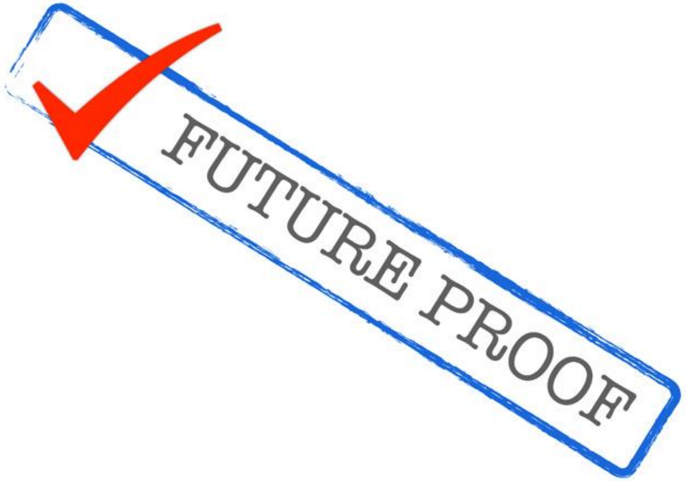


https://static1.squarespace.com

Future-Proof Software-Systems (FPSS)

Part 3D: Special Topics (2)

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

Version 1.0/08.12.2019

Future-Proof Software-Systems [Part 3C]

Special Topics = Specific IT-Architecture Areas related to IT-Systems

- Cyber-Physical Systems (CPS)
- Systems-of-Systems (SoS)
- Cyber-Physical Systems-of-Systems (CPSoS)
- Cloud Computing
- Microservices

Part 3C ✓

- Agile Manifesto and Future-Proof Software-Systems ?
- Domain Software Engineering
- Legacy System Migration/Modernization
- Software Product Lines

Part 3D

Agile Manifesto
and
Future-Proof Software-Systems ?

DEFINITIONS

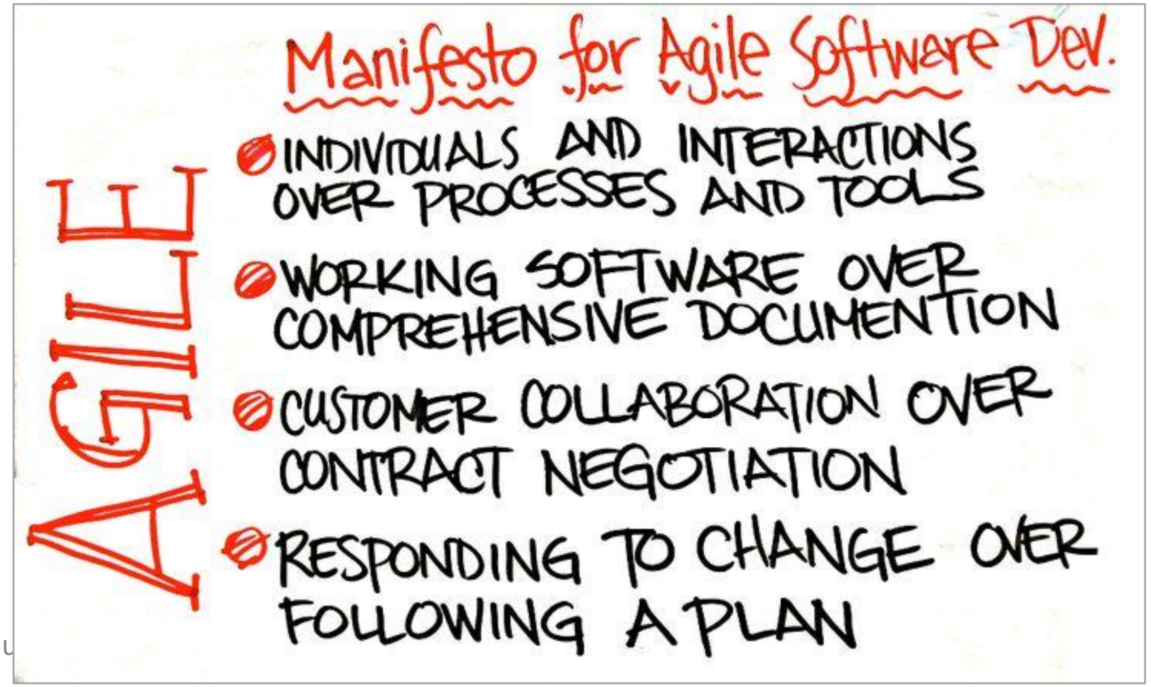
Agile Method:

An agile method is a software development method that is people-focused, communications-oriented, flexible, speedy, lean, responsive, and learning.

Qumer & Sellers, 2007

https://setandbma.files.wordpress.com

In 2001, a group of 17 programming gurus got together and developed the "**Agile Manifesto**"



Manifesto for Agile Software Dev.

AGILE

- INDIVIDUALS AND INTERACTIONS OVER PROCESSES AND TOOLS
- WORKING SOFTWARE OVER COMPREHENSIVE DOCUMENTION
- CUSTOMER COLLABORATION OVER CONTRACT NEGOTIATION
- RESPONDING TO CHANGE OVER FOLLOWING A PLAN

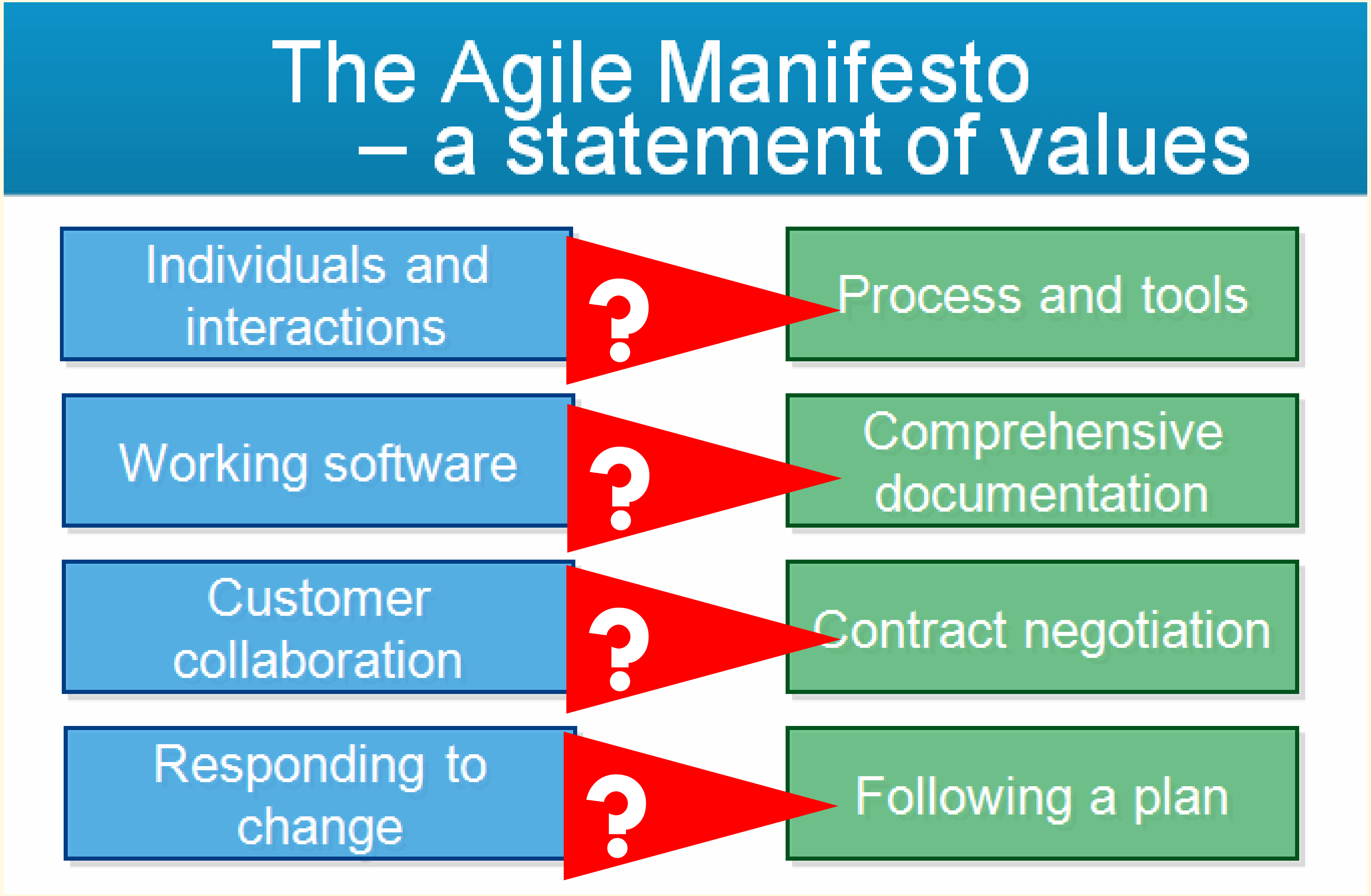
https://s-media-cache-ak0.pinimg.com

Agile Methods ⇔ Future-Proof Software-Systems ??

http://www.isummation.com

1 ~~Agile methods are the pathway to efficient and effective software production~~

2 ~~Agile methods are an excuse to avoid serious software engineering work~~



The Agile Manifesto – a statement of values

Individuals and interactions ? Process and tools

Working software ? Comprehensive documentation

Customer collaboration ? Contract negotiation

Responding to change ? Following a plan

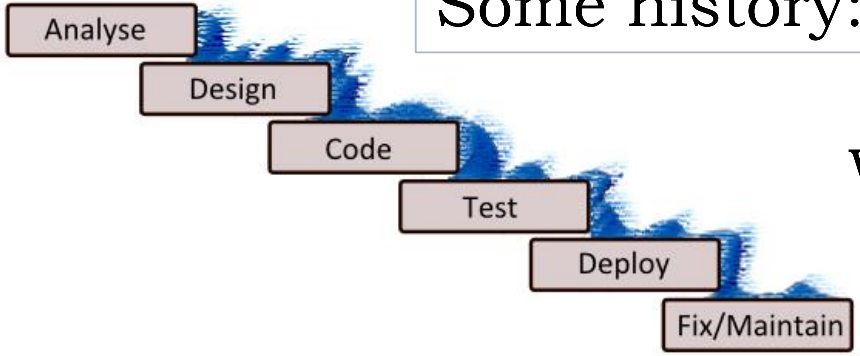
What followed ?

Agilists ⟷ Traditionalists

http://www.trucker.de

Future-Proof Software-Systems [Part 3C]

Some history: 1980 … 2000:

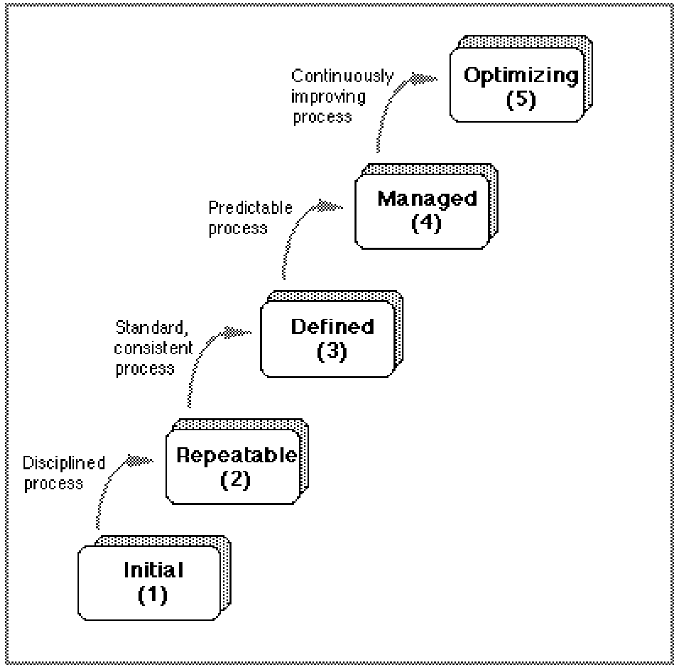


https://french.kwiziq.com

Waterfall development process

Capability Maturity Model Integration

http://www.signavio.com



http://people.cs.ksu.edu

… The software engineering process became seriously overloaded and slow

Consequence:

http://yinyangmother.com

... (very) heavy & slow development processes

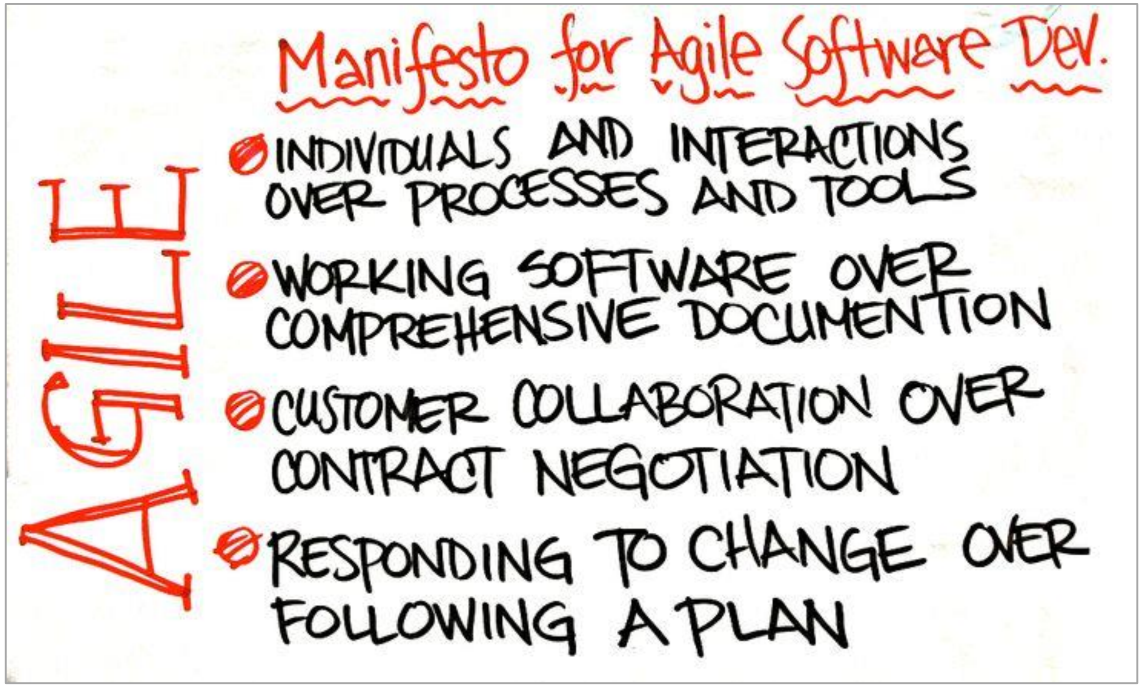
http://img-aws.ehowcdn.com

... Frustrated users and customers

http://portfolio.goldlilys-media.com

«We need something radically new»: Agile Methods

https://s-media-cache-ak0.pinimg.com



Manifesto for Agile Software Dev.

AGILE

- INDIVIDUALS AND INTERACTIONS OVER PROCESSES AND TOOLS
- WORKING SOFTWARE OVER COMPREHENSIVE DOCUMENTION
- CUSTOMER COLLABORATION OVER CONTRACT NEGOTIATION
- RESPONDING TO CHANGE OVER FOLLOWING A PLAN

… soon, new ***lightweight*** SW development processes came up:

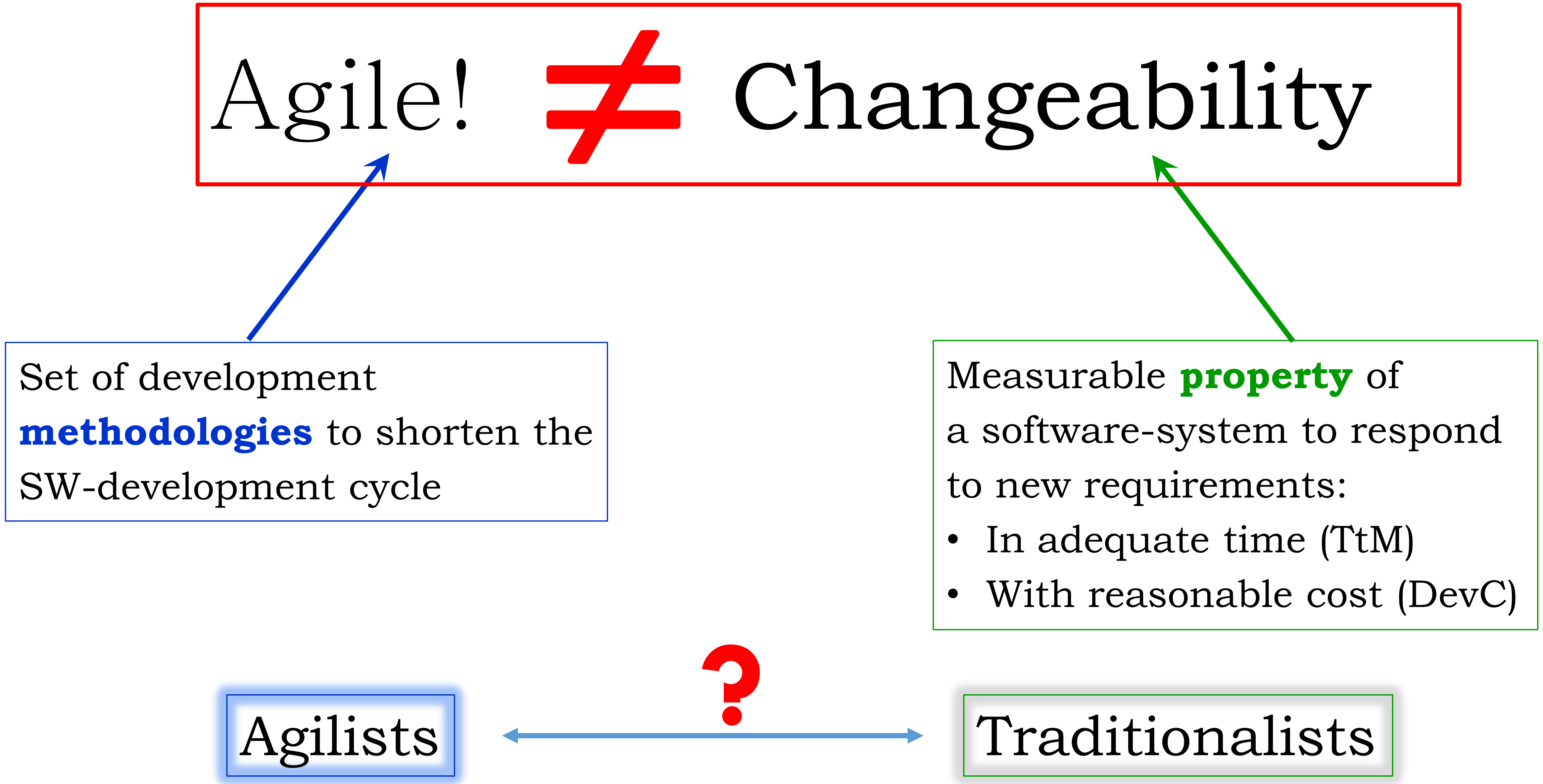
Scrum

http://www.hydromaxx40coupon.com

XP (Extreme Programming)

AUP, or Agile Unified Process

RAD (Rapid Application Development)



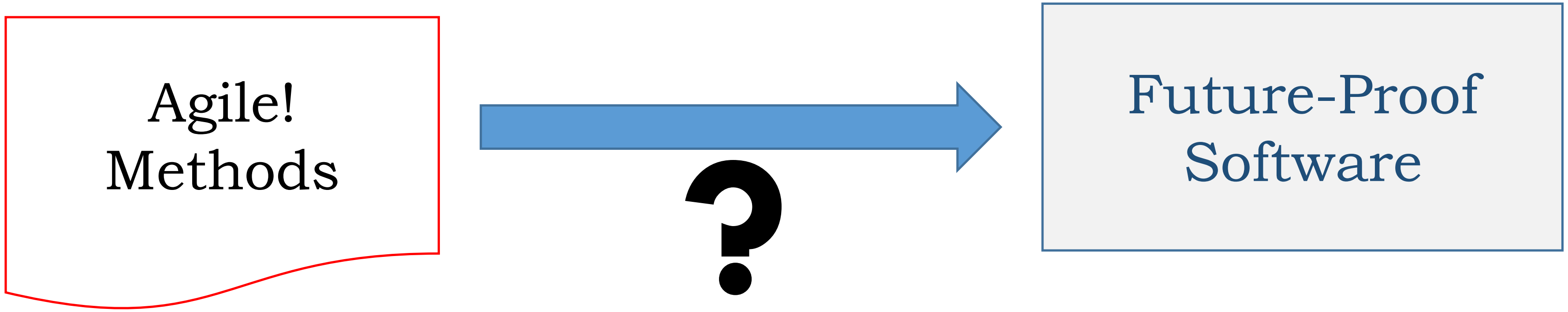
Agile! ≠ Changeability

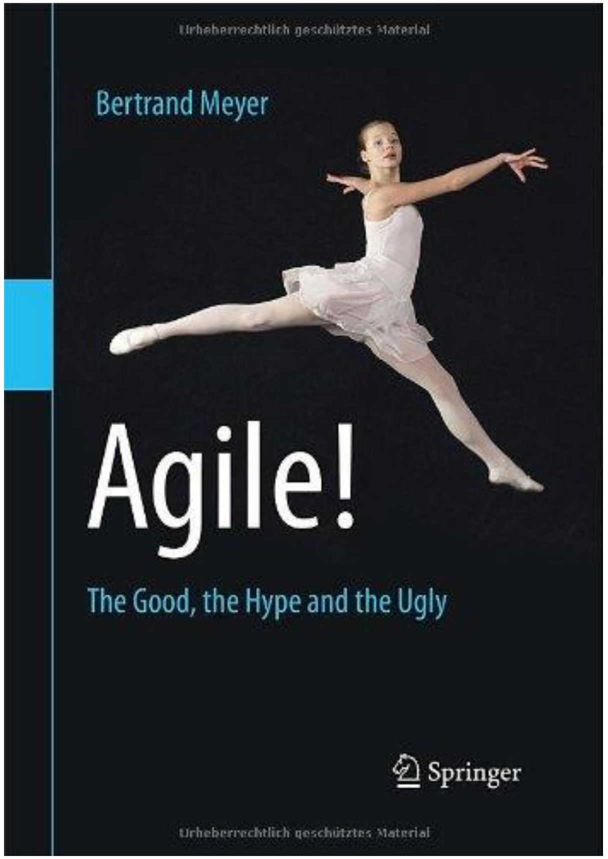
Set of development **methodologies** to shorten the SW-development cycle

Measurable **property** of a software-system to respond to new requirements:

- In adequate time (TtM)
- With reasonable cost (DevC)

Agilists ? Traditionalists




Urheberrechtlich geschütztes Material
Bertrand Meyer
Agile!
The Good, the Hype and the Ugly
Springer
Urheberrechtlich geschütztes Material

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

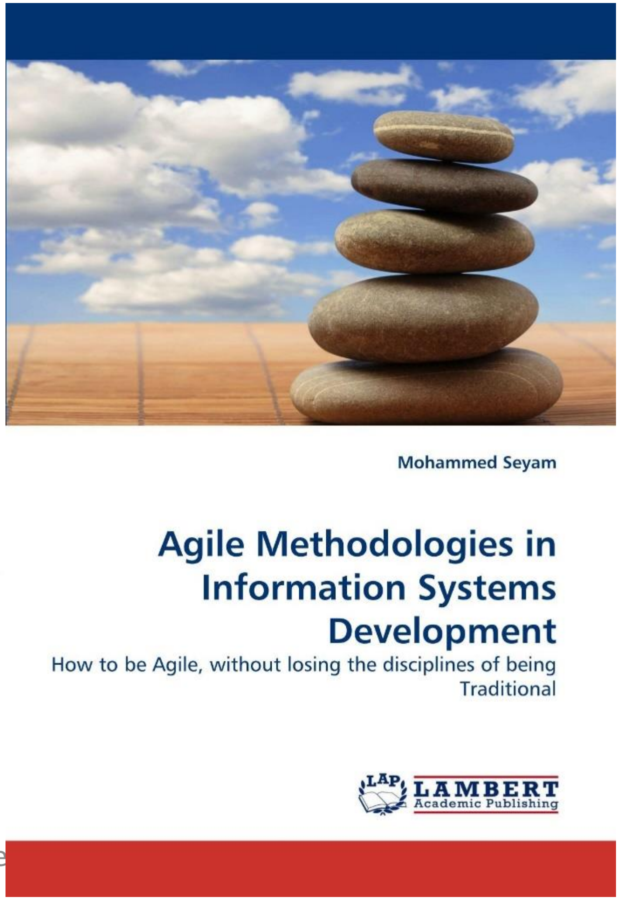
What is the situation today ?

Agilists
Traditionalists

http://i.huffpost.com

There is still serious ***mistrust***

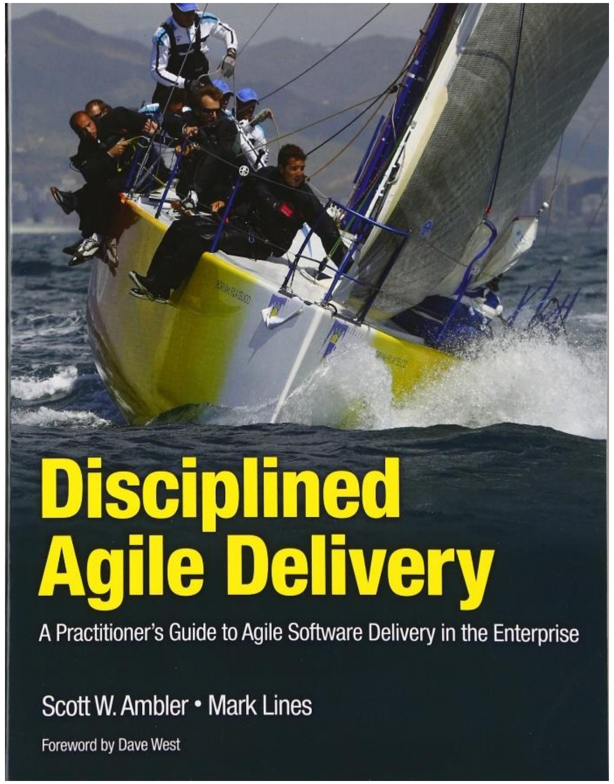
… but even enterprise architects are now learning from agilists


Mohammed Seyam

Agile Methodologies in Information Systems Development

How to be Agile, without losing the disciplines of being Traditional

LAP LAMBERT Academic Publishing


Disciplined Agile Delivery

A Practitioner's Guide to Agile Software Delivery in the Enterprise

Scott W. Ambler • Mark Lines

Foreword by Dave West

Agile Methods:

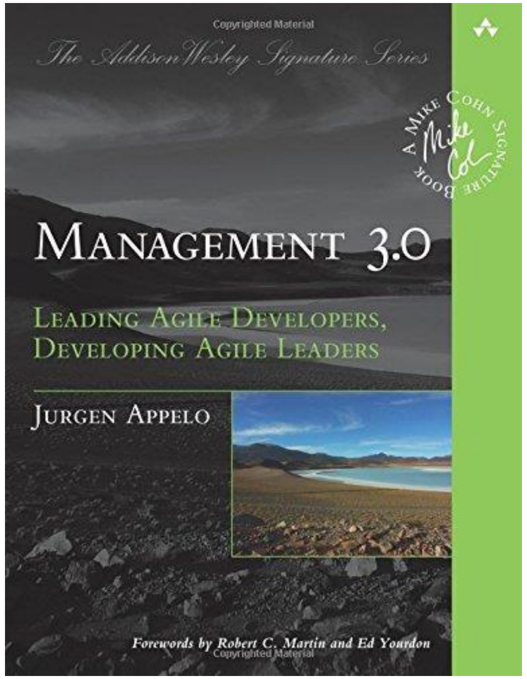
Effect on Future-Proof Software:

https://thumbs.dreamstime.com

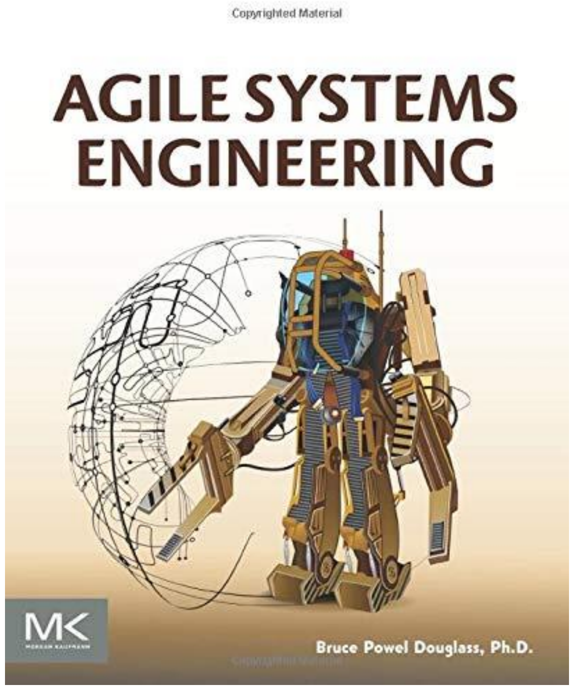
Conclusions:

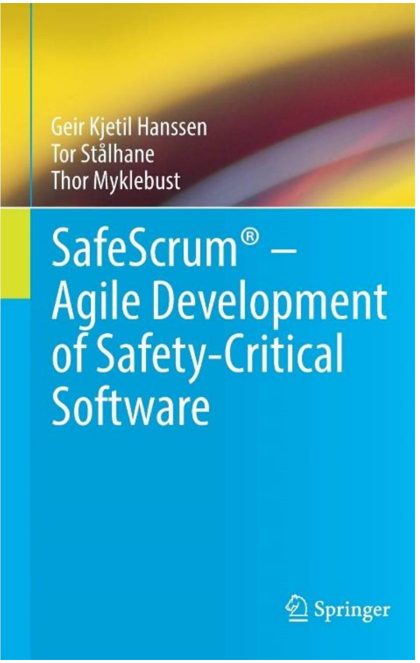
1. The agile canon ***misses*** the foundation of future-proof software (e.g. requirements gathering, formal modeling, **architecture** development & maintenance, system optimization)
2. Agile methods bring benefits to the work of small programming teams (< 25)
3. Some agile ideas are useful in improving processes also for very large information systems («Disciplined Agile Delivery»)

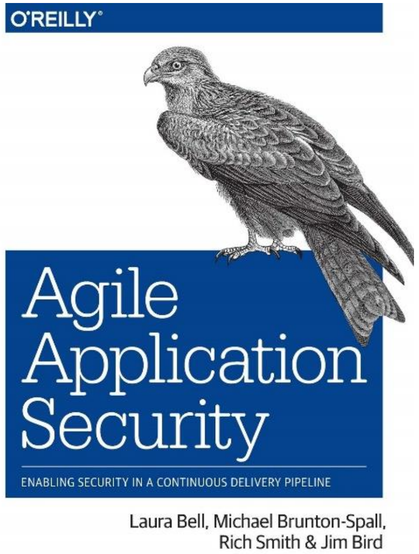
… however – there is some hope

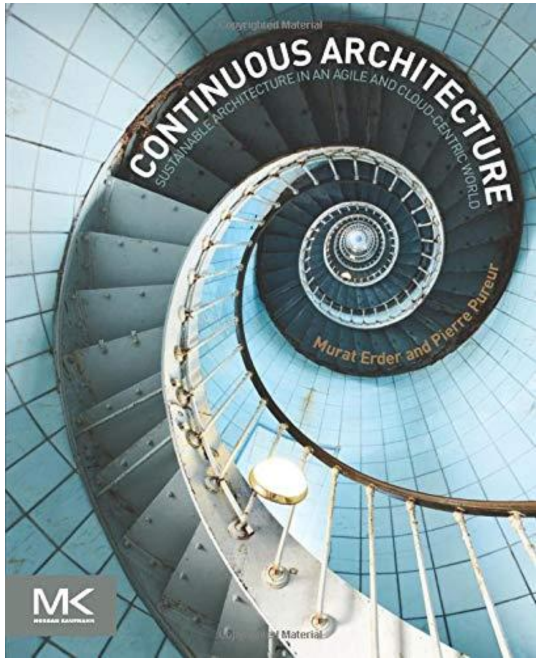

Copyrighted Material
The Addison Wesley Signature Series
A Mike Cohn Signature Book
Management 3.0
Leading Agile Developers, Developing Agile Leaders
Jurgen Appelo
Forewords by Robert C. Martin and Ed Yourdon

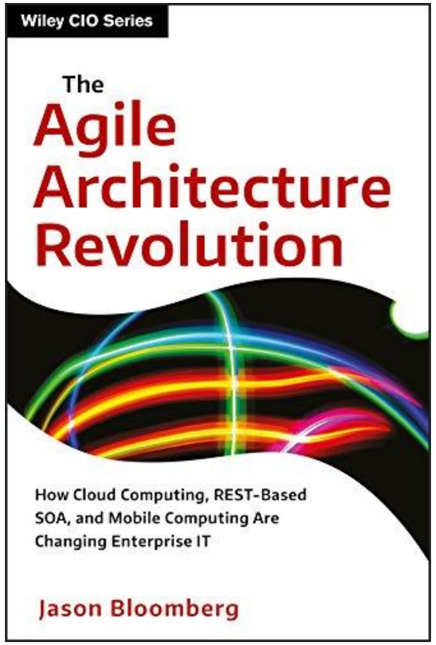
Agile!-methods for large systems and large projects

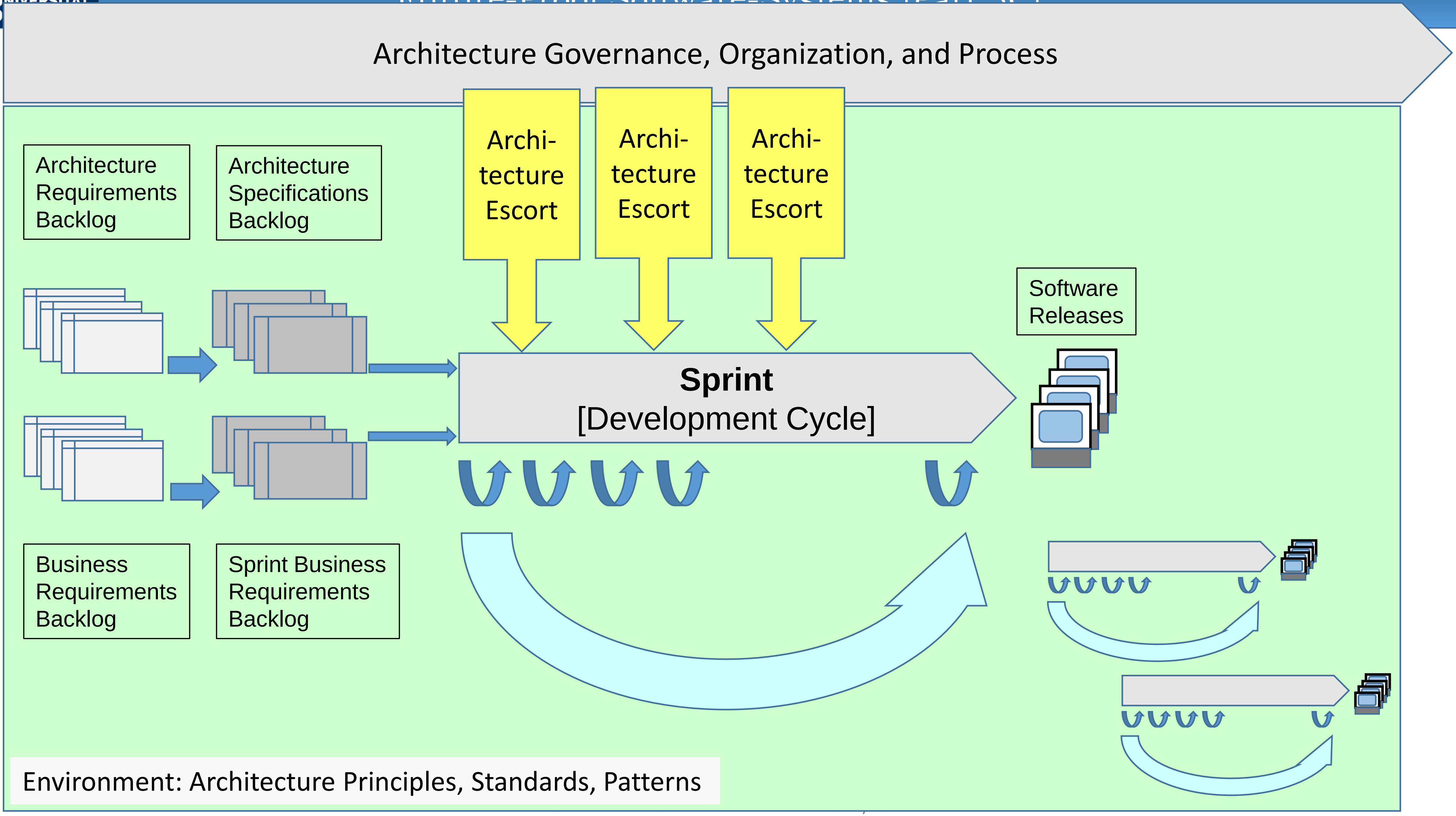





O'REILLY®
Agile Application Security
ENABLING SECURITY IN A CONTINUOUS DELIVERY PIPELINE
Laura Bell, Michael Brunton-Spall, Rich Smith & Jim Bird


Copyrighted Material
CONTINUOUS ARCHITECTURE
SUSTAINABLE ARCHITECTURE IN AN AGILE AND CLOUD-CENTRIC WORLD
Murat Erder and Pierre Pureur
MK


Wiley CIO Series
The Agile Architecture Revolution
How Cloud Computing, REST-Based SOA, and Mobile Computing Are Changing Enterprise IT
Jason Bloomberg



Architecture Governance, Organization, and Process

Architecture Requirements Backlog

Architecture Specifications Backlog

Archi-tecture Escort

Archi-tecture Escort

Archi-tecture Escort

Software Releases

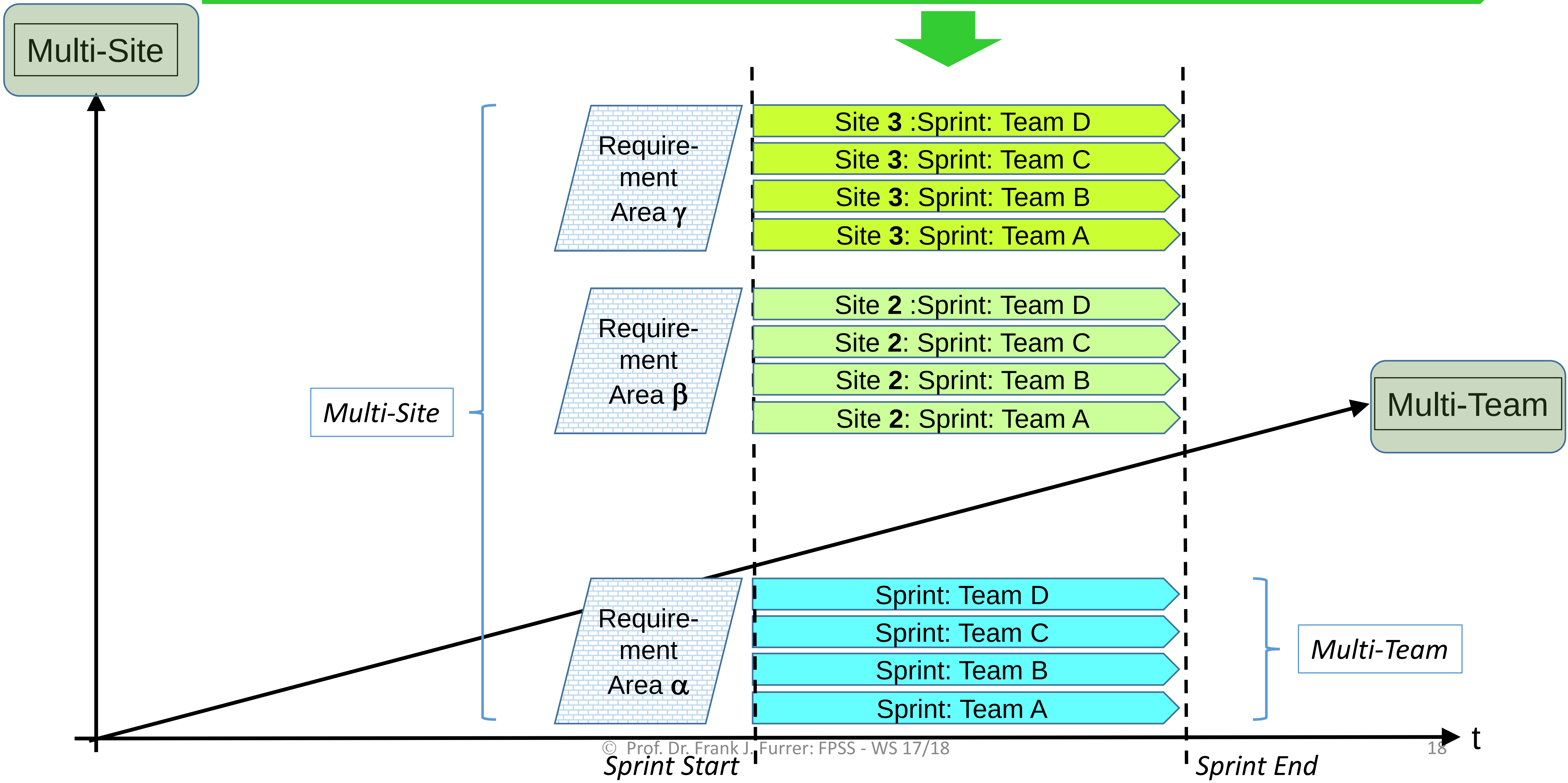
Sprint
[Development Cycle]

Business Requirements Backlog

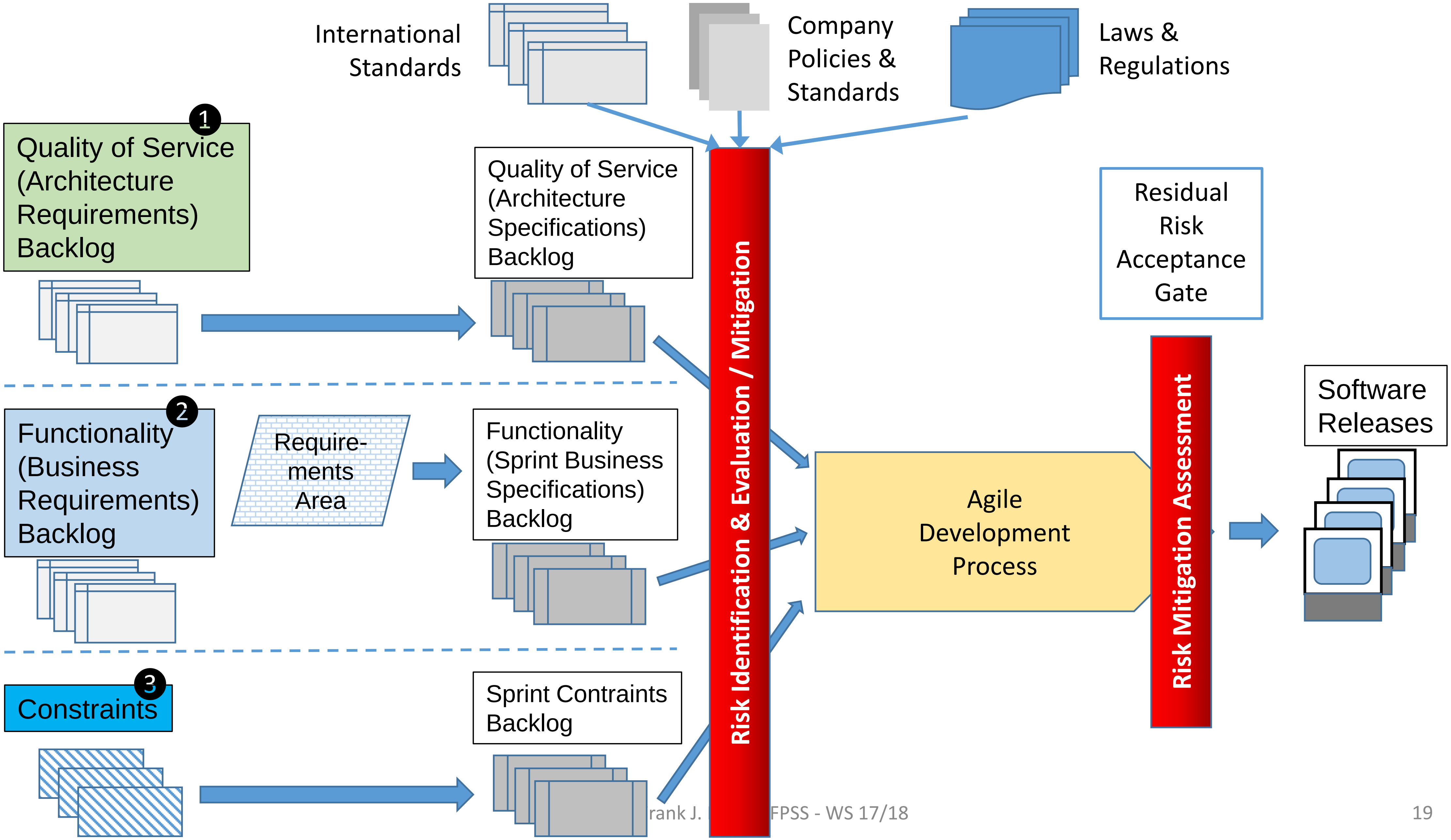
Sprint Business Requirements Backlog

Environment: Architecture Principles, Standards, Patterns

Intentional Architecture: Definition, Documentation, Process & Enforcement



Future-Proof Software-Systems [Part 3C]

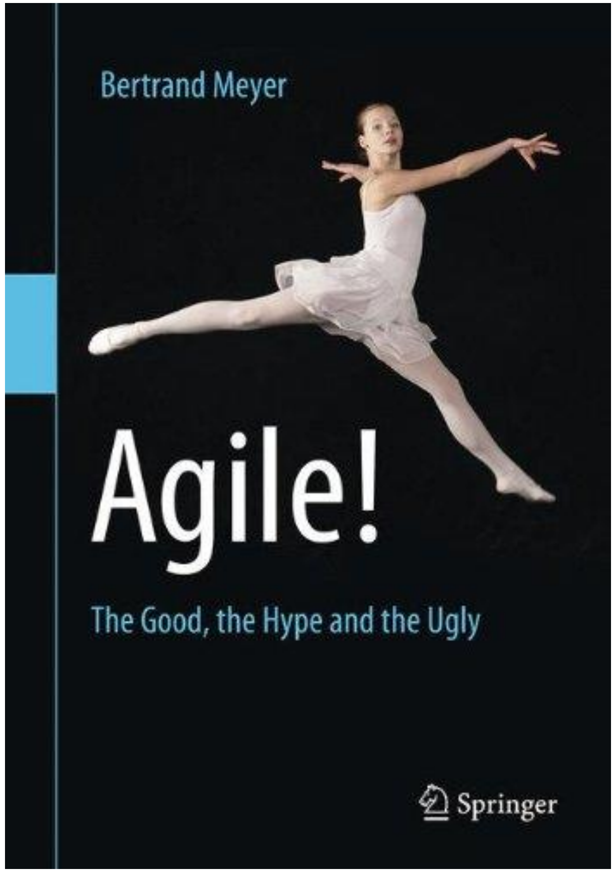

International Standards
Company Policies & Standards
Laws & Regulations
1
Quality of Service (Architecture Requirements) Backlog
Quality of Service (Architecture Specifications) Backlog
2
Functionality (Business Requirements) Backlog
Require-ments Area
Functionality (Sprint Business Specifications) Backlog
3
Constraints
Sprint Contraints Backlog
Risk Identification & Evaluation / Mitigation
Agile Development Process
Residual Risk Acceptance Gate
Risk Mitigation Assessment
Software Releases

Recommendations

Architecture Recommendations for Agile Methods

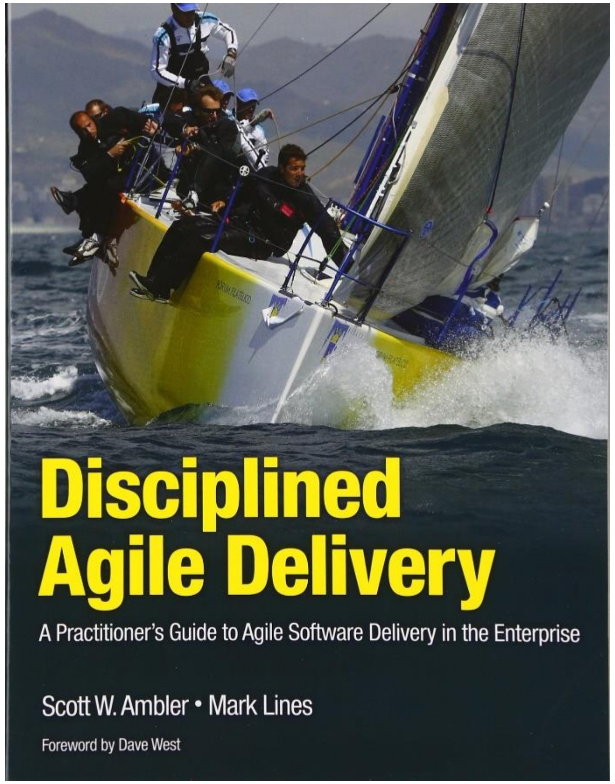
1. **Never** compromise the foundation of future-proof software-systems, i.e. architecture, models etc. for the sake of «agile»
2. Assure in the development process that – despite agile methods – no technical debt is accumulated
3. Compensate architecture erosion/divergence as soon as possible
4. Implement Risk-Managing Gates into the Process

Textbook



Bertrand Meyer:
Agile! – The Good, the Hype and the Ugly
Springer-Verlag, Germany, 2014. ISBN 978-3-3190-51543

Textbook


Disciplined Agile Delivery
A Practitioner's Guide to Agile Software Delivery in the Enterprise
Scott W. Ambler • Mark Lines
Foreword by Dave West

Mark W. Lines, Scott Ambler:
Disciplined Agile Delivery – *A Practitioner's Guide to Agile Software Delivery in the Enterprise*
Prentice Hall Inc. (IBM Press), USA, 2012. ISBN 978-0-132-81013-5

Domain Software Engineering

Future-Proof Software-Systems [Part 3C]

https://cimx.wordpress.com

«Software design is a constant battle with **complexity**»

Eric Evans, 2015

If we don't manage it well
⇒ Divergence

https://i0.wp.com

Future-Proof Software-Systems [Part 3C]

http://www.sutherlandweston.com

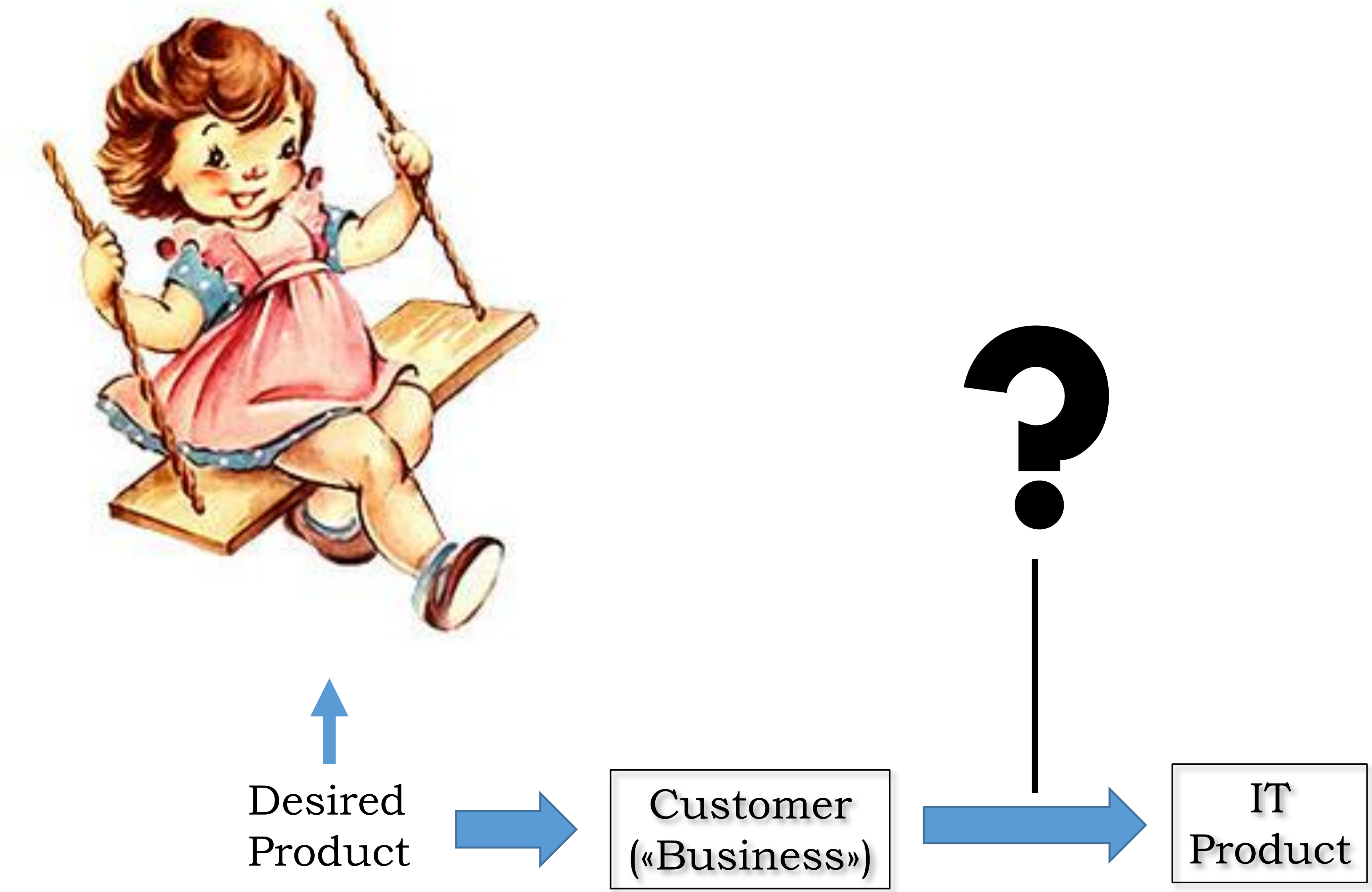
DEFINITIONS

Divergence =

Mismatch between *Business Needs* and *IT-Implementation*

Example: Swing

http://de.clipartlogo.com



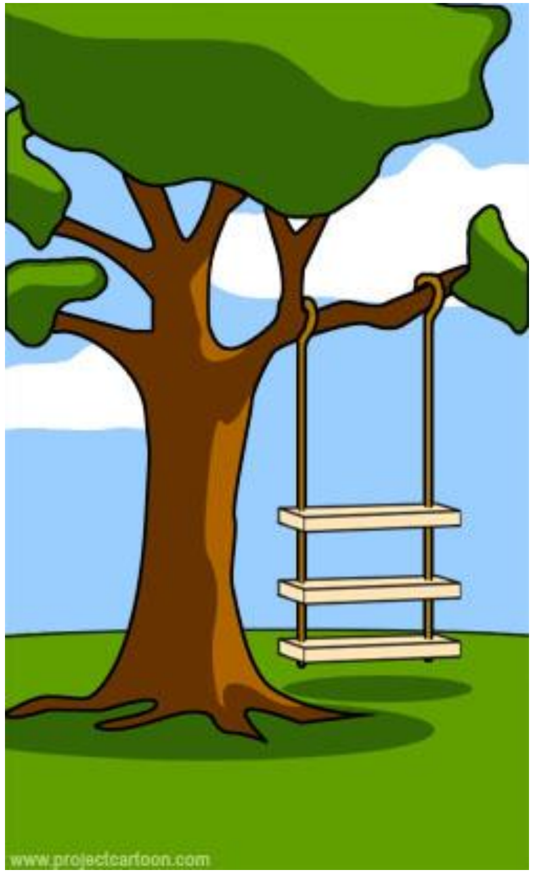
Desired Product

Customer («Business»)

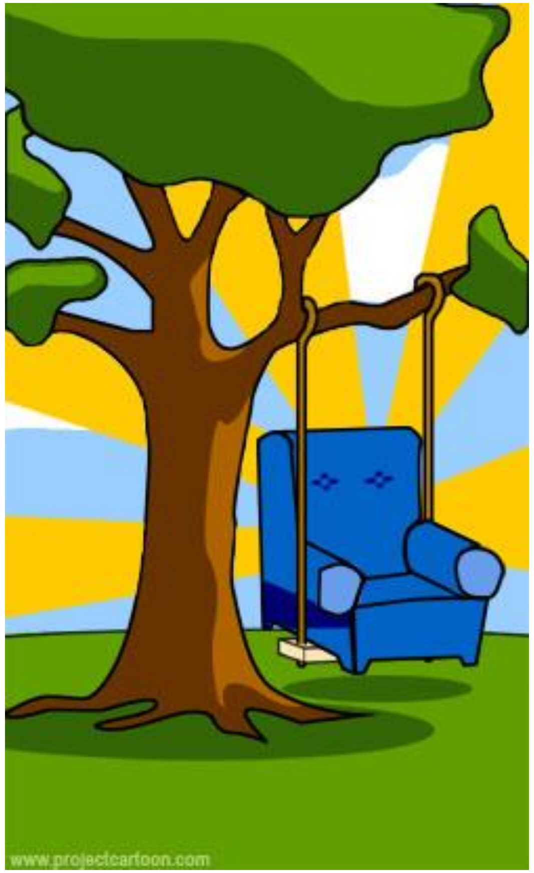
IT Product

Requirements

http://projectcartoon.com/create/

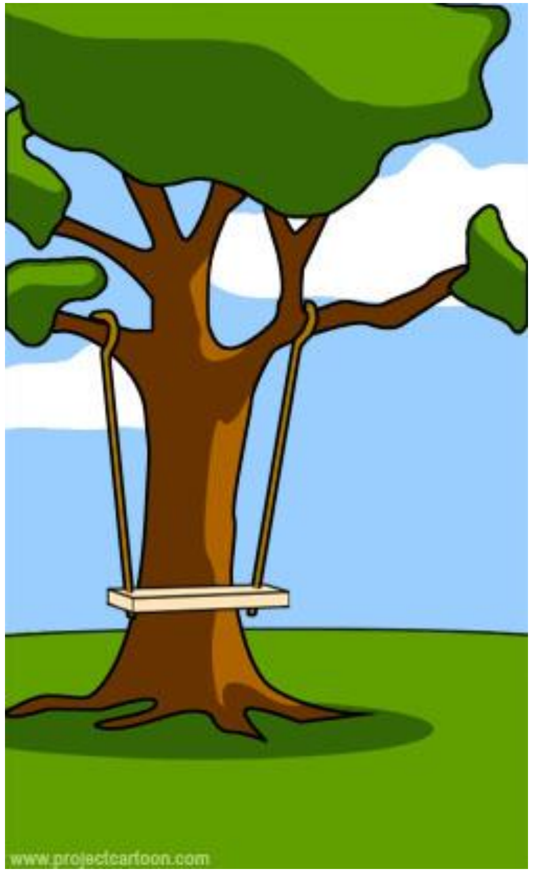


How the customer explained it



How the business consultant described it

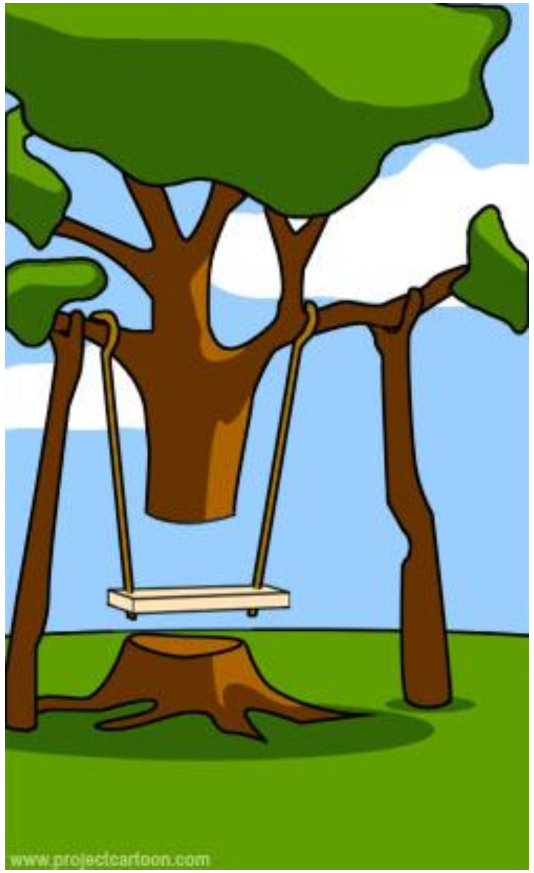
Specifications



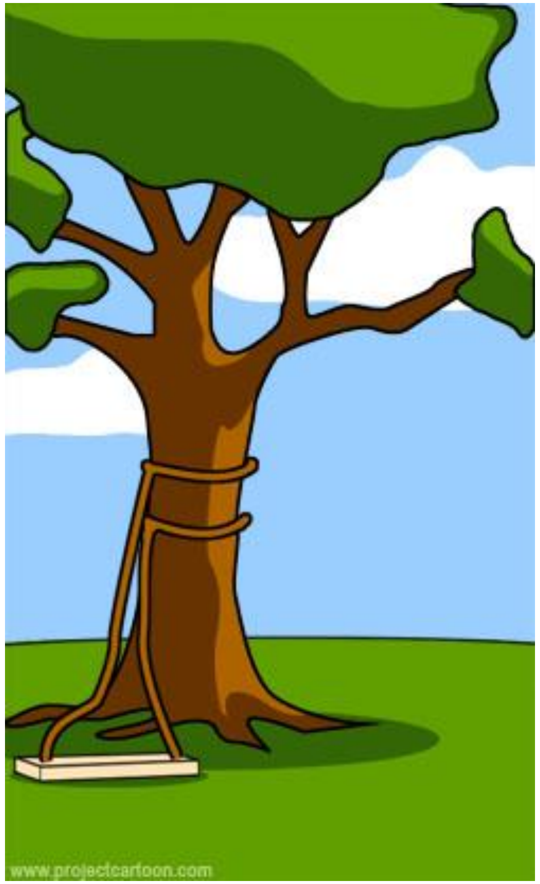
How the project leader understood it

Implementation

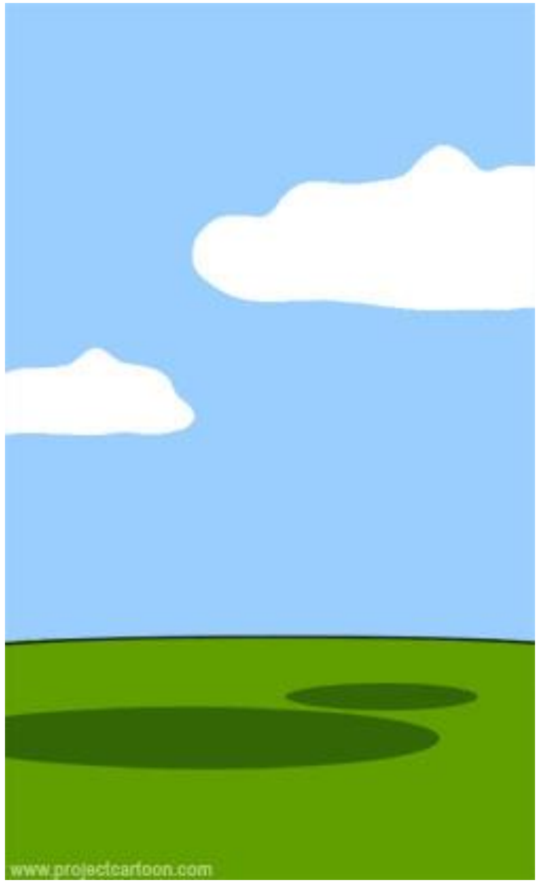
http://projectcartoon.com/create/



How the analyst designed it

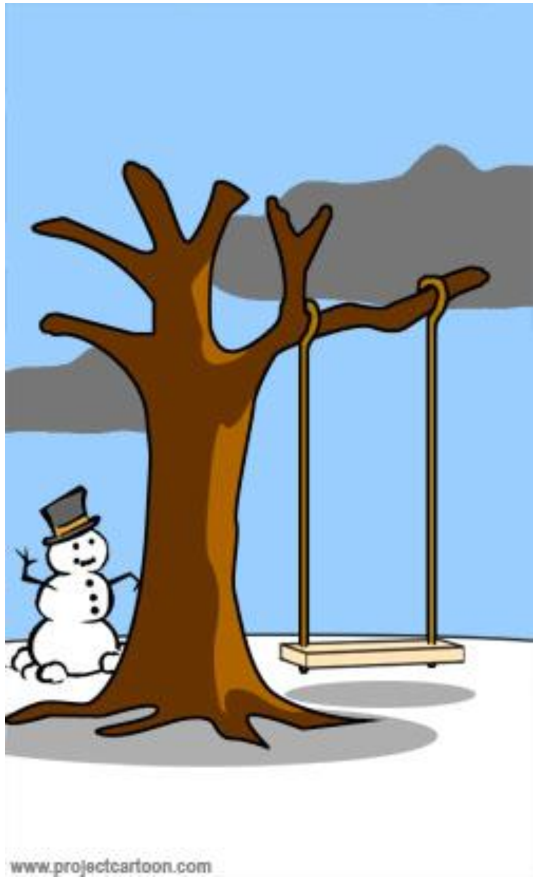


How the programmers implemented it



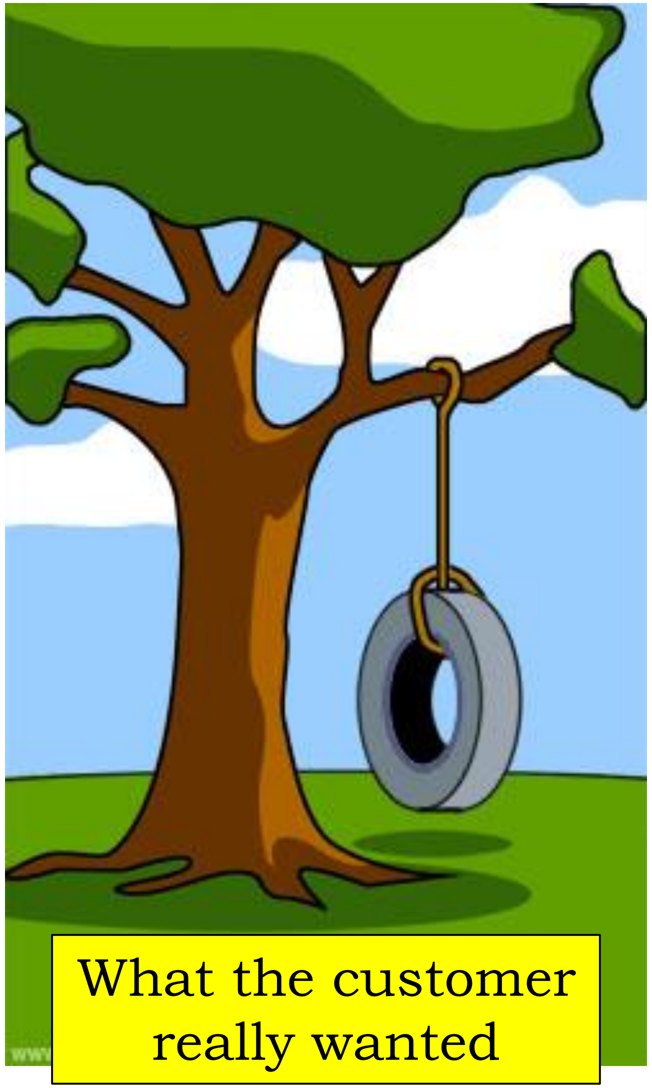
How the project was documented

Deployment → Operation



When the project was delivered

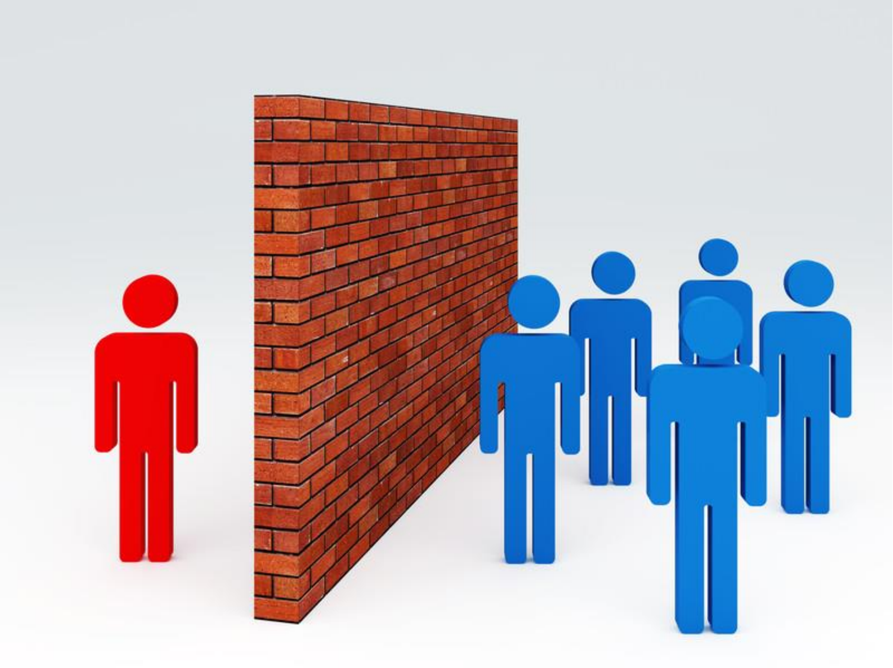
http://projectcartoon.com/create/



What the customer really wanted

What is the reason?

Failed Communications!

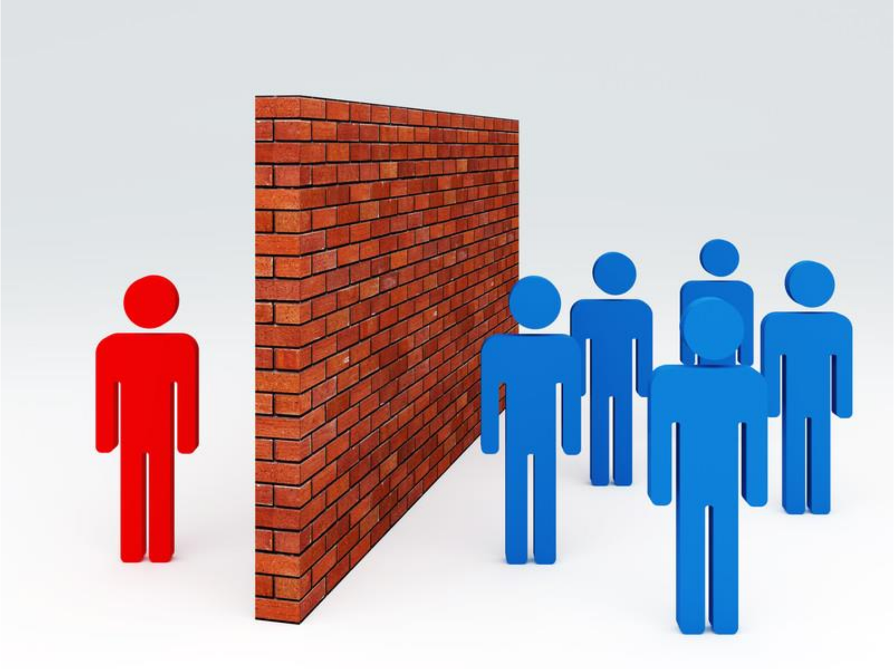


http://mayrsom.com

- Different *vocabulary* between business and IT
- Lots of *implicit* knowledge and assumptions
- No common *model*

Future-Proof Software-Systems [Part 3C]

Failed Communications!

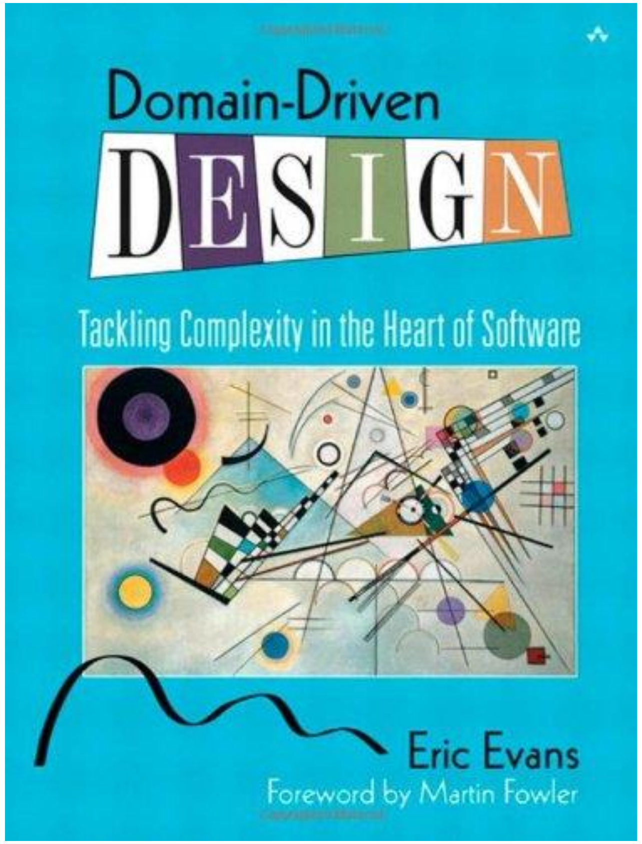


How can we significantly ***improve*** the communications between business and IT ?

⇒ Domain Software Engineering

Eric Evans, 2003

http://mayrsom.com


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

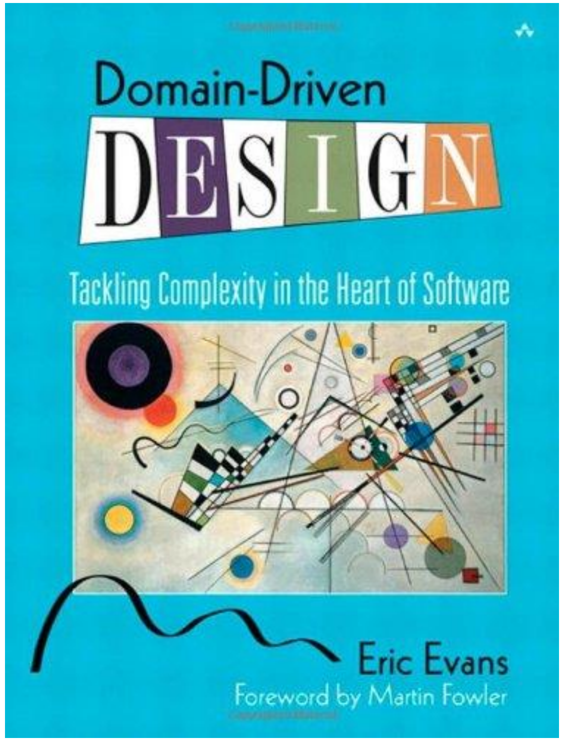
Future-Proof Software-Systems [Part 3C]

Domain-Driven Design [DDD]
Domain Engineering [DE]
Domain-Specific Languages [DSL]
Domain Language Engineering [DLE]
Domain-Specific Modeling [DSM]

→

Domain Software Engineering [DSE]

The start:


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

Seminal Work 2003

Excellent Summary:

Abel Avram, Floyd Marinescu:
Domain-Driven Design - Quickly
C4Media Inc., USA, 2006.
ISBN 978-1-4116-0925-9
Download:
http://www.infoq.com/minibooks/domain-driven-design-quickly
[last accessed: 2.12.2015]

http://blogs.msdn.com

DEFINITIONS

Domain Software Engineering [DSE] =

an architectural ***methodology***

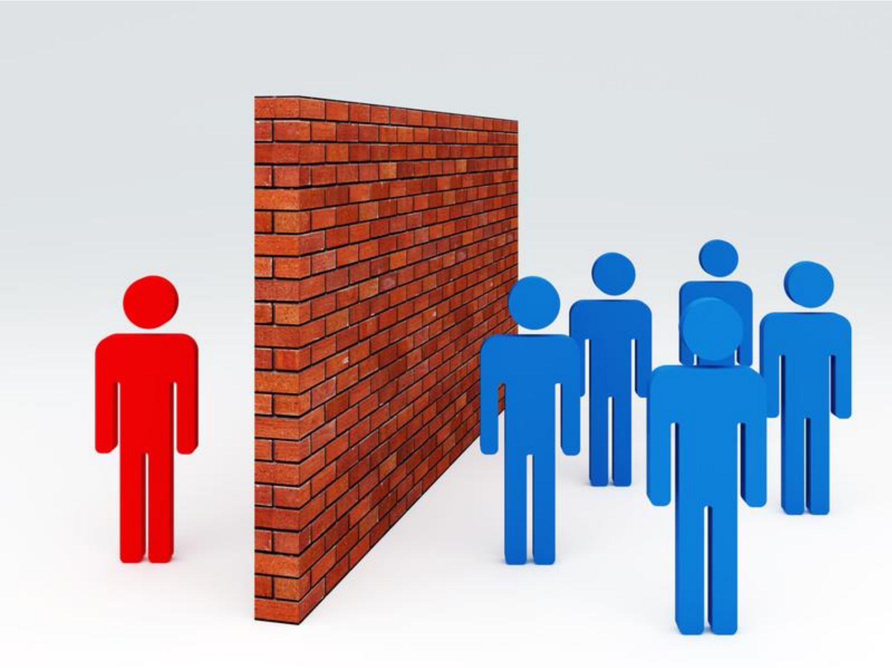
for evolving a *software* system

that closely aligns to *business* domains

Important note:

All architecture principles remain strictly valid in DSE

Future-Proof Software-Systems [Part 3C]



http://mayrsom.com

Why does communications between business and IT increase the complexity ?

Because it needs a ***translation*** between the «business world» and the «IT world»

Error-prone
Time-consuming
Annoying

Essential complexity

… is the *inherent* complexity of the system to be built.

Essential complexity for a given problem *cannot* be reduced.

It can only be lessened by *simplifying* the requirements for the system extension.

http://www.sherweb.com

Manage essential complexity

Accidental Complexity

… is *introduced* by our development activities or by constraints from our environment.

This is unnecessary and can be *reduced* or eliminated.

⇒ Development methodology!

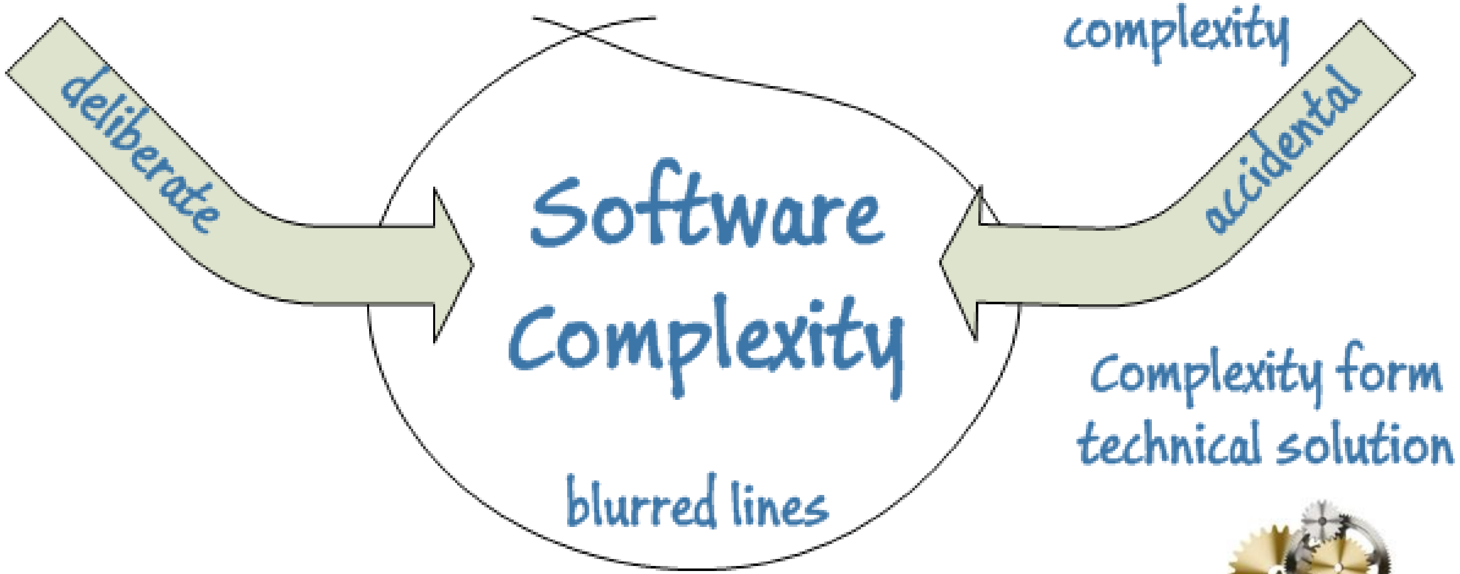
Combat accidental complexity

http://www.experto.de

Domain Logic Complexity

http://year7historygr.edublogs.org

Legacy code base complexity



Software Complexity

blurred lines

Complexity form technical solution

https://cimx.wordpress.com

© Toni Esteves

Frustration !

Divergence =

Mismatch between:

Business Needs ⇔ IT-Implementation

Essential Complexity

DSE

Accidental Complexity

Misunderstandings

Lack of Precision

Semantic Differences

http://clipartzebraz.com

https://cimx.wordpress.com

Customer/Business Needs

IT Implementation

http://clipartzebraz.com

DSE

https://cimx.wordpress.com

Customer/Business Needs

IT Implementation

Which are the key elements of DSE (Domain Software Engineering?)

1. Understanding the Business/Application Domain in terms of the business
   (⇒ Domain Model)
2. Use of an ubiquitous language
   (Business ⇔ IT alignment)
3. Software: Implementation of Business Domain concepts
   (Concepts ⇒ Business objects ⇒ Programm objects)

Universale Ausdrucksform

Future-Proof Software-Systems [Part 3C]

http://blogs.msdn.com

Domain Software Engineering (DSE)

DSE Concepts

- Business/Application Domain
- Bounded Context
- Domain Model
- Anticorruption Layer

DEFINITIONS

Business/Application Domain =

A Domain is a Sphere of Knowledge, Influence or Activity.

A Domain lives within a Bounded Context.

A Domain represents a well-defined Part of the Real World.

A Domain encapsulates a Domain Model.

www.thinkddd.com

http://blogs.msdn.com

DEFINITIONS

Business/Application Domain =

A Domain is a Sphere of Knowledge, Influence or Activity.

A Domain lives within a Bounded Context.

A Domain represents a well-defined Part of the Real World.

A Domain encapsulates a Domain Model.

www.thinkddd.com

Future-Proof Software-Systems [Part 3C]

https://thoughtsfromthisflower.wordpress.com

DEFINITIONS

Bounded Context =

The Bounded Context is the Boundary of a Model.

When you have multiple Models you should define Bounded Contexts.

To map between Bounded Contexts you use a Context Map.

www.thinkddd.com

http://cliparts.co/meeting-pictures

DEFINITIONS

Domain Model =

A Domain Model is a representation of the Entities, Relationships and their Properties in your Domain

The Domain Model should be recognizable and understandable by the business and IT

The domain model has sufficient essential details

www.thinkddd.com

DEFINITIONS

Anticorruption Layer =

An Anti-Corruption Layer is a method to isolate two domains or systems, allowing systems to be integrated without knowledge of each other

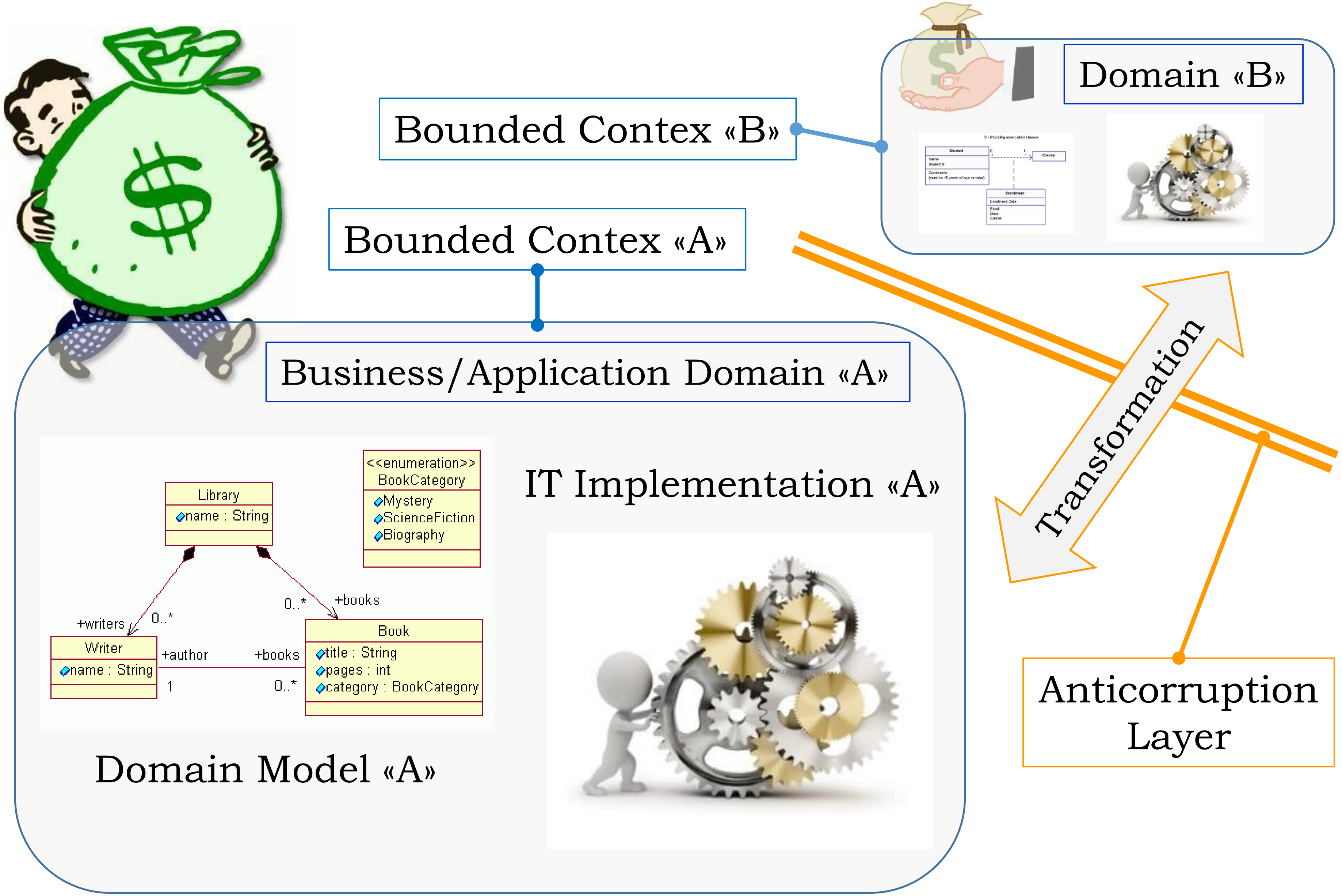
An Anti-Corruption Layer presents a Facade to both systems, defined in terms of their specific models

Anti-Corruption Layers maintain the integrity of differing systems and models

www.thinkddd.com

DSE Definitions: **Summary**

http://clipartzebraz.com



Bounded Contex «B»

Domain «B»

Bounded Contex «A»

Business/Application Domain «A»

Domain Model «A»

IT Implementation «A»

Transformation

Anticorruption Layer

Example: Business/Application Domain

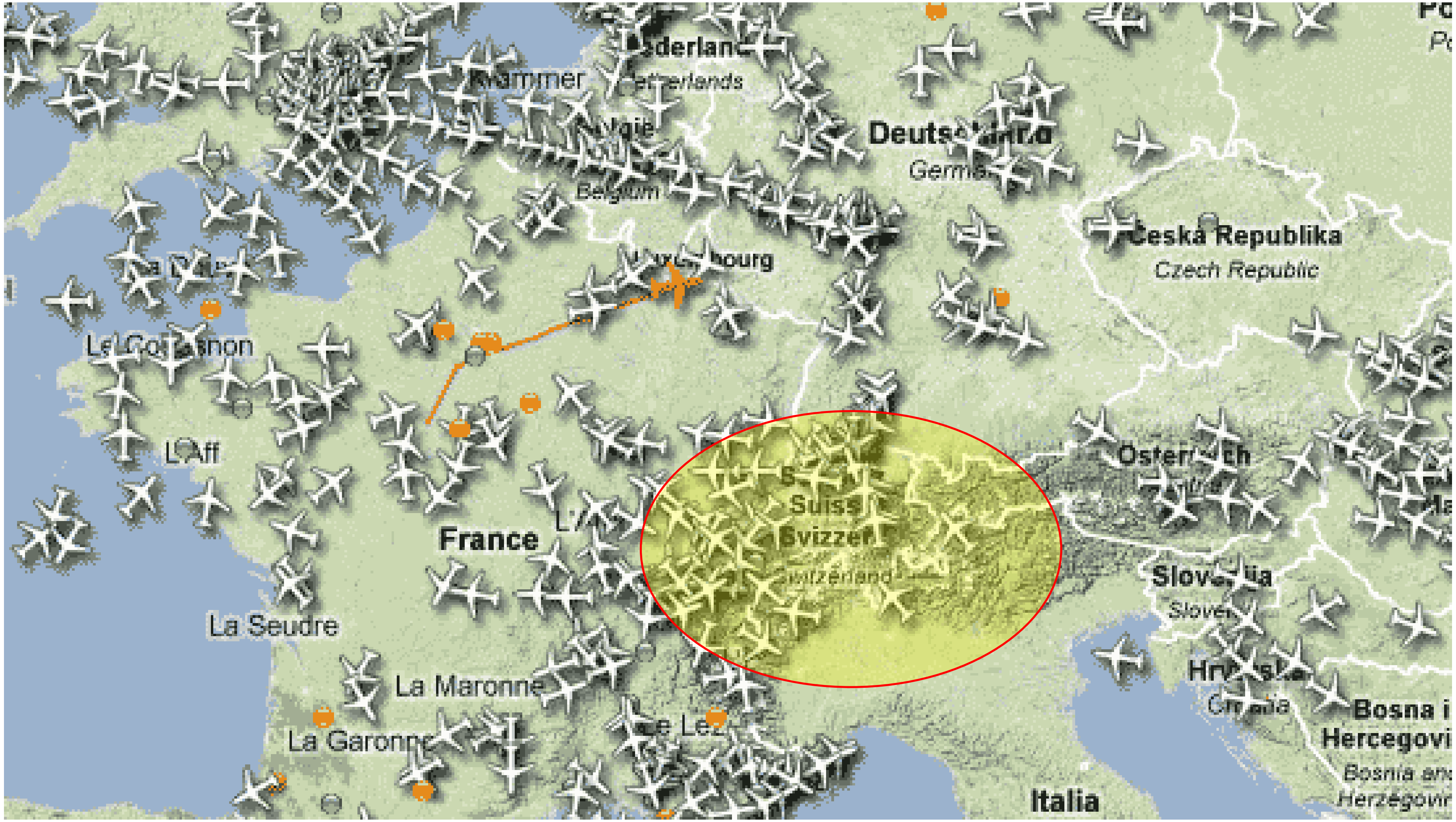
http://www.skyguide.ch

Domain = Flight Monitoring

Context:

Thousands of planes are in the air all over the planet. The flight monitoring systems track every flight and avoid mid-air collisions

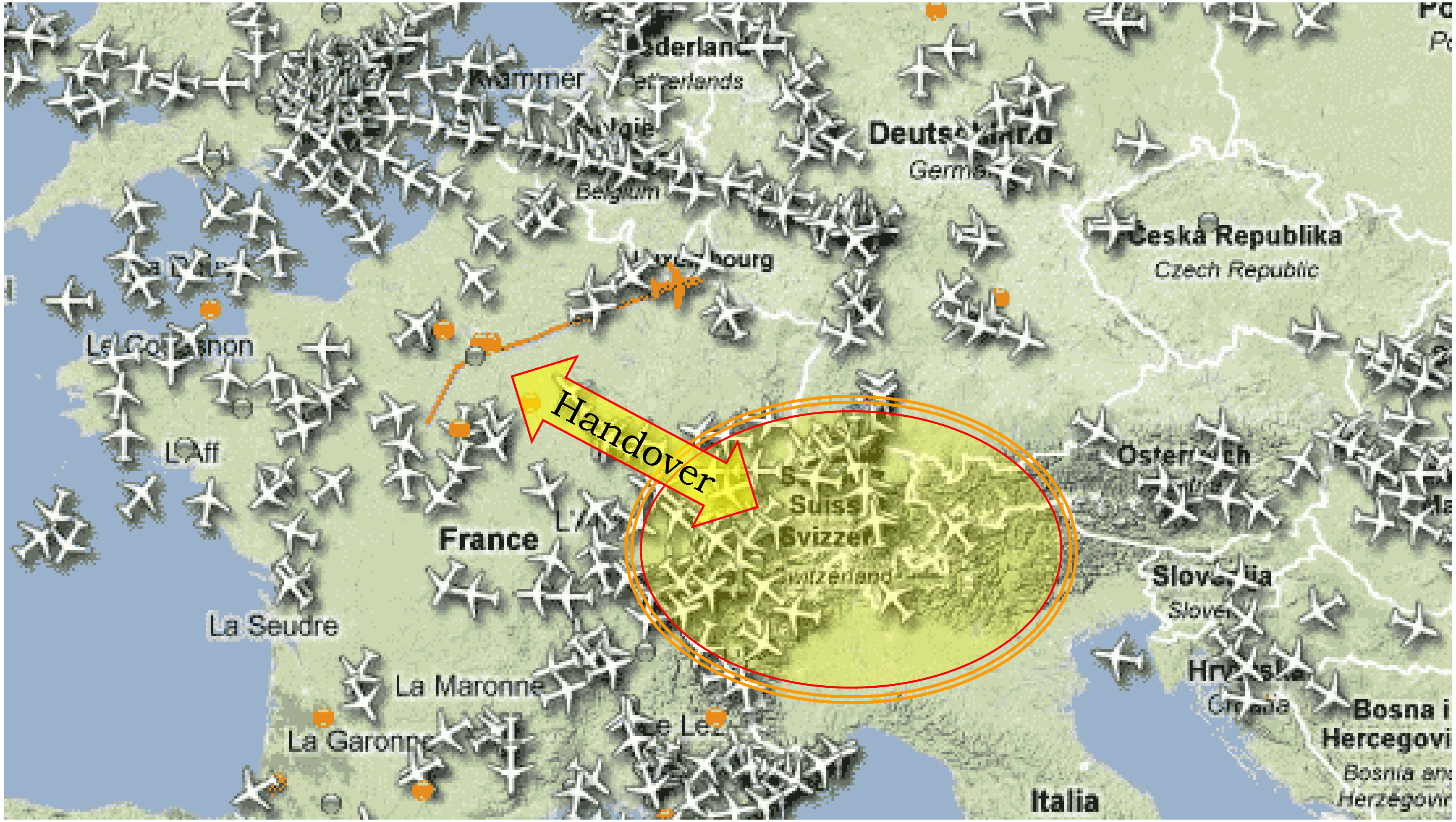
Example: Bounded Context ⇔ SKYGUIDE Switzerland



https://www.flightradar24.com

Boundary = Contractual Responsibility within the European System

Example: Bounded Context

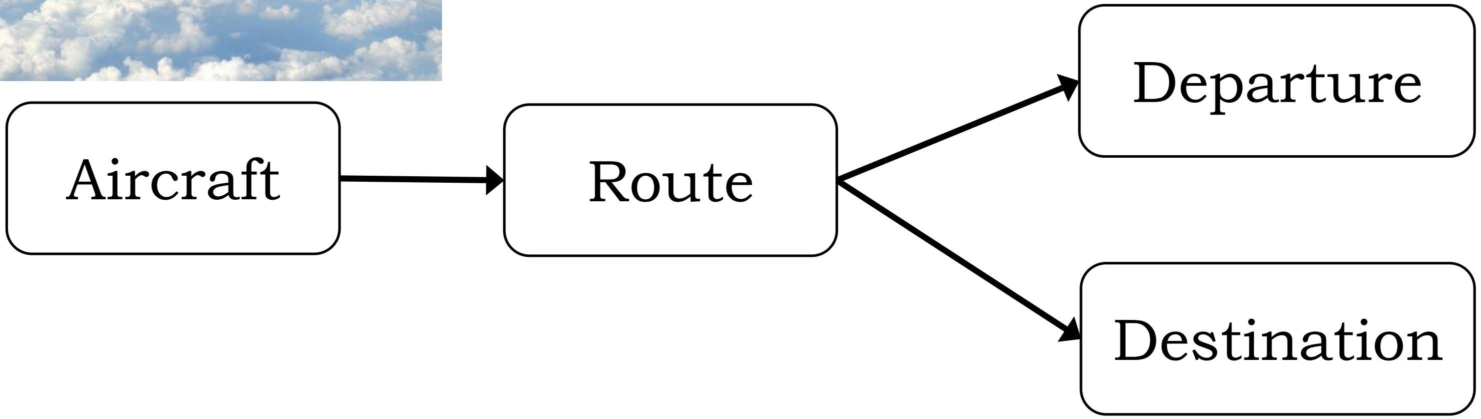


https://www.flightradar24.com

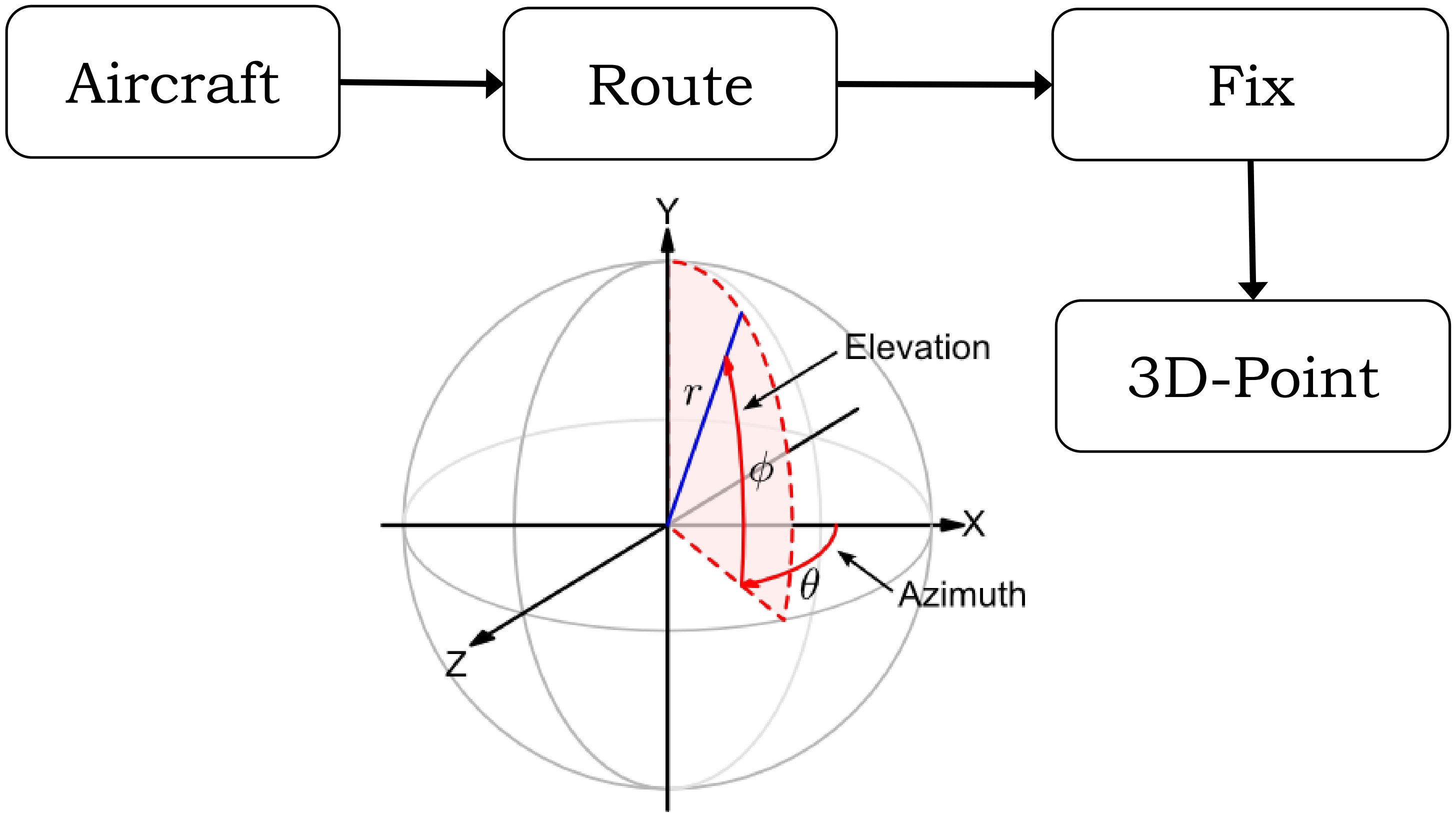
Anticorruption Layer = X-Compatibility Layer

Example:
Flight Monitoring **Domain Model**

… Development of the Domain Model
⇒ Search & Definition of **Key Concepts**



Example:
Flight Monitoring **Domain Model**



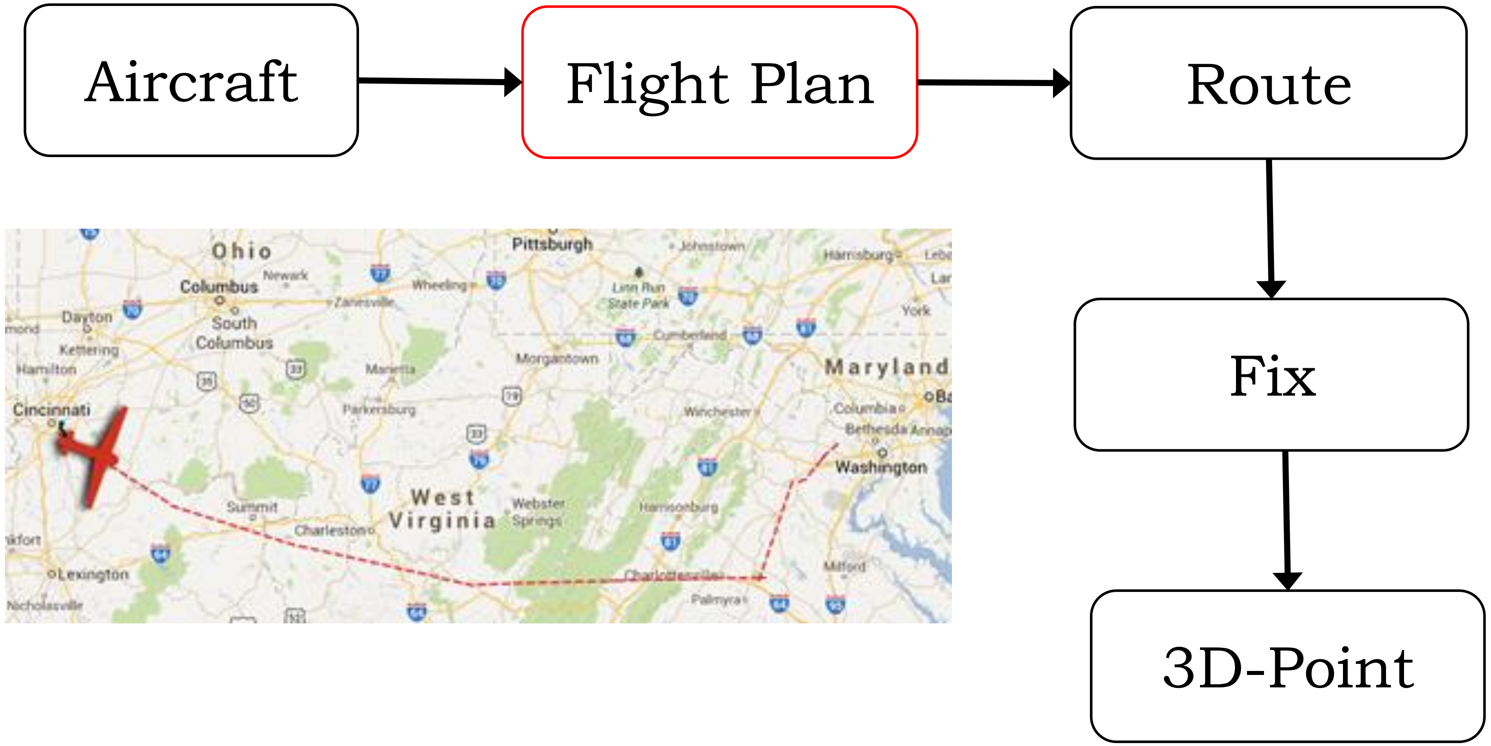
Aircraft

Route

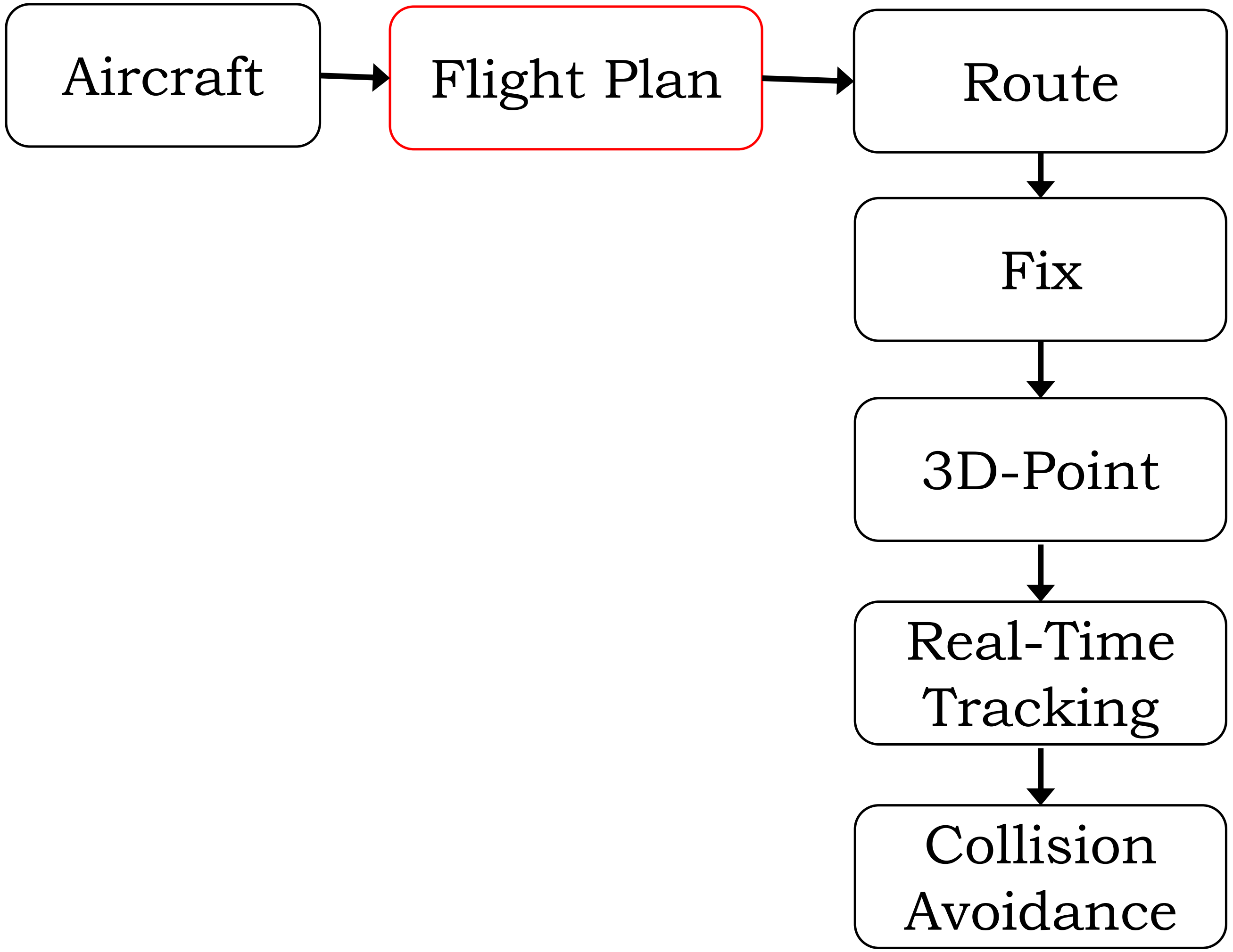
Fix

3D-Point

Example:
Flight Monitoring **Domain Model**



Example:
Flight Monitoring **Domain Model**

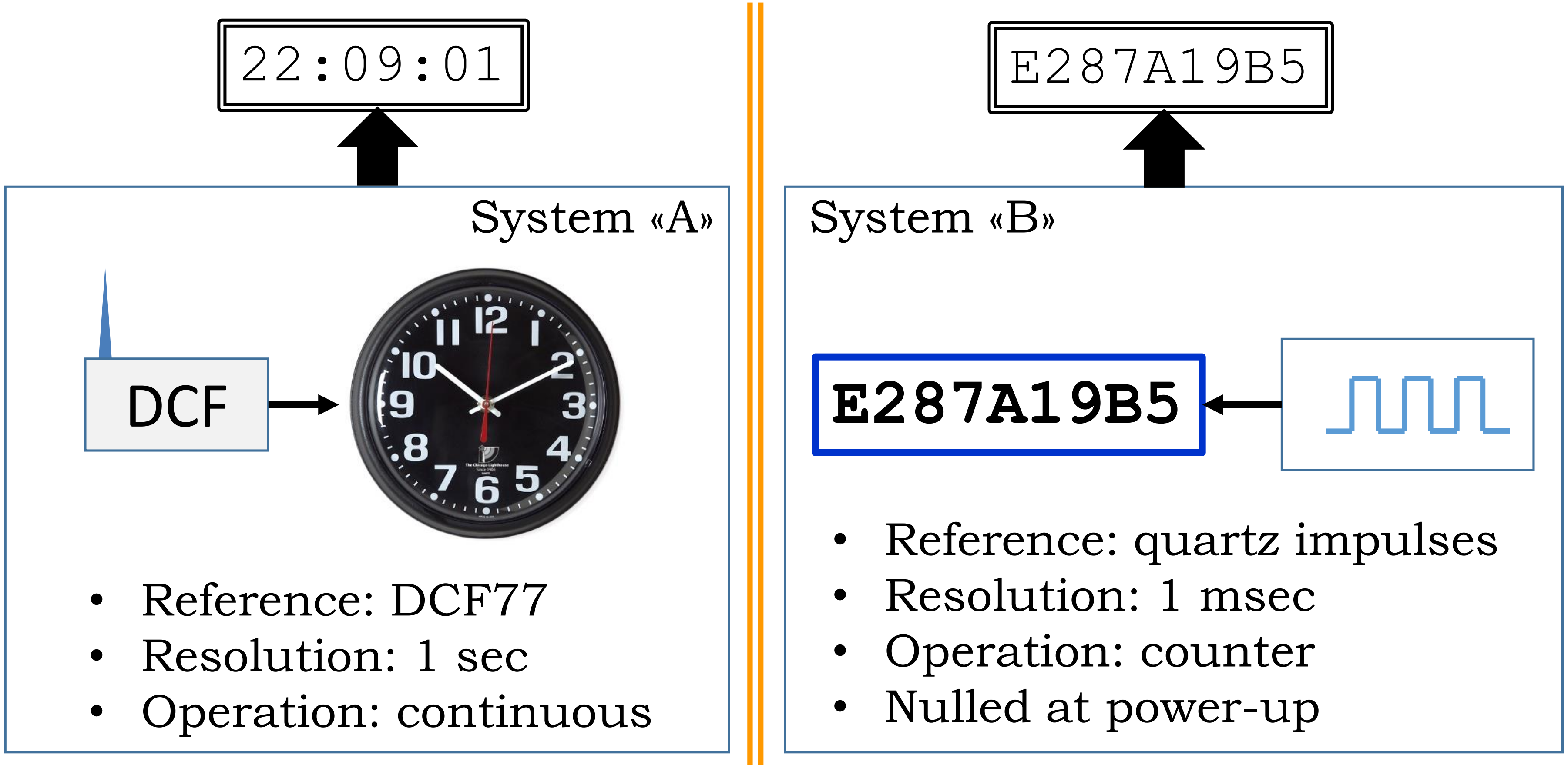


Future-Proof Software-Systems [Part 3C]

Example:
Concept «Time»

Anticorruption Layer
- Domain concepts
- Models
- Implementation

Get{time}



System «A»

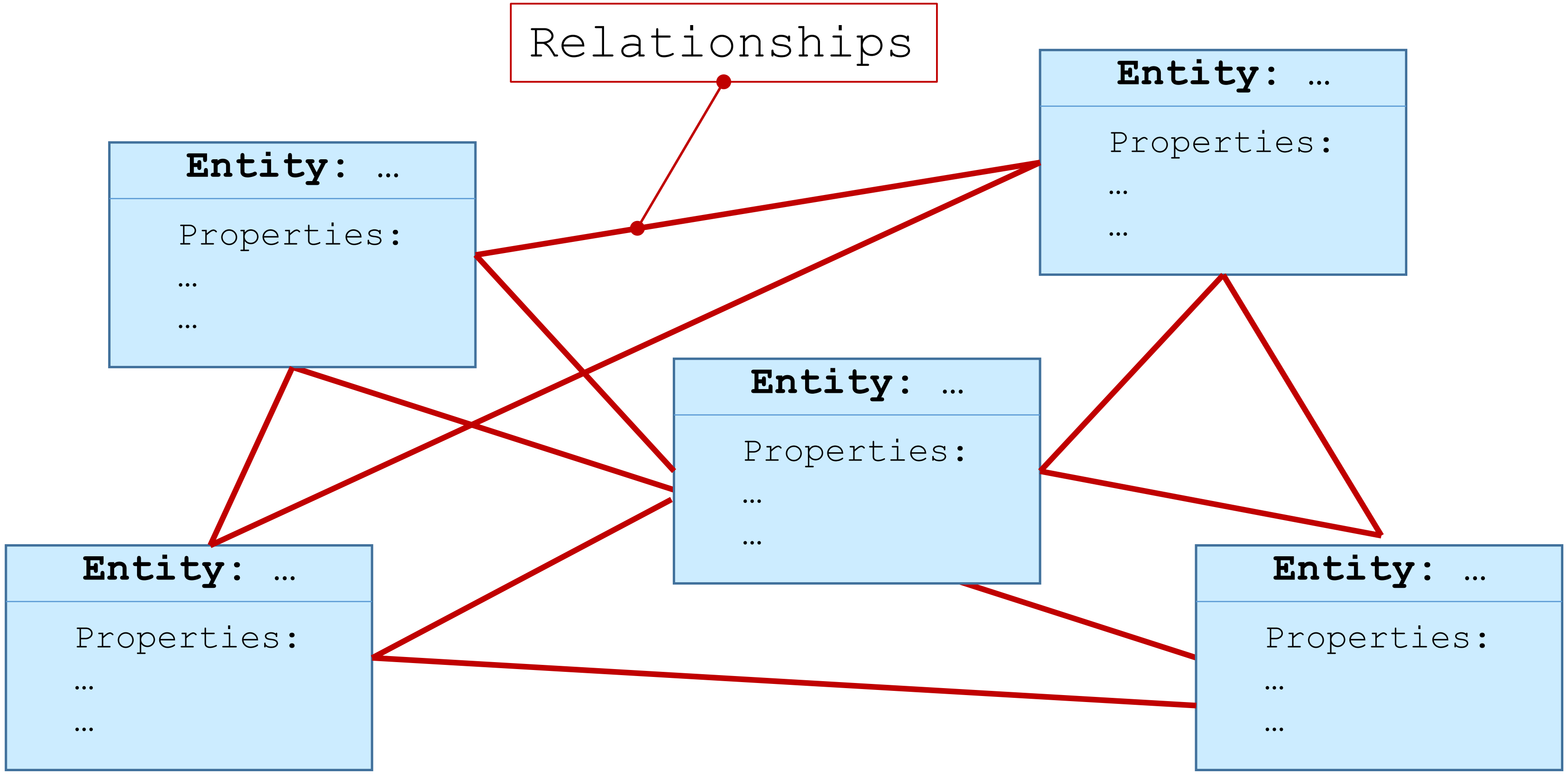
- Reference: DCF77
- Resolution: 1 sec
- Operation: continuous

System «B»

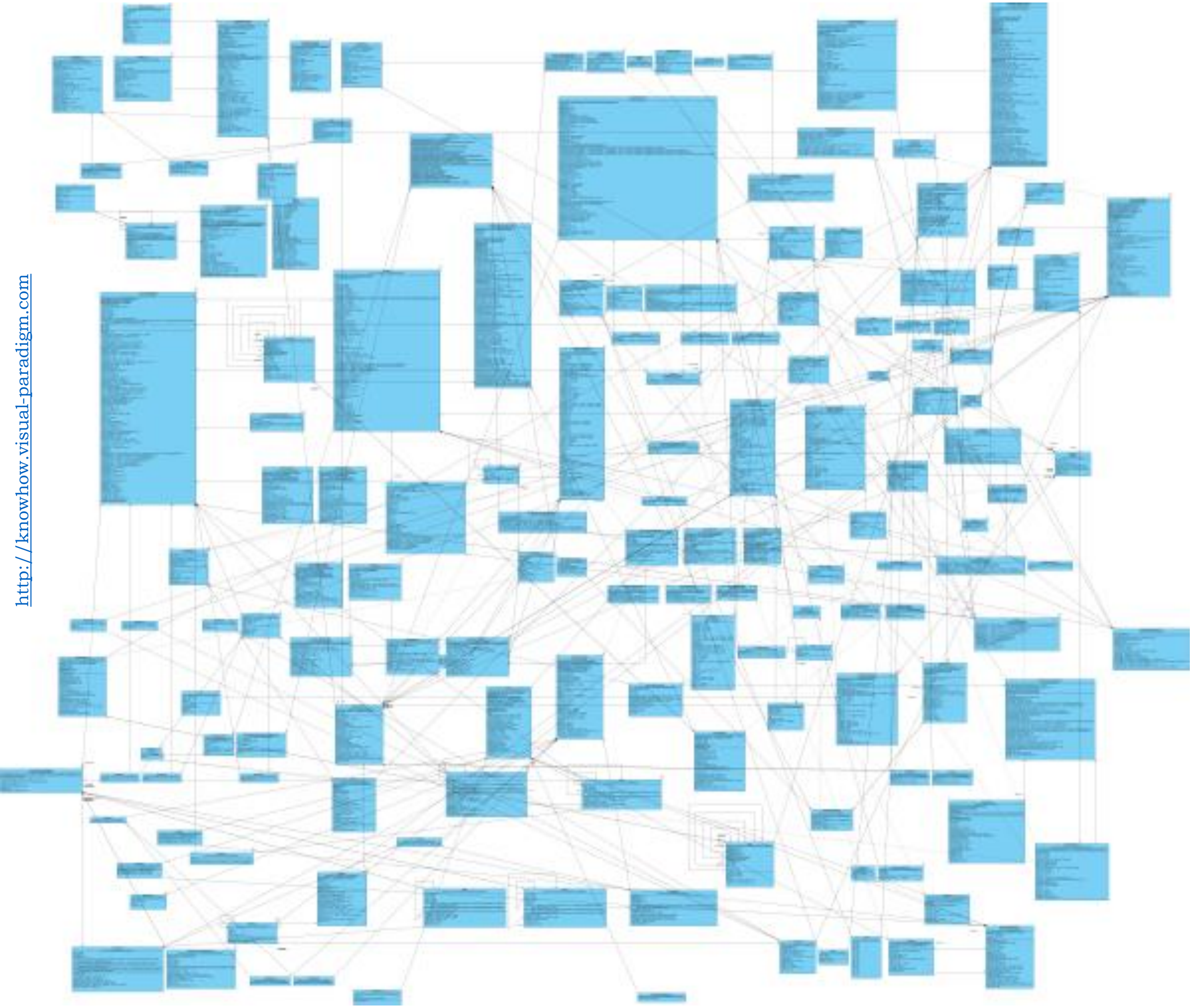
- Reference: quartz impulses
- Resolution: 1 msec
- Operation: counter
- Nulled at power-up

Domain Model =

Reminder: A Domain Model is a representation of the Entities, Relationships and their Properties in your Domain



Future-Proof Software-Systems [Part 3C]



http://knowhow.visual-paradigm.com

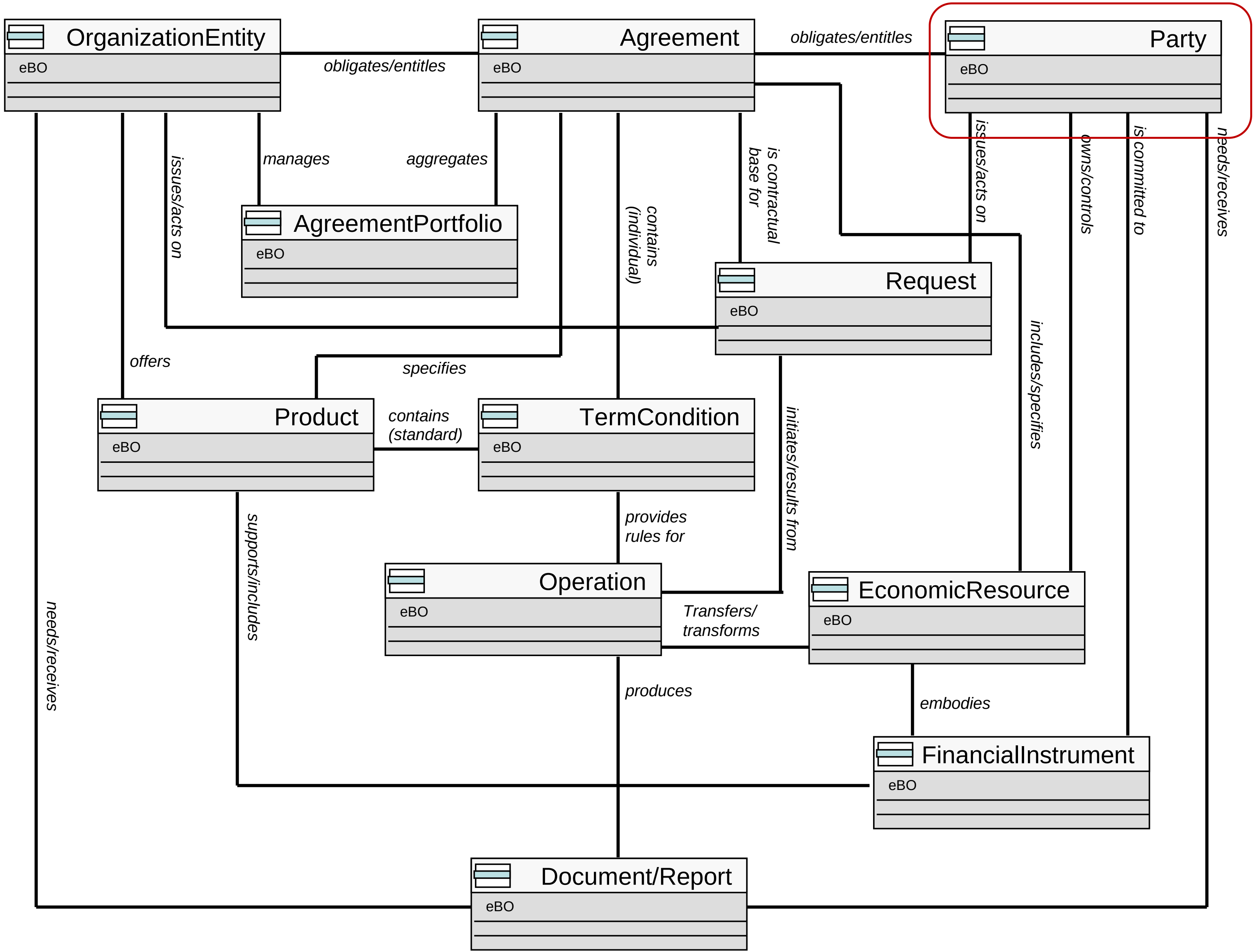
Problem:

Model-Explosion.

⇒ Size of the models grows!

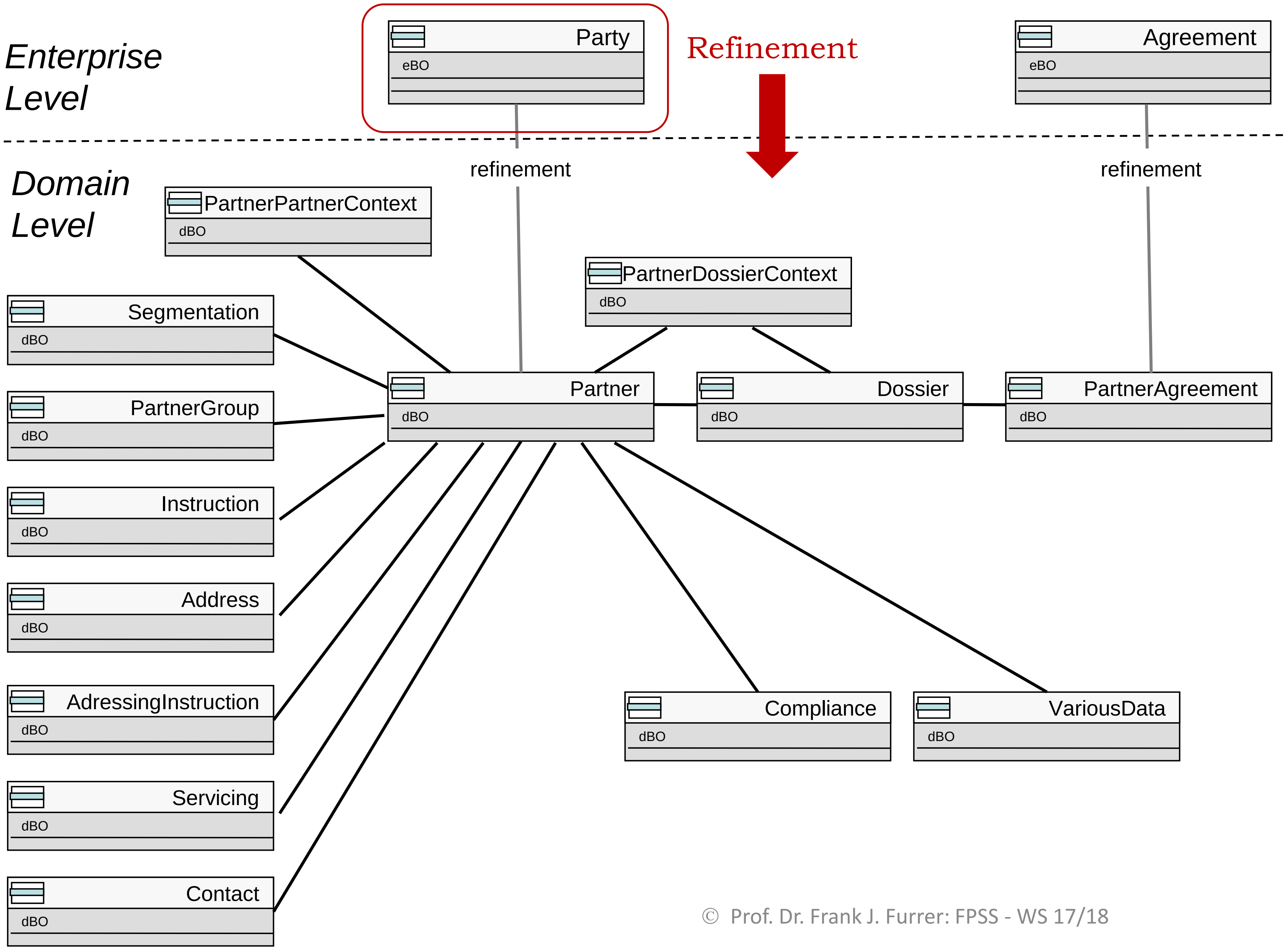
Build hierarchical models

Future-Proof Software-Systems [Part 3C]



Example:
Top-Level
Domain *Model*
for a Financial
Institution

Future-Proof Software-Systems [Part 3C]



Hierarchy:

2nd Level

Future-Proof Software-Systems [Part 3C]

http://blogs.msdn.com

A major reason for failure of software projects is a failure of people = the failure to *communicate*

DEFINITIONS

Ubiquitious Language [UL] =

The Ubiquitous Language is a *shared* language between the business and the development teams

The Ubiquitous Language comes from the *business*, and is enriched by the development team

www.thinkddd.com

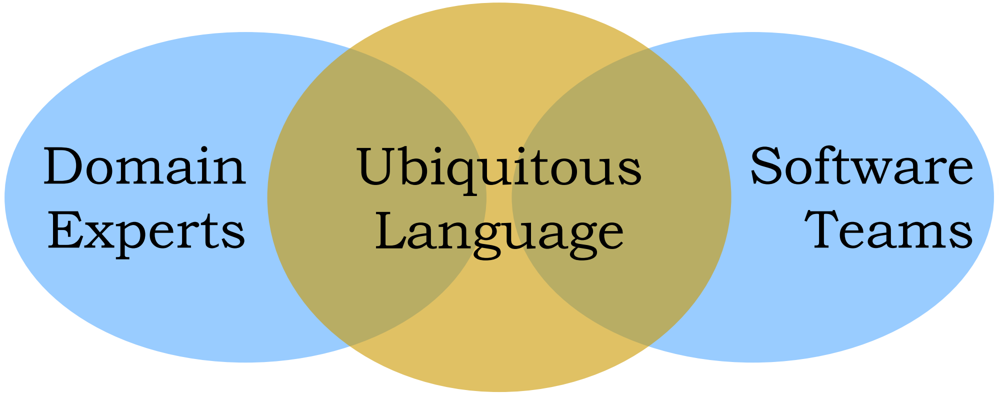
UL

Customer/Business

http://clipartzebraz.com

IT Organization

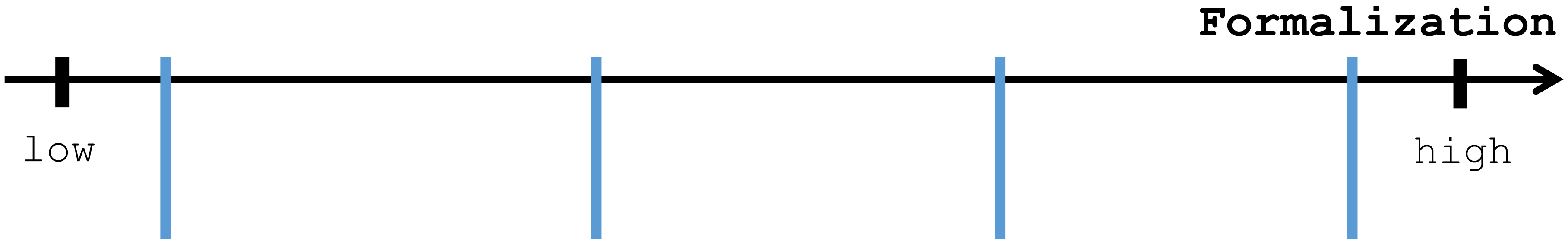
https://cimx.wordpress.com



Domain Experts

Ubiquitous Language

Software Teams



«Boxes & Lines» Text

Boxes & Lines with semantics

UML, SysML

Ontologies

How is an Ubiquitous Language developed?

… very often a good start is a textual table

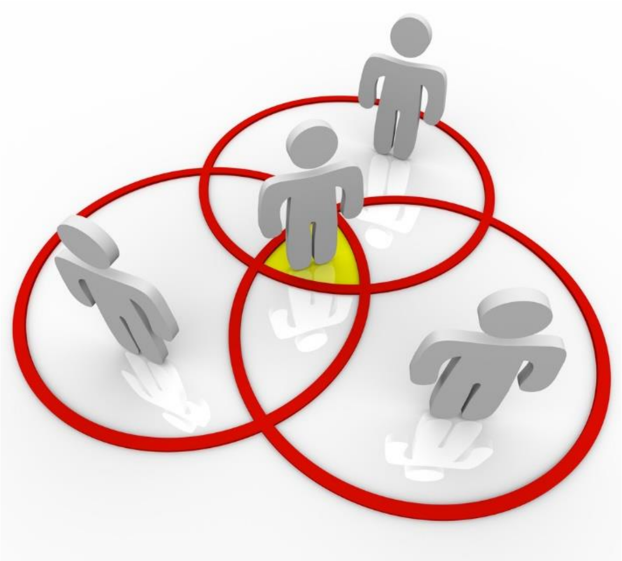
High Level Domain Entities (Enterprise Level)

Domain Concept	Description	Operations
Organization Entity	Legal Entity for executing business	- Create the entity - Internal organization of the entity - Agreements with other parties - Creation of financial products - Collaborate with other parties - Create reports - …
Operation	Value-transferring activity with adherence to legal & regulatory requirements	- Define parties - Oblige parties - Check legal & regulatory requirements - Execute operation - Document & archive operation - …
etc.		
etc.		

Definition

Concepts

Operations



http://blog.asha.org

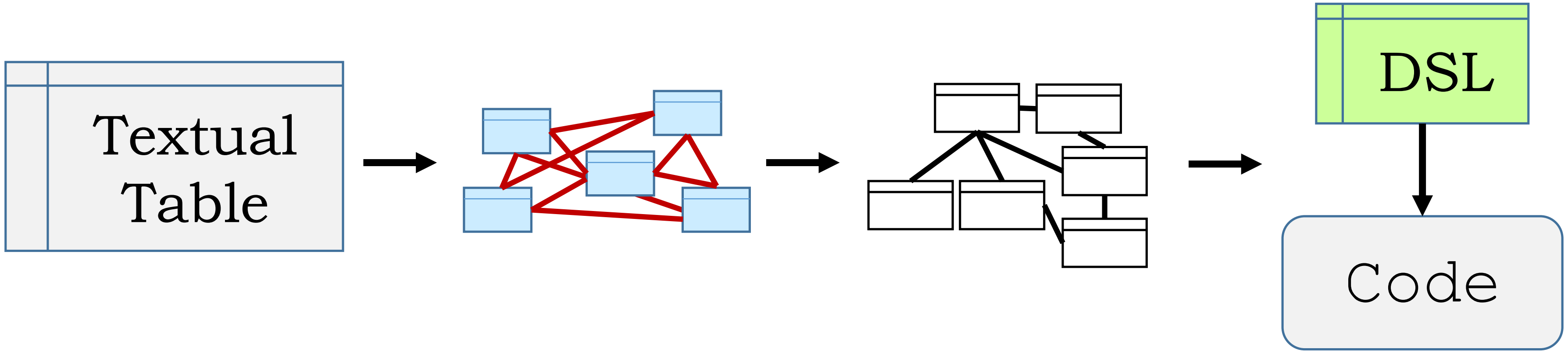
DEFINITIONS

Domain Specific Language =

A computer programming language of limited expressiveness focused on a particular domain.

The domain focus is what makes a limited language worthwhile.

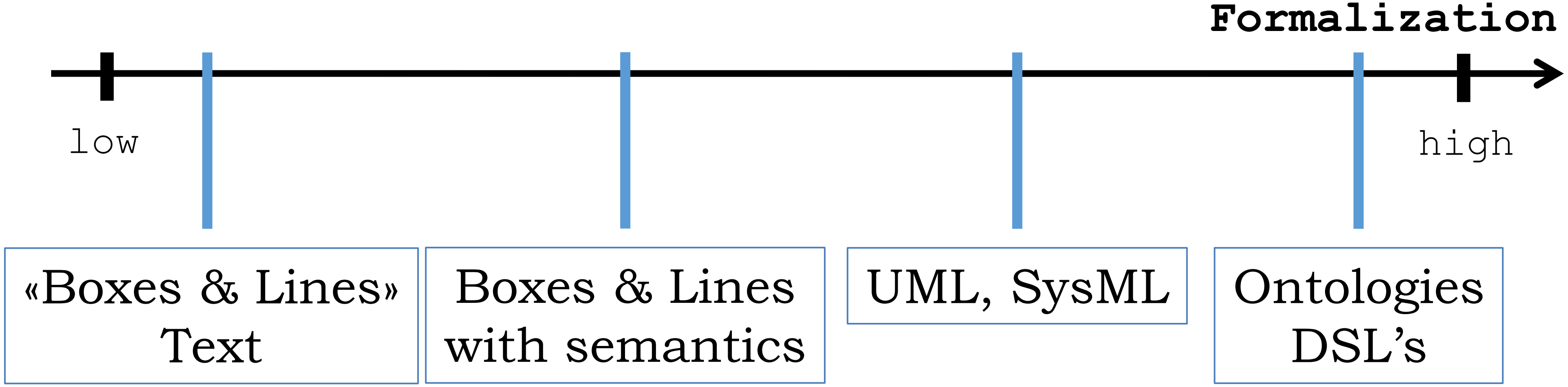
Fowler/Parsons 2011



Textual Table

DSL

Code



Formalization

low

high

«Boxes & Lines» Text

Boxes & Lines with semantics

UML, SysML

Ontologies DSL’s

Implementing Domain-Specific Engineering (DSE) in a company is a very demanding task

DSE "light"

http://www.aiming.in

```
if(top!=self)
function calcWidth() {
    var wW = 0;
    if (typeof window.innerWidth == 'number') {
        wW = window.innerWidth;
    } else if (document.documentElement && docum
        wW = document.documentElement.clientWid
    } else if (document.body && document.body.c
        wW = document.body.clientWidth;

    }
    if (sH = document.documentElement.scrollHe
        var wH = window.innerHeight || docume
        sW = !document.all && (sH > wH)
                                    'menu', 'wid
```

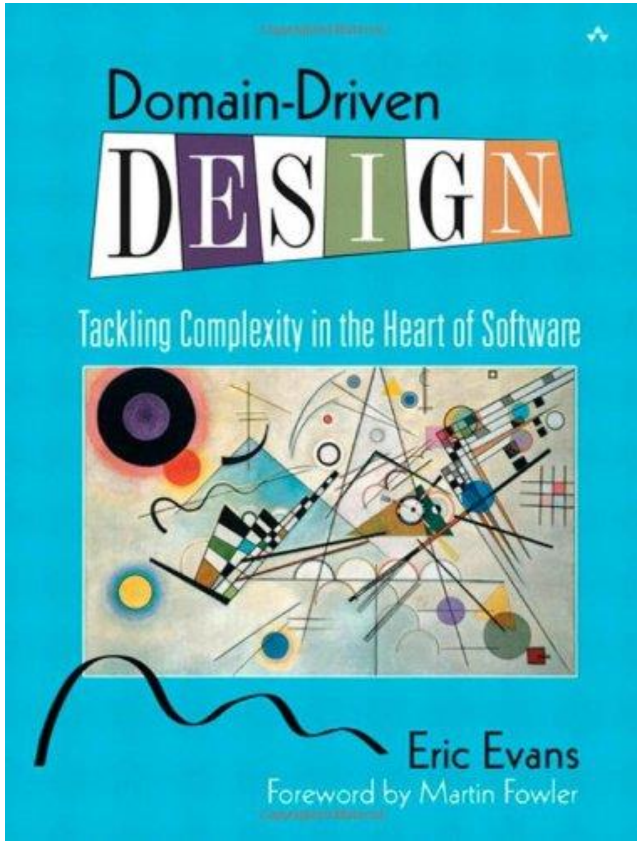
http://i1-news.softpedia-static.com

Use the business terminology in your code:

- Business objects → classes
- Business operations → services/methods
- Business terms → Variables

Future-Proof Software-Systems [Part 3C]

Textbook

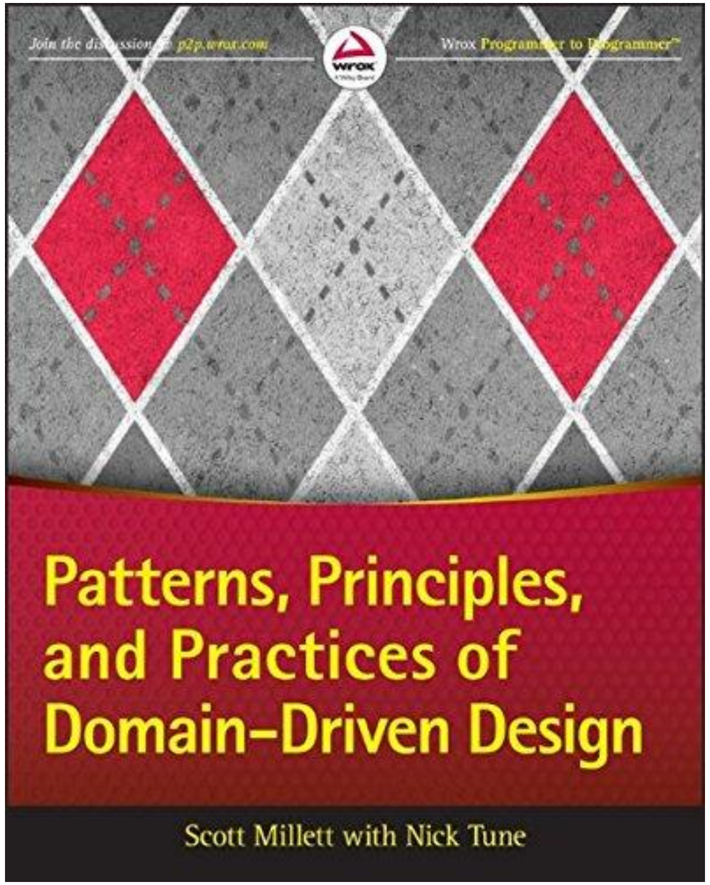

Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

Eric J. Evans :

Domain-Driven Design – *Tackling Complexity in the Heart of Software*

Addison Wesley Inc., USA, 2003. ISBN 978-0-321-12521-7

Textbook


Join the discussion @ p2p.wrox.com
wrox
Wrox Programmer to Programmer™
Patterns, Principles, and Practices of Domain-Driven Design
Scott Millett with Nick Tune

Scott Millett, Nick Tune:

Patterns, Principles, and Practices of Domain-Driven Design

Wrox, 2015. ISBN 978-1-1187-1470-6

Recommendations

Architecture Recommendations for Domain Software Engineering (DSE)

1. Gracefully build up an Ubiquituous Language between Business/Customer and IT (Implementer)
2. Define a consistent and complete domain model (hierarchical because of the size)
3. Push the formalization as far as possible (without losing the business/customer)
4. Use the terminology from the domain model/ubiquitous language in the code
5. Keep the domain model and the code implementation strictly synchronized at all times

Legacy System Migration/Modernization

What is a legacy system ?

… „a system built yesterday“

- and still in use today

… 25 years

http://www.123rf.com

http://www.nzz.ch/aktuell

?

Liability

Asset

Legacy System:

DEFINITIONS

Obsolete computer system which is still in use,

because its **data** can not be changed to newer or standard formats,

its **application programs** can not be upgraded,

or its **development/execution platform** can not be changed

"***can not***" = with an unreasonable effort (money, time & people)

http://www.123rf.com

Liability

http://www.nzz.ch/aktuell

Asset

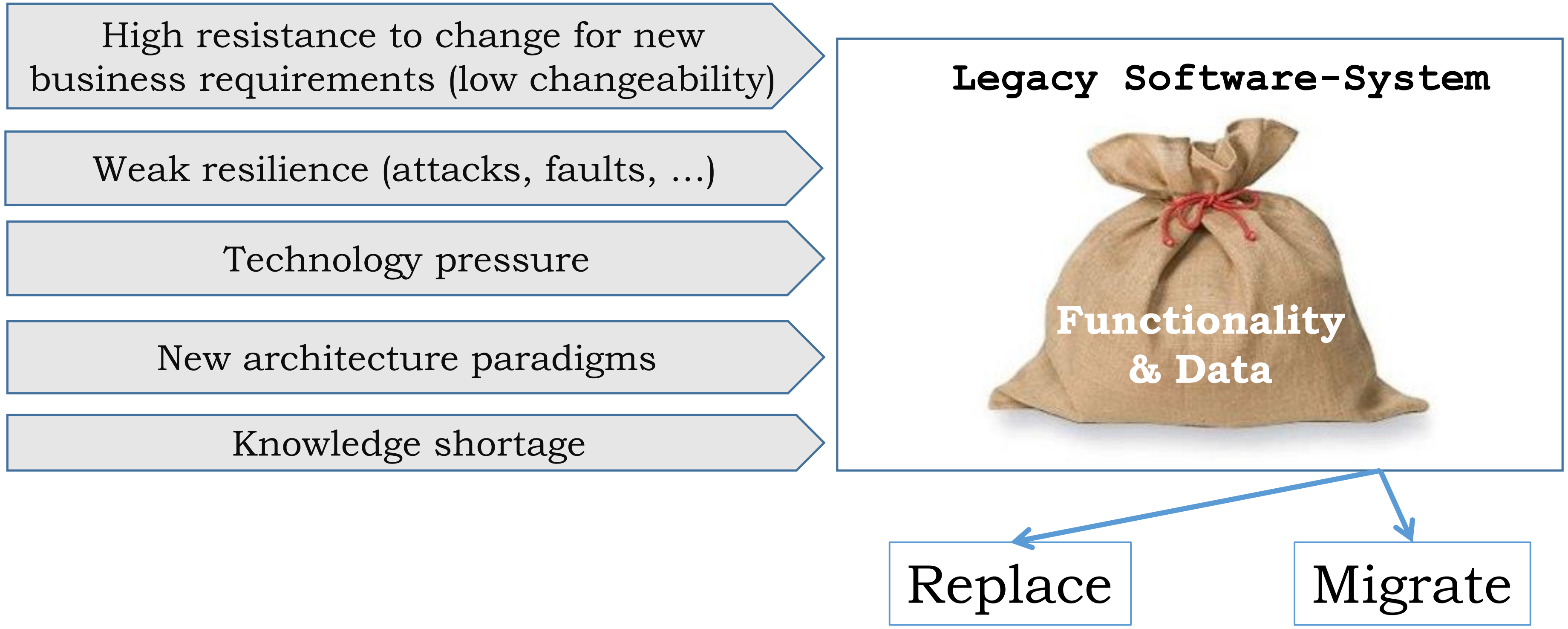
bad:

- very low changeability (= high resistance to change)
- weak resilience
- eroded architecture
- badly or not documented
- obsolete technology (HW & SW)
- large technical debt
- lost knowledge (people left)
- difficult integration context

good:

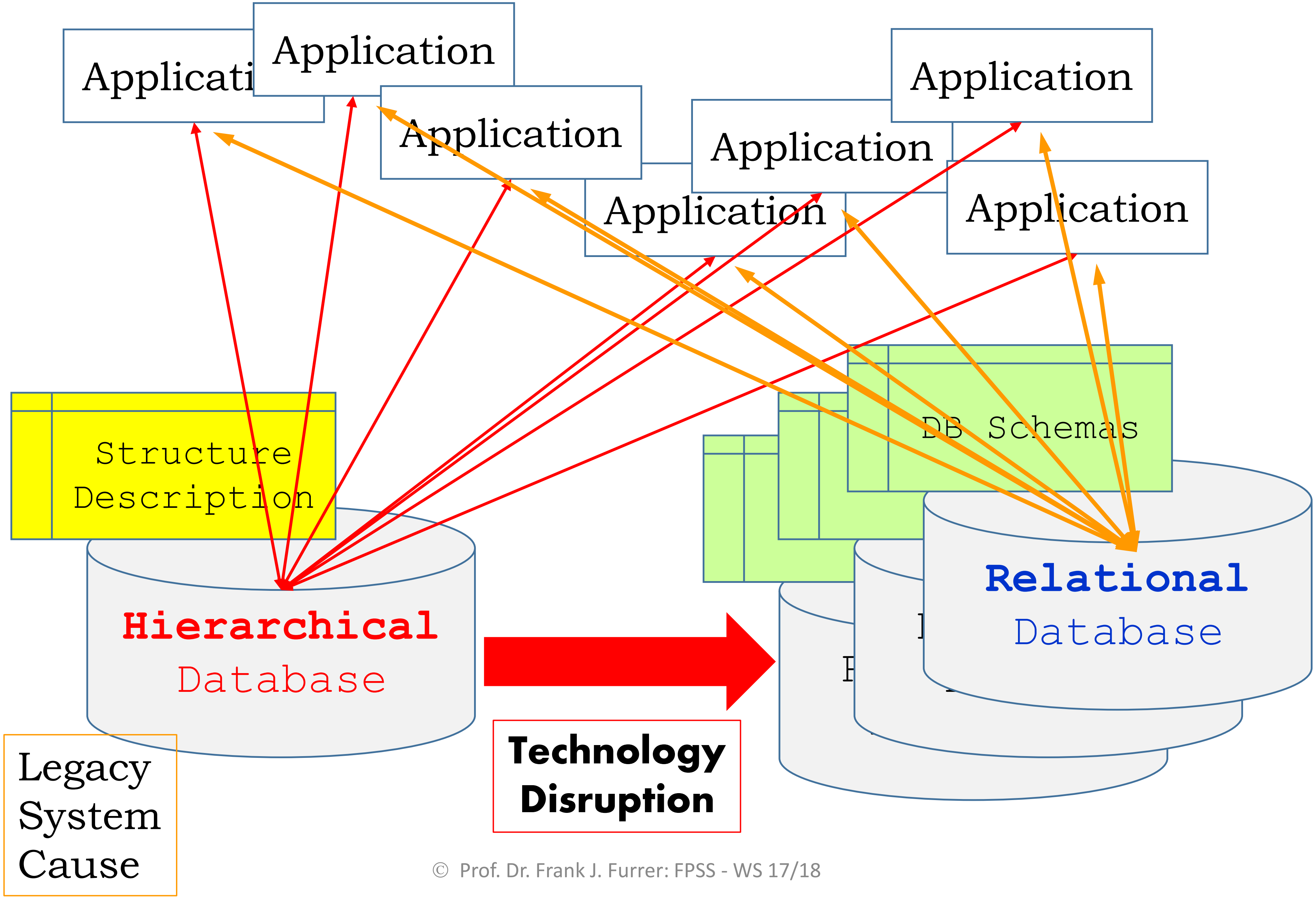
- invaluable *implicit* knowledge of the domain and the business processes
- stable operation (mature)
- good solutions/algorithms
- often: suprisingly good code

Why must we **modernize** legacy systems ?


High resistance to change for new business requirements (low changeability)
Weak resilience (attacks, faults, ...)
Technology pressure
New architecture paradigms
Knowledge shortage
Legacy Software-System
Functionality & Data
Replace
Migrate

Decision Criteria:

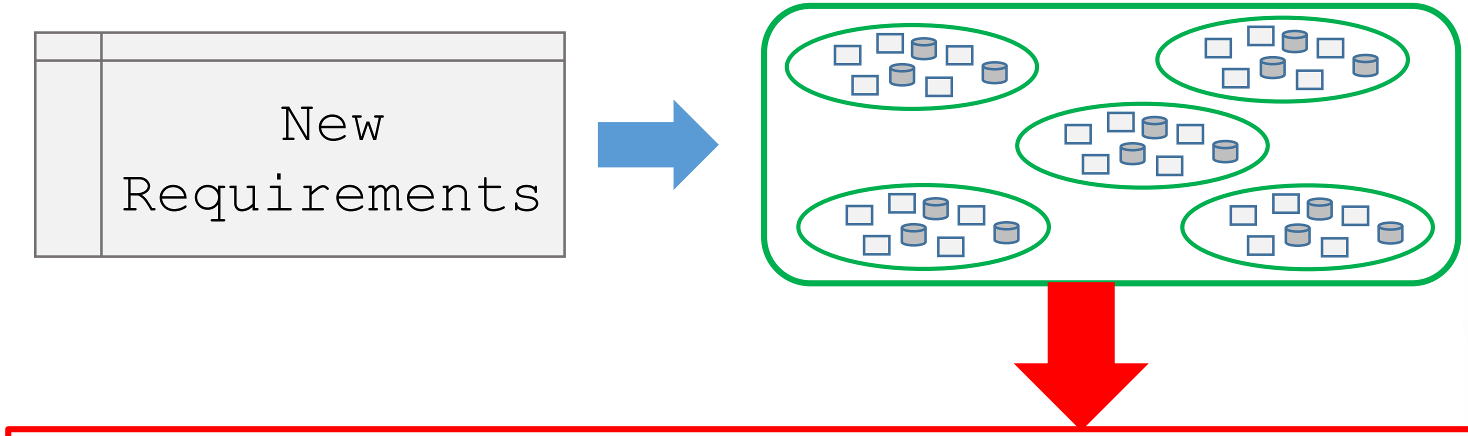
Operational Risk	Minimizing the risk of operational faults
Fit-for-Future	Technical Debt?
Cost/Time	Total Effort
Additional Constraints	e.g. Certification



Technology Disruption

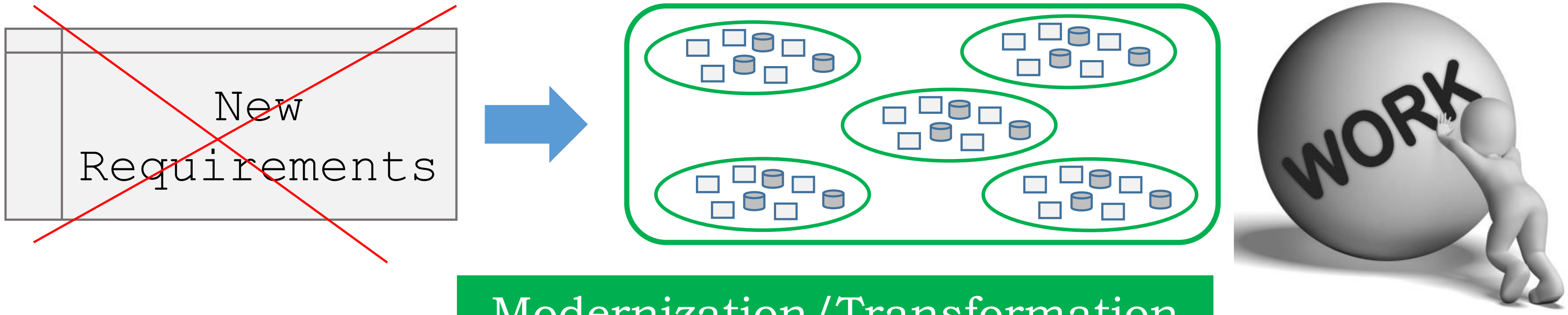
Legacy System Cause

Legacy system modernization strategies



The **evolution** becomes unmanageable (*low* architecture quality)

Legacy system modernization **strategies**



Modernization/Transformation

Replacing:
Completely new development starting from systems requirements

Re-Architecting:
Transforming to new architecture paradigm

Re-Engineering:
Transforming to new technology (new infrastructure or software technology)

Re-Factoring:
Improving existing code (no functionality change)

⇒ may require *reverse engineering* – if no or insufficient information (code/doc)

Example: Code Reverse-Engineering:

~~Source Code~~ ← COBOL Compiler ←

http://roycebits.blogspot.ch

→ ~~Doc~~

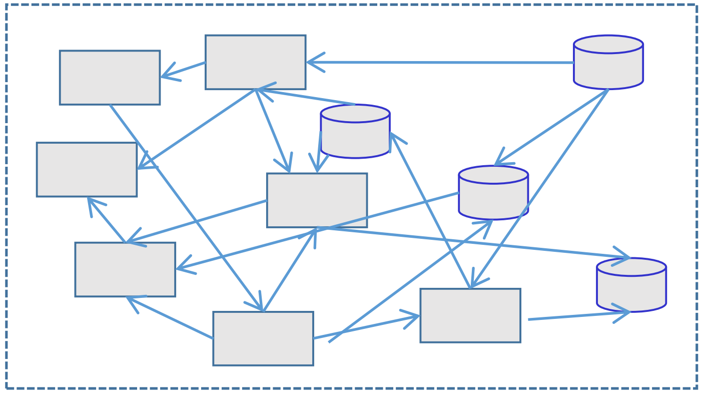
Executable machine code:

```
8020 78
8021 A9 80
8023 8D 15 03
8026 A9 2D
8028 8D 14 03
802B 58
802C 60
802D EE 20 D0
8030 4C 31 EA
```

http://c2.com/cgi/wiki?MachineCode

Future-Proof Software-Systems [Part 3C]

Example: Code Reverse-Engineering:



Tool support

Structure Analyzer

Assembly language code:

```
Start: .org $8020
SEI
LDA #$80
STA $0315
LDA #$2D
STA $0314
CLI
RTS
INC $D020
JMP $EA31
```

Executable machine code:

```
8020 78
8021 A9 80
8023 8D 15 03
8026 A9 2D
8028 8D 14 03
802B 58
802C 60
802D EE 20 D0
8030 4C 31 EA
```

De-Assembler

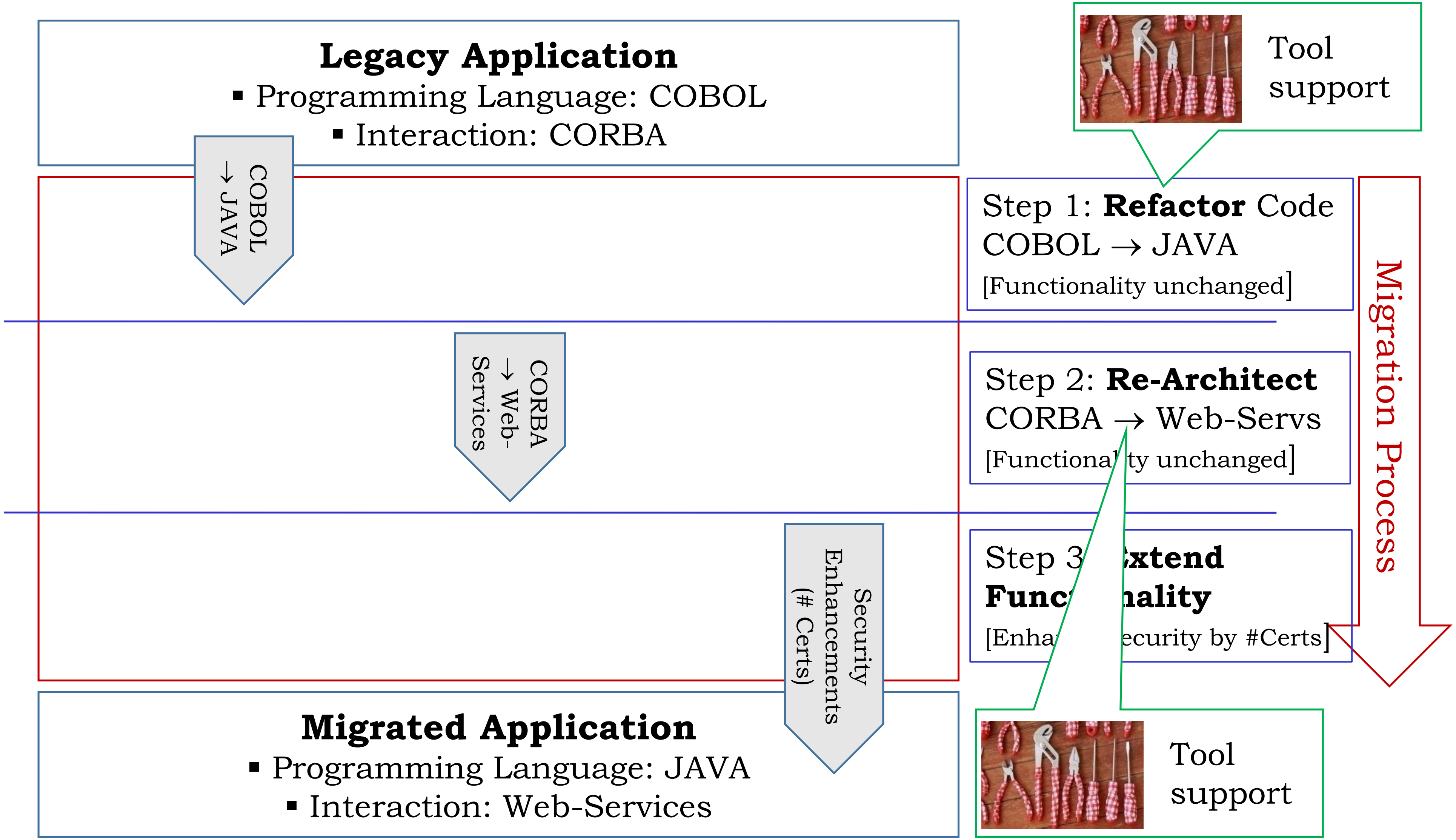
Tool support

http://c2.com/cgi/wiki?MachineCode

Legacy system modernization techniques

Type of Migration	Current State	Target State
Replacing - Completely new development starting from systems requirements	Operational software. Cost, time and risk for a migration to high	Software has completely been rewritten, starting from the initial requirements
Re-Architecting - Transforming to new architecture paradigm (Considerable functional change)	Operational software. Architecture paradigm has changed [e.g. monolithic architecture ⇒ service-oriented architecture]	Software runs under the new architecture paradigm
Re-Engineering - Transforming to new technology base, e.g. new infrastructure or software technology (limited functional change)	Operational code running on an outdated execution platform or using an obsolete software technology	Code runs on the modern execution platform or uses modern software technology
Re-Factoring - Improving existing code (no functionality change)	Operational code, deficiencies in the program implementation	Improved code (quality criteria)
Reverse Engineering - No or insufficient information (code + doc)	Operational code, massive lack of documentation, of knowledge and of source code	System is sufficiently understood and documented to start migration

Example: COBOL/CORBA ⇒ JAVA/Web-Services Migration + Security



Legacy Application
- Programming Language: COBOL
- Interaction: CORBA

COBOL → JAVA

CORBA → Web-Services

Security Enhancements (# Certs)

Migrated Application
- Programming Language: JAVA
- Interaction: Web-Services

Tool support

Step 1: **Refactor** Code
COBOL → JAVA
[Functionality unchanged]

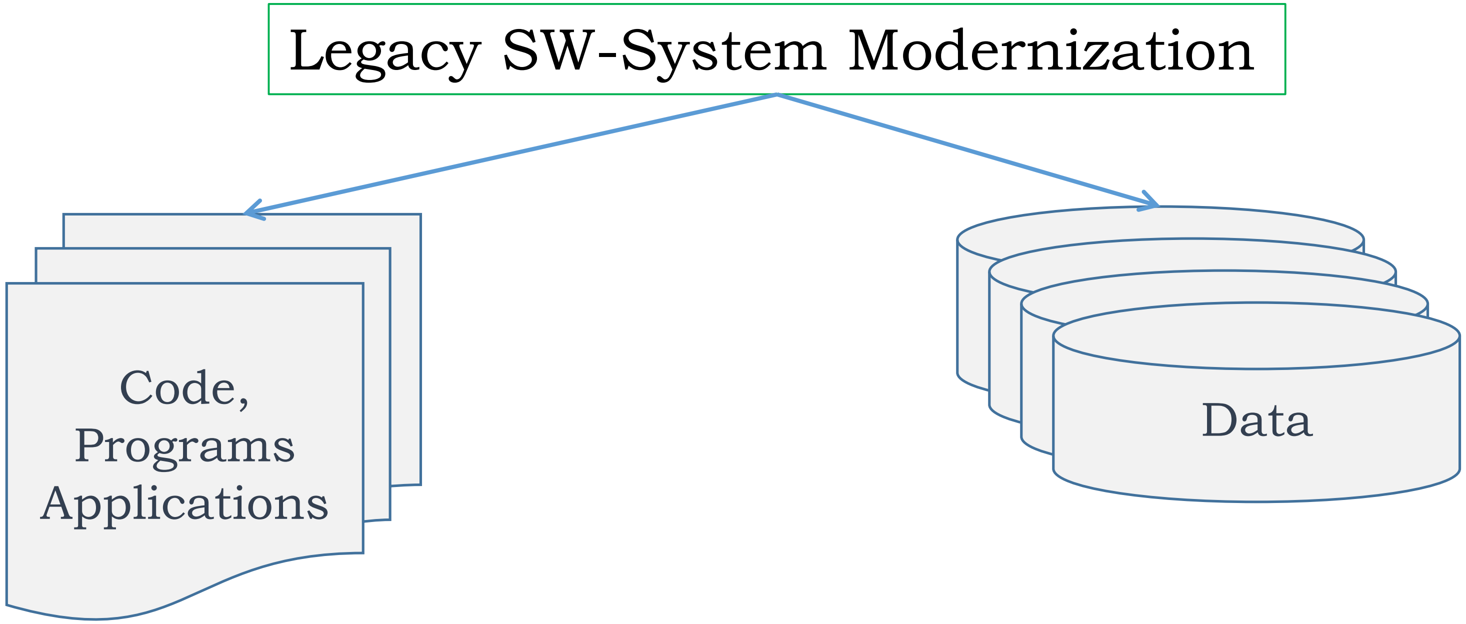
Step 2: **Re-Architect**
CORBA → Web-Servs
[Functionality unchanged]

Step 3: **Extend Functionality**
[Enhance Security by #Certs]

Migration Process

Tool support

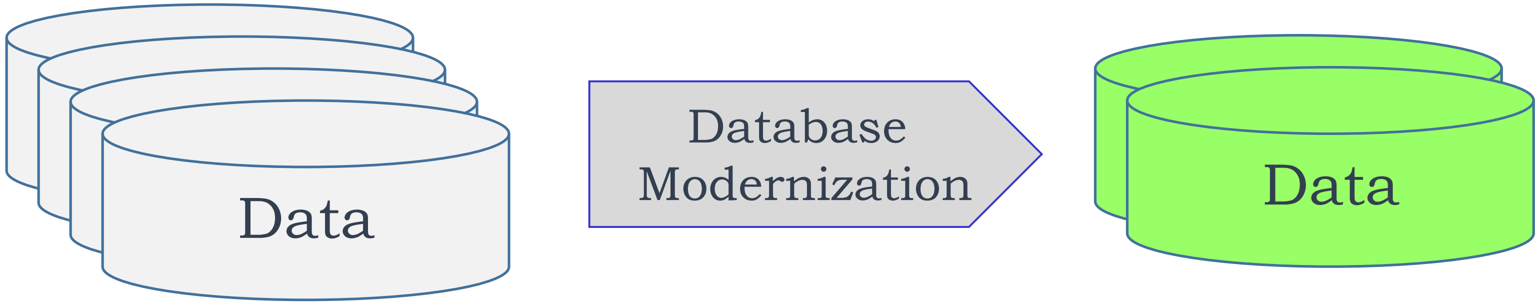
Future-Proof Software-Systems [Part 3C]



- ➢ **Replacing**
- ➢ **Re-Architecting**
- ➢ **Re-Engineering**
- ➢ **Re-Factoring**
- ➢ **Reverse Engineering**

- ➢ **Replacing**
- ➢ **Re-Architecting**
- ➢ **Re-Engineering**
- ➢ **Re-Factoring**
- ➢ **Reverse Engineering**

Legacy SW-System Modernization



- ➢ **Replacing**
- ➢ **Re-Architecting**
- ➢ **Re-Engineering**
- ➢ **Re-Factoring**
- ➢ **Reverse Engineering**

Objectives:

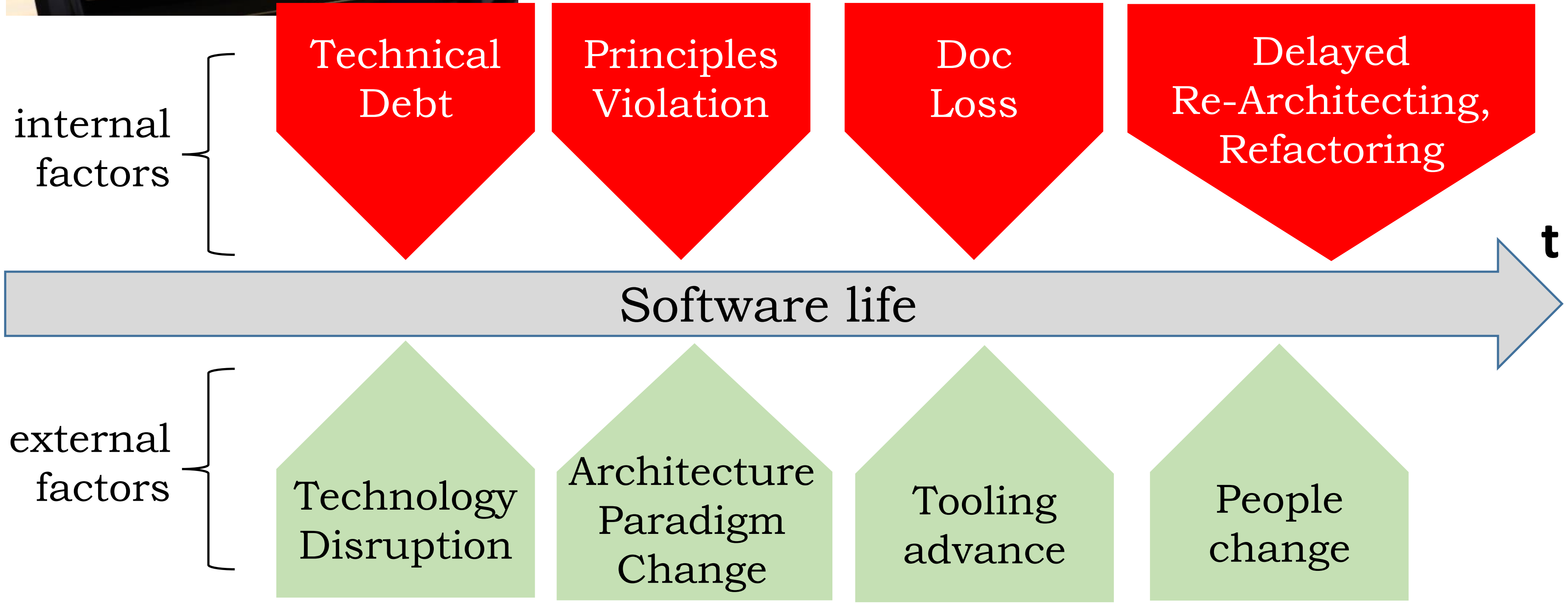
- ▪ Redundancy elimination
- ▪ Syntactic/semantic integrity
- ▪ Database technology (relational)
- ▪ Access performance
- ▪ Transactional integrity
- ▪ Modeling

Future-Proof Software-Systems [Part 3C]

http://www.rakenapp.com

Can we avoid the production of «Legacy software»

… **NO!** – only make the legacy easier



internal factors

Technical Debt

Principles Violation

Doc Loss

Delayed Re-Architecting, Refactoring

t

Software life

external factors

Technology Disruption

Architecture Paradigm Change

Tooling advance

People change

Future-Proof Software-Systems [Part 3C]

http://www.rakenapp.com

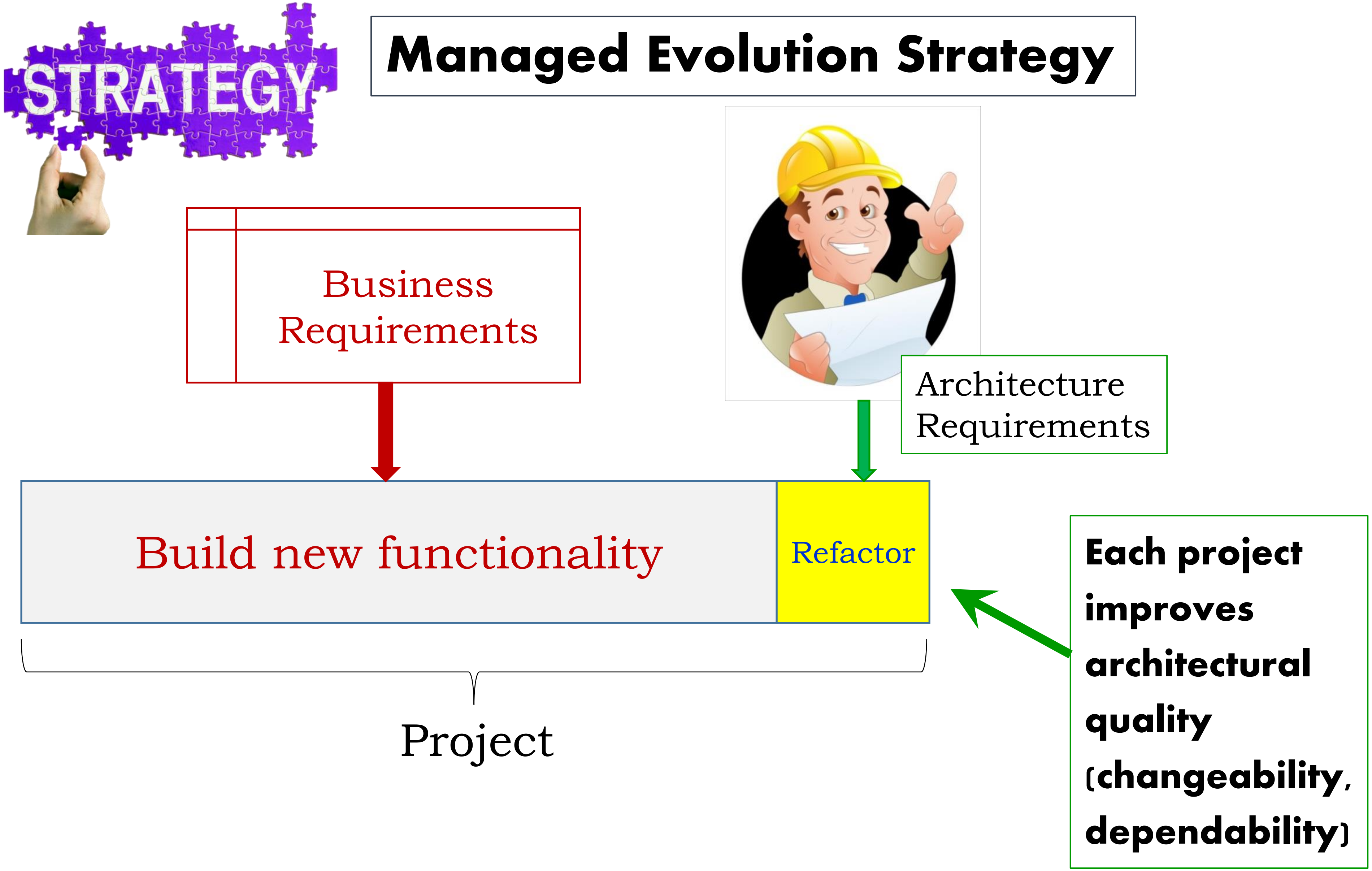
Can we avoid the production of «Legacy software»

… **NO!** – only make the legacy easier

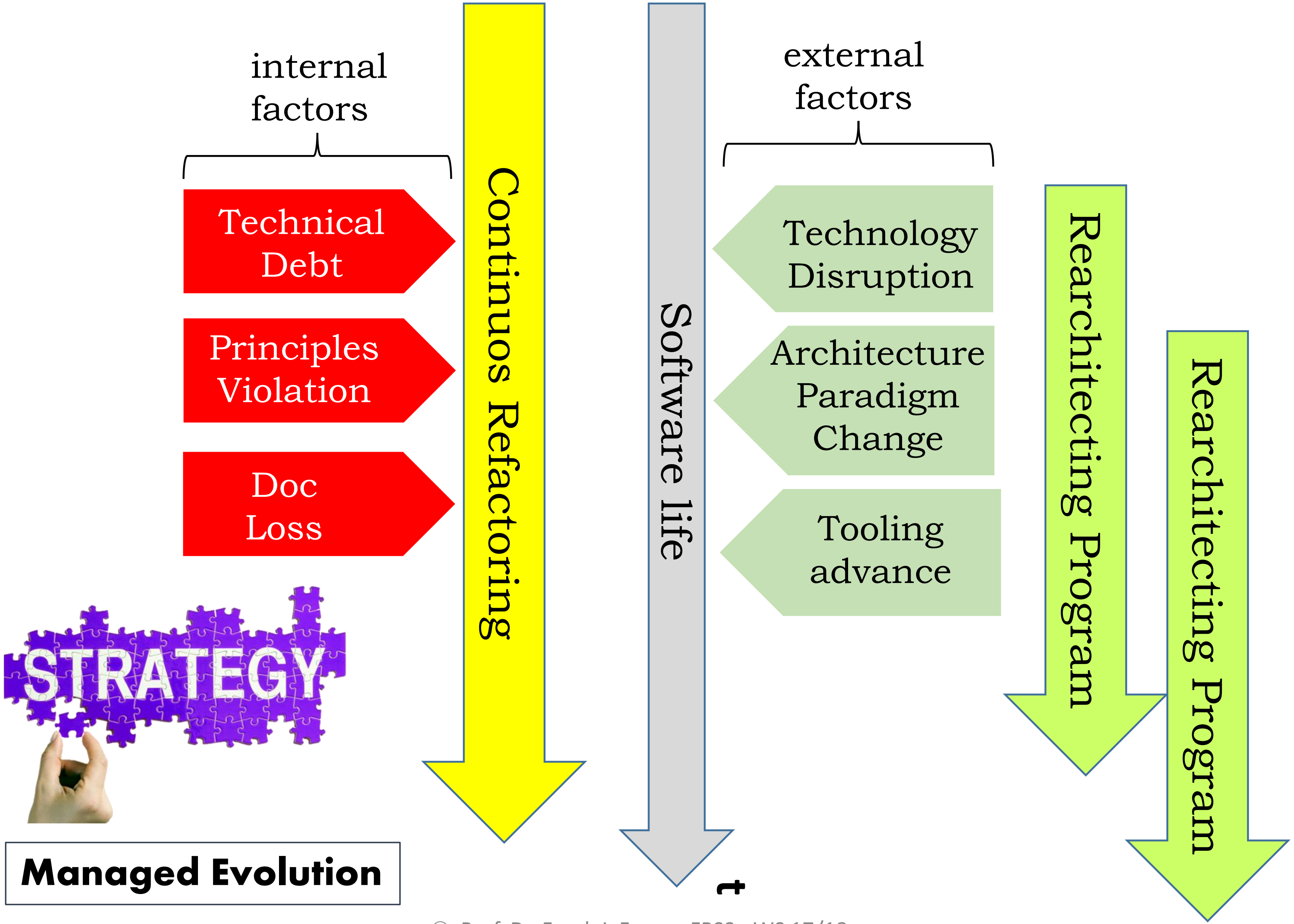
Solution:
Continuous Rearchitecting/Refactoring

Managed Evolution Strategy

Managed Evolution Strategy



Each project improves architectural quality (changeability, dependability)



internal factors

Technical Debt

Principles Violation

Doc Loss

Continuos Refactoring

Software life

t

external factors

Technology Disruption

Architecture Paradigm Change

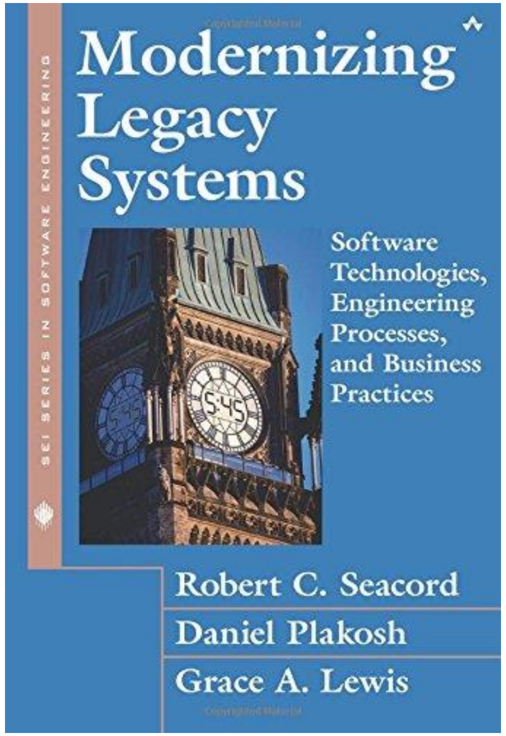
Tooling advance

Rearchitecting Program

Rearchitecting Program

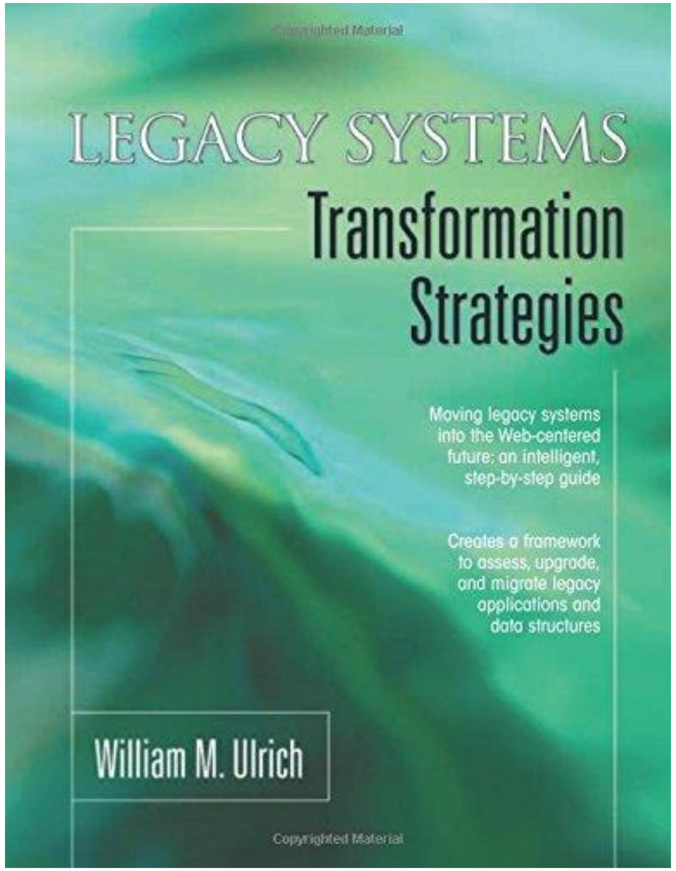
Managed Evolution

Textbook


Modernizing Legacy Systems
Software Technologies, Engineering Processes, and Business Practices
SEI SERIES IN SOFTWARE ENGINEERING
Robert C. Seacord
Daniel Plakosh
Grace A. Lewis

Robert C. Seacord, Daniel Plakosh, Grace A. Lewis:
Modernizing Legacy Systems - *Software Technologies, Engineering Processes, and Business Practices*
Addison-Wesley Professional, USA, 2003. ISBN 978-0-321-11884-4

Textbook


LEGACY SYSTEMS
Transformation Strategies
Moving legacy systems into the Web-centered future: an intelligent, step-by-step guide
Creates a framework to assess, upgrade, and migrate legacy applications and data structures
William M. Ulrich

William M. Ulrich:
***Legacy Systems* – Transformation Strategies**
Prentice Hall Inc., USA, 2002. ISBN 978-0-130-44927-6

Recommendations

Architecture Recommendations for Legacy System Modernization

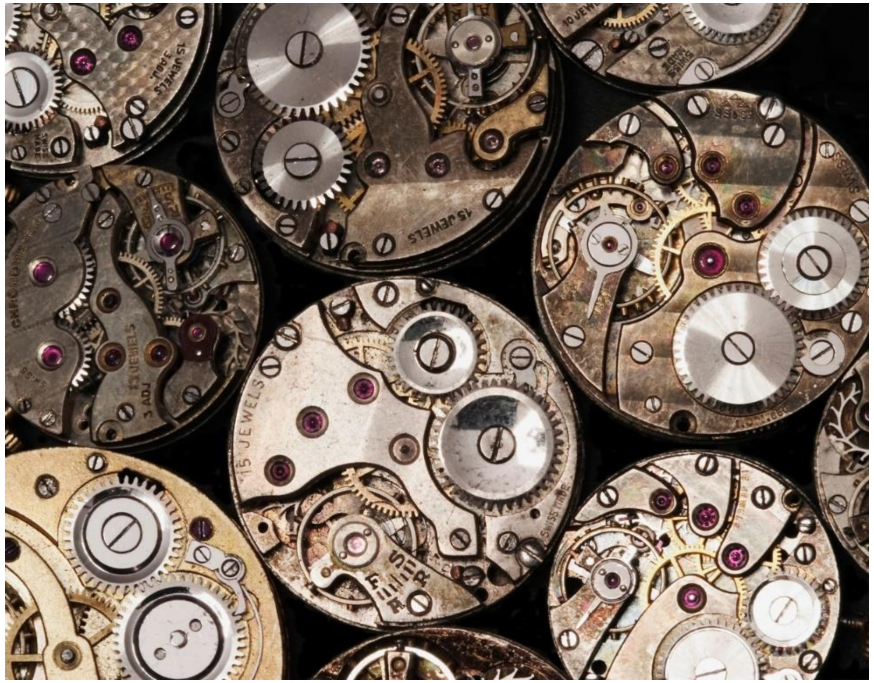
1. Unambigously specify the boundary of the system (Code & Data) to be migrated/modernized
2. Clearly assess the state of the legacy system (code, data, documentation, value)
3. Precisely define the migration/modernization goals (for code & data)
4. Choose a migration/modernization strategy based on risk, fit-for-future, cost & time and quality attributes (e.g. certification or validation etc.)
5. Select optimum tool support [Note: Many excellent tools available, search www]

Software Product Lines

DEFINITIONS

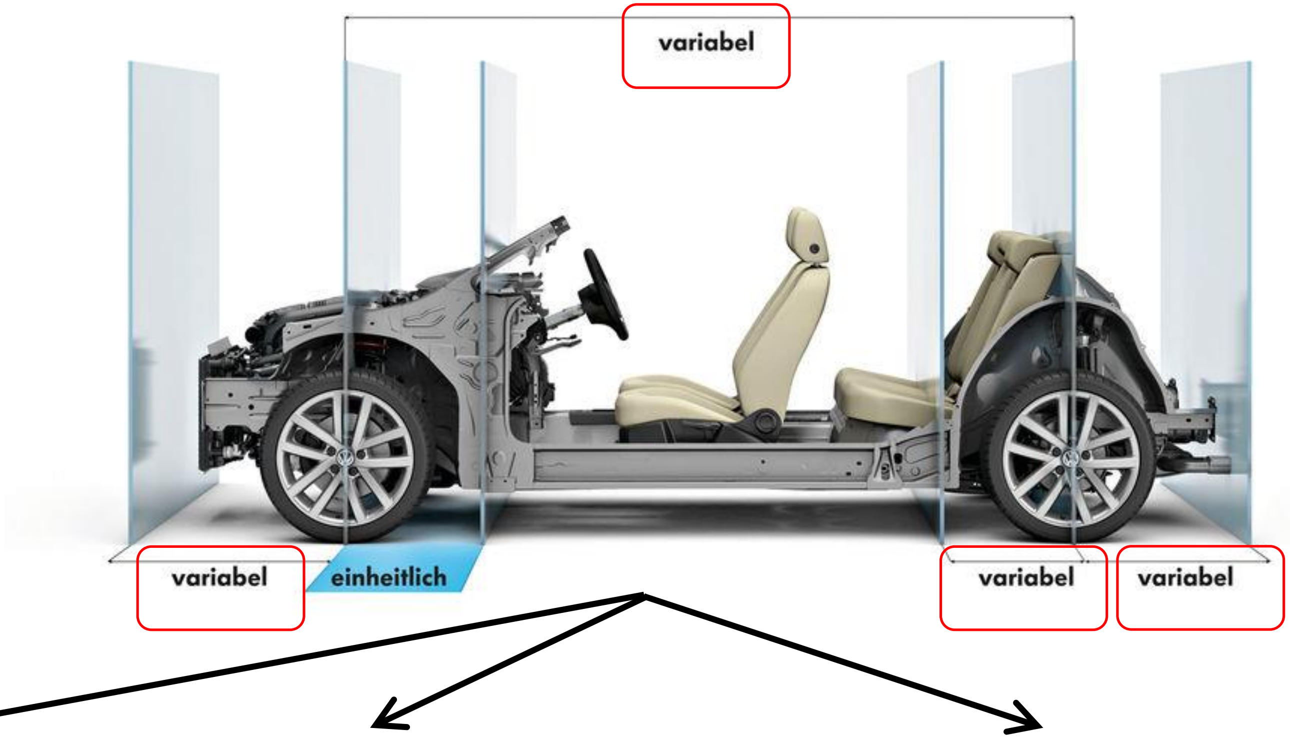
A **Software Product Line** is
a **set** of software-intensive systems
sharing a **common, managed set of features**
that satisfy the specific needs of a **particular market segment or mission**
and that are developed from a **common set of core assets**
in a **prescribed way**

[Clements02]



http://www.ahubaux.com

Example:
Product line in automotive development



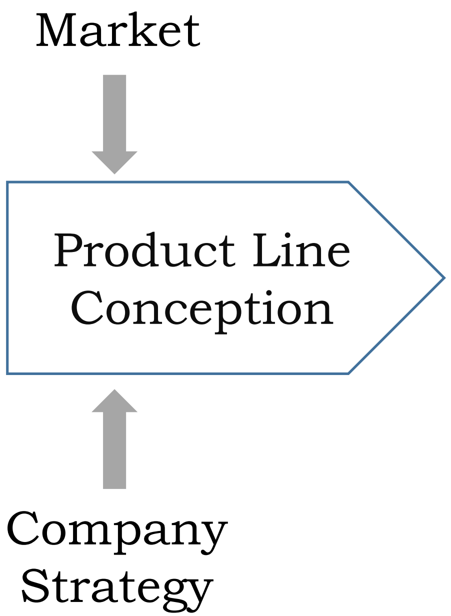
http://www.auto-motor-und-sport.de/

VW Jetta

VW Golf

VW Passat

SW Product Line Development

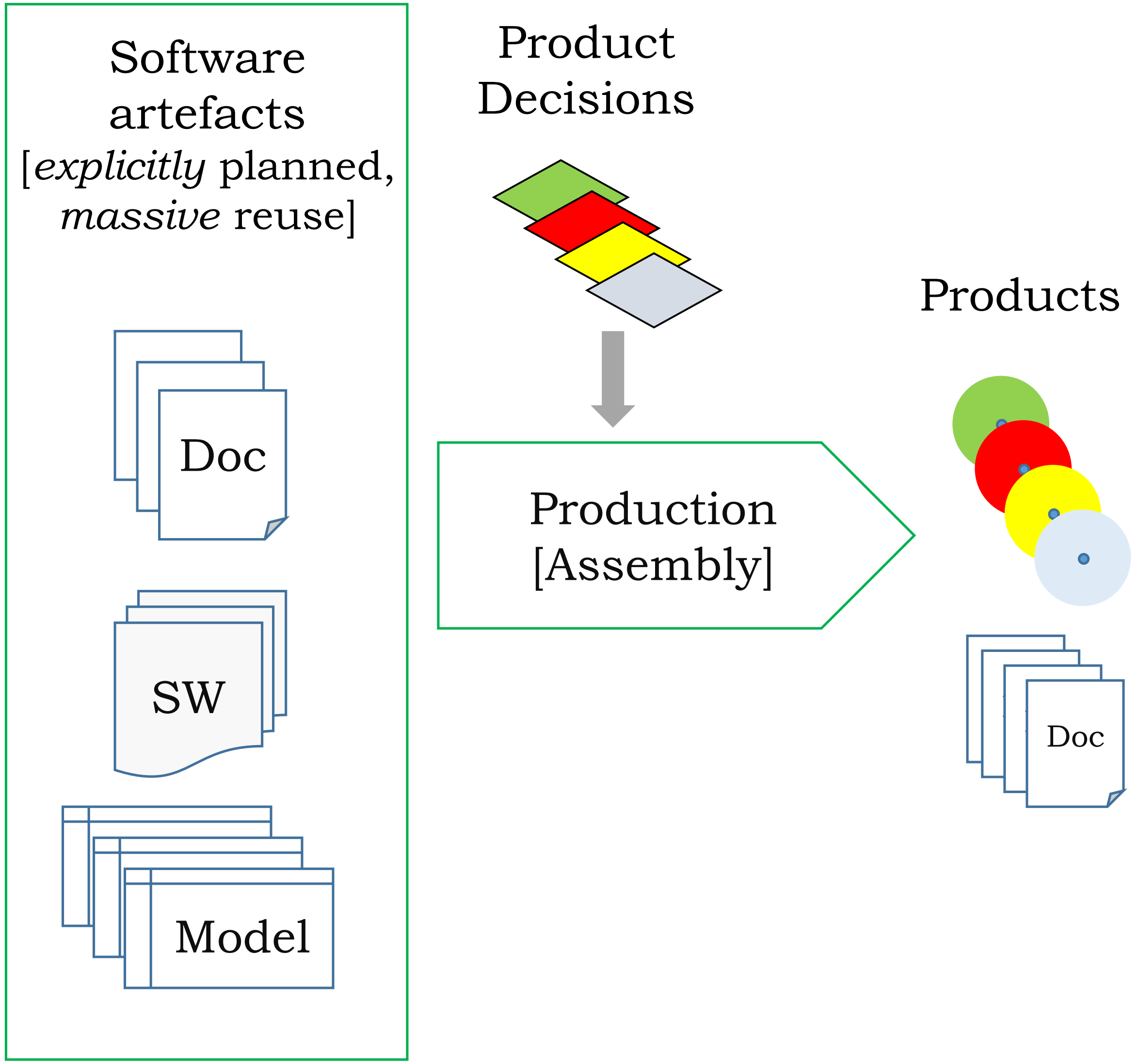


Market

Product Line Conception

Company Strategy

http://blog.enerdynamics.com



Software artefacts [*explicitly* planned, *massive* reuse]

Doc

SW

Model

Product Decisions

Production [Assembly]

Products

Doc

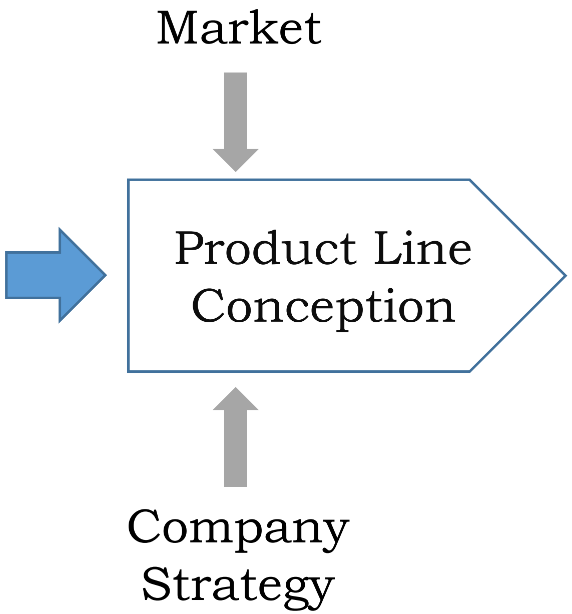
SW Product Line Development

Feature Model

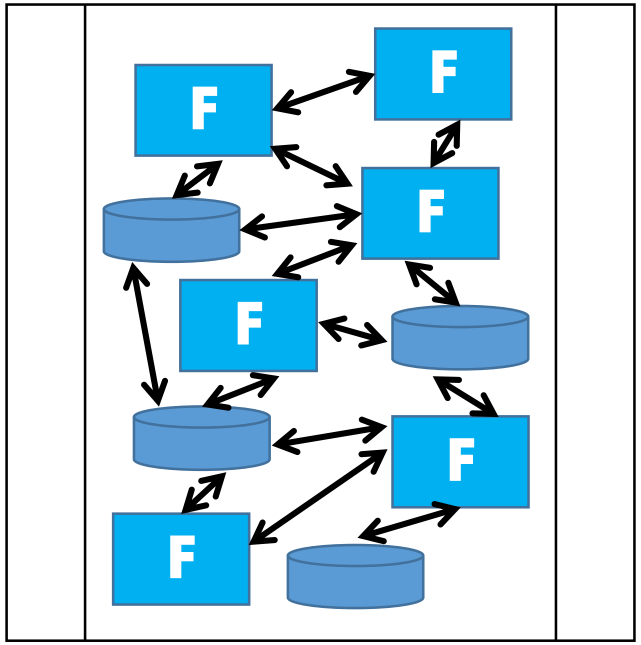
Wave 1:
- X
- Y

Wave 2:
- Z
- R

…

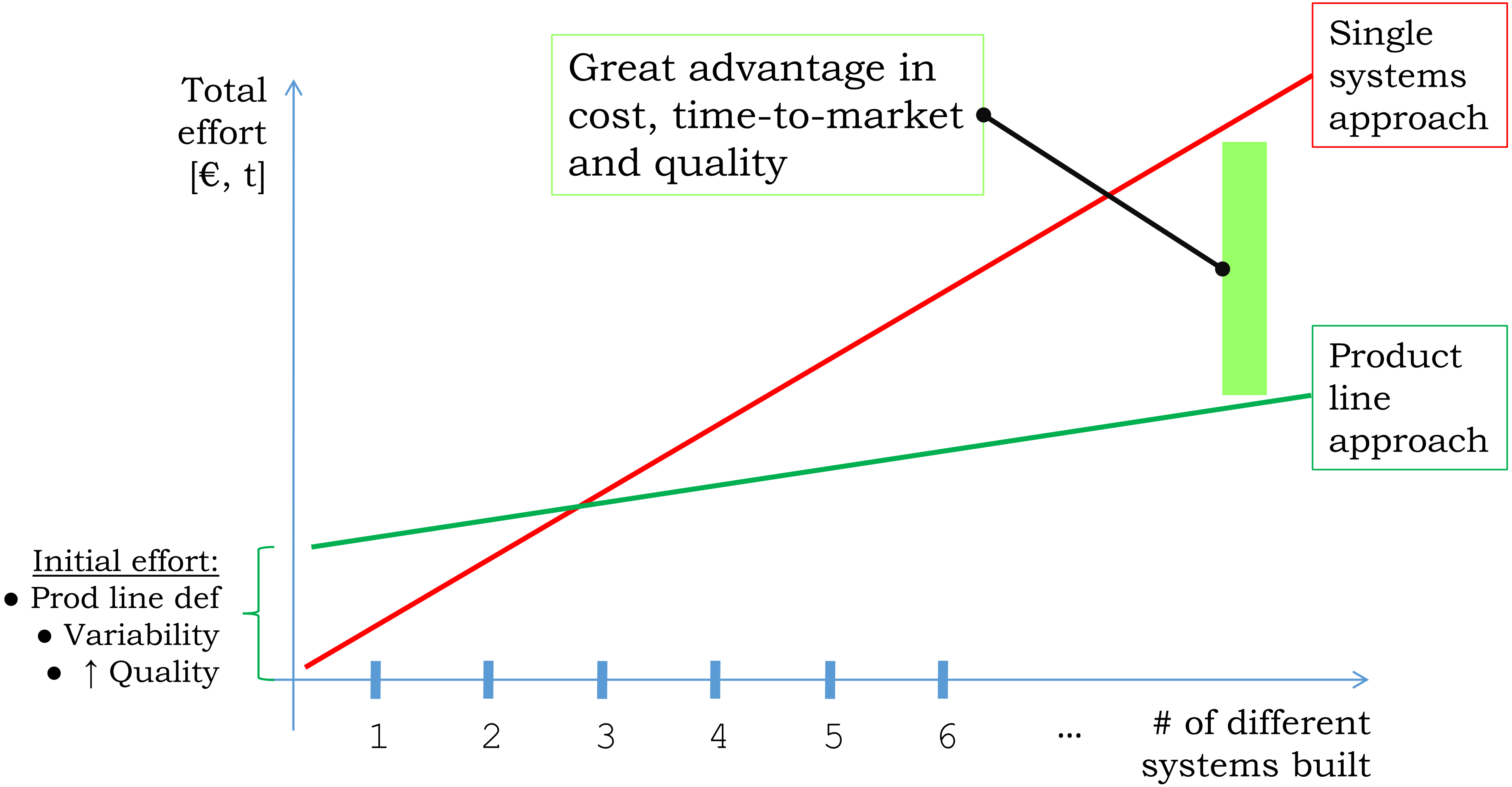


Product Line **Architecture**



http://grandyouth.org

Economics of Product Line Development:



Software Product Line

Effect on Future-Proof Softw

Superbly planned and executed managed redundancy

✓ Product lines make use of planned, massive reuse

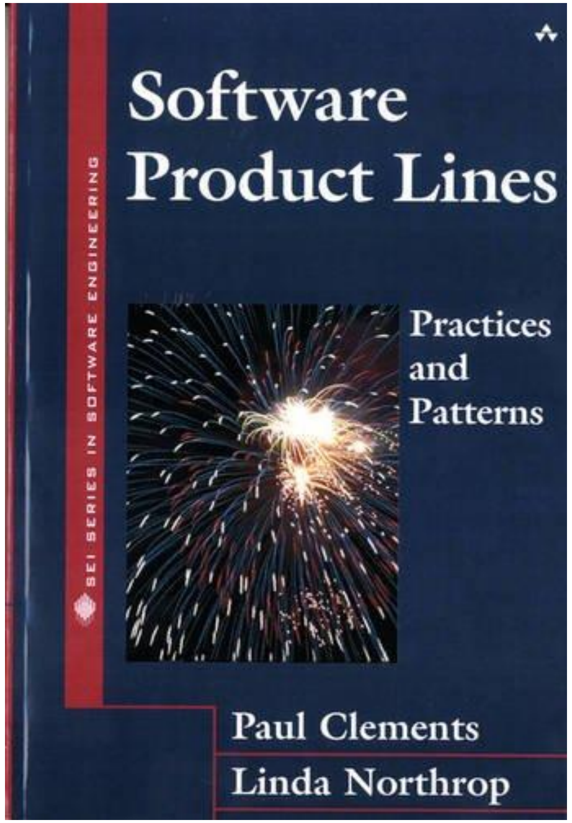
✓ The product line approach promises significant advantages in development cost, time-to-market and quality of the products = strong amplifier for agility)

✓ Product line engineering requires specific organizational structures and a new software development process

✓ The product line approach is a mature, proven technology which leads to considerable competitive advantages for companies

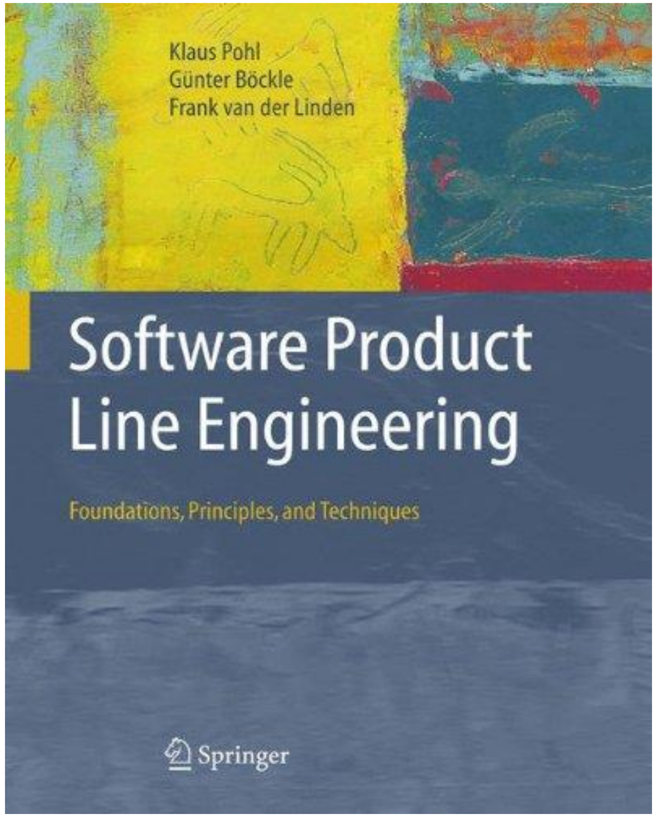
Future-Proof Software-Systems [Part 3C]

Textbook



Paul Clements, Linda Northrop:
Software Product Lines – *Practices and Patterns*
Addison Wesley Inc., USA, 2015. ISBN 978-0-134-42408-8

Textbook


Klaus Pohl
Günter Böckle
Frank van der Linden
Software Product Line Engineering
Foundations, Principles, and Techniques
Springer

Klaus Pohl, Gunter Bockle, Frank J. Linden:
Software Product Line Engineering – *Foundations, Principles and Techniques*
Springer-Verlag, Berlin, 2010. ISBN 978-3-642-06364-0

Recommendations

Software Product Lines

1. Product lines make use of planned, massive reuse
2. The product line approach promises significant advantages in development cost, time-to-market and quality of the products = strong amplifier for agility)
3. Product line engineering requires specific organizational structures and a new software development process
4. The product line approach is a mature, proven technology which leads to considerable competitive advantages for companies

Part 3 D