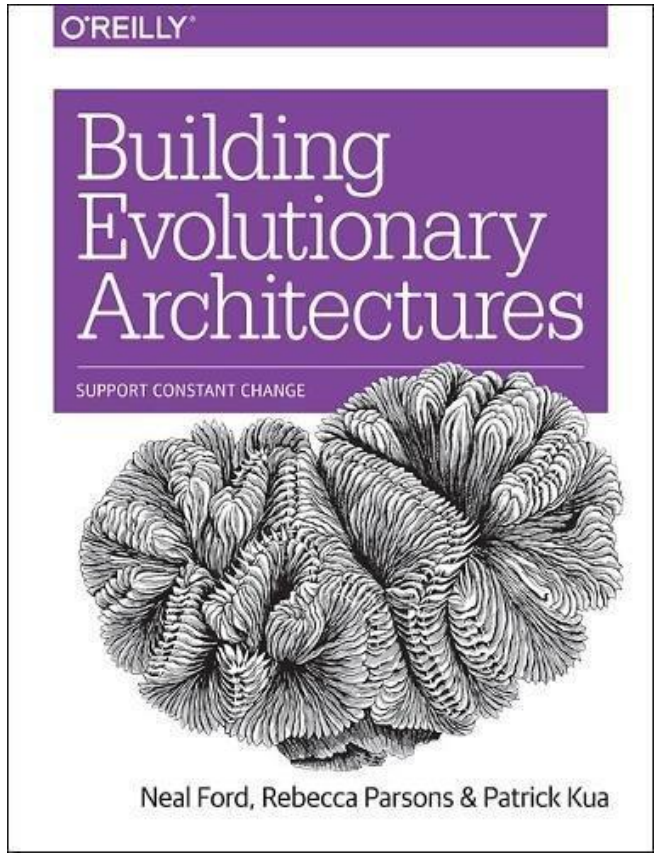
Architecture is a continuously evolving, managed, **highly valuable key artefact**!

Textbook



Robert C. Martin:
Clean Architecture – *A Craftsman's Guide to Software Structure and Design*
Prentice Hall Inc., USA, 2017. ISBN 978-0-134-49416-6

Textbook



Neal Ford, Rebecca Parsons, Patrick Kua:
Building Evolutionary Architectures – *Support Constant Change*
O'Reilly UK Ltd., 2017. ISBN 978-1-491-98636-3

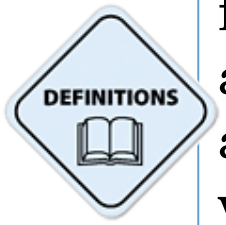
Industrial Architecture Framework

http://robertvbinder.com

Complex System

Categorization

https://www.indoortrend.com

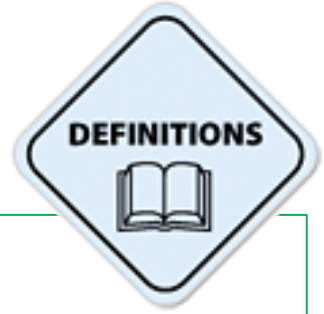


An **architecture framework** establishes a common practice for creating, interpreting, analyzing and using architecture descriptions within a particular domain of application or stakeholder community.
http://www.iso-architecture.org/42010/cm

Framework = Definition of Categories

Definition: **Industrial Architecture Framework**

Long-lived, industrially or commercially relevant IT-system

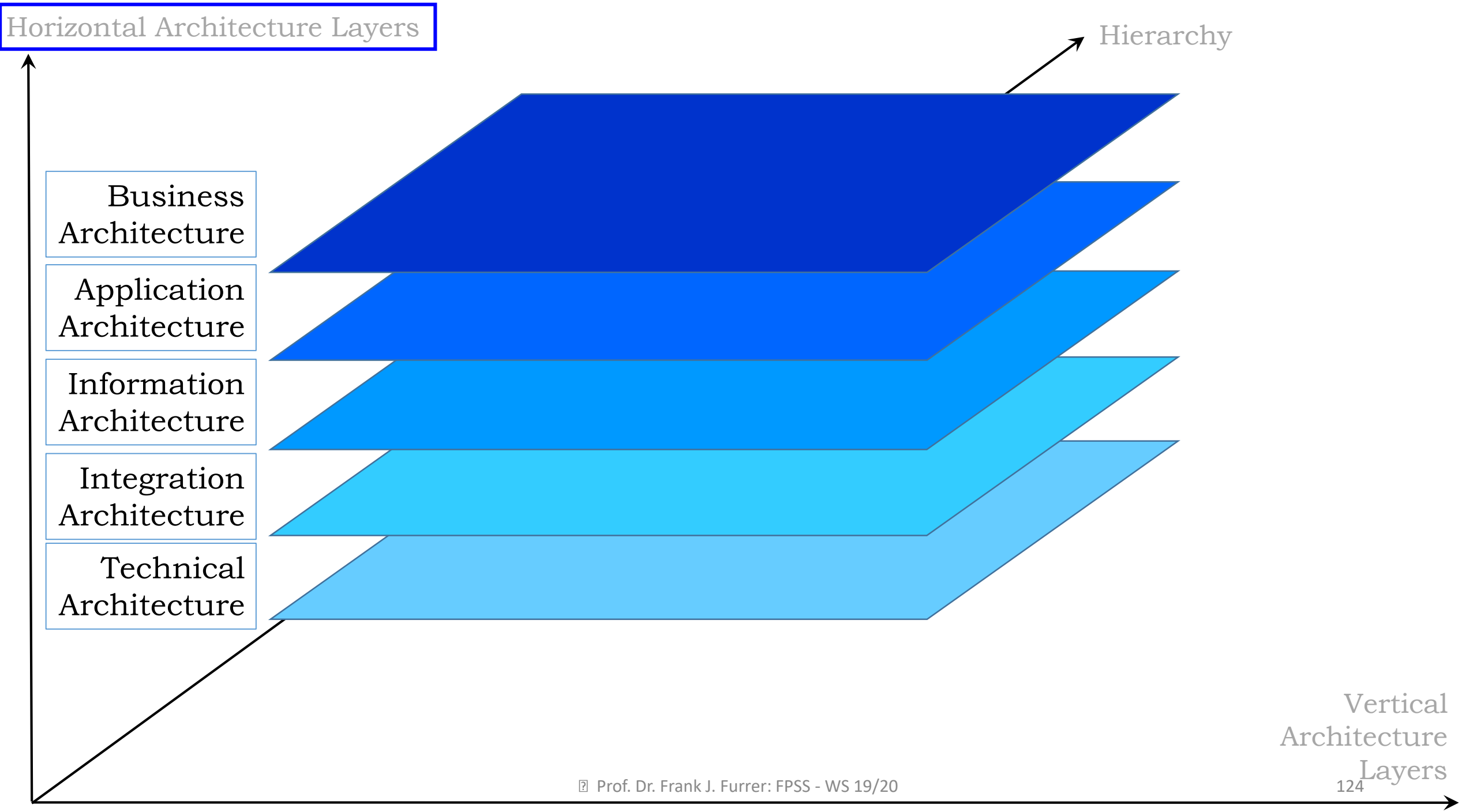


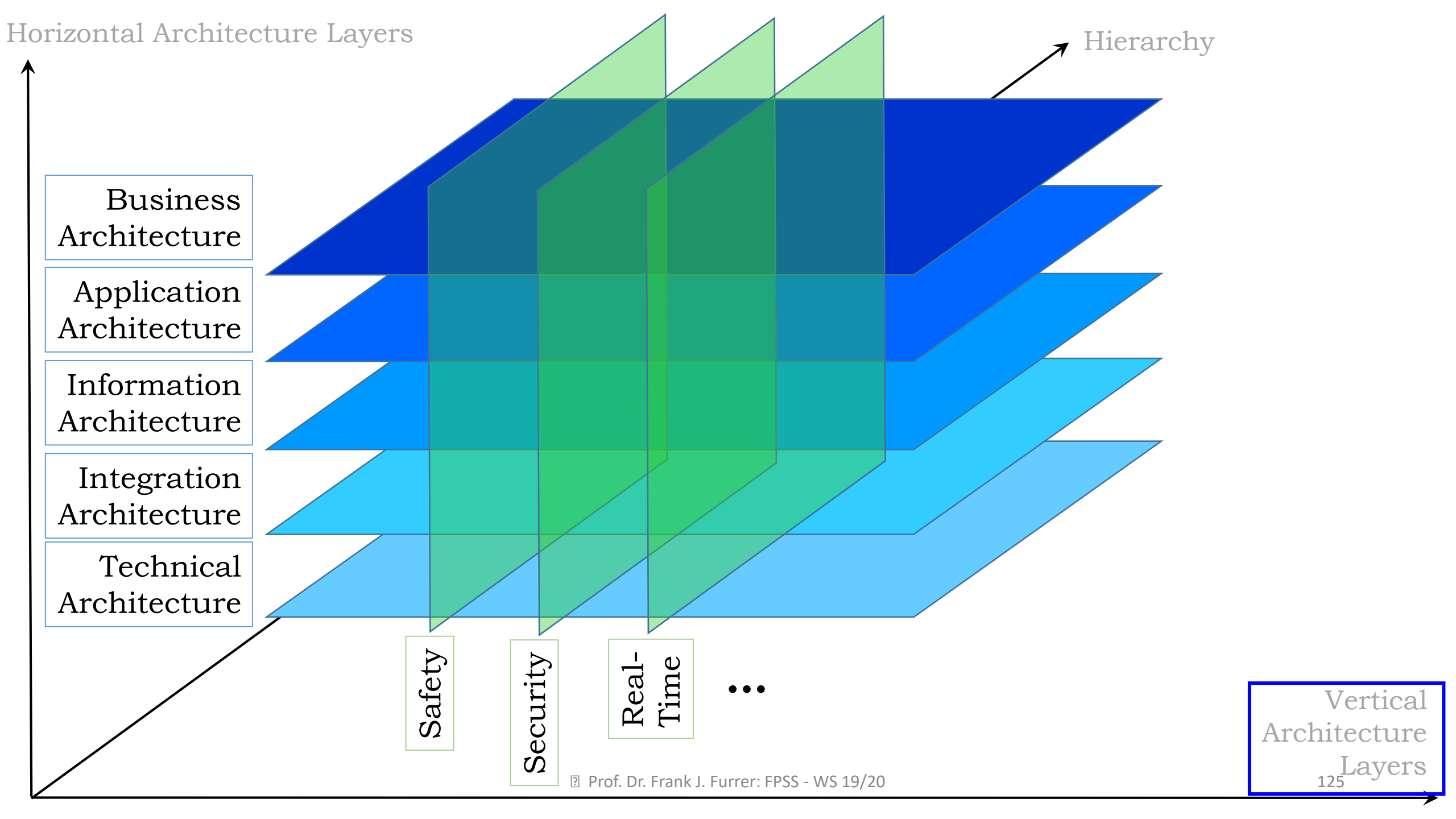
Industrial Architecture Framework =

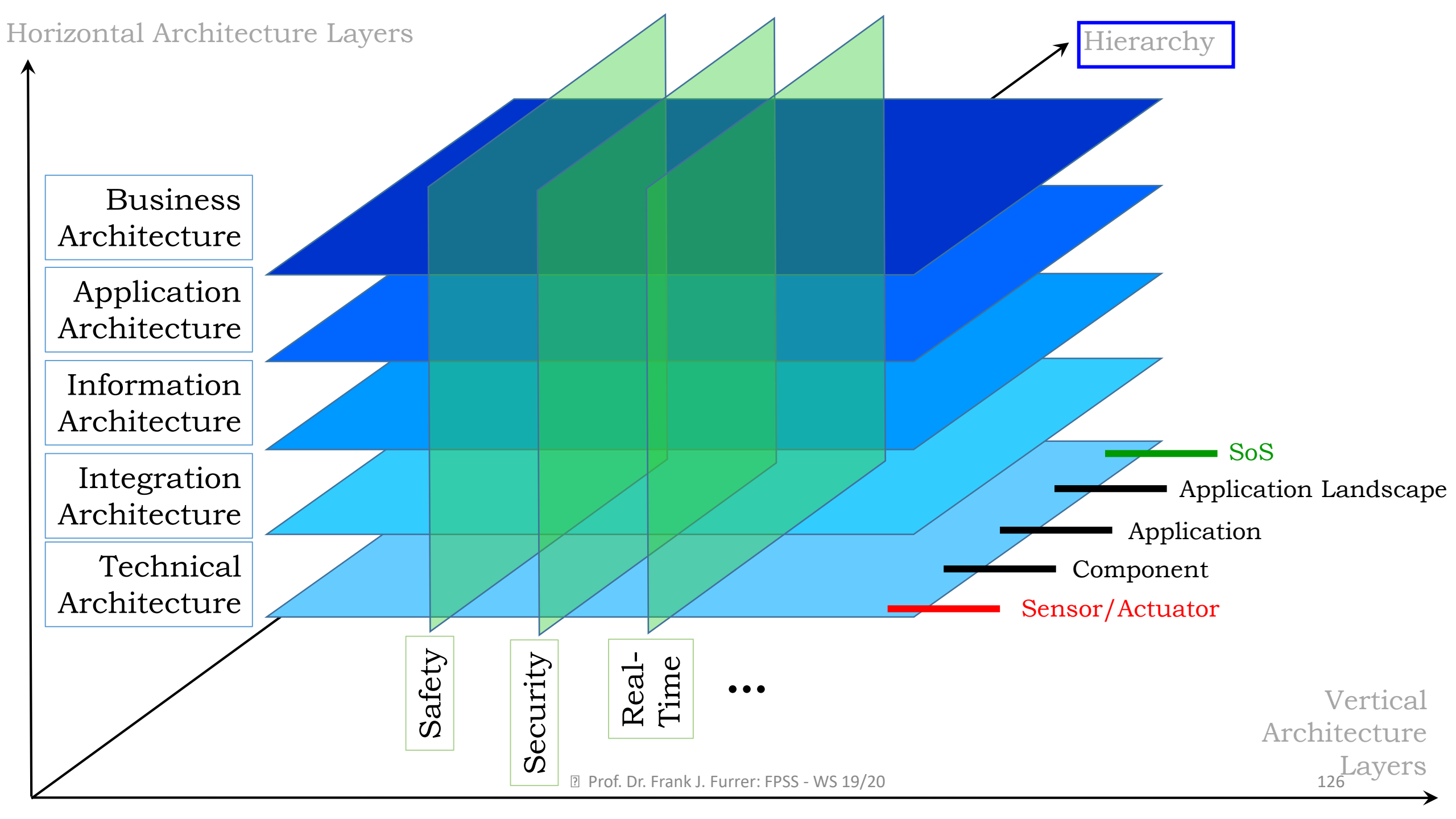
A conceptual framework for structuring and separating the functionality and the quality properties of IT-systems to enable partitioning and life-cycle management.

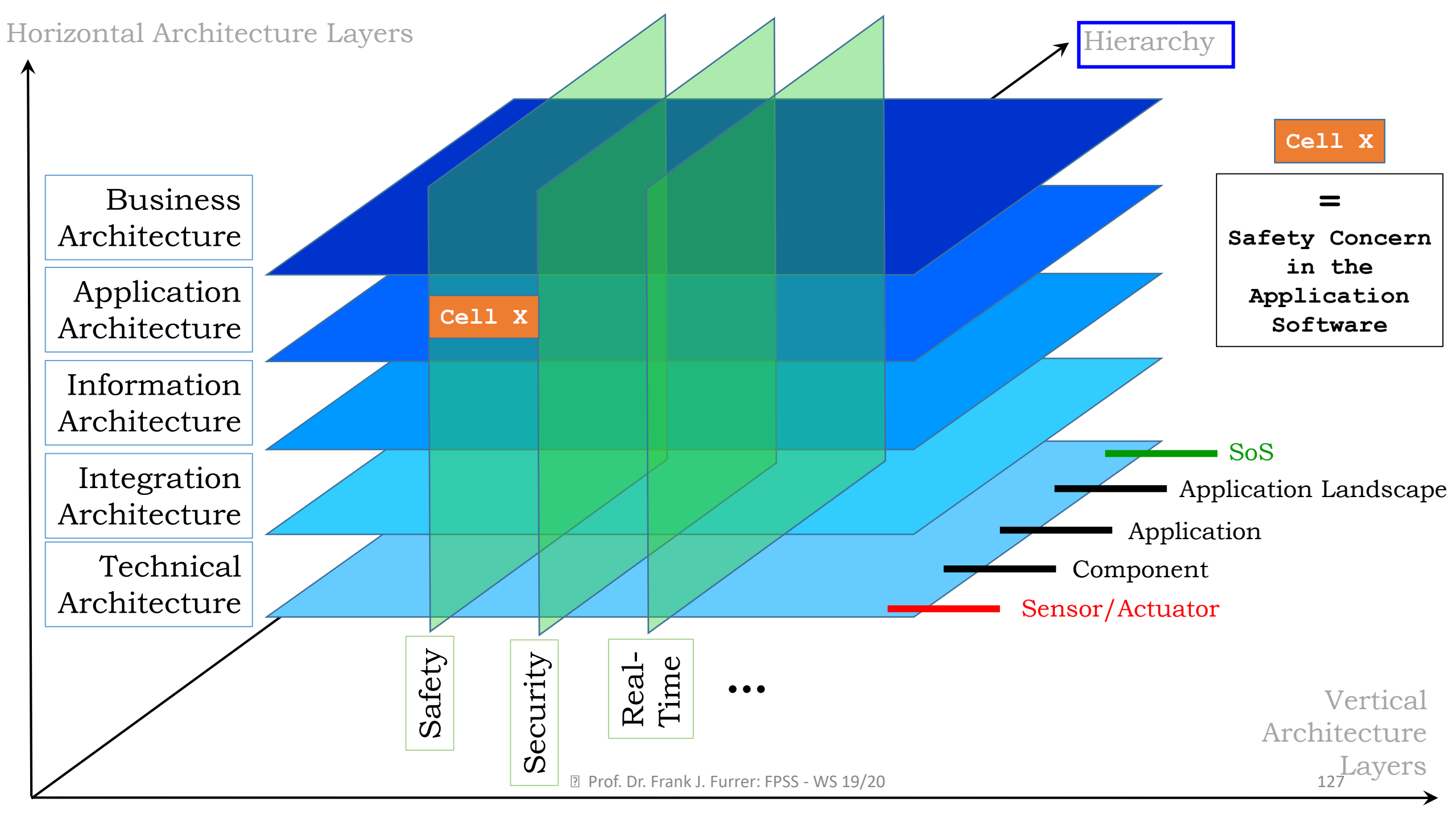
http://www.dedoose.com

Objective:
Separate and partition the dimensions of an IT-system in order to organize and manage both complexity and the stakeholders


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


Horizontal Architecture Layers
Hierarchy
Cell X
=
Safety Concern in the Application Software
Business Architecture
Application Architecture
Information Architecture
Integration Architecture
Technical Architecture
Cell X
SoS
Application Landscape
Application
Component
Sensor/Actuator
Safety
Security
Real-Time
...
Vertical Architecture Layers

Cell X = Safety Concern in the Application Software

Industrial Architecture Framework Cells =

Allow assignment, structuring, and separating of the functionality and of the quality properties of IT-systems to enable partitioning and life-cycle management.

⇒ **Formulation of Powerful Set of Architecture Principles**, e.g.:

NEVER implement security functionality in the applications software

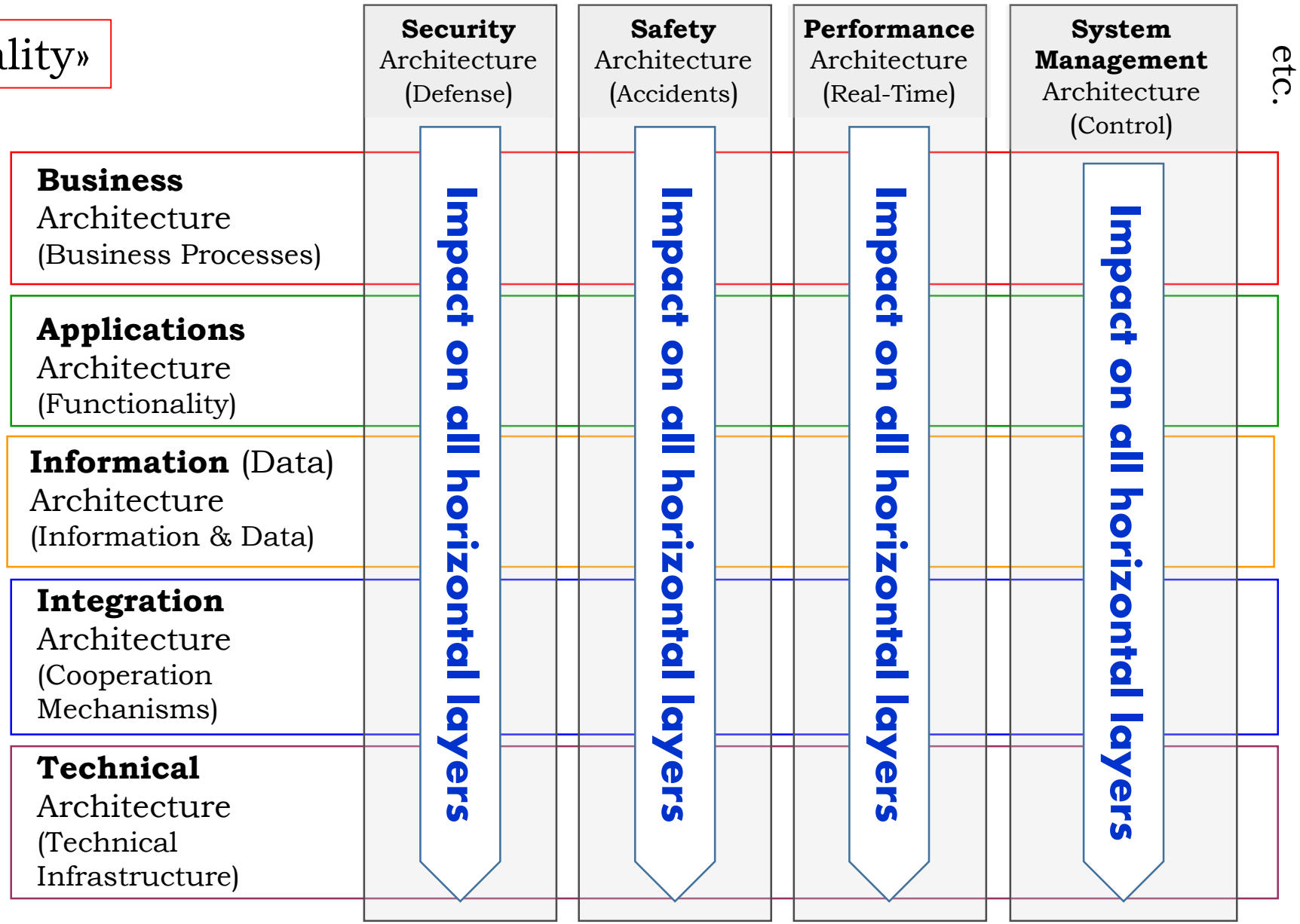
… but only allow calls to the security functionality

«Canon of Orthogonality»

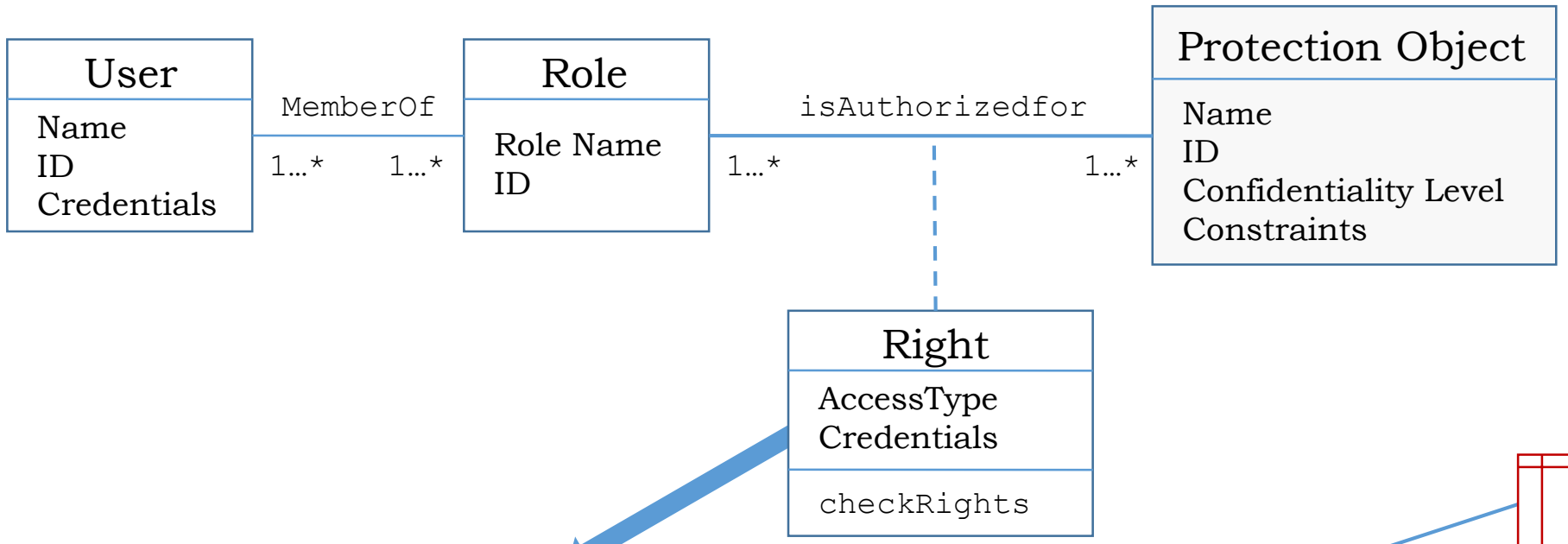
https://www.npmjs.com

Future-Proof Software-Systems [Part 2]

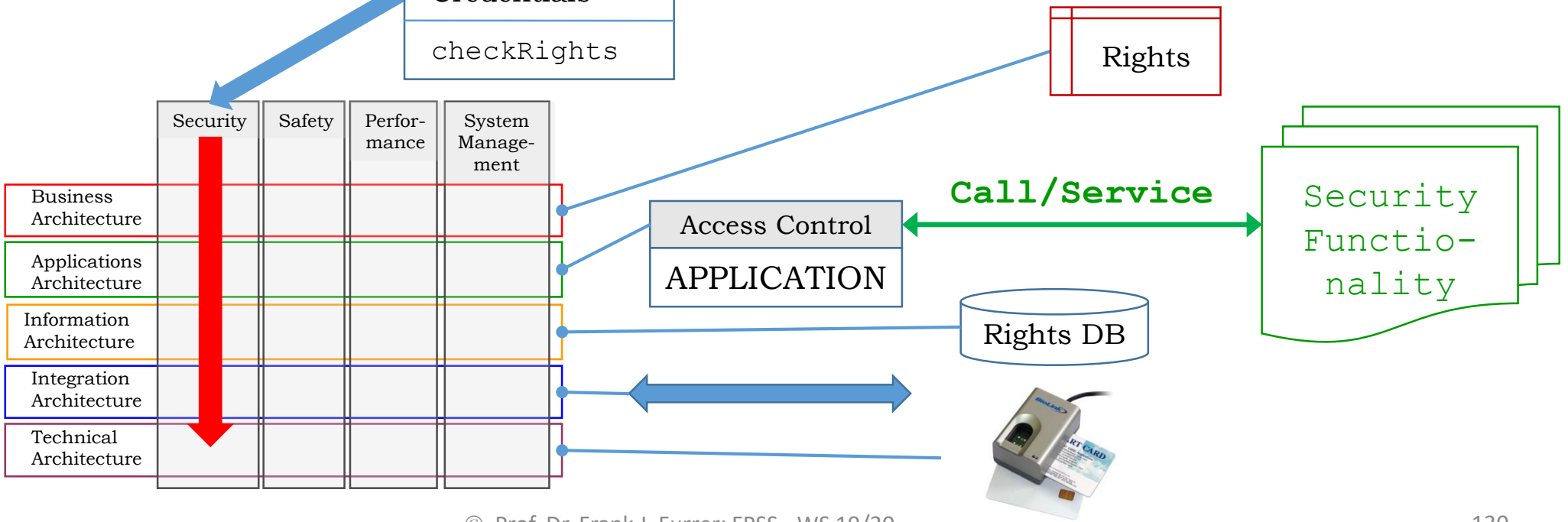
«Canon of Orthogonality»



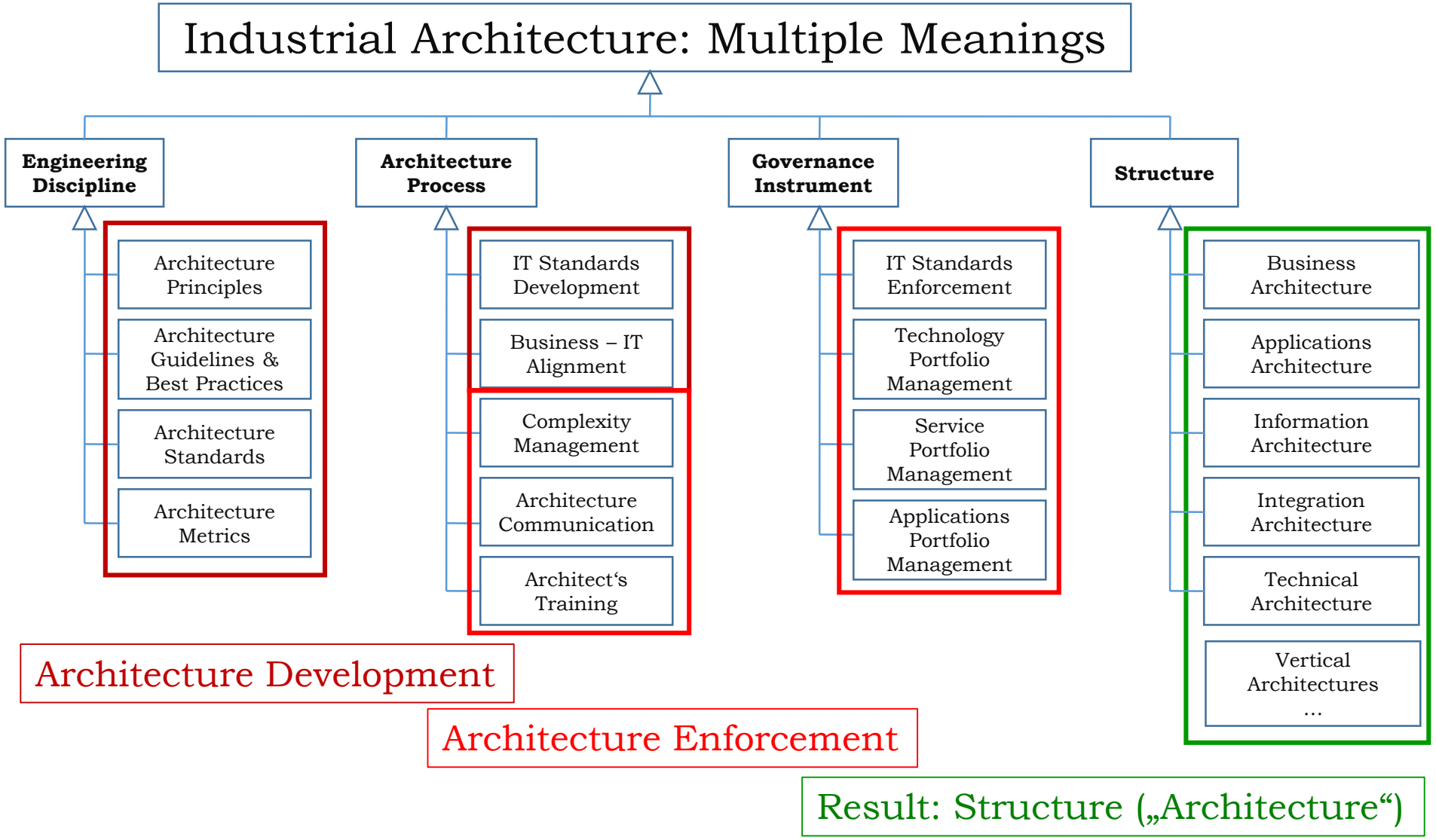
Example: Access Control (Security Architecture)



«Canon of Orthogonality»
NO security or technical functionality is implemented in the application software – these are all provided via calls/services



http://www.tecnwencn.com


Industrial Architecture: Multiple Meanings
Engineering Discipline
Architecture Process
Governance Instrument
Structure
Architecture Principles
Architecture Guidelines & Best Practices
Architecture Standards
Architecture Metrics
IT Standards Development
Business – IT Alignment
Complexity Management
Architecture Communication
Architect's Training
IT Standards Enforcement
Technology Portfolio Management
Service Portfolio Management
Applications Portfolio Management
Business Architecture
Applications Architecture
Information Architecture
Integration Architecture
Technical Architecture
Vertical Architectures
...
Architecture Development
Architecture Enforcement
Result: Structure („Architecture")

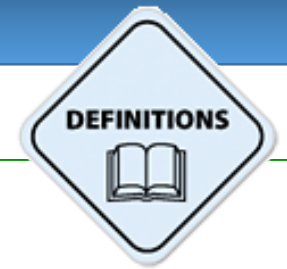
Architecture Principles and their use

Future-Proof Software-Systems [Part 2]

	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)					

For each of the horizontal and vertical architectures there are:

- **Architecture Principles**
- Architecture Patterns
- Frameworks
- Reference Architectures
- Industry Standards
- Best Practices



Architecture Principles:

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

Architecture Principles:

→ highly valuable architecture knowledge in proven & easily accessible form

→ teachable & enforcable

→ the foundation for the design, implementation and evolution of future-proof software-systems

… the birth of architecture principles (1972):

Programming Techniques — R. Morris Editor

On the Criteria To Be Used in Decomposing Systems into Modules

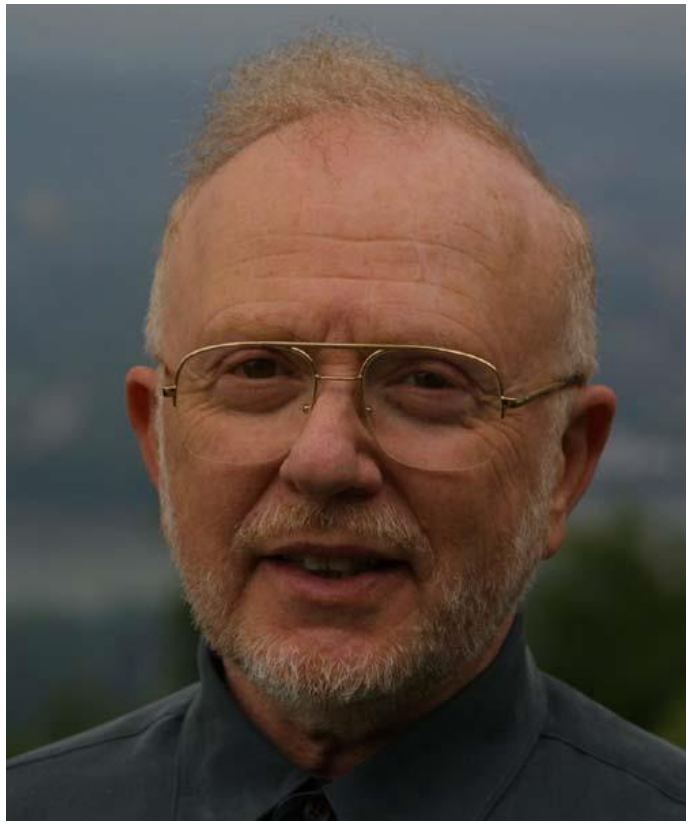
D.L. Parnas
Carnegie-Mellon University

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]

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



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

http://datapeak.net/computerscientists.htm

… a little bit of history:

The first computer program:

Lady Ada Lovelace (**1843**)

„*Computation of Bernoulli Numbers*“

http://axsoris.com

http://www.itp.net

2019:

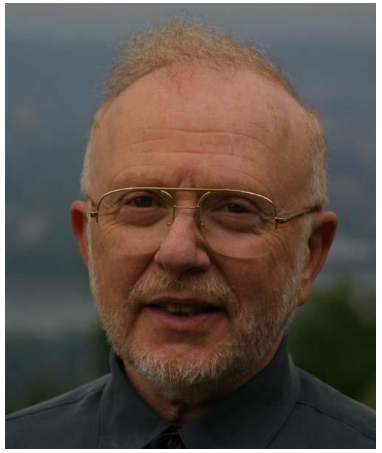
The world software market exceeds **$ 500 billion**

(not including embedded software!)

… Software production has become a major industry
⇒ and needs industrial rules, methods, processes

… some more history:

1843 – 1972: Software engineering is somewhat of a „black art“, mastered by experienced, talented individuals only



David L. Parnas

1972 – today:
Software engineering slowly becomes an *engineering discipline* with increasing maturity. Principles for good software engineering are discovered, applied and formal methods appear

2025 (?) … ?: Software engineering will evolve to *formal model engineering* with automatic software generation and provably correct programs

Recipes for good – i.e. future-proof – software-systems:

- **Architecture Principles**
- **Patterns**

Architecture Principles:

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

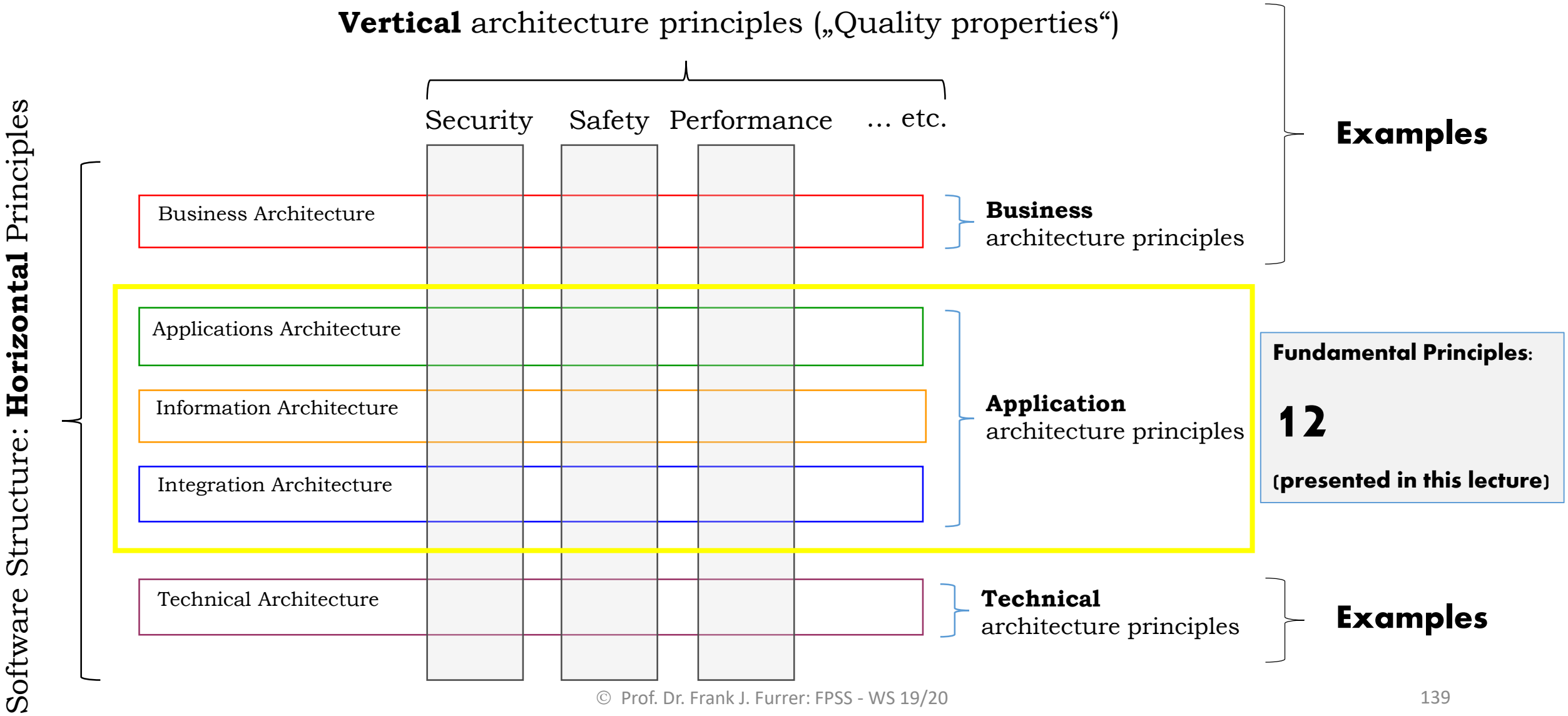
Patterns:

Proven, generic solutions to clearly specified architectural problems which can be adapted to the task at hand

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

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

How **many** architecture principles are needed ?



Fundamental Architecture Principles

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

http://www.holyoke.org

Architecture principles and patterns are the **knowledge-carriers** for future-proof software-systems

The **future-proof software-systems engineer** must know and understand the architecture principles and patterns. He must correctly apply them to his/her design

Architecture

Principles

=

Knowledge

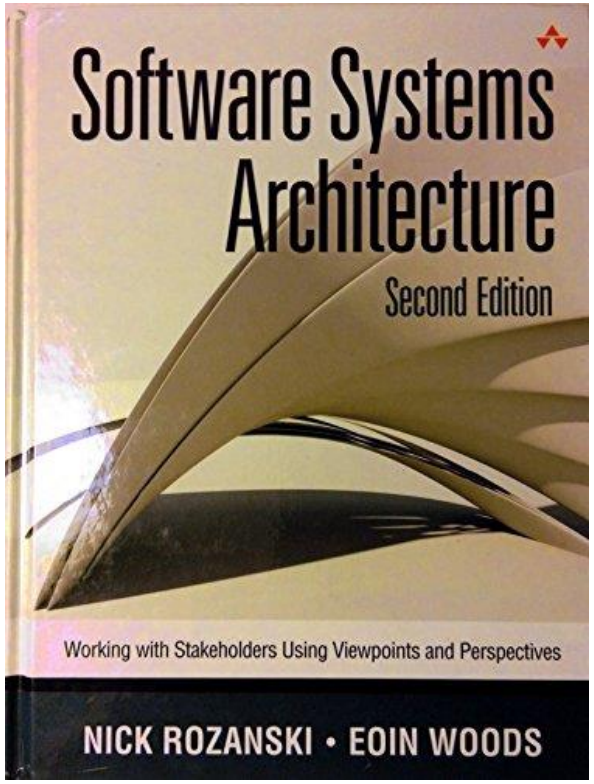
Toolbox

of the

Systems Architect

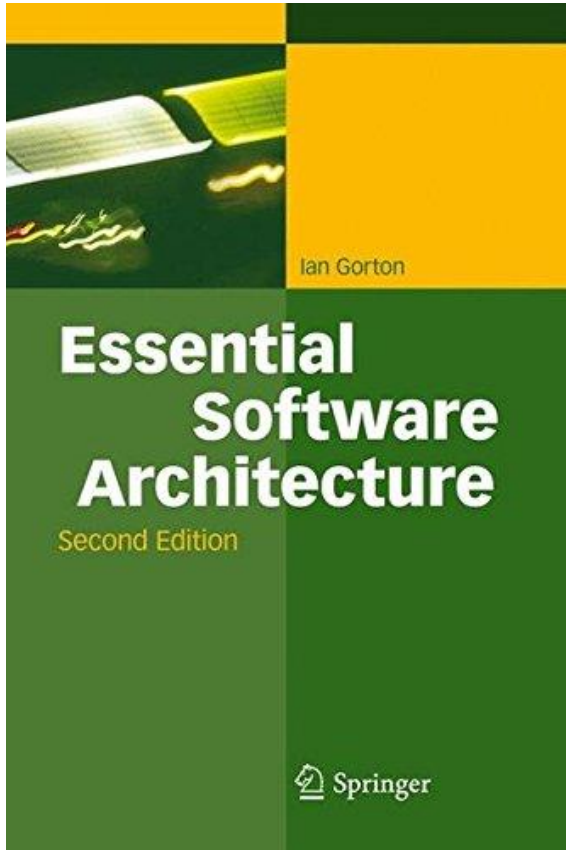
© NATURE

Textbook



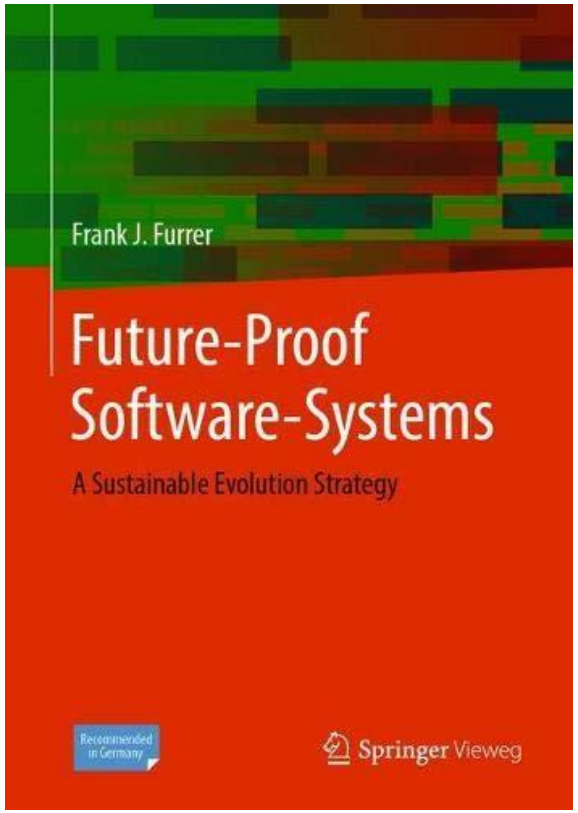
Nick Rozanski, Eoin Woods:
Software Systems Architecture – *Working With Stakeholders Using Viewpoints and Perspectives*
Addison Wesley, USA, 2nd revised edition, 2011. ISBN 978-0-321-71833-4

Textbook



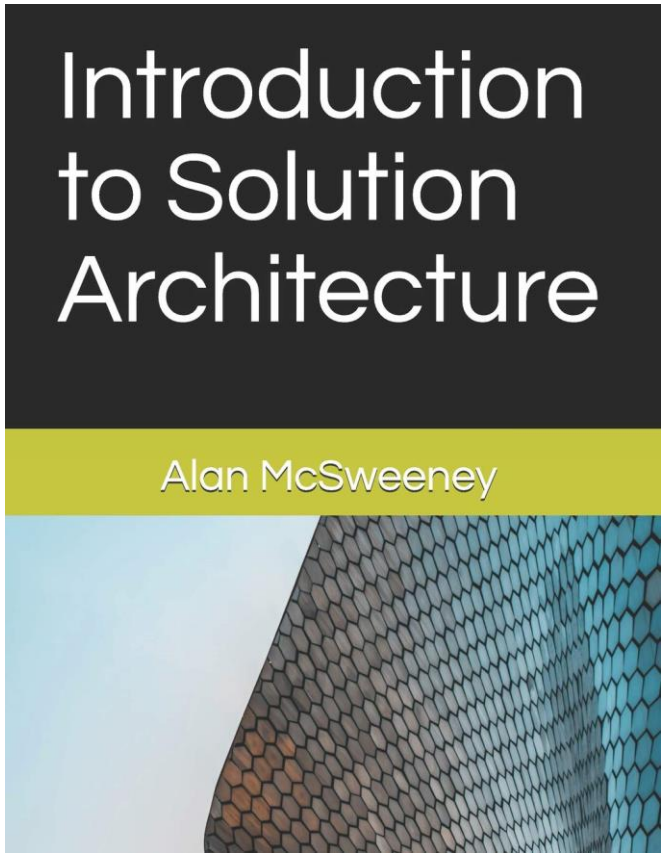
Ian Gorton:
Essential Software Architecture
Springer-Verlag, Germany, 2nd edition, 2011.
ISBN 978-3-642-19175-6

Textbook



Frank J. Furrer: **Future-Proof Software-Systems**
Springer Vieweg, Wiesbaden, Germany, 2019.
ISBN 978-3-658-19937-1

Textbook



Alan McSweeney: **Introduction to Solution Architecture**. Independently published, 2019. ISBN 978-1-7975-6761-7

Part 2