1. – 4. September 2014
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Built To Last

Nachhaltige Software-Entwicklung

Frank Pientka

Materna GmbH

Wer ist Frank Pientka?

Dipl.-Informatiker (TH Karlsruhe)

Software Architect in Dortmund

iSAQB-Gründungsmitglied

heise.de/developer/Federlesen-Kolumne

Über 20 Jahre IT-Erfahrung
Veröffentlichungen und Vorträge zu:

Datenbanken, Applikations- und Portalserver

Built to last: Inhalt

- Warum Nachhaltigkeit?
- Was ist Nachhaltigkeit?
- Kriterien und Prinzipien für nachhaltige Software
- Der Weg zu einer API-Ökonomie
- API-Design-Beispiele
- Was muss sich ändern?
- Fazit


Investitionen
Zukunft
Lösungen
Ideen
Nutzung
Gesellschaft
Natur
Politik
Umwelt
Nachhaltigkeit
erneuerbar
Forschung
nachhaltig
Förderung
Energie
Technologie
Wirtschaft
Entwicklung

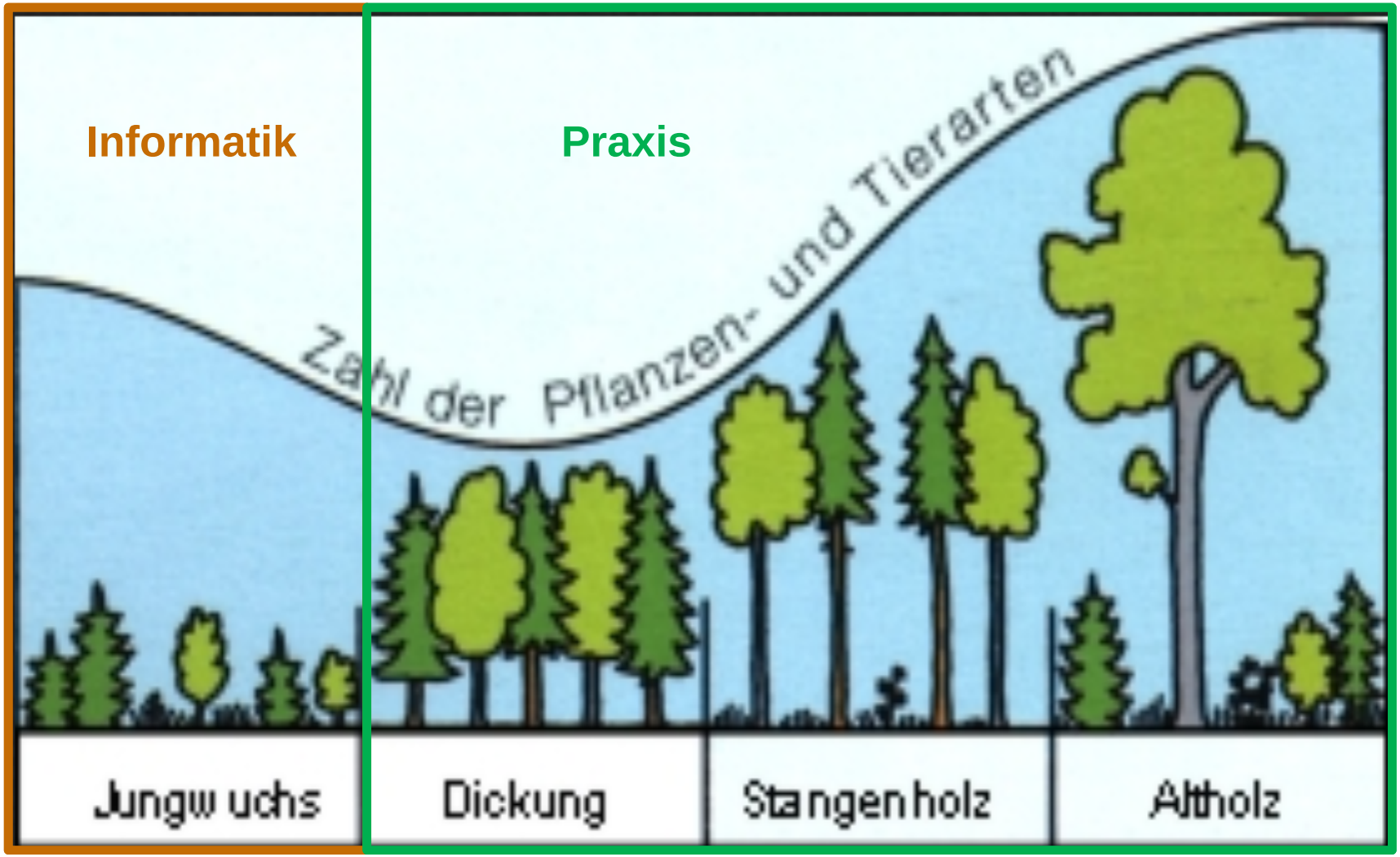
Seit 300 Jahren: Was ist Nachhaltigkeit?

Hans Carl von Carlowitz (1645 – 1714)
Oberberghauptmann aus Freiberg (Sachsen)
Begründer des Prinzips der Nachhaltigkeit

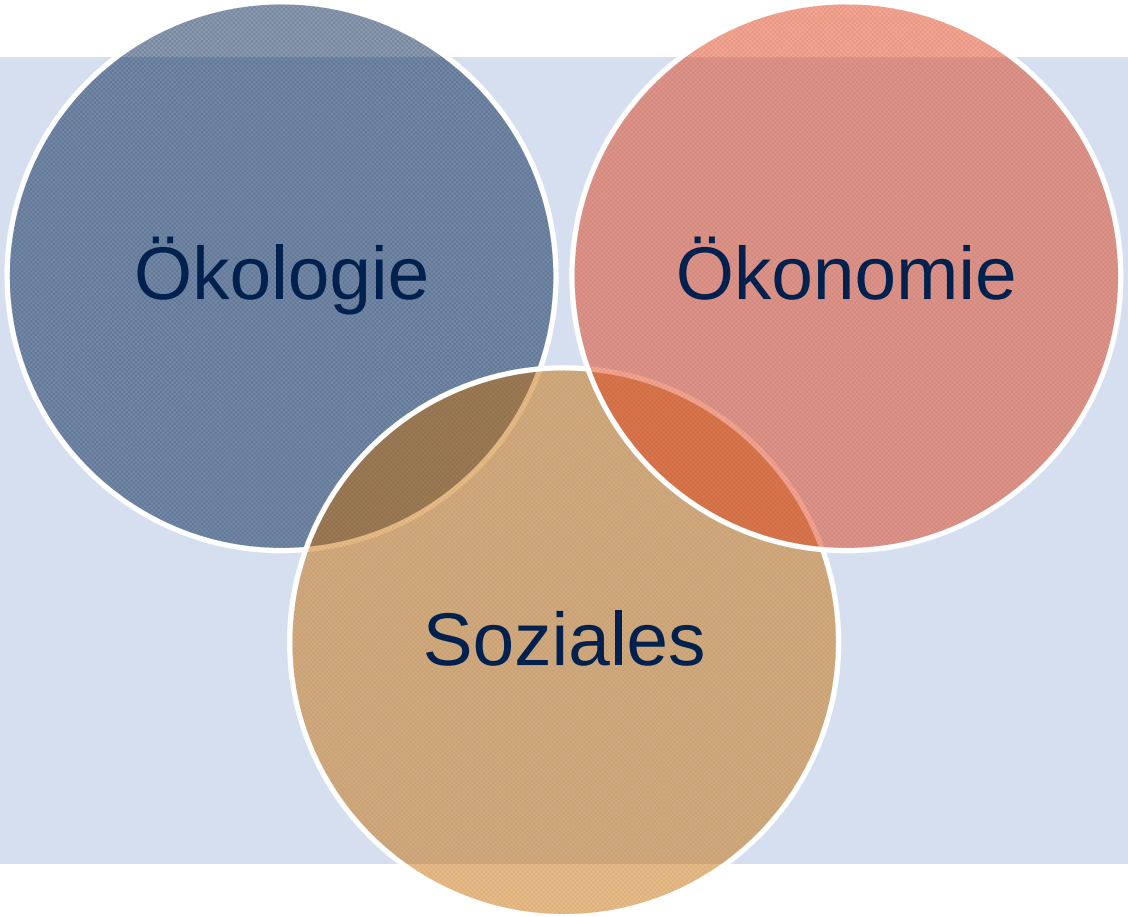
Holzmangel in ganz Europa als das Problem des 17. JH
Buch "**Sylvicultura oeconomica oder Anweisung zur wilden Baum-Zucht**“
Anleitung zu kontinuierlicher, beständiger und nachhaltender
Nutzung des Holzes

300 JAHRE
NACHHALTIGKEIT
IN SACHSEN

Schema der nachhaltigen Forstwirtschaft



Nachhaltigkeit



Gebaut für die Ewigkeit: Was ist eine nachhaltige Architektur?

Pick Your Battles, Zef Hemel, Cloud9, December 13, 2012

Altbewährt und trotzdem gut

…the most boring technology you can find in use for years and years…

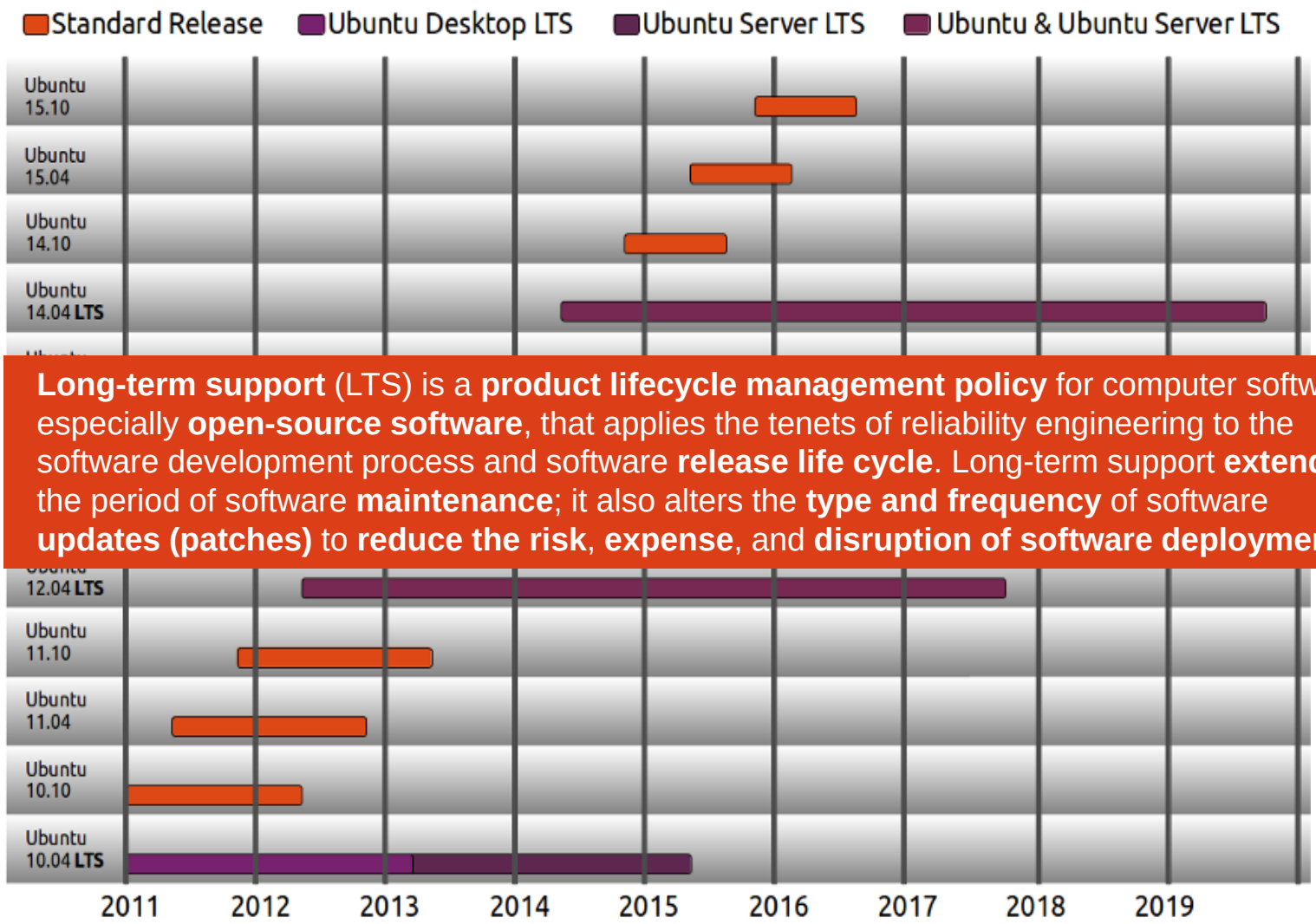
In Defense of Boring, Grady Booch, May/June 2013, IEEE Software

Digitale Nachhaltigkeit Aspekte

*“development that meets the needs of the **present***
*without compromising the ability of **future** generations*
*to meet their own **needs**”*
*(*Brundtland UN-Report 1987*)*

Der digitale Graben - Digitale Nachhaltigkeit
*"Open Source und **informationelle Nachhaltigkeit**"*
(Thorsten Busch in Open Source Jahrbuch 2008)

http://en.wikipedia.org/wiki/Long-term_support



Long-term support (LTS) is a **product lifecycle management policy** for computer software, especially **open-source software**, that applies the tenets of reliability engineering to the software development process and software **release life cycle**. Long-term support **extends** the period of software **maintenance**; it also alters the **type and frequency** of software **updates (patches)** to **reduce the risk**, **expense**, and **disruption of software deployment**

http://wiki.eclipse.org/LTS/LTS_Ready

A **build** that builds on Eclipse Foundation hardware and

- Can be cloned/checked out with one step
- Is **documented**
- Is **version controlled**
- Is **automated**
- Is deterministic given the same source code and third party libraries
- Is easily reproducible on **suitably-configured systems**
- Can refer to compilers and other tools **from a configurable location**
- Capable of building without an active Internet connection
- Capable of pulling dependencies from a **known controlled source**

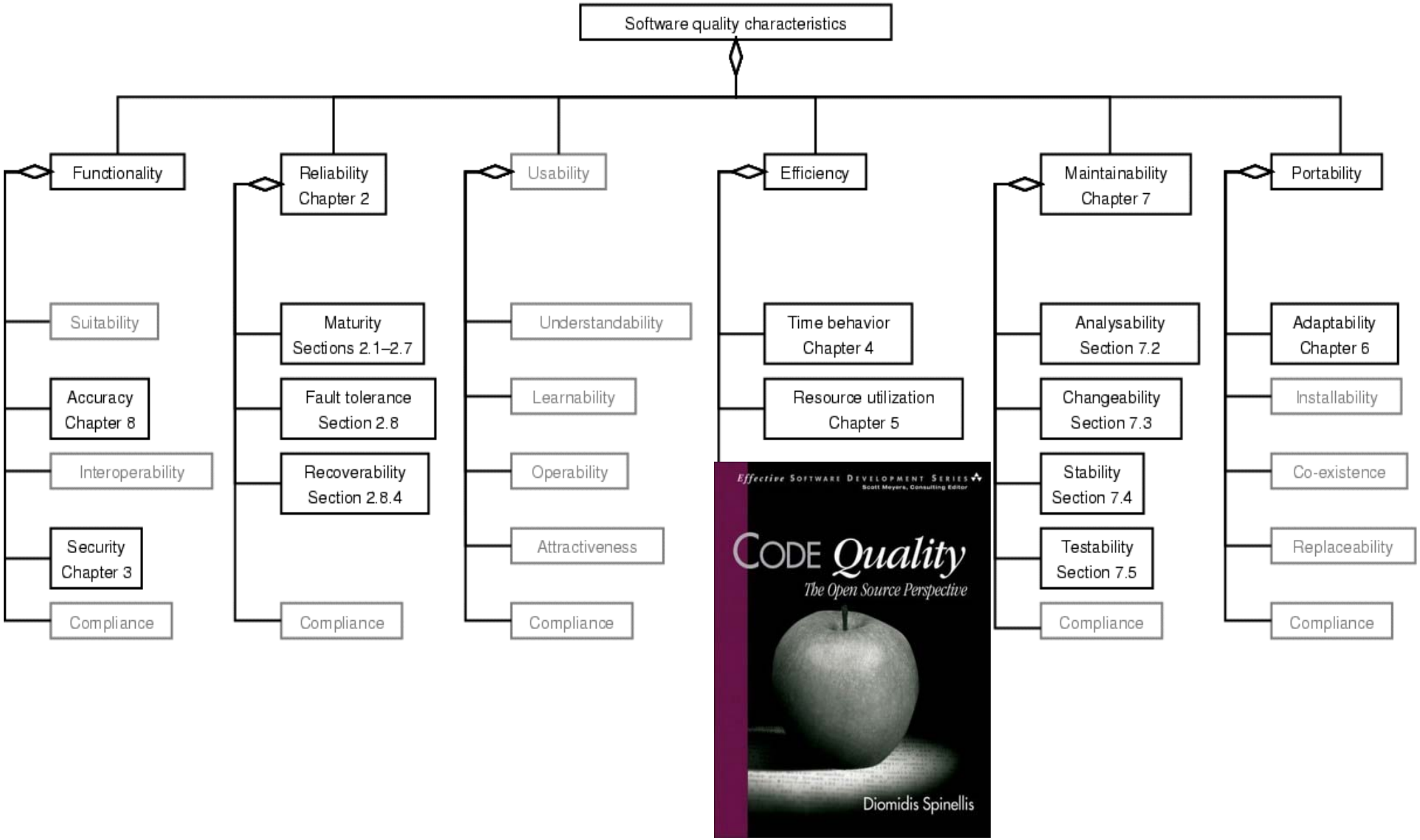
Adheres to Eclipse IP **policies**

Bug Tracking: no code change can be released without proper bugzilla entry

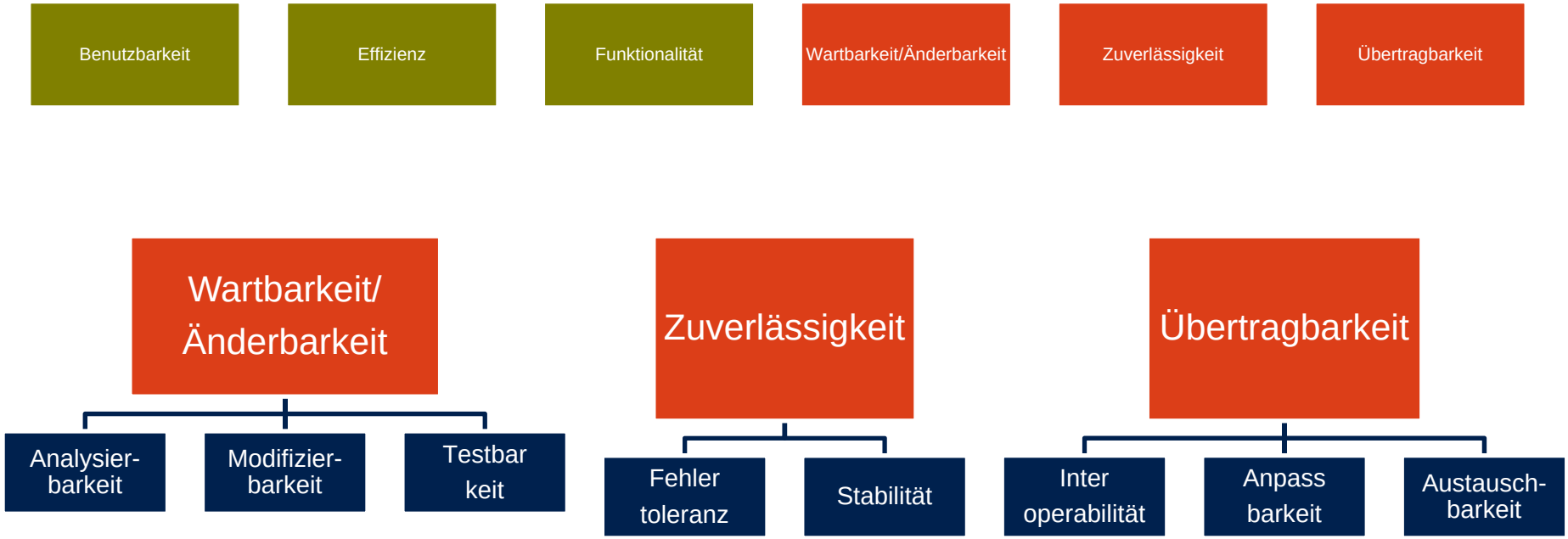
Release Management: part of the annual simultaneous release

Supply & Demand: At least two LTS IWG member companies offering support

Code Quality: The Open Source Perspective, Diomidis Spinellis



ISO/IEC 9126 für Softwarequalität

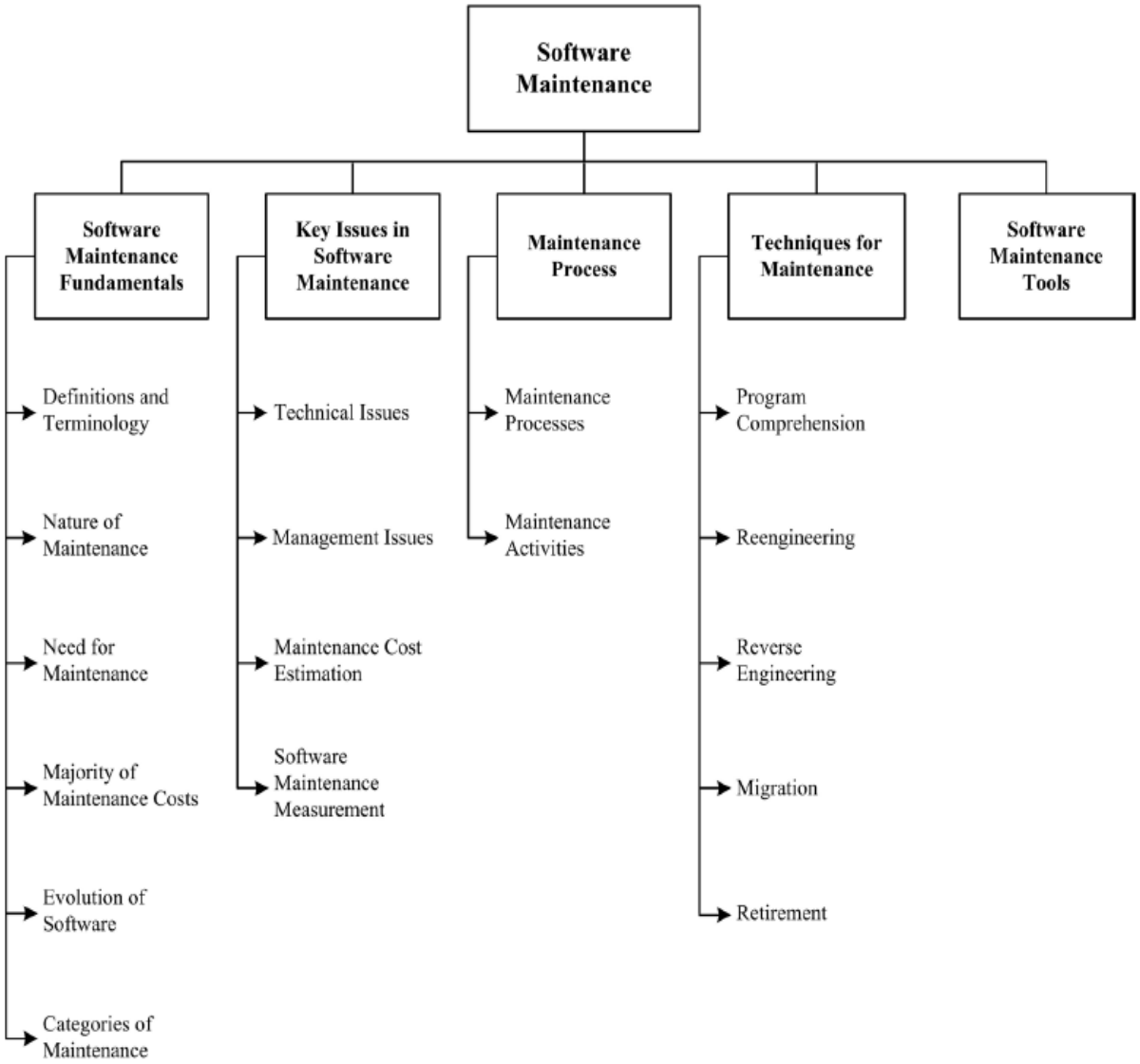


Softwarequalität

http://www.computer.org/portal/web/swebok 3.0 (2014)

1. Software Requirements
2. Software Design
3. Software Construction
4. Software Testing
5. **Software Maintenance**
6. Software Configuration Management
7. Software Engineering Management
8. **Software Engineering Process**
9. Software Engineering Models and Methods
10. **Software Quality**
11. Software Engineering Professional Practice
12. Software Engineering Economics

5. Software Maintenance SWEBOK® Guide V3.0

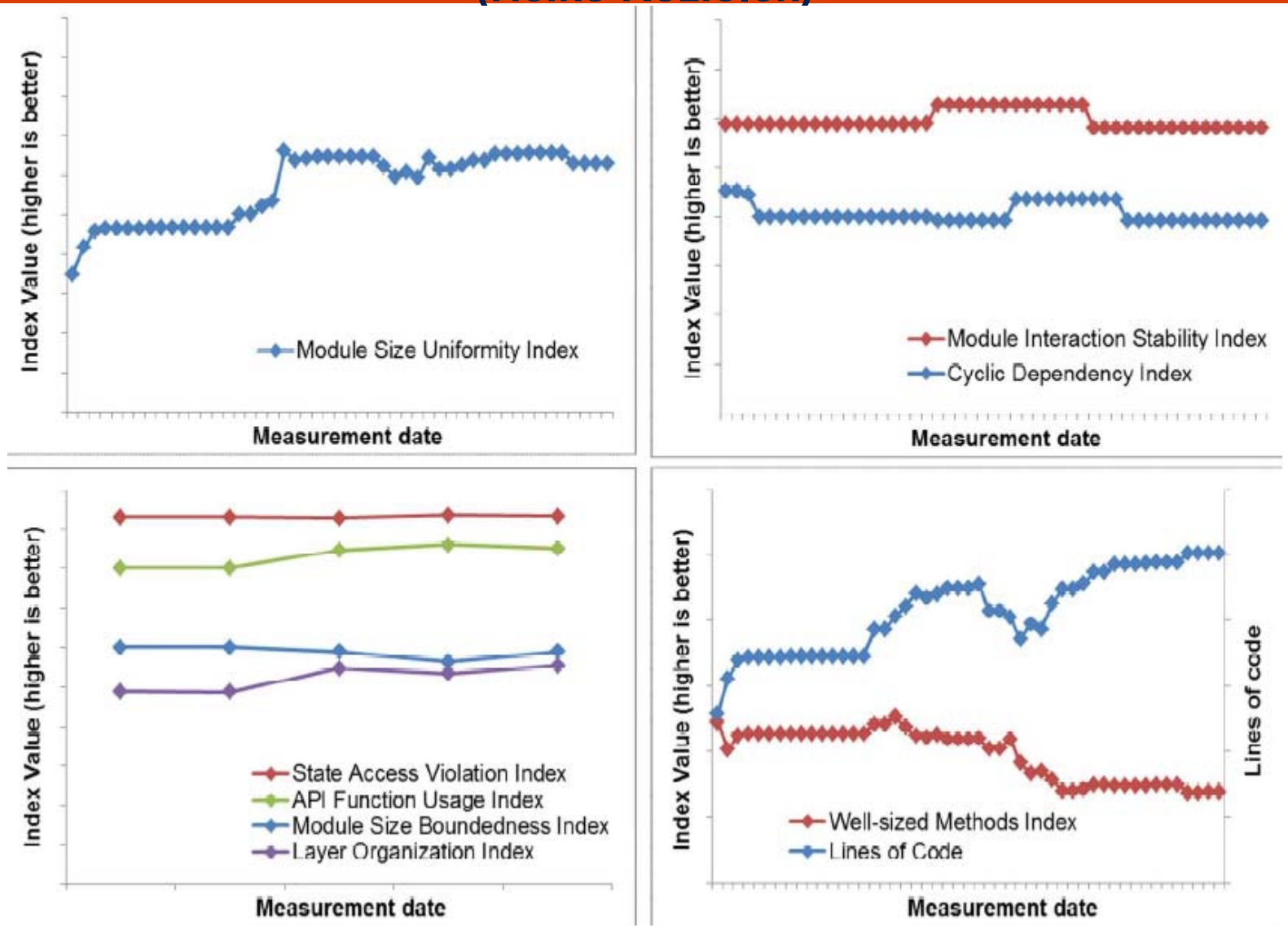


Measuring Architecture Sustainability (Heiko Koziolek)

Evolution Scenario	Assessment 2011	Assessment 2012	Assessment 2013
Change of operating system implementation	Highly likely, but limited impact (preparation done)	Change occurred, limited impact	No further changes expected in the near future
Increase of system workload	Likely, high impact	Internal testing done, impact on architecture lower than expected	Still likely, but no additional measures implemented
Integration of a new hardware component	Likely, limited impact	Change occurred, project started	Under development
Change of the deployment platform	Scenario recognized, but no immediate preparation desired	Scenario refined, likelihood increased	Large research project started
Change of a client-facing plug-in interface	Unlikely, but impact would be high	Did not occur, still possible	Did not occur, still possible
Change of the third-party middleware implementation	Medium likelihood, limited impact	Did not occur, still possible	Did not occur, still possible
GUI framework change	Unlikely, but impact would be high	Research project executed	Planned as an extension

Änderungs-/Wachstumsszenarien entwerfen → Entscheidungen validieren, Alternativen betrachten

Überwachen wichtiger Metriken für nachhaltige Architektur (Heiko Koziolek)



Wenige, einfache Metriken

Cyclic Dependency Index (to improve modifiability): percentage of modules with namespace dependency cycles

Well-sized Methods Index (to improve modifiability): percentage of well-sized (<30 LOC) methods in the code base

Distance from Main Sequence (to improve modifiability): fraction of modules that are either concrete and stable (that is, difficult to maintain) or abstract and unstable (that is, difficult to use)

Module Interaction Stability Index (to improve layering): percentage of modules that depend on modules with higher instability

Layer Organization Index (to improve layering): extent to which module dependencies skip adjacent layers

Normalized Testability Index (to make testing more efficient): extent to which modules are independently testable

Module Interaction Index (to improve API usage): effectiveness of how a module's API functions are used

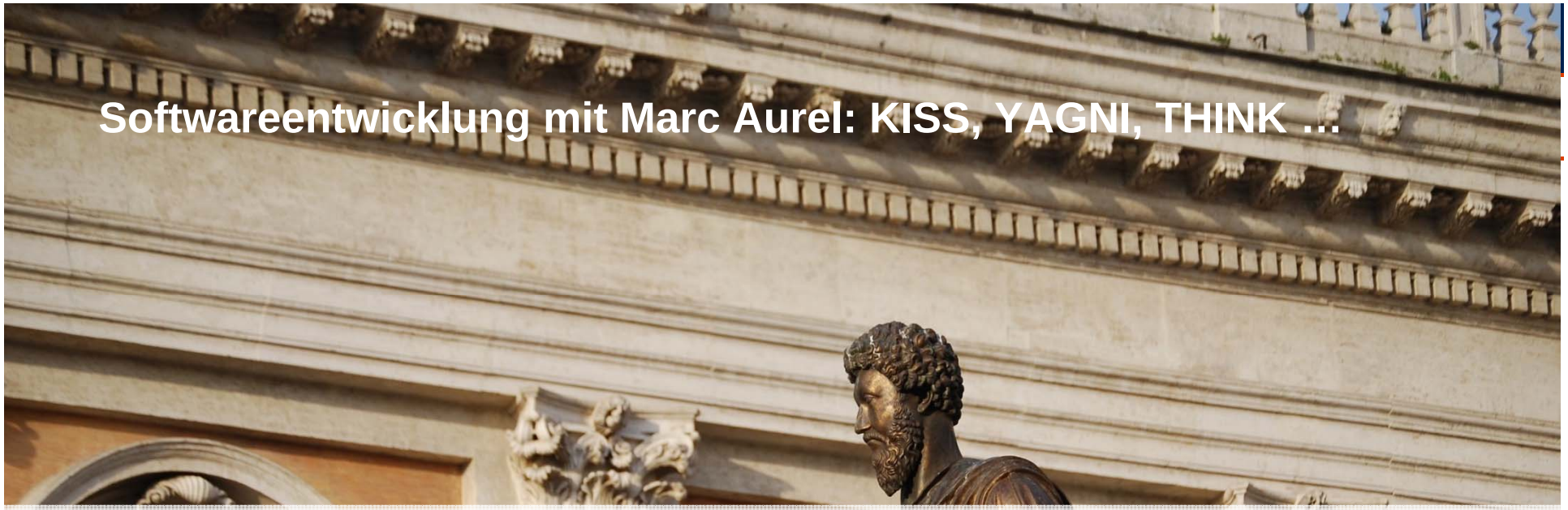
API Function Usage Index (to improve API usage): averaged percentage of the ratio of non-API functions to API functions

Module Size Uniformity Index (to limit the sizes of modules): extent of heterogeneously sized modules

Module Size Boundedness Index (to limit the sizes of modules): extent of module sizes differing from a maximum size threshold

State Access Violation Index (to reduce internal variable usage): extent to which state variables are accessed directly across module boundaries

Association-Induced Coupling (to reduce internal variable usage): extent of coupling between modules due to class association



Softwareentwicklung mit Marc Aurel: KISS, YAGNI, THINK …

„Wir sind **zur Zusammenarbeit geboren**.“

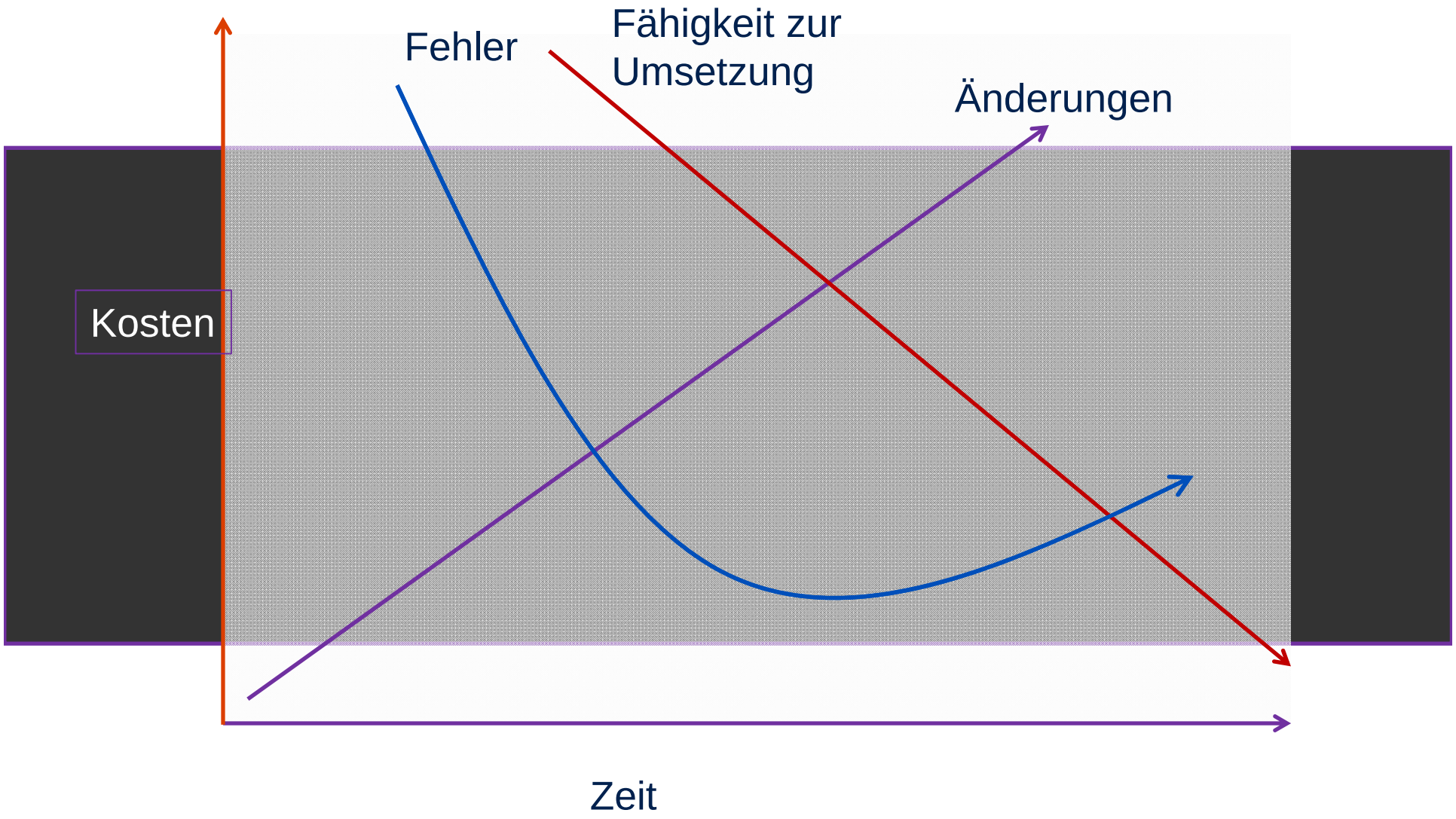
„Beachte immer, dass **nichts bleibt, wie es ist** und denke daran, dass die Natur immer wieder ihre Formen wechselt.“

„Wir müssen **von ganzem Herzen** alles, **was uns trifft, willkommen heißen**, wir dürfen auch innerlich nicht murren, ja uns nicht einmal wundern.“

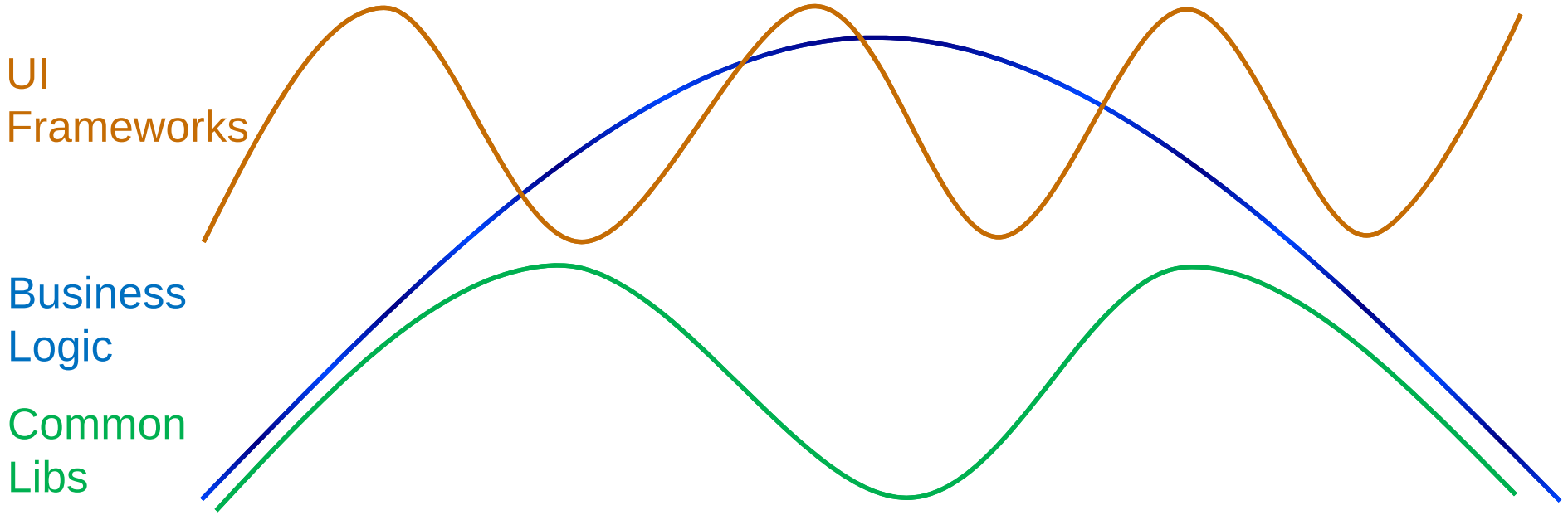
„Denke lieber an das, **was du hast**, als an das, **was dir fehlt**!

Suche von den Dingen, die du hast, **die besten aus** und bedenke dann, wie eifrig du nach ihnen gesucht haben würdest, wenn du sie **nicht hättest**.“

Kosten von Änderungen und Fehlern (Kevin Tate 2006)



Alles ändert sich: Lebenszyklus UI & Anwendung



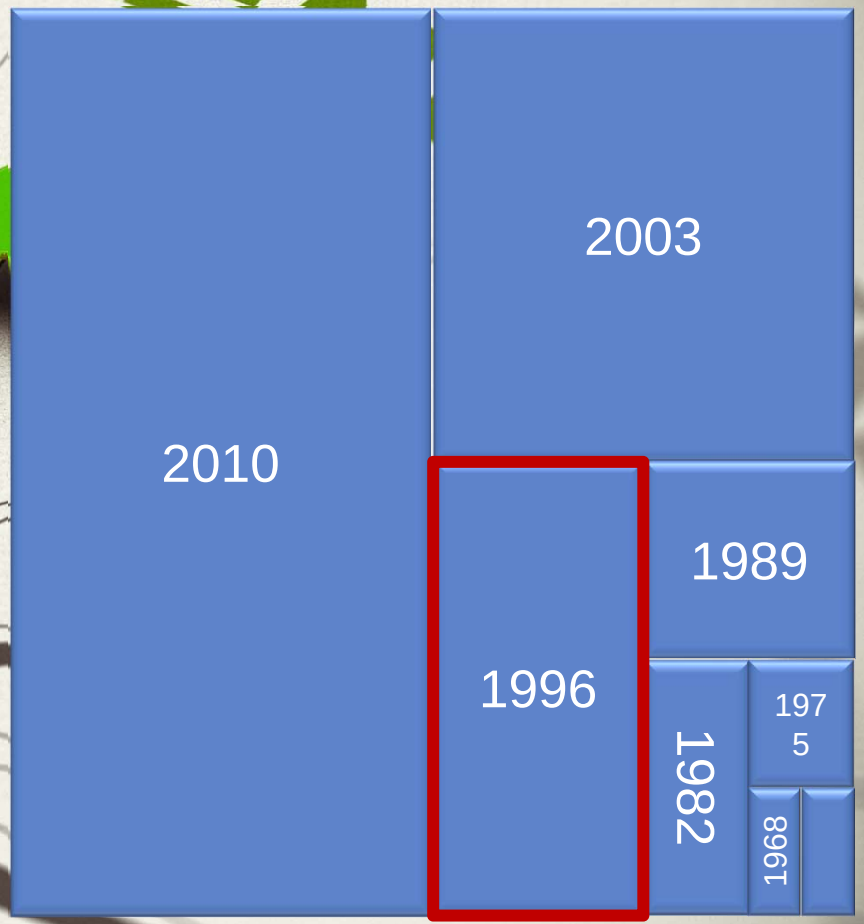
Änderungsgeschwindigkeit

- Endgeräte: PC, Notebook → Smartphone, Email, SMS, Messenger …
- UI: Terminal, Desktop, Web, mobile Apps, multi devices
- Server: HW, Virtualisierung, Cloud, XaaS

- Wie wirkt sich das auf Entwicklungsumgebung, - Prozess aus?
- Wie wirkt sich das auf die Arbeitsergebnisse, Produktivität aus?
- Nicht nur Optimierung Betrieb, sondern auch Entwicklung

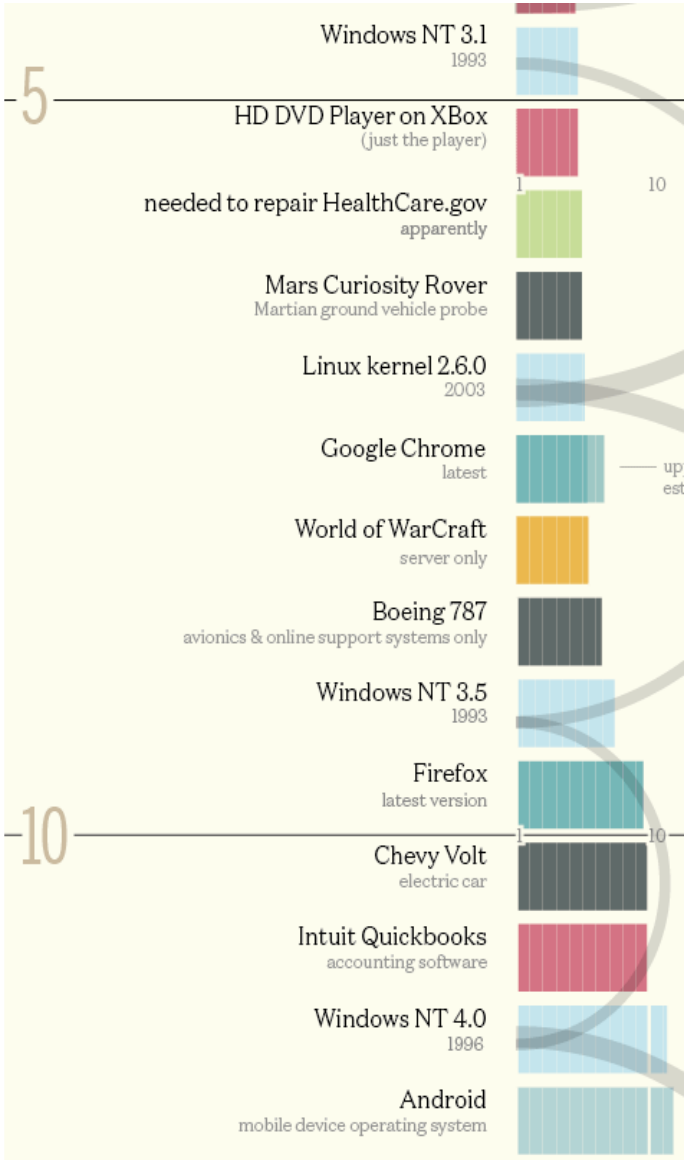
Wie entwickelt sich Software über die Zeit?

Verdoppelung alle 7 Jahre
Wie viele Codezeilen 2017?
Wie wird Komplexität beherrschbar?

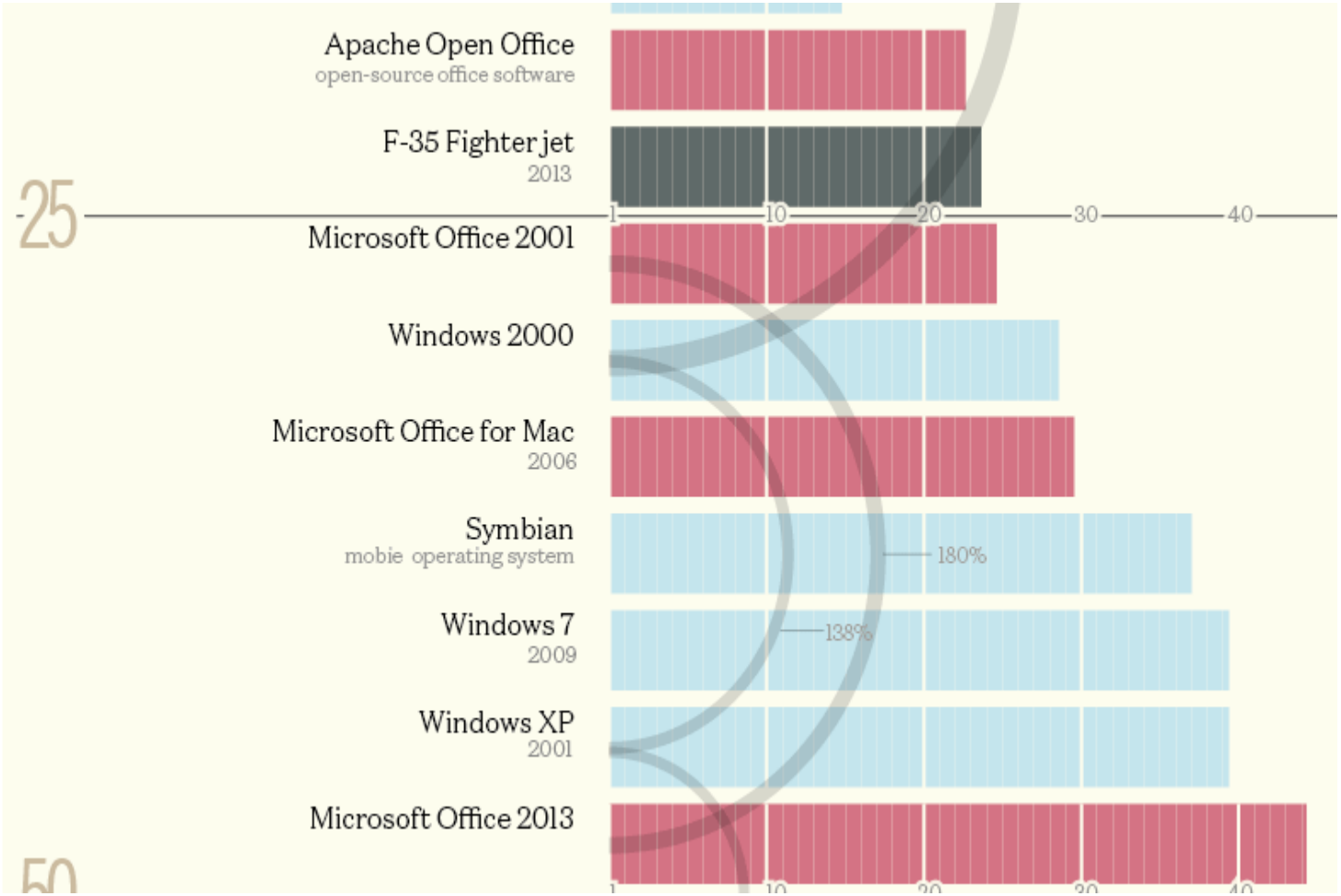


http://users.jyu.fi/~koskinen/smcosts.htm

5 – 10 Millionen Codezeilen



25 Millionen Codezeilen: Office&Windows

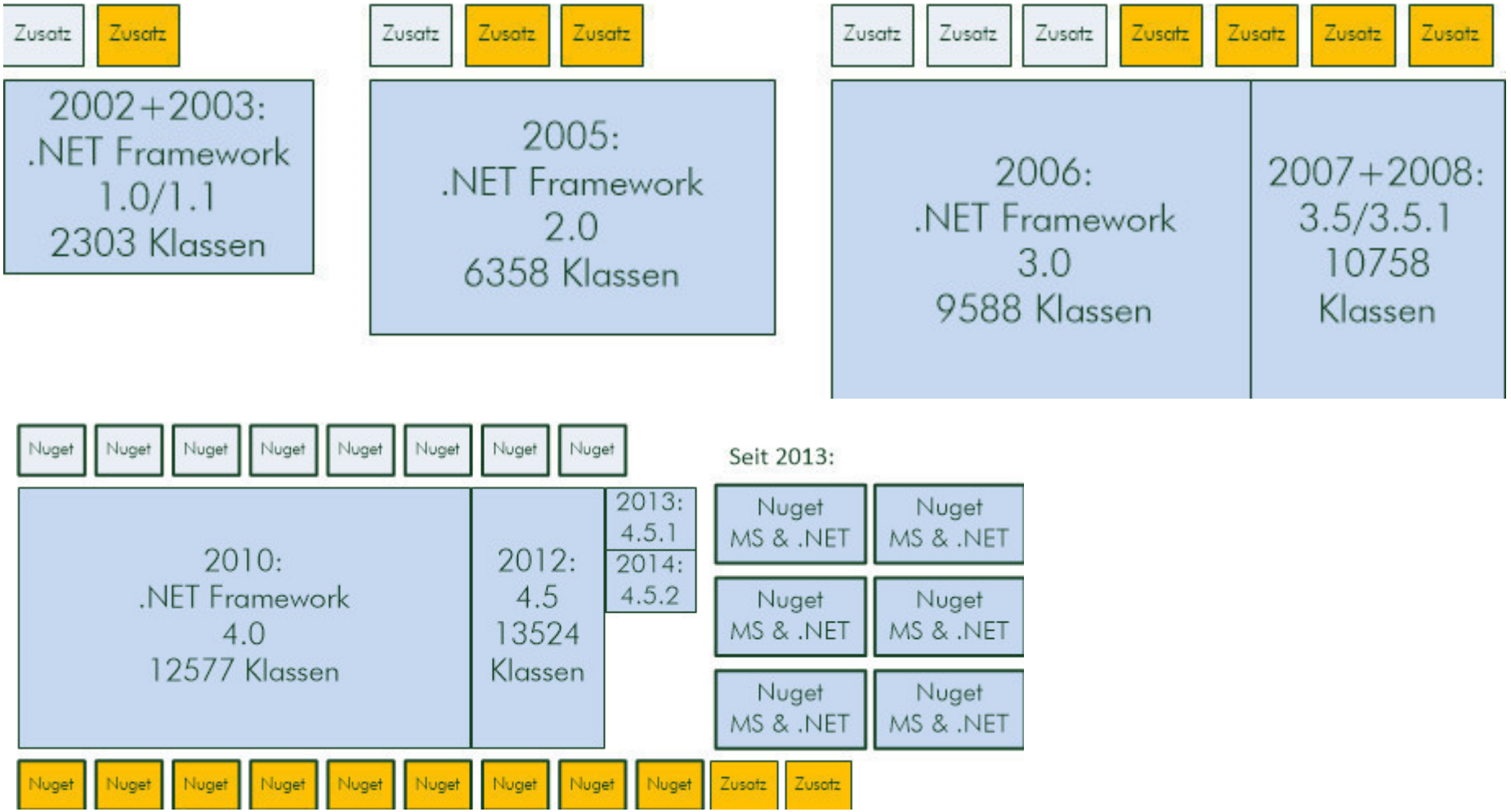


Too big to scale !

Entwicklung der Windows-Größe – Conways Law

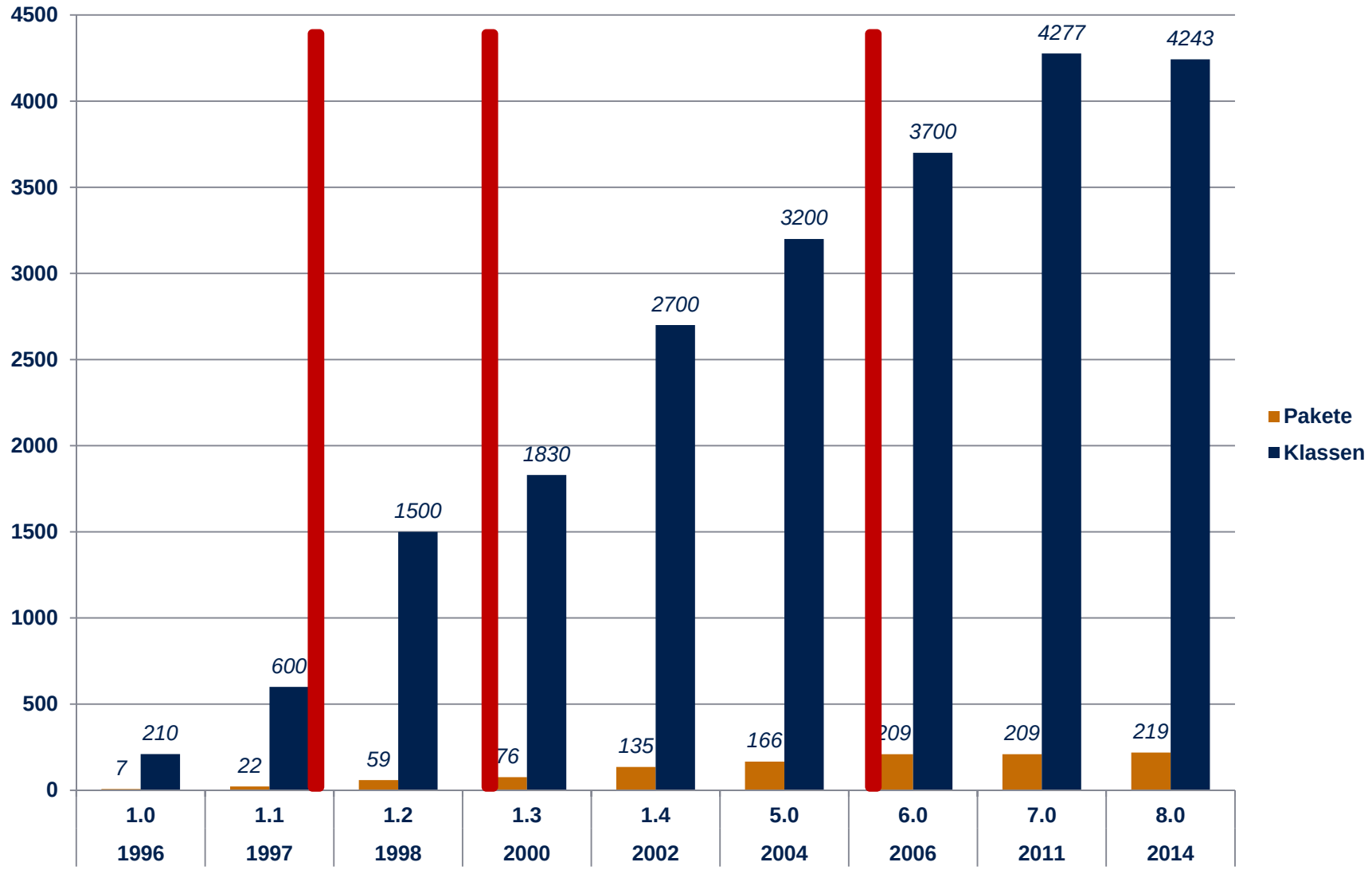
Windows-Version	SLOC (in Mio.)
Windows 3.1 (1992)	3
Windows NT 3.1 (1993)	**5**
Windows NT 3.5 (1994)	8
Windows NT 3.51 (1995)	10
Windows NT 4.0 (1996)	12
Windows 2000 (2000)	29
Windows XP (2001)	40
Windows Vista (2007)	**50**
Windows 7 (2009)	40

.NET-Klassenanzahl über die Zeit



Bildquelle: Holger Schwichtenberg

Java-Pakete- und -Klassenanzahl über die Zeit



Modularität statt Monolithen gefragt

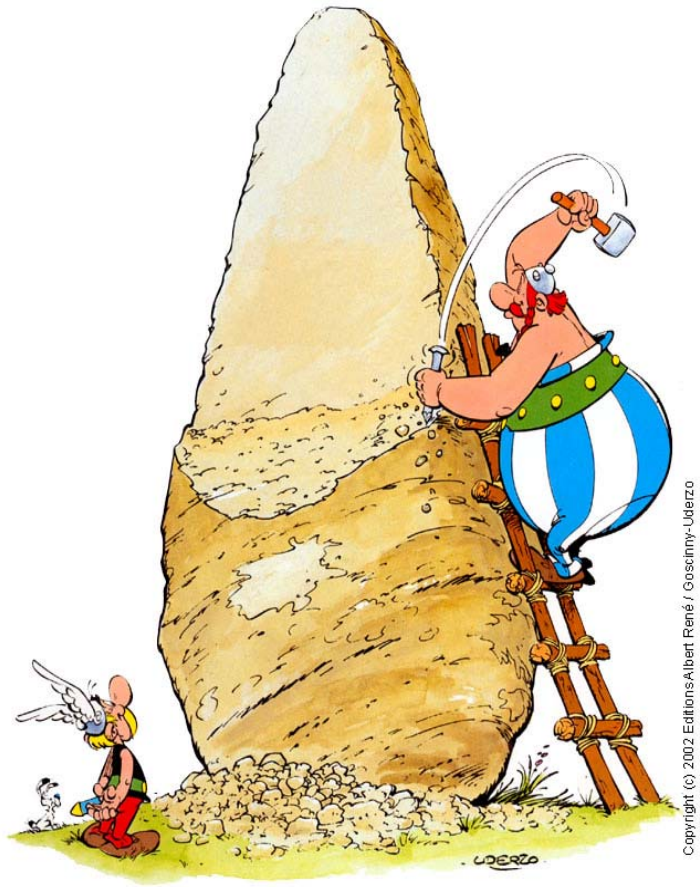
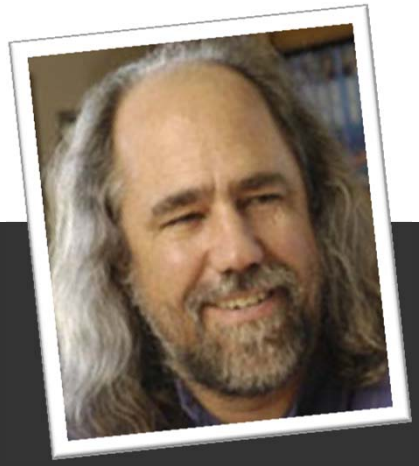


Foto: Ehapa

Software ist abstrakt



“The entire history of software engineering is one of rising levels of abstraction”

Grady Booch (2008)

“Software development has advanced in large part by increasing the granularity of the aggregations that we have to work with” Steve McConnell (2004)

Das Offen-Geschlossen Prinzip

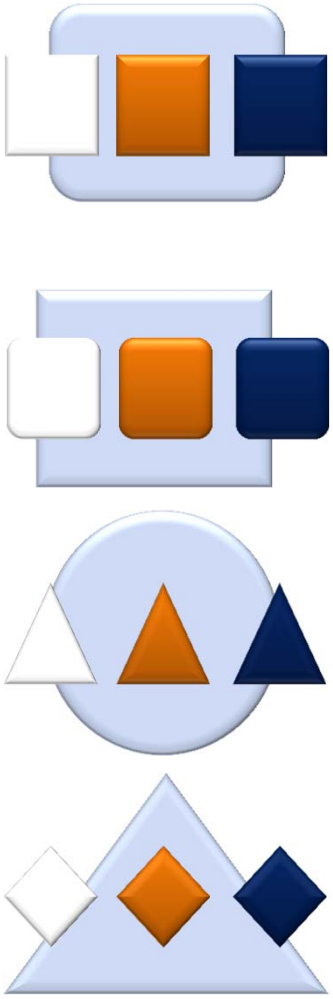
"*software entities (classes, modules, functions, etc.) should be **open for extension**, but **closed for modification***"

Bertrand Meyer (1988)

Beispiel: Design by Contract

Trennung in Schnittstelle und Implementierung

Stufen der Modularität



Modules

Packages

Classes

Functions

OSGi, Java 9?

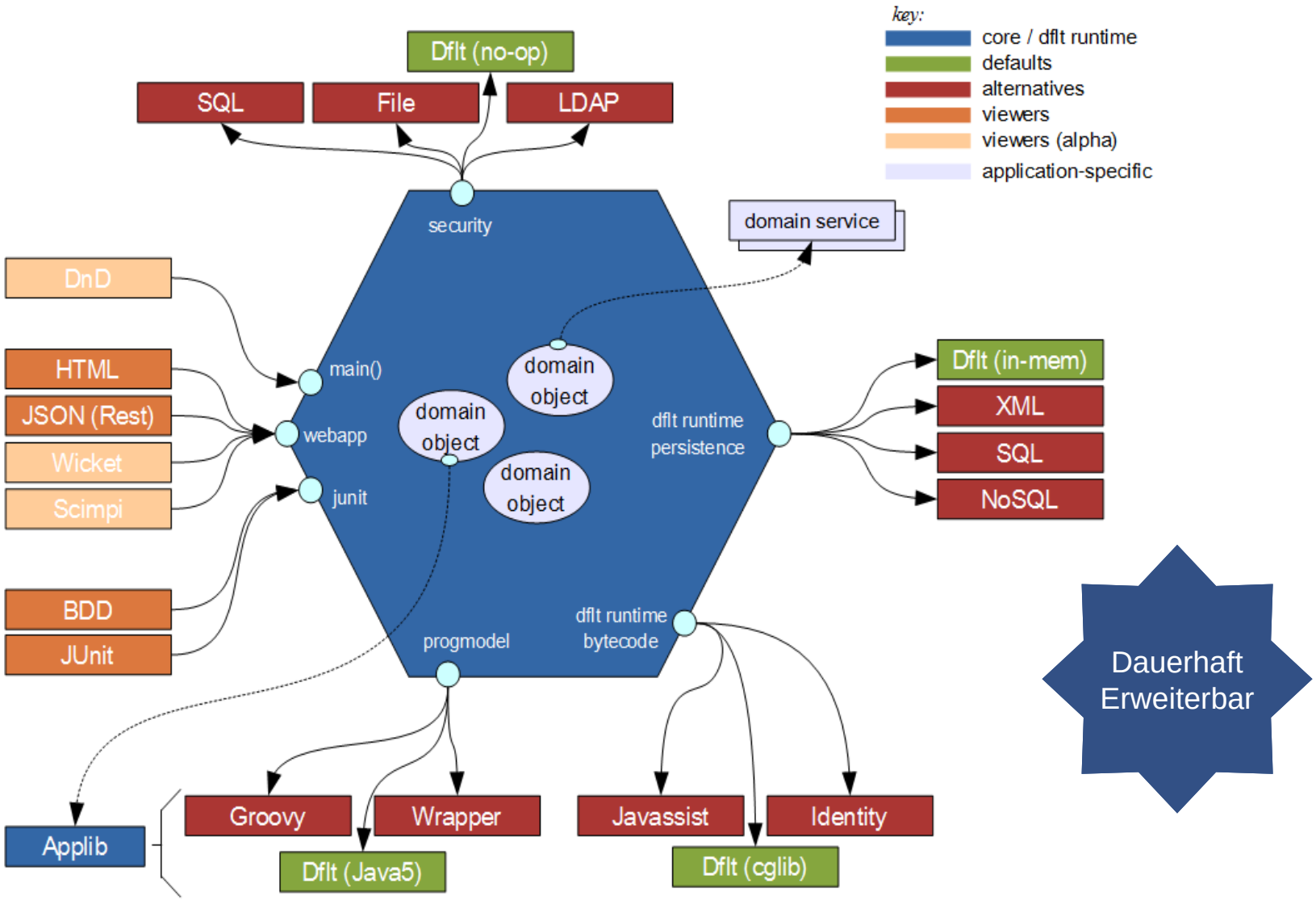
Java

Pasta-Theorie in der Softwareentwicklung: Schichtenarchitektur

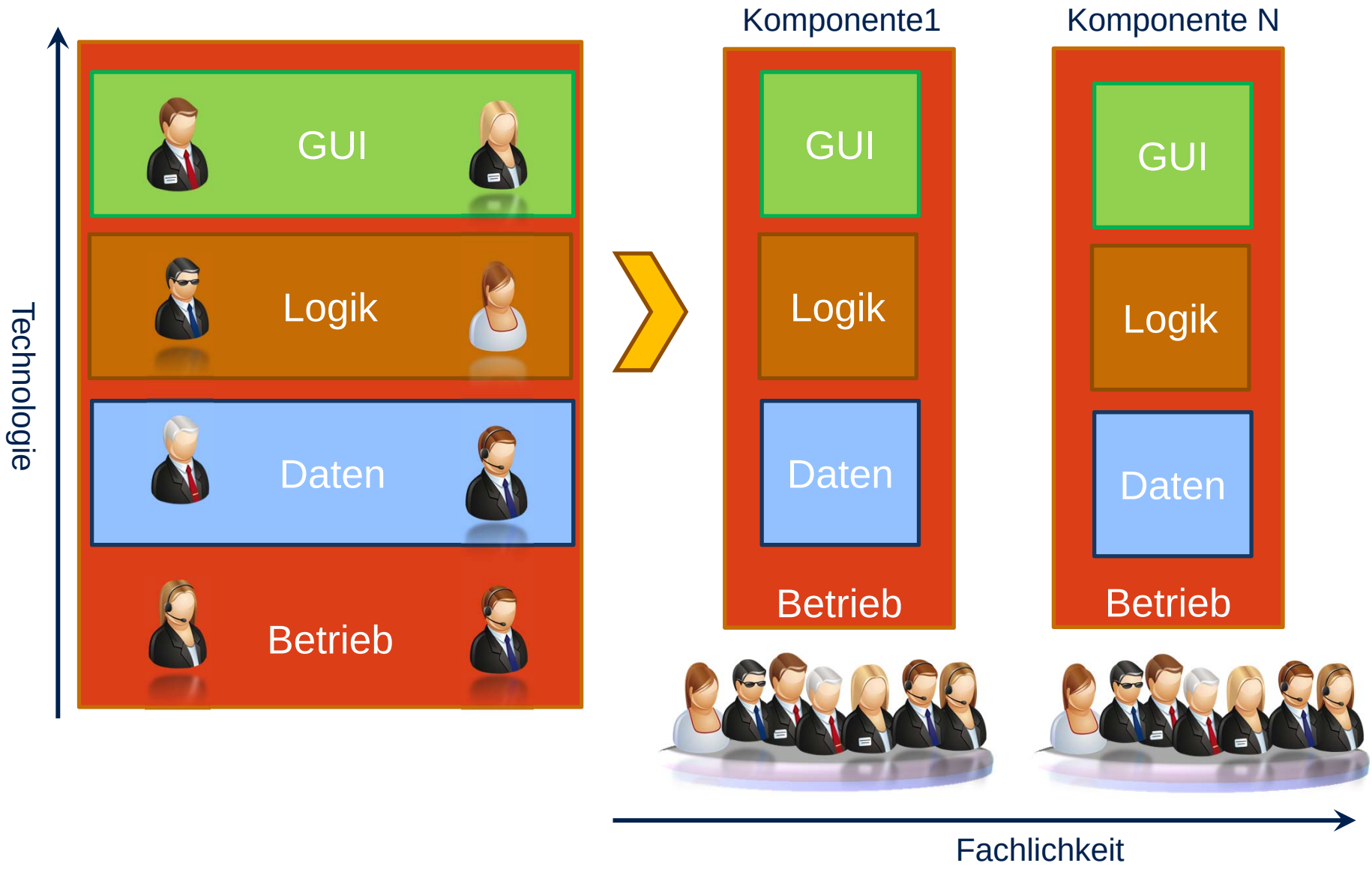
Spaghetti-Orientierte Architektur (SOA) vs. Micro Services

Nonoservice is an Anti-pattern where a service is too fine grained.
Nanoservice is a service whose overhead out-weights its utility.
(SOA Patterns, Arnon Rotem-Gal-Oz, 2012)

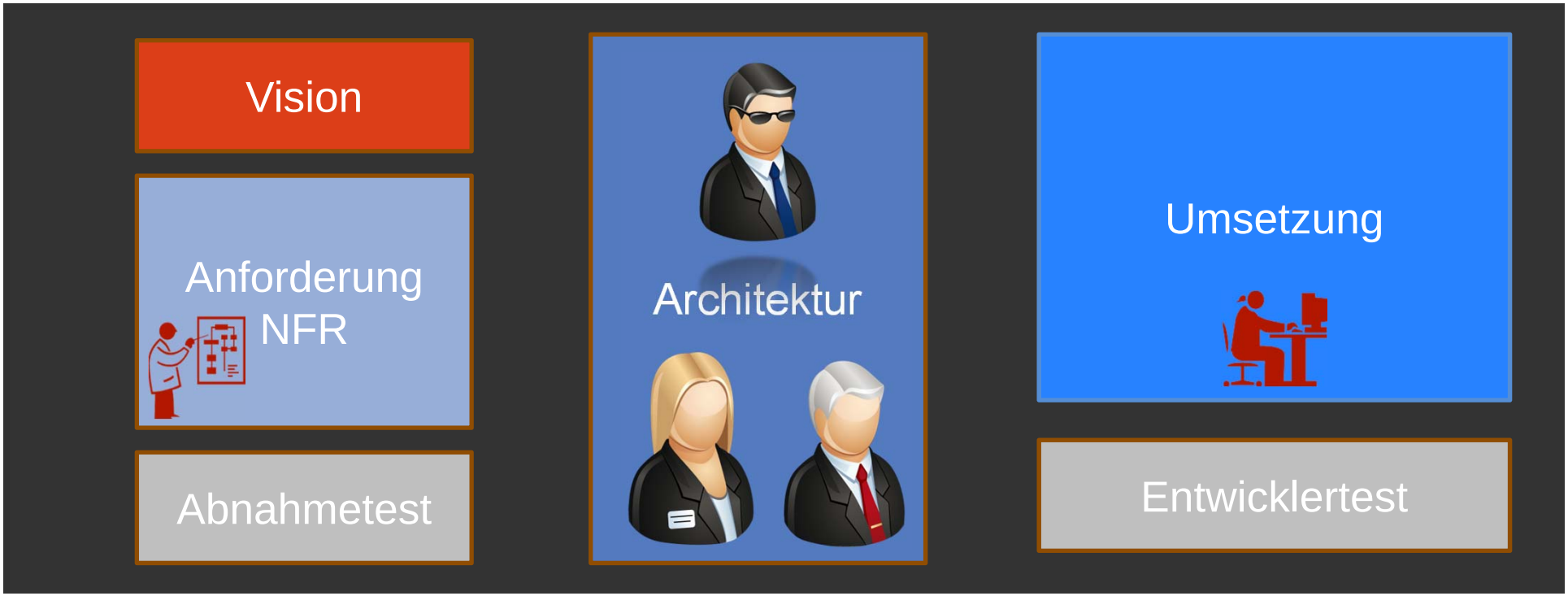
Port-Adapter-Architektur (Apache Isis)



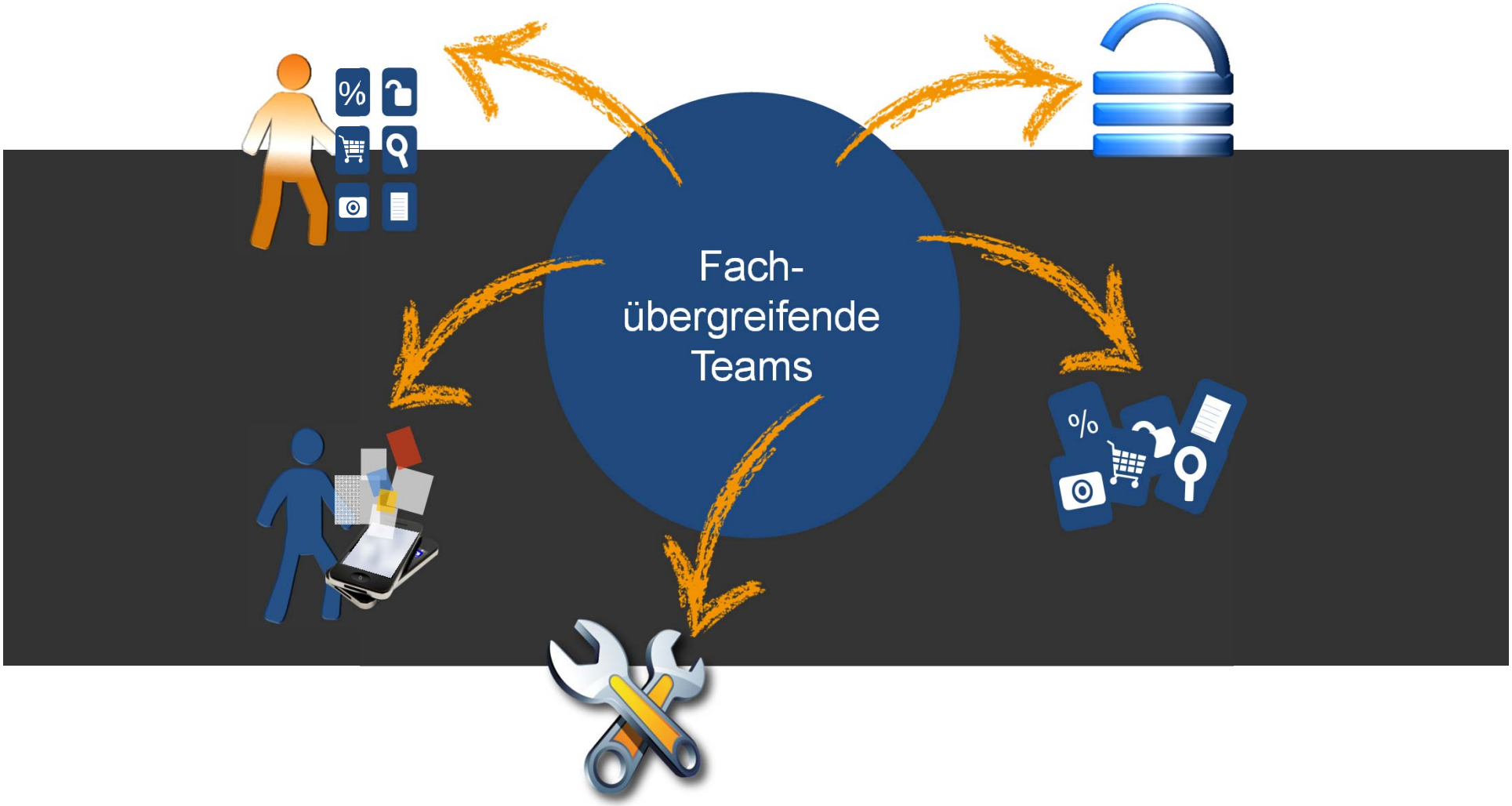
Schichten vs Vertikale



Wichtige Bausteine



FachübergreifendeTeams



FachübergreifendeTeams

Architektur-Entwickler-Tandem: wechselnde Position, immer im Tritt

Ziel & Weg im Auge

Was braucht nachhaltige Softwareentwicklung?

Erfahrung

Sprachen

Methoden

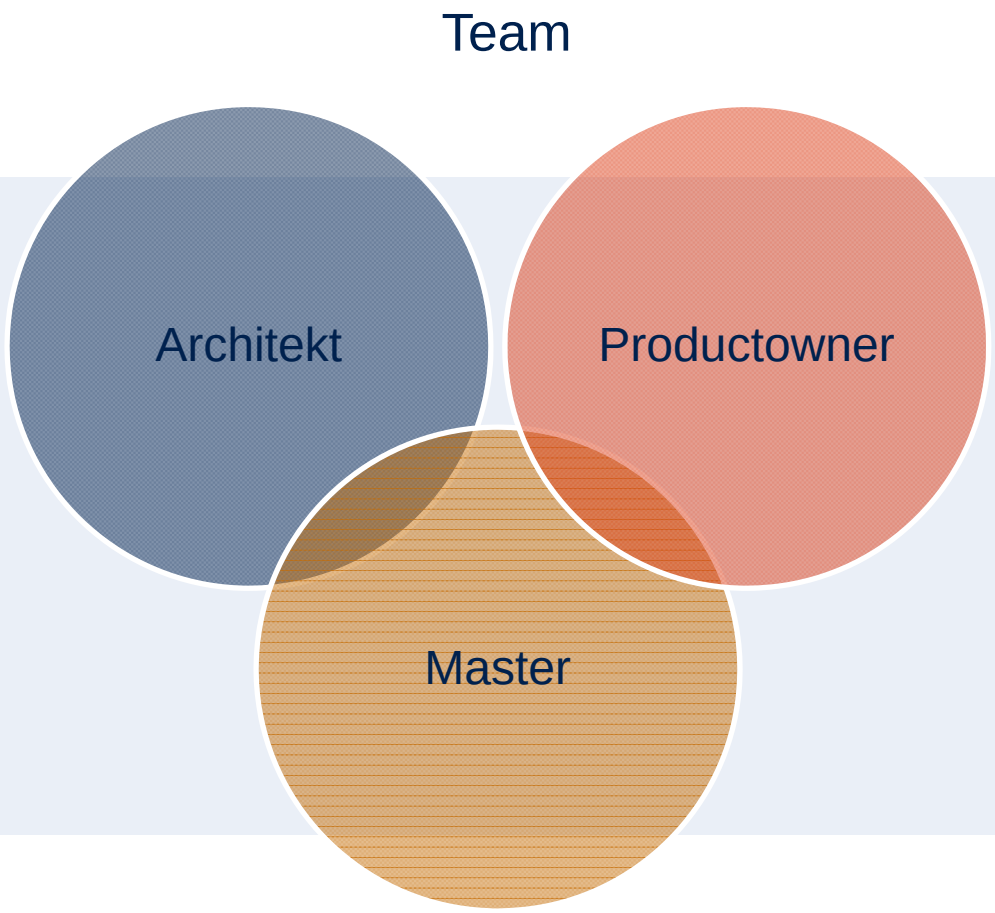
Soft Skills

Werkzeuge

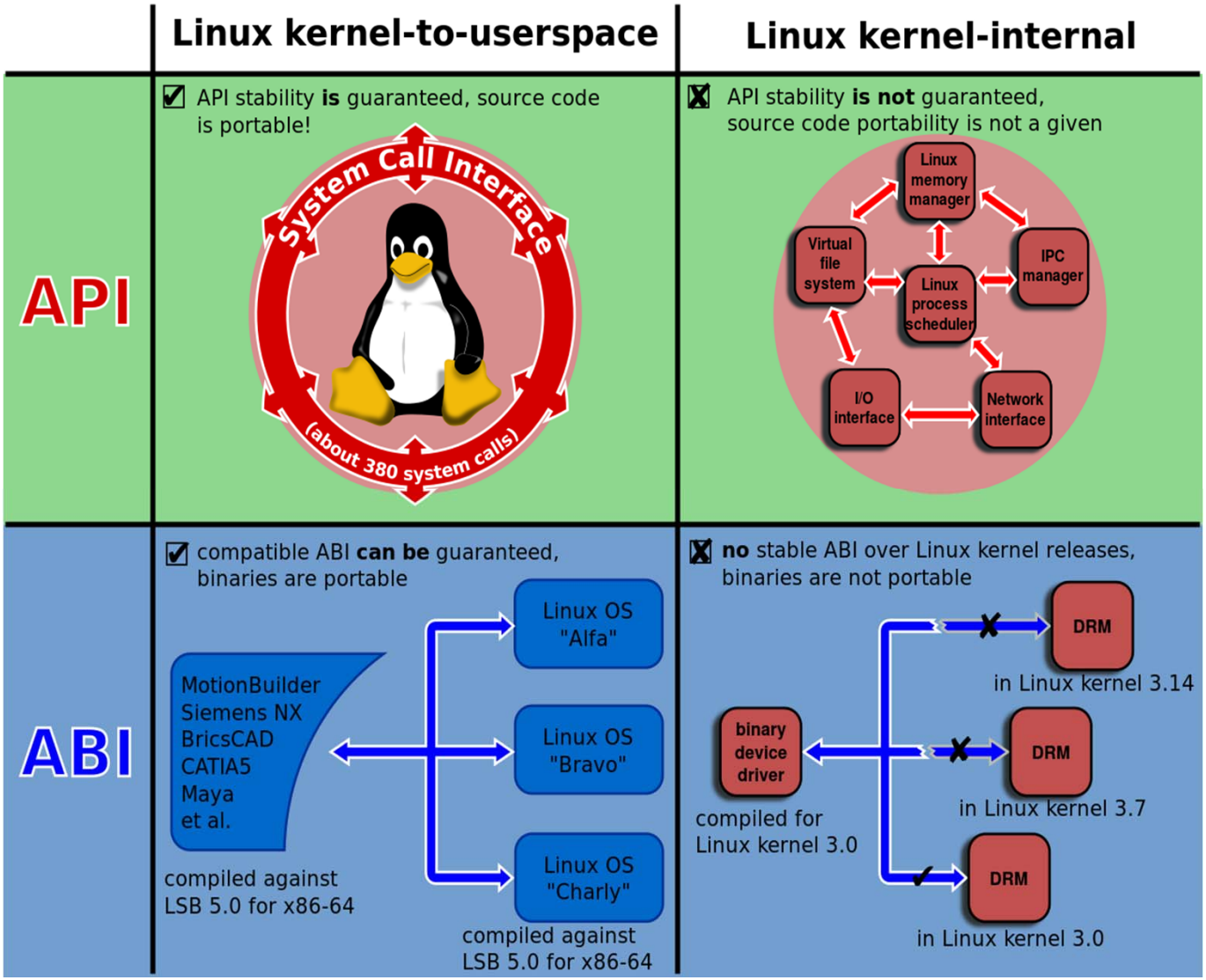
Werte

Vertrauen

Nachhaltigkeit in der Softwareentwicklung

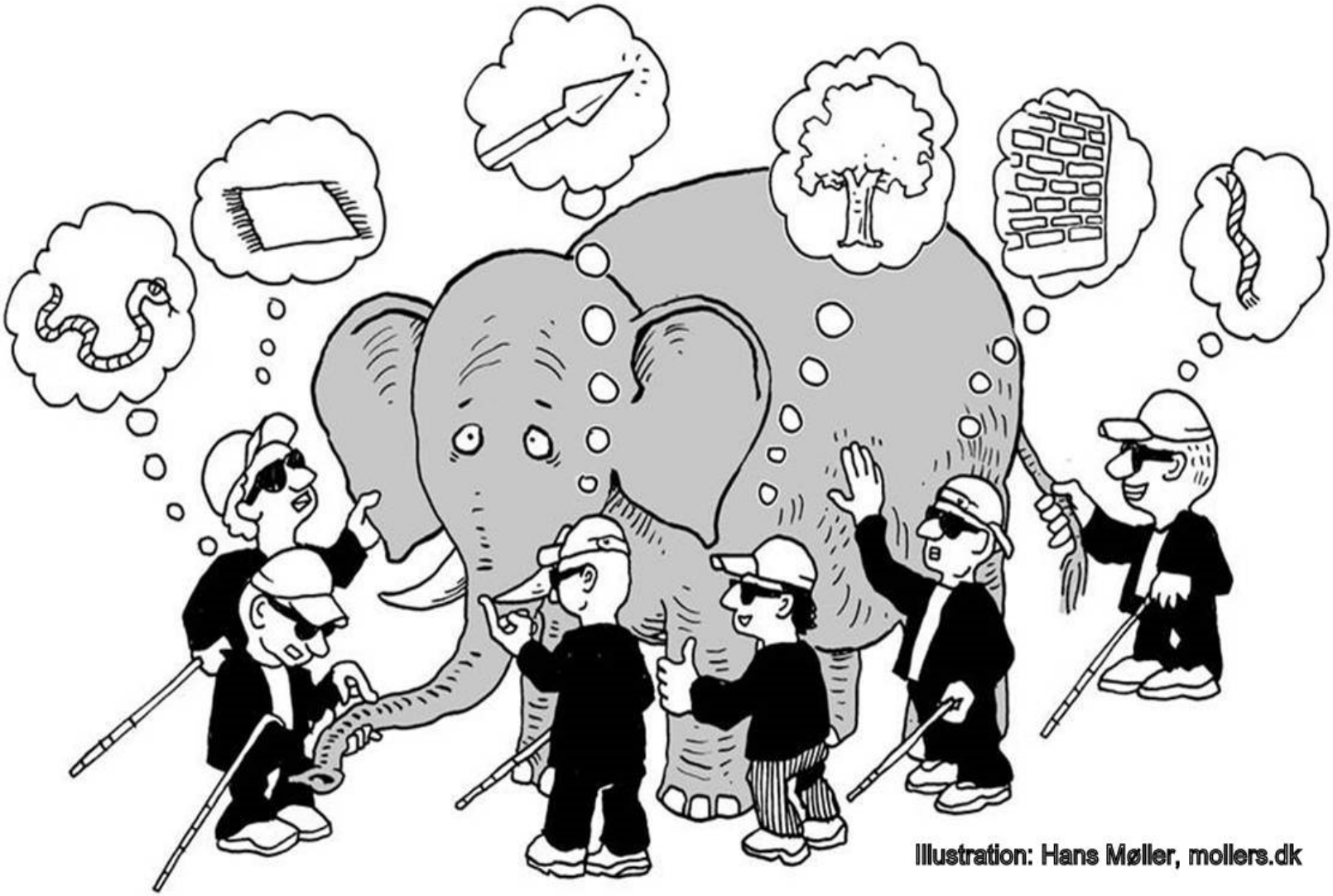


API != ABI: Kompatibilitätsunterschiede

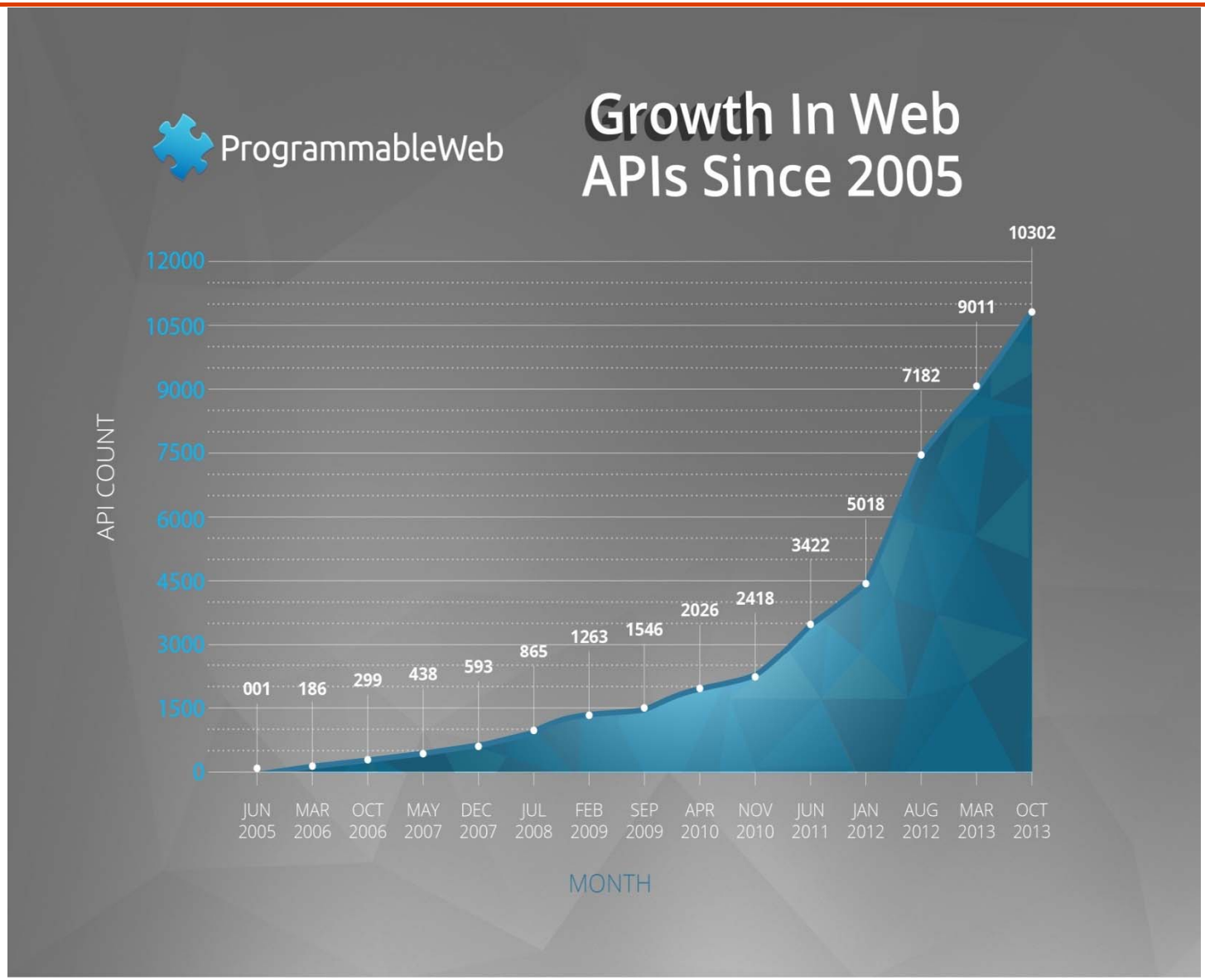


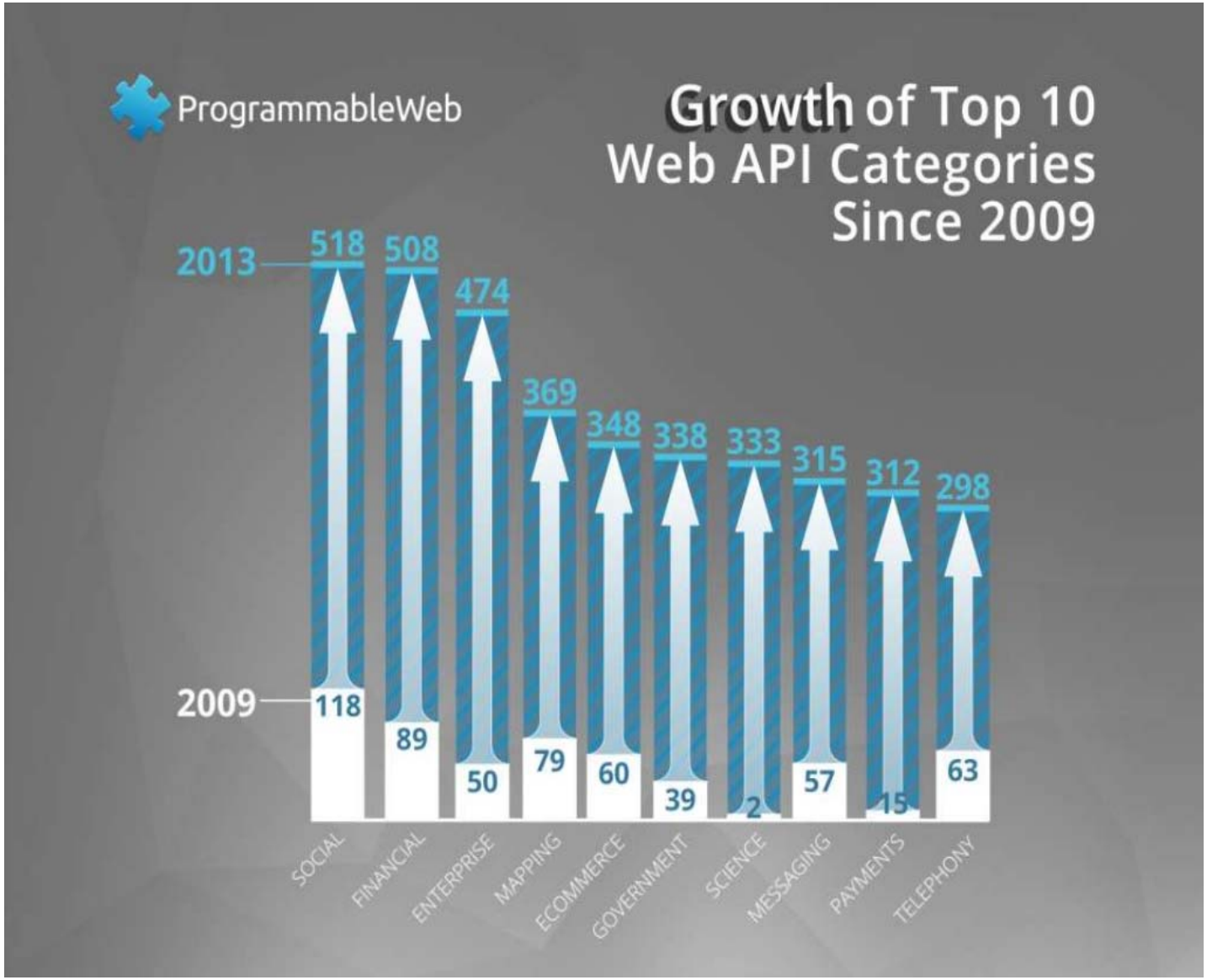
http://de.wikipedia.org/wiki/Bin%C3%A4rschnittstelle
http://de.wikipedia.org/wiki/Binärkompatibilität

Was ist eine Schnittstelle: es hängt von der Nutzung ab …

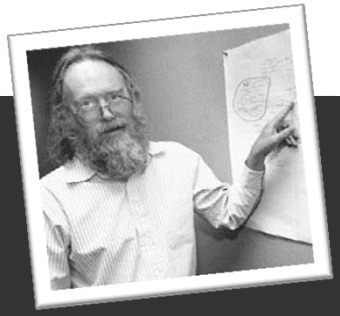


Über 10.000 Web-APIs

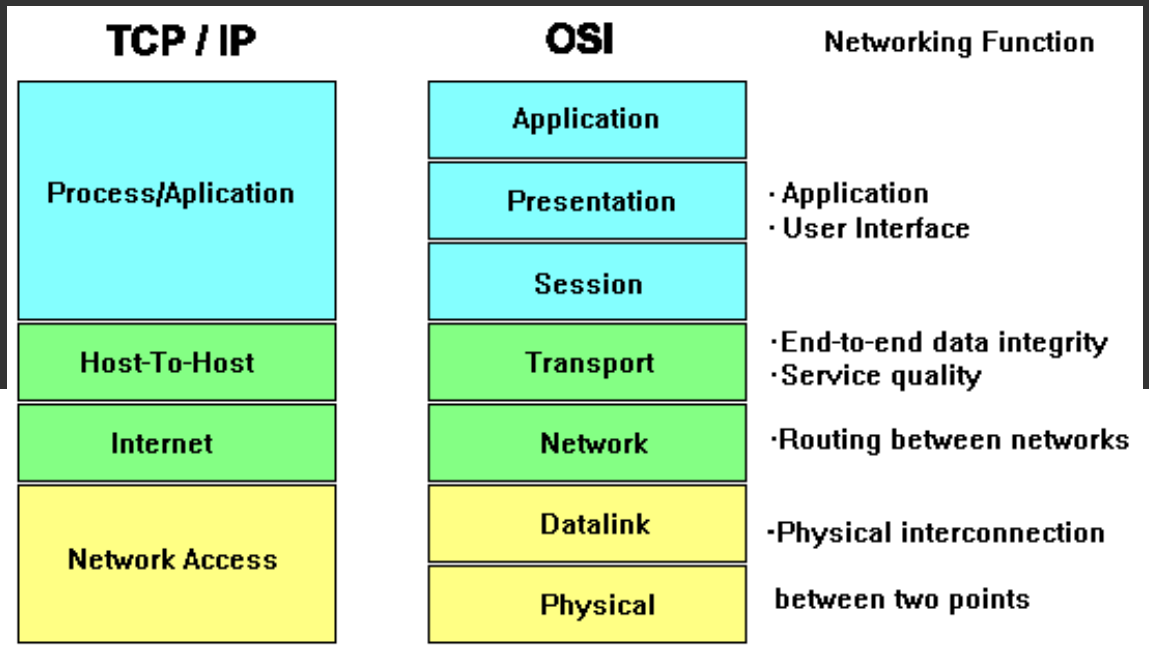



ProgrammableWeb
Growth of Top 10
Web API Categories
Since 2009
2013
2009
518
508
474
369
348
338
333
315
312
298
118
89
50
79
60
39
2
57
15
63
SOCIAL
FINANCIAL
ENTERPRISE
MAPPING
ECOMMERCE
GOVERNMENT
SCIENCE
MESSAGING
PAYMENTS
TELEPHONY

Wie Schnittstelle designen? – das OSI-Schichtenmodell



TCP Robustness Principle (RFC793) Jon Postel (1981)
*"be **conservative** in what you do,*
*be **liberal** in what you accept from others"*

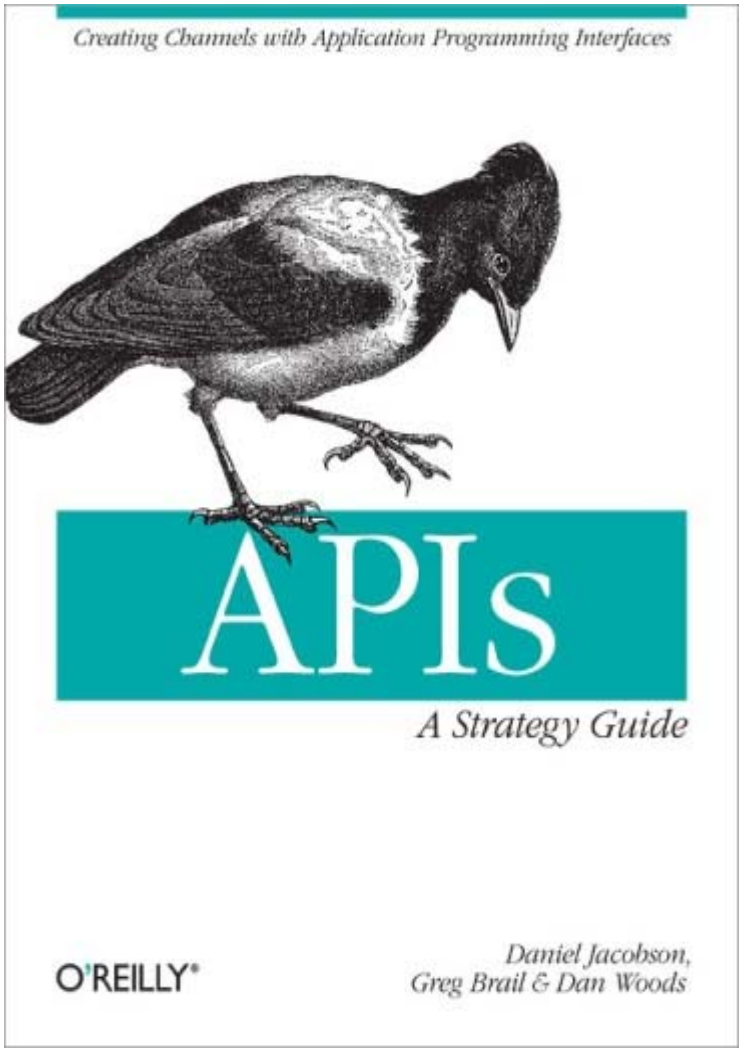


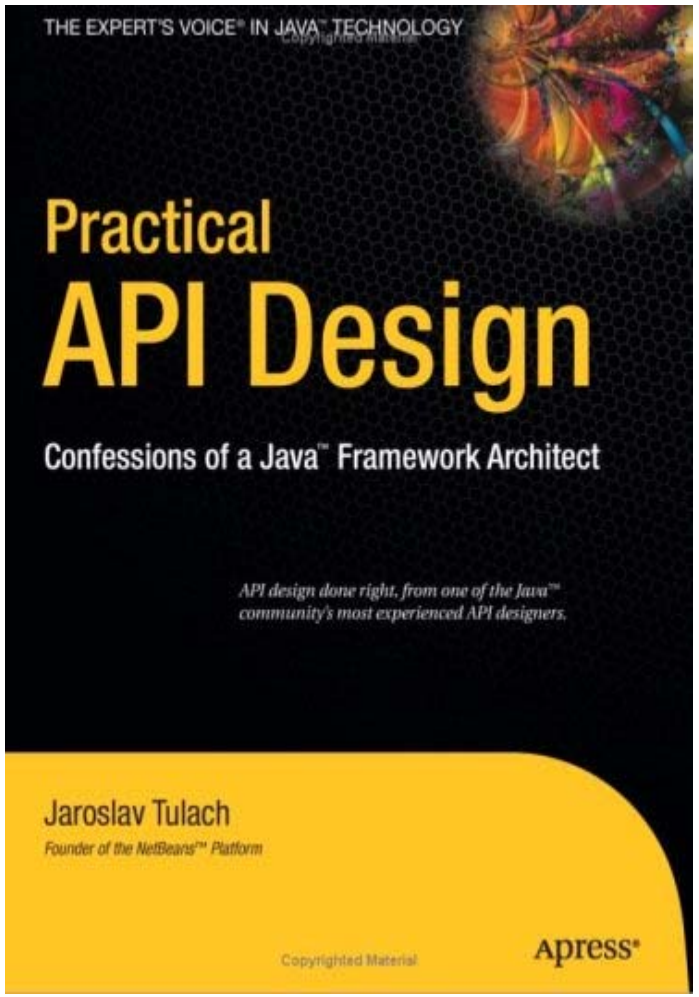
Risikopotential APIs

- Schnittstelle
- Protokoll
- Implementierung
- Infrastruktur
- Betrieb
- Überwachung
- Änderung

- Hohe Kosten, Risiken

http://wiki.apidesign.org/wiki/APIDesignPatterns

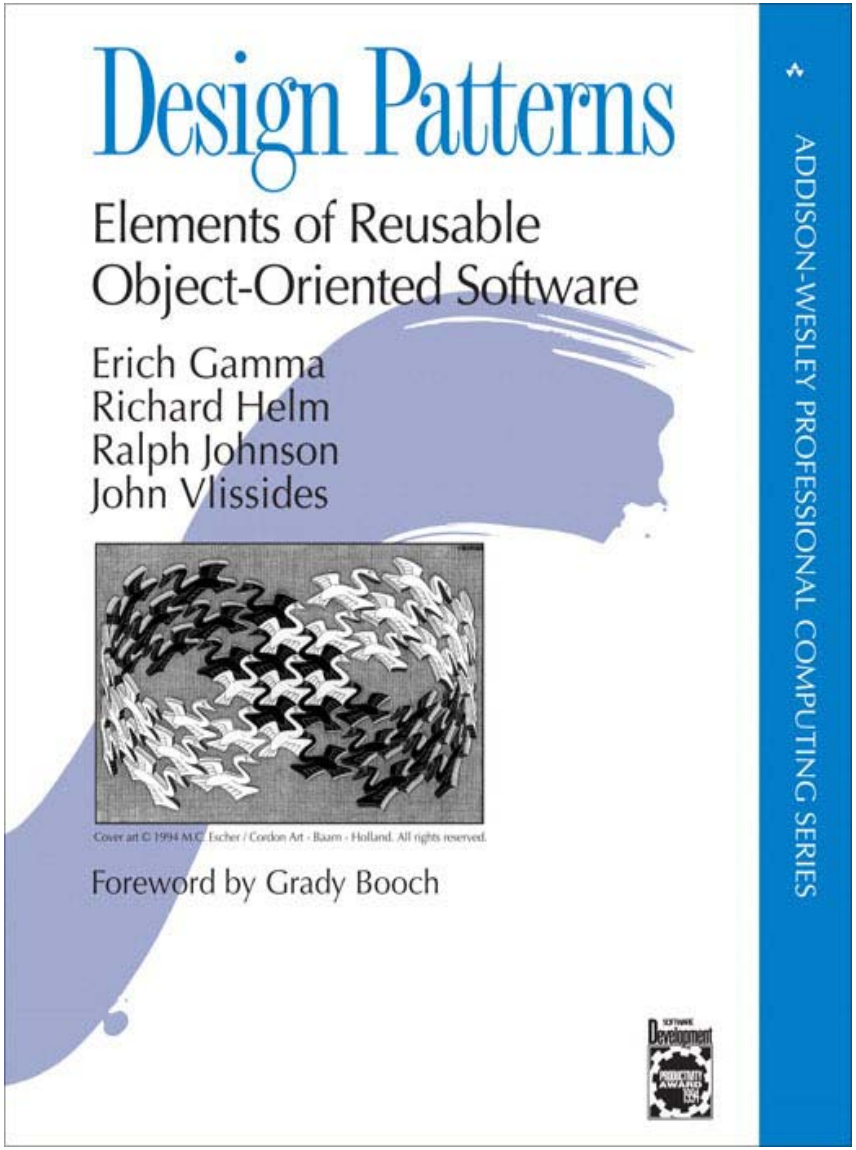




http://www.oracle.com/technetwork/java/seccodeguide-139067.html

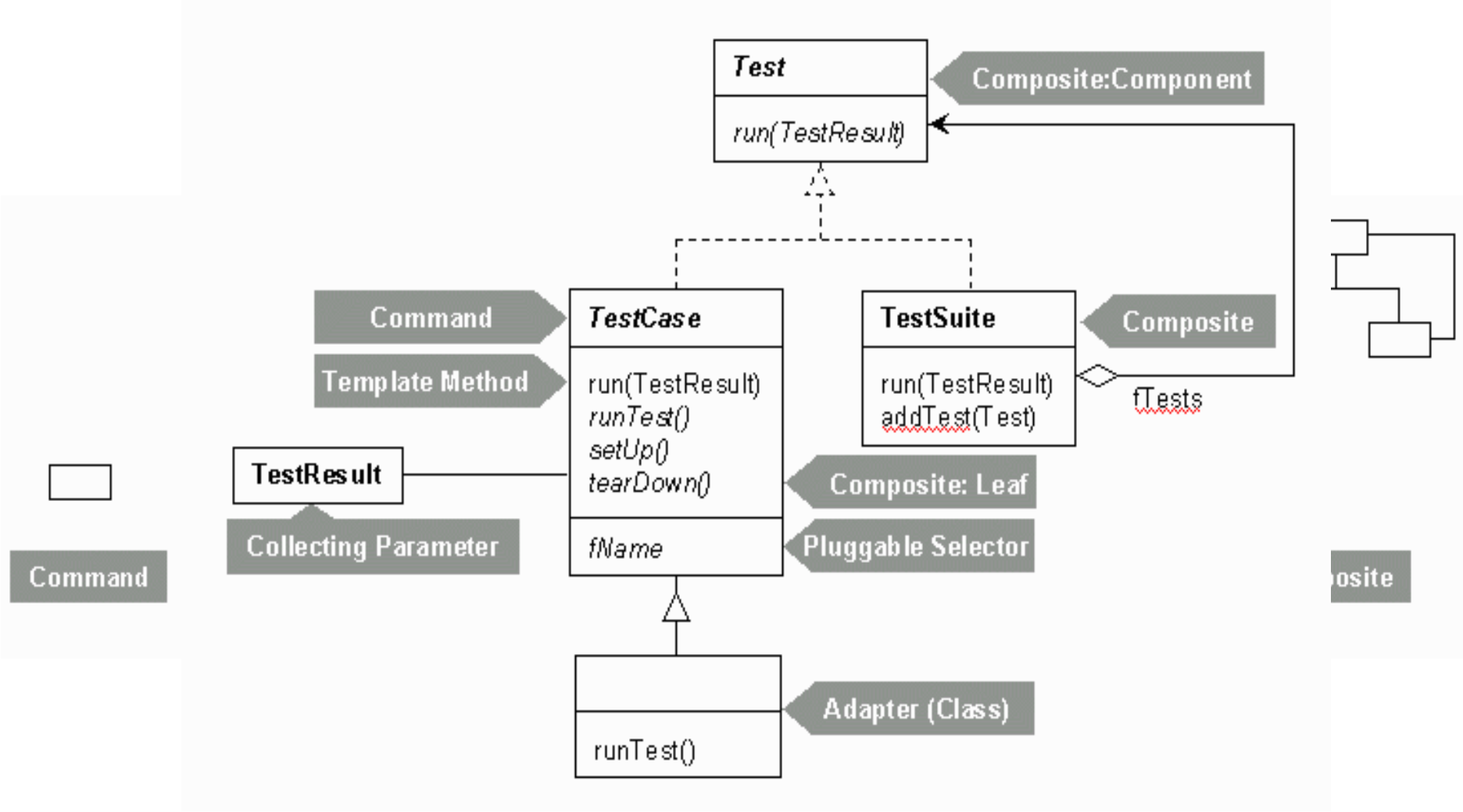
- Guideline 0-0 / FUNDAMENTALS-0: Prefer to have obviously no flaws rather than no obvious flaws
- Guideline 0-1 / FUNDAMENTALS-1: Design APIs to avoid security concerns
- Guideline 0-2 / FUNDAMENTALS-2: Avoid duplication
- Guideline 0-3 / FUNDAMENTALS-3: Restrict privileges
- Guideline 0-4 / FUNDAMENTALS-4: Establish trust boundaries
- Guideline 0-5 / FUNDAMENTALS-5: Minimise the number of permission checks
- Guideline 0-6 / FUNDAMENTALS-6: Encapsulate.
- Guideline 0-7 / FUNDAMENTALS-7: Document security-related information
- Guideline 1-2 / DOS-2: Release resources in all cases
- Guideline 2-1 / CONFIDENTIAL-1: Purge sensitive information from exceptions
- Guideline 2-2 / CONFIDENTIAL-2: Do not log highly sensitive information
- Guideline 2-3 / CONFIDENTIAL-3: Consider purging highly sensitive from memory after use
- Guideline 3-1 / INJECT-1: Generate valid formatting
- Guideline 3-2 / INJECT-2: Avoid dynamic SQL
- Guideline 5-1 / INPUT-1: Validate inputs

Design Patterns 20 Jahre danach



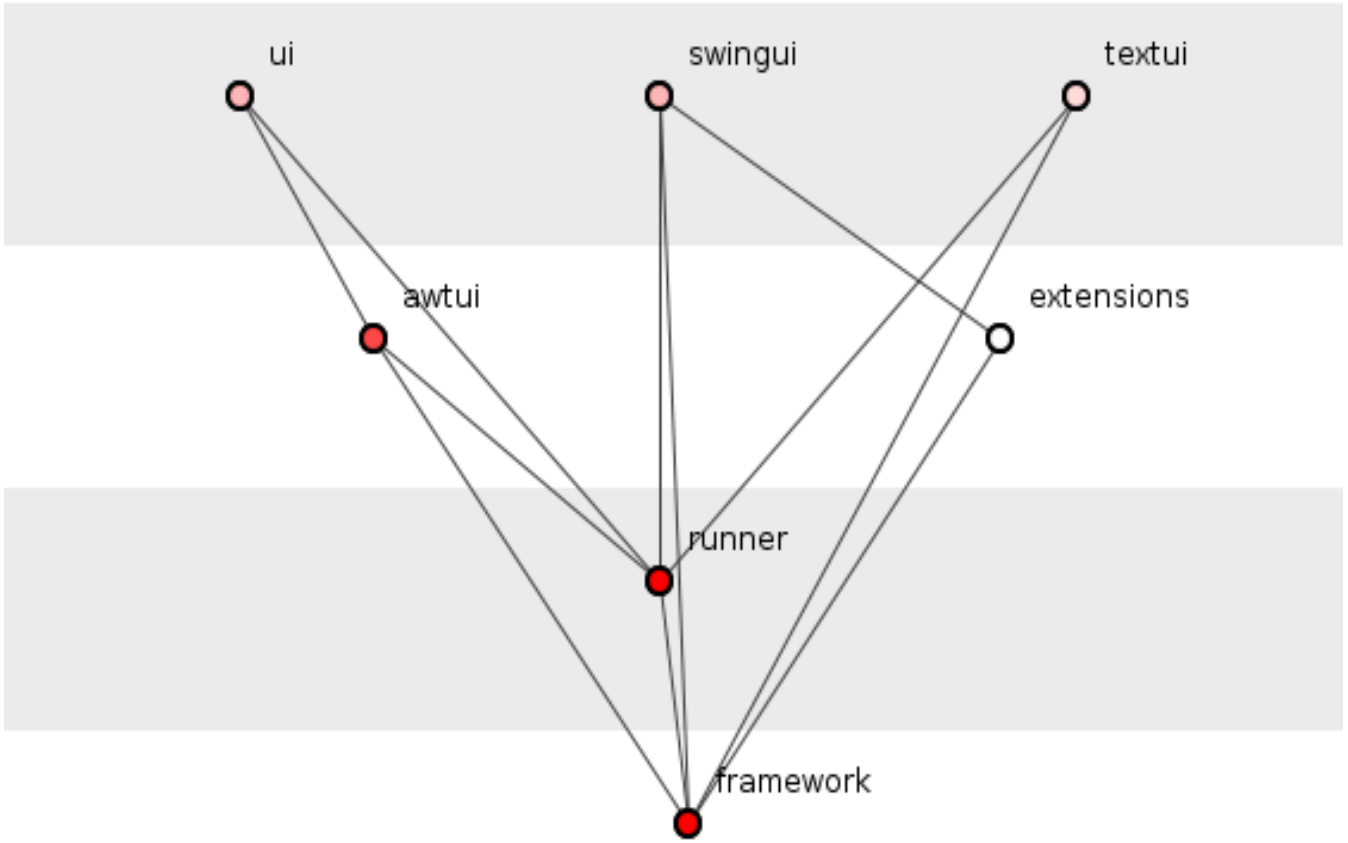
Design Patterns 15 Years Later: An Interview with Erich Gamma, Richard Helm, and Ralph Johnson
http://www.informit.com/articles/printerfriendly/140

JUnit Design (1998, Kent Beck, Erich Gamma)



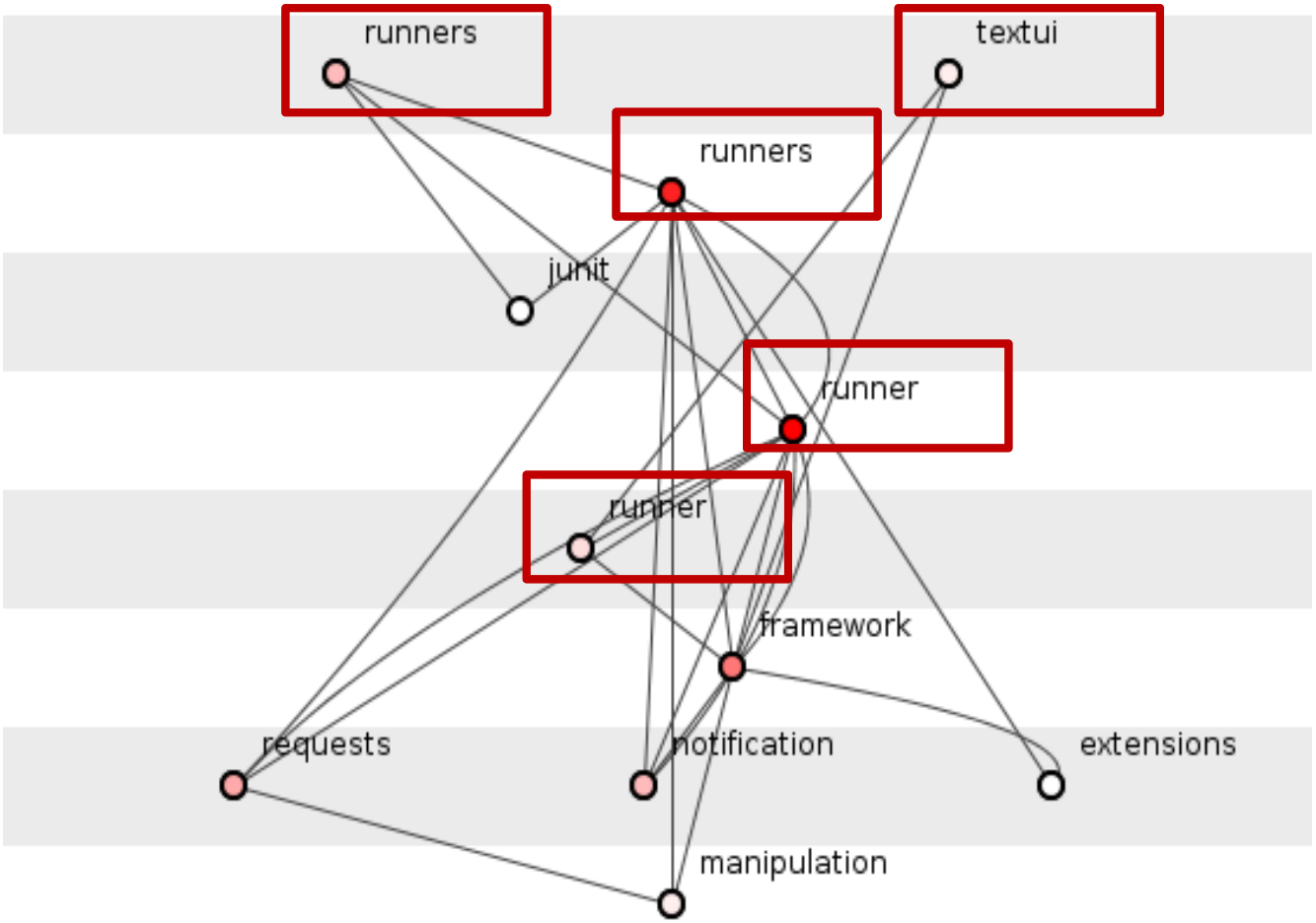
http://junit.sourceforge.net/doc/cookstour/cookstour.htm
Test Infected: Programmers Love Writing Tests, Java Report, July 1998, Volume 3, Number 7

JUnit version 3.7 (2001) – wie alles begann



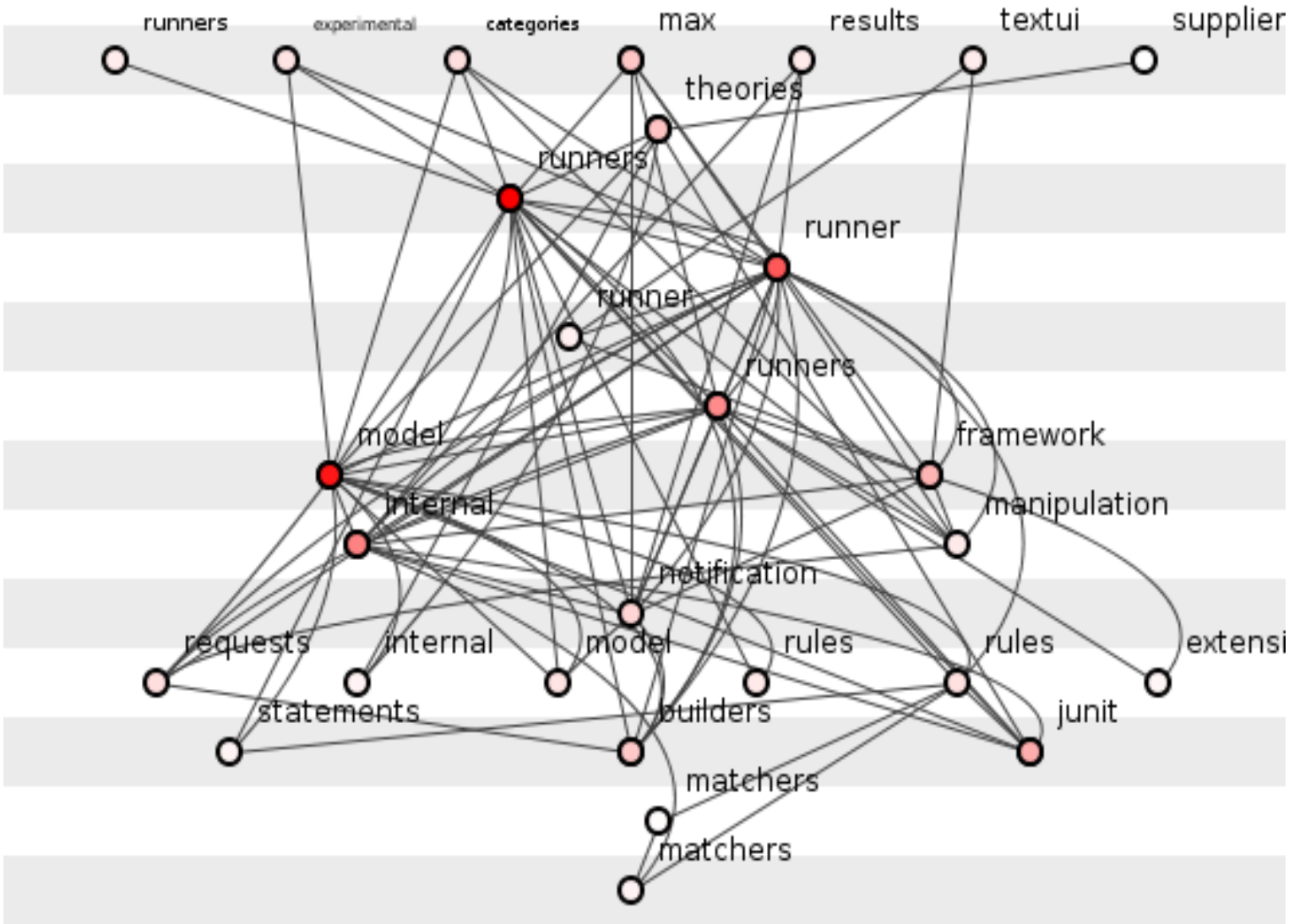
http://edmundkirwan.com/general/junit.html

JUnit version 4.0 (2006) – entwickelt sich weiter



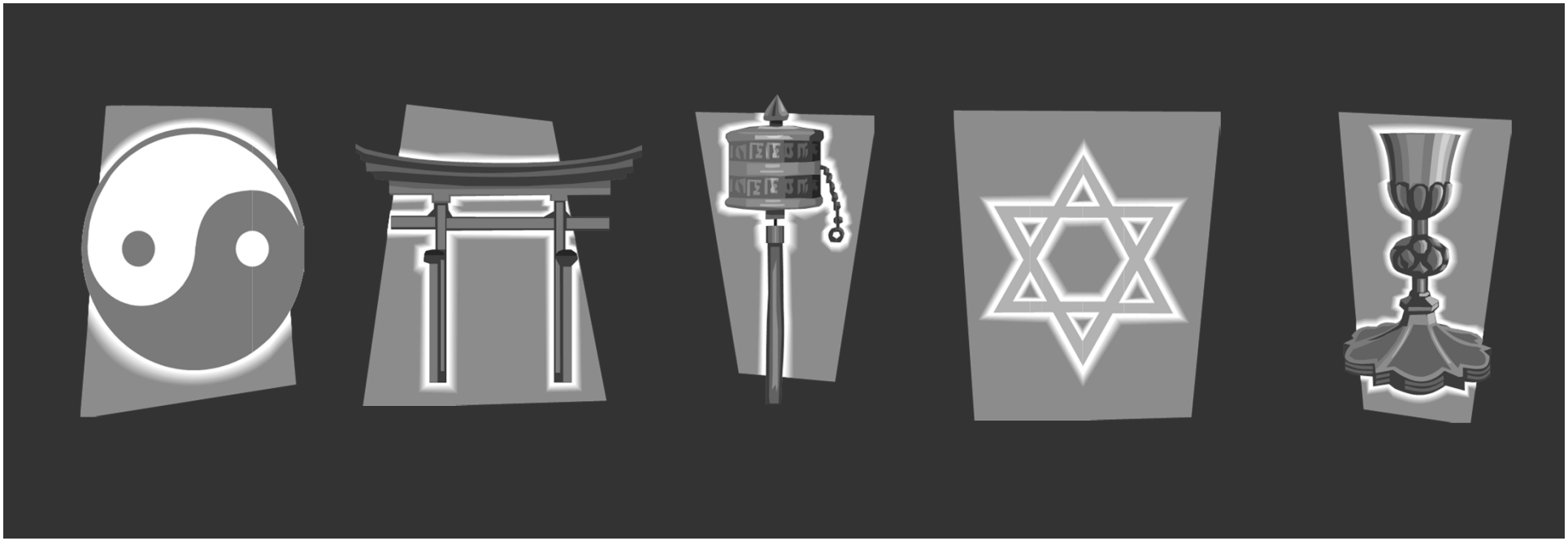
http://edmundkirwan.com/general/junit.html

JUnit version 4.11 (2012) – und weiter



Patterns & Antipattern

GOF ist keine Religion oder Magie!



Kompatibilität

- Binärkompatibilität (Big- oder Little-Endian)
- Quelltextkompatibilität: Quelltext kompiliert ohne Anpassung auf unterschiedlichen Systemen (GCC)
- Abwärtskompatibilität: UTF-8, 7-Bit-ASCII
- Aufwärtskompatibilität: Browser HTML-Erweiterungen
- Inkompatibilität: Major-Release
- Fehlerkompatibilität: IE6-Modus, Emulation

API-Versionierungsbeispiele bei Apache

Commons Lang 2.6 API

Packages	
org.apache.commons.lang	Provides highly reusable static utility methods, chiefly concerned with adding value to the java.lang classes.
org.apache.commons.lang.builder	Assists in creating consistent equals(Object), toString(), hashCode(), and compareTo(Object) methods.
org.apache.commons.lang.enum	**Deprecated** and replaced by org.apache.commons.lang.enums and wil
org.apache.commons.lang.enums	Provides an implementation of the C style enum in the Java world.
org.apache.commons.lang.exception	Provides JDK 1.4 style Nested Exception functionality for those on prior
org.apache.commons.lang.math	Extends java.math for business mathematical classes.
org.apache.commons.lang.mutable	Provides typed mutable wrappers to primitive values and Object.
org.apache.commons.lang.reflect	Accumulates common high-level uses of the java.lang.reflect APIs.
org.apache.commons.lang.text	Provides classes for handling and manipulating text, partly as an extensio
org.apache.commons.lang.time	Provides classes and methods to work with dates and durations.

The Apache Software Foundation

Class ClientDataSource40

java.lang.Object
  org.apache.derby.jdbc.ClientBaseDataSourceRoot
    org.apache.derby.jdbc.ClientBaseDataSource
      org.apache.derby.jdbc.ClientDataSource
        org.apache.derby.jdbc.ClientDataSource40

All Implemented Interfaces:

java.io.Serializable, java.sql.Wrapper, javax.naming.Referenceable, javax.sql.Co
org.apache.derby.jdbc.ClientDataSourceInterface

```
public class ClientDataSource40
extends org.apache.derby.jdbc.ClientDataSource
implements javax.sql.DataSource
```

This datasource is suitable for a client/server use of Derby, running on the following platforms:

- Java SE 7 (JDBC 4.1) and
- full Java SE 8 (JDBC 4.2).

Use BasicClientDataSource40 if your application runs on Java 8 Compact Profile 2.

Use ClientDataSource if your application runs on the following platforms:

- JDBC 4.0 - Java SE 6
- JDBC 3.0 - J2SE 5.0

Apache Commons Lang 3.3.2 API

Packages

Package	Description
org.apache.commons.lang3	
org.apache.commons.lang3.builder	
org.apache.commons.lang3.concurrent	
org.apache.commons.lang3.event	
org.apache.commons.lang3.exception	
org.apache.commons.lang3.math	
org.apache.commons.lang3.mutable	
org.apache.commons.lang3.reflect	
org.apache.commons.lang3.text	
org.apache.commons.lang3.text.translate	
org.apache.commons.lang3.time	
org.apache.commons.lang3.tuple	

All Classes

AbstractConnPool
AbstractCookieAttributeHandlerHC4
AbstractCookieSpecHC4
AbstractExecutionAwareRequest
AbstractHttpEntityHC4
AbstractMessageParserHC4
AbstractMessageWriterHC4
AbstractVerifierHC4
AIMDBackoffManager
AllowAllHostnameVerifierHC4
Args
Asserts
AuthCache
AuthenticationStrategy
AuthOption
AuthProtocolState
AuthSchemeBaseHC4
AuthSchemeProvider

Overview Package Class Tree Deprecated Index Help

PREV NEXT FRAMES NO FRAMES

httpclient-android 4.3.3 API Deprecated API

Contents

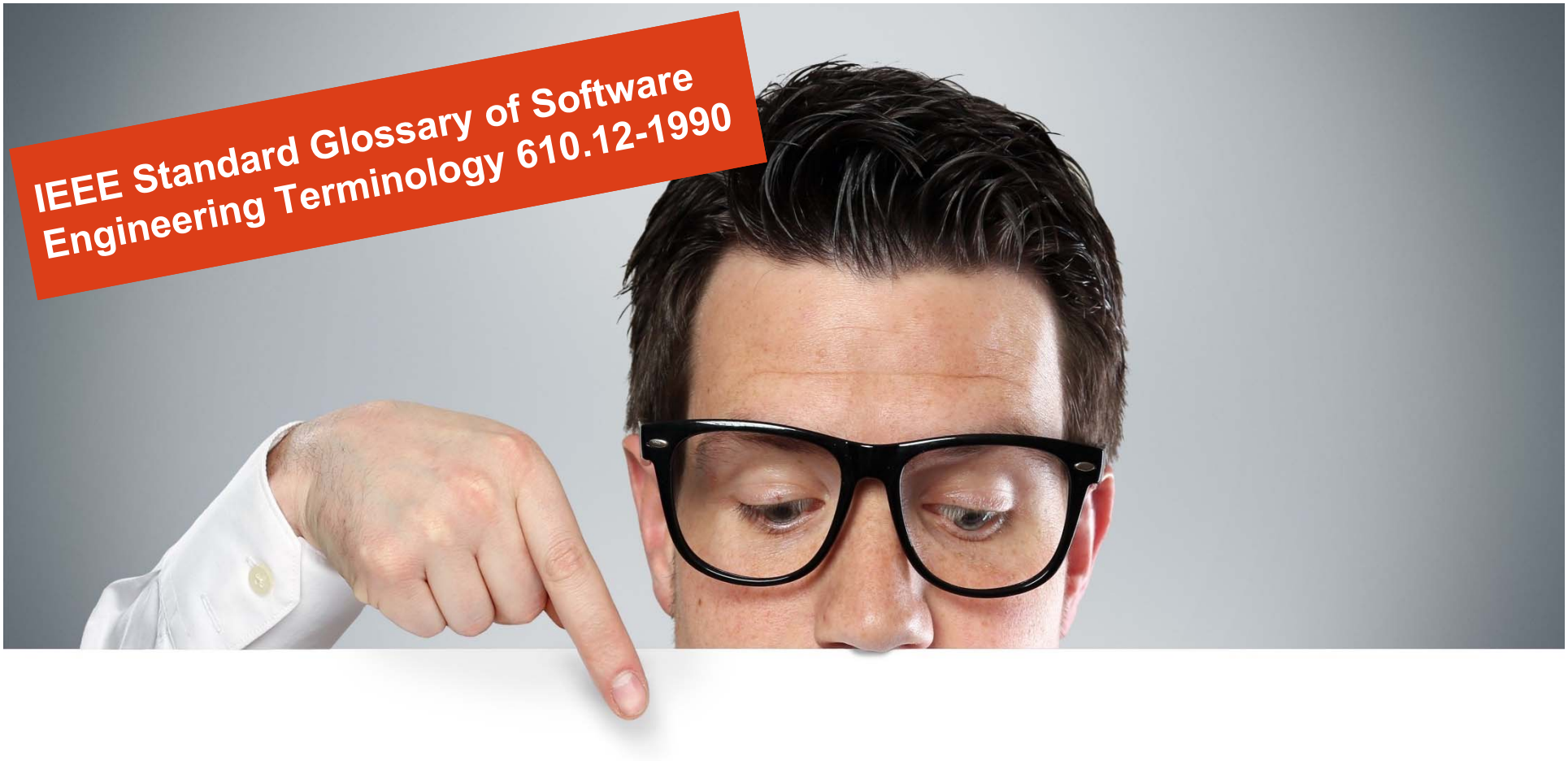
- Deprecated Interfaces
- Deprecated Classes
- Deprecated Fields
- Deprecated Methods
- Deprecated Constructors

Deprecated Interfaces

org.apache.http.params.HttpParamsNames
(4.3) use configuration classes provided 'org.apache.http.config' and 'org.apache.http.client.config'

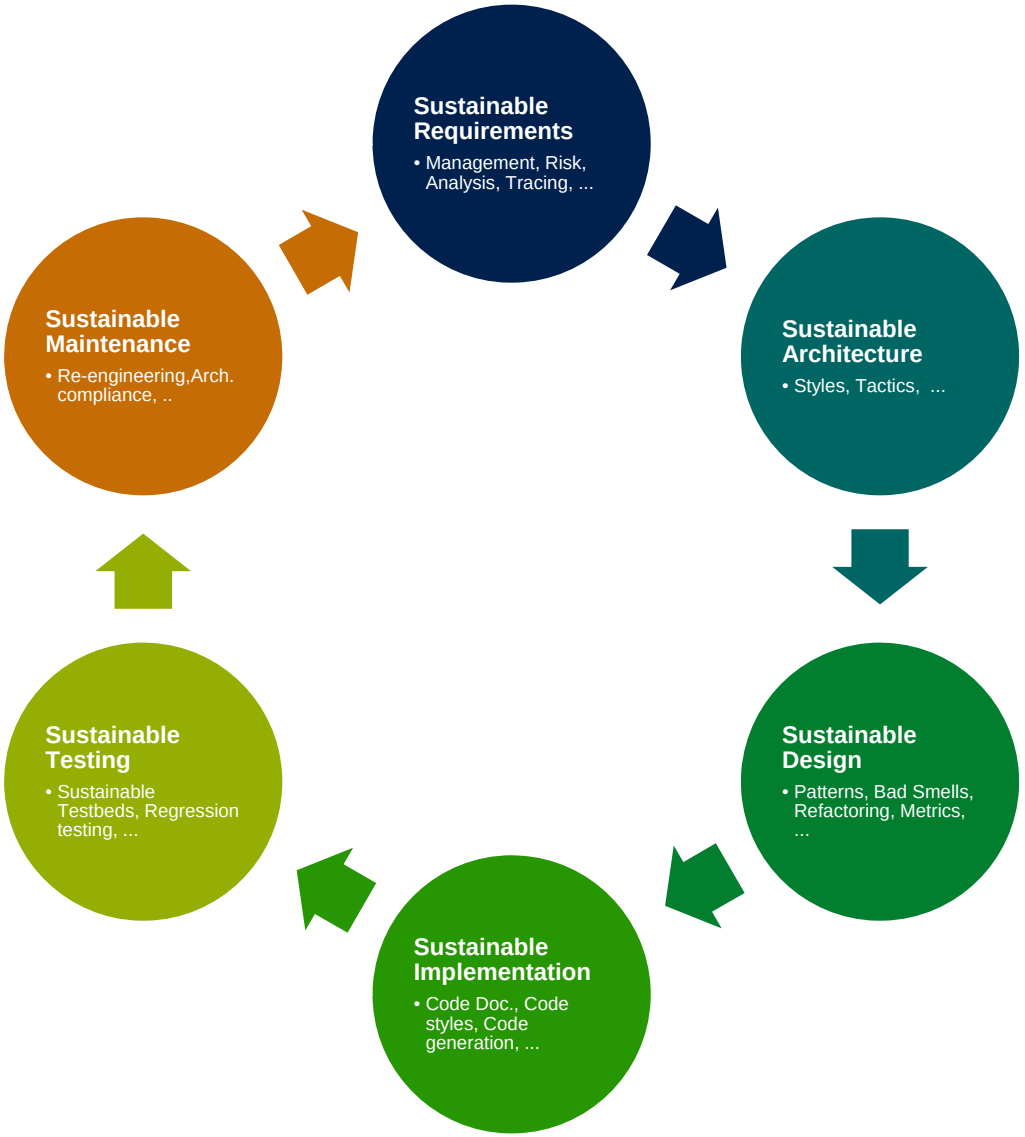
Softwareentwicklung:

ein komplexes Ökosystem
mit vielen Beteiligten



"**Software engineering** is the application of a *systematic, disciplined*, *quantifiable* approach to the **development**, **operation**, and **maintenance** of software"

Sustainability Guidelines for Long-Living Software Systems



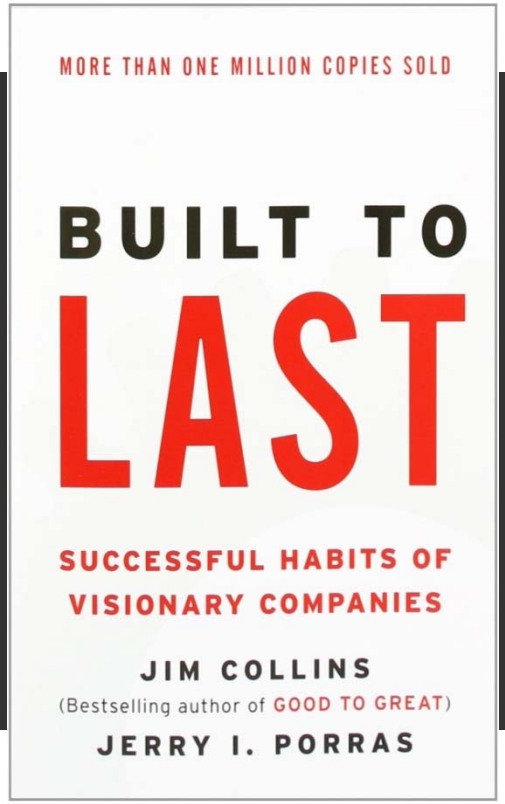
Phase independent:

- Documentation, Knowledge,
- Management, Process,
- Improvement,
- Organizational Structures, ...

Quelle: Heiko Koziolek

Fachkräftemangel als
das Problem des 21. Jh.

Build to last: Kriterien für nachhaltige Software



- Verwendbarkeit
- Anpassbarkeit
- Änderbarkeit
- Ersetzbarkeit

Maßnahmen für Nachhaltige Softwareentwicklung

- Einfachheit, Unabhängigkeit und Robustheit als oberstes Ziel
- Gemeinsame Nutzung durch das OpenSource-Prinzip erhöhen
- Reparatur geht vor Erweiterungen
- Ersatz geht vor Wiederverwendung (Replacement statt Reuse)
- Wissenstransfer der wichtigen Konzepte -Transparenz, statt Wissenslöcher/Kopfmonopole
- Qualitätszenarien frühzeitig entwerfen und anpass
  (Risikominimierung)
- Qualität vor Quantität


Ökologie
Reparatu-vor
Wiederverwendung
API-
Ökonomie
Gebaut für die
Ewigkeit
Soziales
Wissen
Robust
Dauerhaft
Erweiterbar

Fazit

- **Nachhaltigkeit**
  - Spielt in der Produktentwicklung eine große Rolle
  - Versöhnung von Ökonomie und Ökologie: Vermeide Verschwendung!
  - Softwareentwicklung als soziales System
  - API-Management & -Design: fehlertolerant, defensiv, konservativ-liberal
  - Behutsames IT-Trend-Management (Radar, Landkarte, Prozess)
  - Qualität vor Geschwindigkeit, Quantität

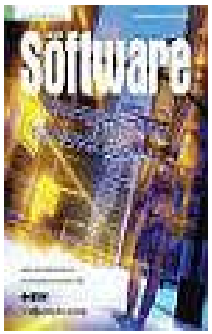
- **Menschen** gestalten Nachhaltigkeit
  - Einstellungssache/Werte - agile, flexibel, elastisch
  - Umdenken für die Zukunft – weniger Altlasten produzieren
  - Keine Anweisung zur wilden Software-Zucht
  - Mehr Qualität in der Softwareentwicklung
  - Wichtige Dokumente sollten zentral und frisch verfügbar sein

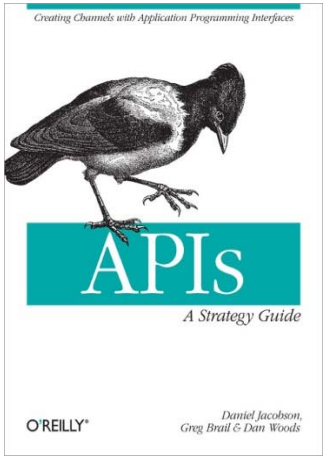
Was hilft?

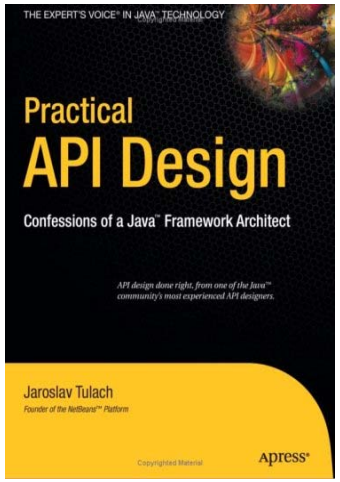
- **Langfristige** Qualitätsziele im Auge haben
- Eine **API-Ökonomie** entwickeln
- **Open Source** aktiv integrieren (Reuse, shared maintenance)
- **Simplify your complexity** – Reduce your dependencies (HW, SW)
- Ein **Wertekoordinatensystem** ist flexibler, als Plan
- Software-Entwicklung als **ganzheitlicher Prozess**
- Lernen **von anderen**, **lebenslanges Lernen**
- **Vom Ende her** denken und designen
- Weniger disruptive, mehr **evolutionäre, taktische** Änderungen

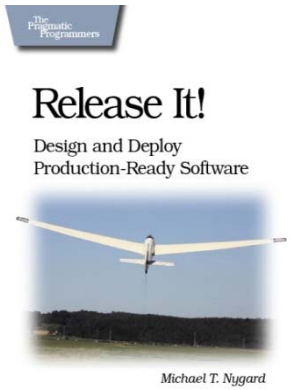
Weitere Infos:

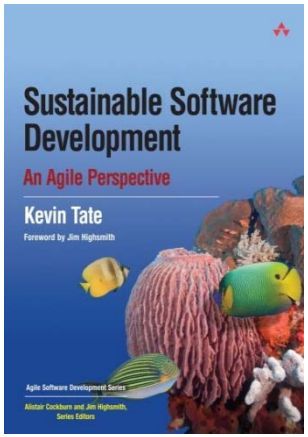
- Free and Open Source Software Technology for Sustainable Development: Sulayman K. Sowe, Govindan Parayil, Atsushi Sunami, September 2012
- Sustainable Software Development: An Agile Perspective: Kevin Tate, 2005
- Measuring Architecture Sustainability: Heiko Koziolek, Dominik Domis, Thomas Goldschmidt, Philipp Vorst, IEEE Software, 2013 (vol. 30) http://www.infoq.com/articles/measuring-architecture-sustainability
- Software Engineering with an Agile Development Framework/Whole process/Sustainability, WikiBook, 2012
- Michael T. Nygard, Release It!: Design and Deploy Production-Ready Software, 2007
- Sustainable Software Development With Clean C++, Stephan Roth, leanpub, 2014











1. – 4. September 2014
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Vielen Dank!

Frank Pientka

Materna GmbH