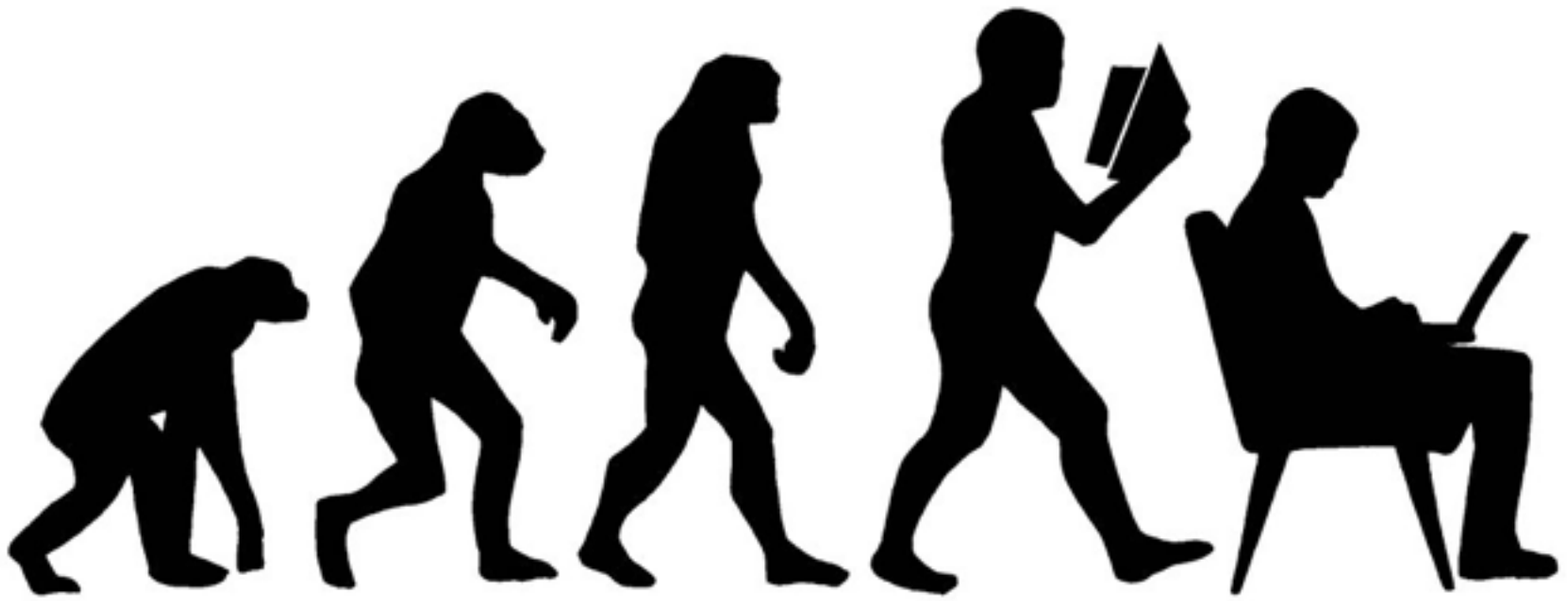
# Software-Evolution

## Warum und Wie



http://commons.wikimedia.org/wiki/File:Evolution-des-wissens.jpg

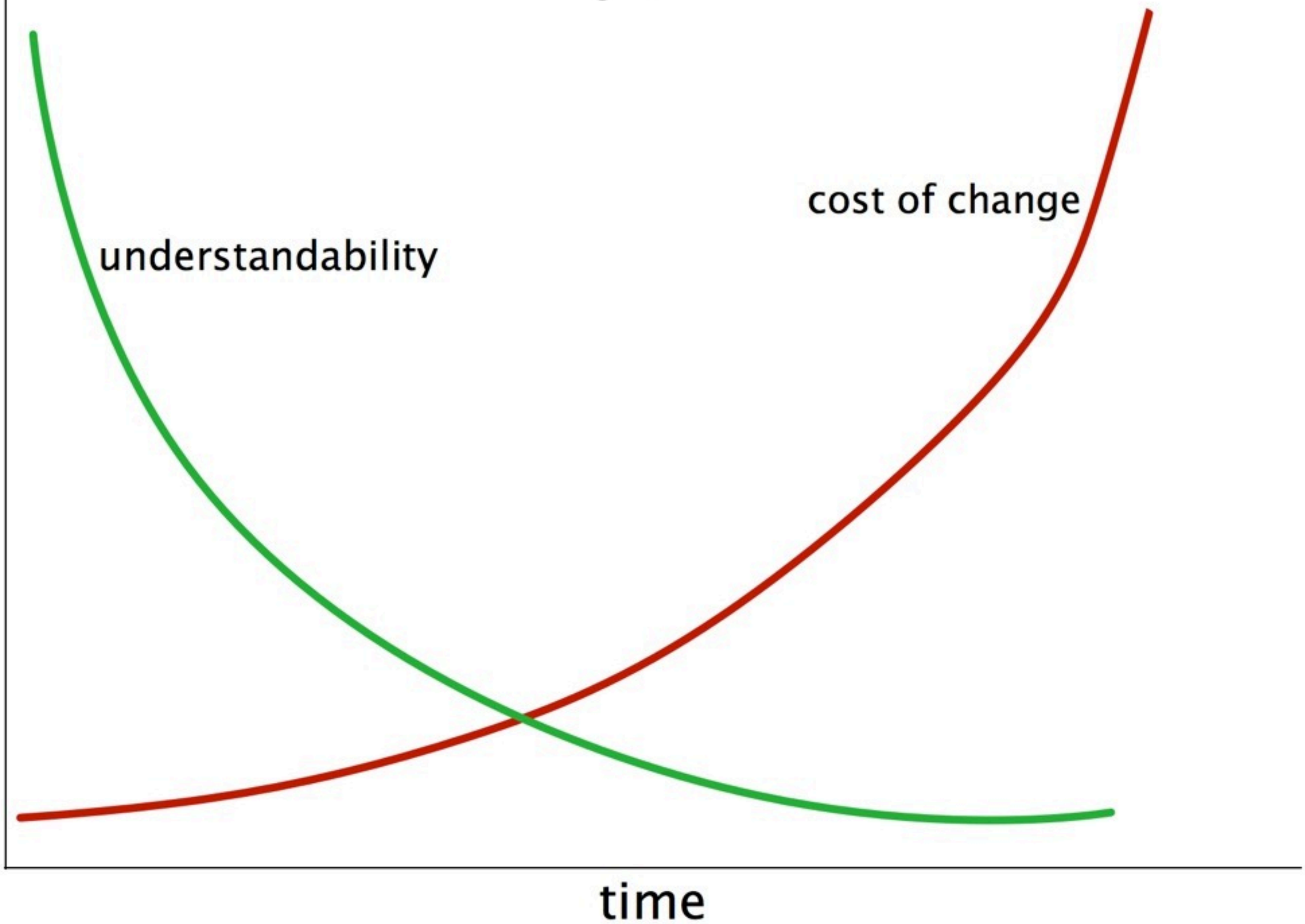
## Ändern - aber richtig!

innoQ

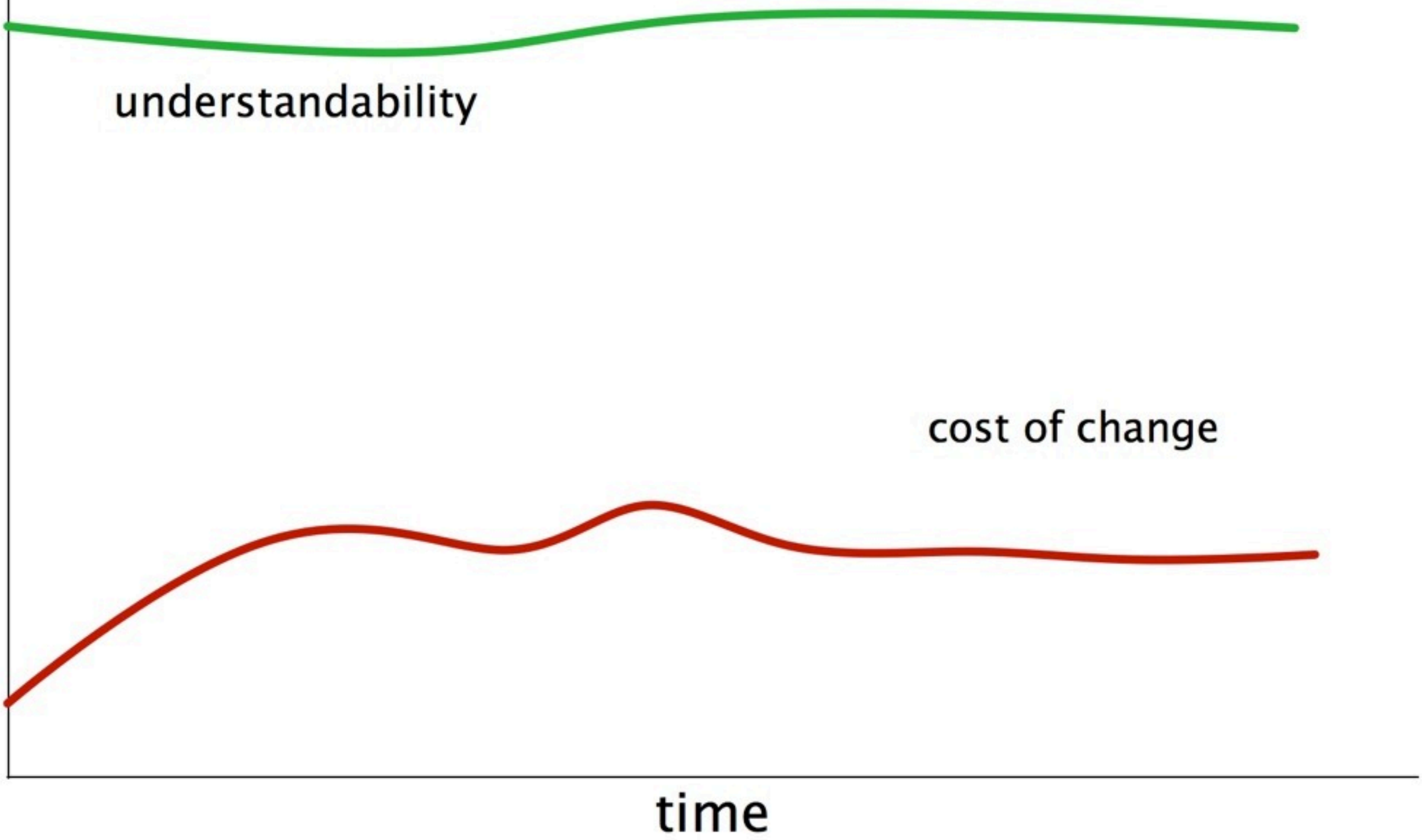
Follow @stilkov

Follow @gernotstarke

# Ist!



# Soll!



These:

Informatik

# Ausbildung fokussiert auf Neubau von Systemen

## These:

Informatik

# Praxis benötigt mehr Änderungskompetenz

an Systemen

These:

an Systemen

# Verbesserung

einzelner Klassen

# ist mehr als Refactoring

These:

Verbesserung
ist mehr als Refactoring

„Große“ Umbauten bedeuten (oft):

- Umbau DB-Struktur, Datenmigration
- Austausch von Software-Infrastruktur
  (z.B. Frameworks)
- umfangreiche Änderung interner Abläufe
- massive Änderung interner Schnittstellen

These:

kommerziell

# Erfolgreiche Systeme

meistens

# sind verbaut

These:

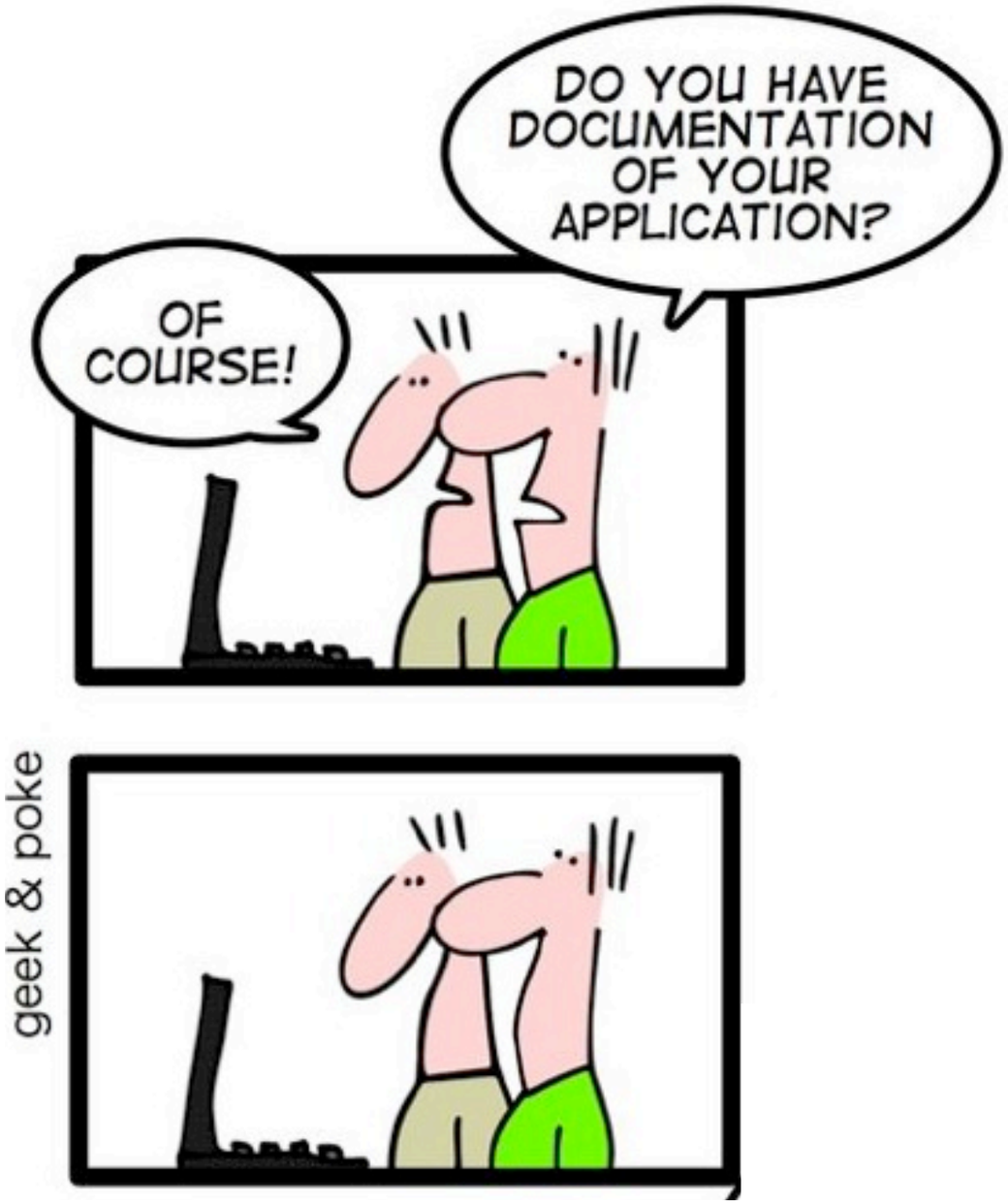
Erfolgreiche Systeme sind verbaut

Kommerziell erfolgreich:

- Anwender fordern schnell viele Features
- Schnelle Lieferung widerspricht „Engineering-Prinzipien“

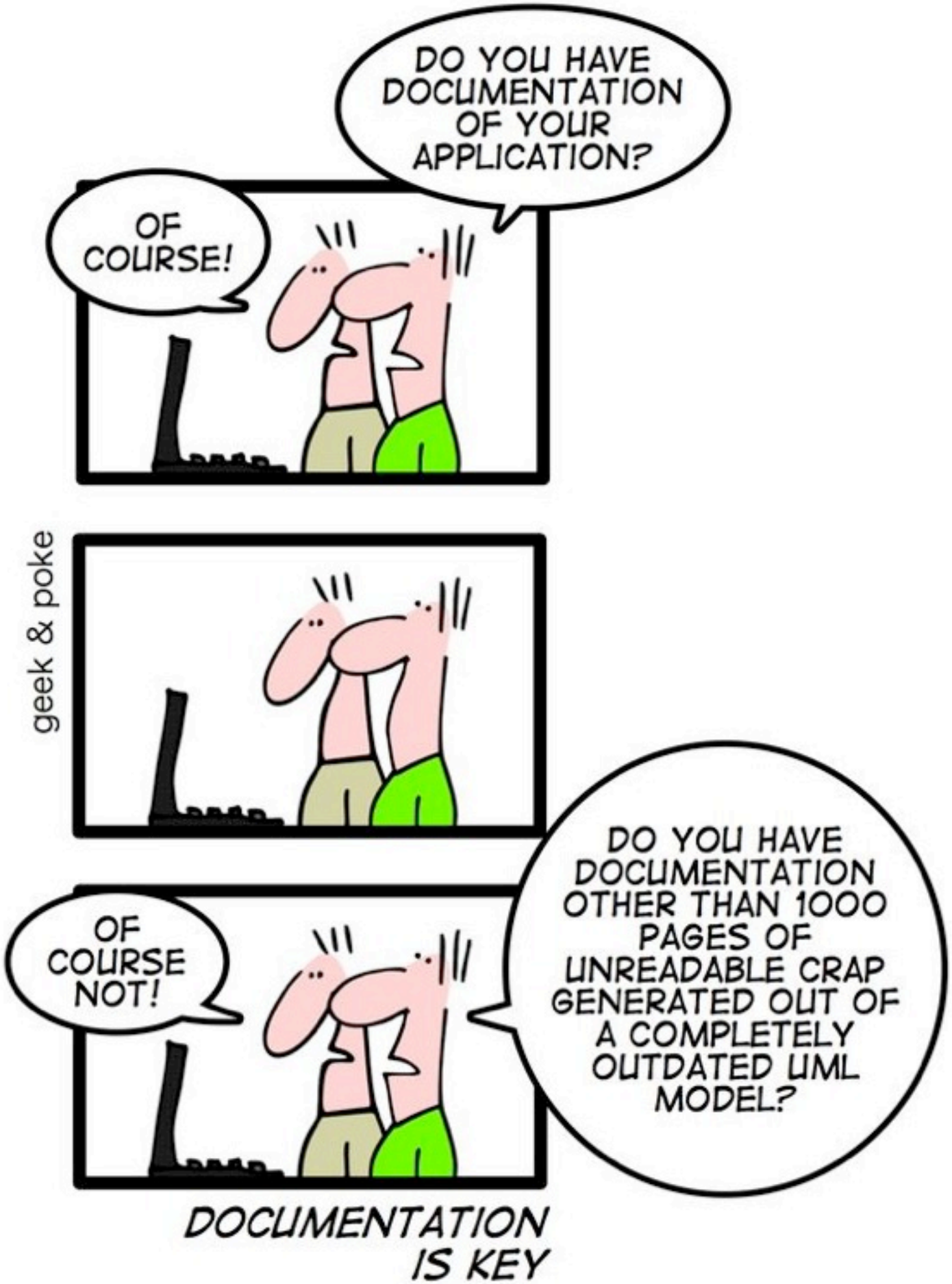
These:

# Erfolgreiche Systeme sind verbaut



These:

# Erfolgreiche Systeme sind verbaut



These:

# Erfolgreiche Systeme sind verbaut

http://www.flickr.com/photos/pasukaru76/9824401426

These:

# Änderungen an Systemen sind durch Geld motiviert

These:

## Änderungen an Systemen sind durch Geld motiviert

- neue Features
- veränderte NFA (z.B. Performance)
- hohe Betriebskosten
- hohe Änderungskosten

These:

# Budgetverantwortliche ignorieren Architekturprinzipien

These:
Budgetverantwortliche ignorieren Architekturprinzipien

# Architekturprinzipien und Clean-Code für

- Entwickler
- Architekten
- sonstige „intrinsisch Motivierte“

3 Fragen:

- Wissen alle Stakeholder, wo die Probleme in der Architektur liegen?
- Sind sich alle über die Konsequenzen einig?
- Gibt es klare Strategien zur Verbesserung?


# Architecture Improvement Method

# Architecture Improvement Method

**free**:

- ✓(as arc42!) open, contributions welcome

**easy** to apply:

- ✓methodical patterns and practices
- ✓no specific tooling required

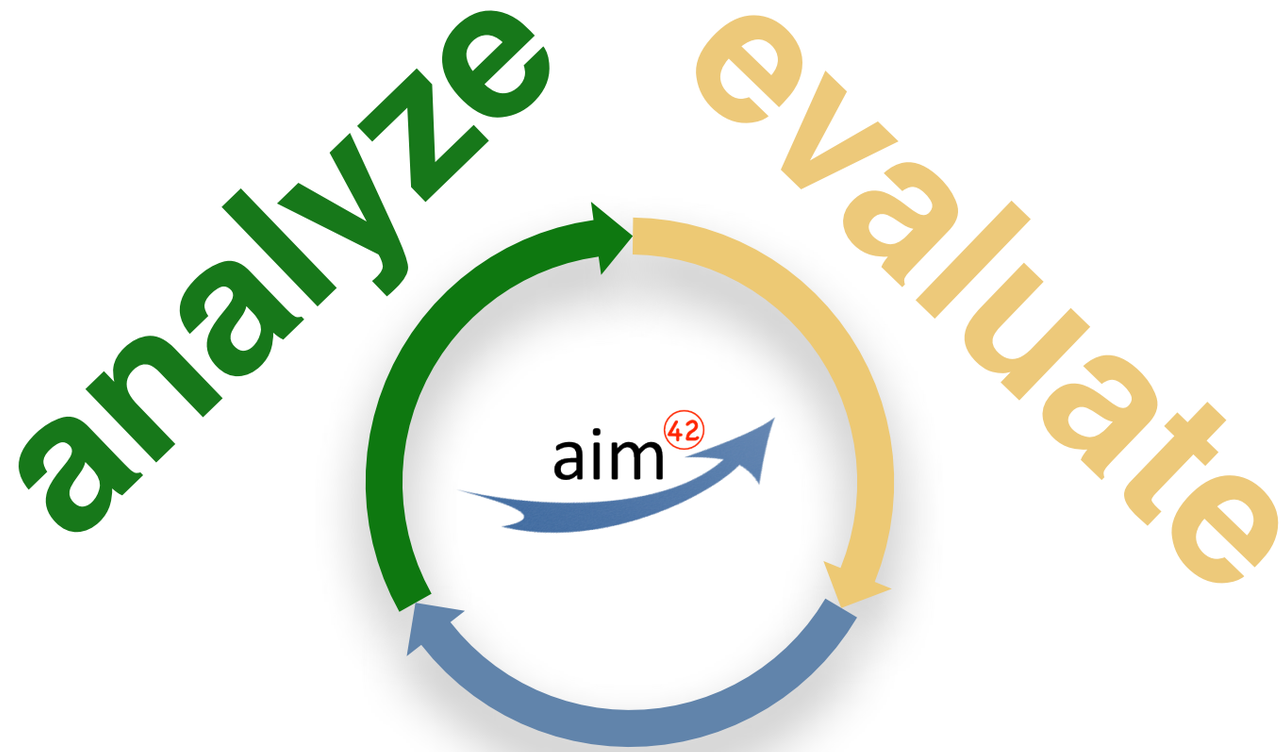
**grounded**:

- ✓combines new and established practices
- ✓industry-proven


analyze
evaluate
aim 42
improve

- architecture
- code
- runtime
- organization

determine „value“ of
problems / risks / issues
and their remedies




# improve

- define strategy
- setup regression testing
- refactor
- re-architect
- re-organize
- remove debt


search for symptoms, problems, issues, risks, faults, mis-behavior, technical-debt...


- qualitative
  - ATAM
  - nominal-actual comparison
- quantitative
- structural
- runtime
  - efficiency
  - resource consumption
  - logs, protocolls
- known issues
  - bugs and bug clusters
- organization
  - stakeholder
  - (development) process


# The Patterns...

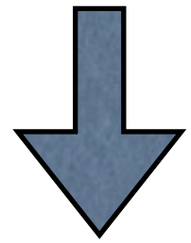
- Capture-Quality-Requirements
- Context-Analysis
- Data-Analysis
- Development-Process-Analysis
- Documentation-Analysis
- Issue-Tracker-Analysis
- Profiling
- Take What They Mean
- Qualitative-Analysis
- Quantitative-Analysis
- Pre-Interview-Questionnaire
- Requirements-Analysis
- Runtime-Artifact-Analysis
- Software-Archeology
- Stakeholder-Analysis
- Stakeholder-Interview
- Static-Code-Analysis
- View-Based-


- Capture-Quality-Requirements
- Context-Analysis
- Data-Analysis
- Development-Process-Analysis
- Documentation-Analysis
- Issue-Tracker-Analysis
- Profiling

# Patterns to find issues...

- **Qualitative-Analysis**
- Quantitative-Analysis
- Pre-Interview-Questionnaire
- Requirements-Analysis
- Runtime-Artifact-Analysis
- Software-Archeology
- **Stakeholder-Analysis**
- Stakeholder-Interview
- **Static-Code-Analysis**


# Stakeholder Analysis



## who MIGHT have problems

**Beispiele:**

Management, Auftraggeber, Projekt-Steuerkreis, sonstige Projekt-Gremien, PMO, Projektmanager, Produktmanager, Fachbereich, Unternehmens-/Enterprise-Architekt, **Architekten**, Methoden-Abteilung, QS-Abteilung, IT-Strategie, Software-Architekt, Software-Designer, **Entwickler**, **Tester**, Konfigurationsmanager, Build-Manager, Release-Manager, Wartungsteam, externe Dienstleister, Zulieferer, Hardware-Designer, Rollout-Manager, Infrastruktur-Planer, **Sicherheitsbeauftragter**, Behörde, Aufsichtsgremium, Auditor, Mitbewerber/Konkurrent, **Endanwender**, Fachpresse, Fachadministrator, **Systemadministrator**, Operator, **Hotline**, Betriebsrat, Lieferant von Standardsoftware, verbundene Projekte, Normierungsgremium, Gesetzgeber...


# Stakeholder Analysis (II)

**who MIGHT have problems**

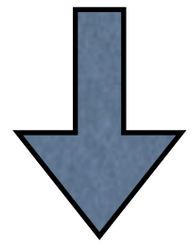
- use pre-interview questionnaire

- conduct personal interviews:

  e.g. what are your top-3 issues with...

  1. the system
  2. the development / maintenance process
  3. operation / infrastructure of the system
  4. ...


Capture-Quality-Requirements



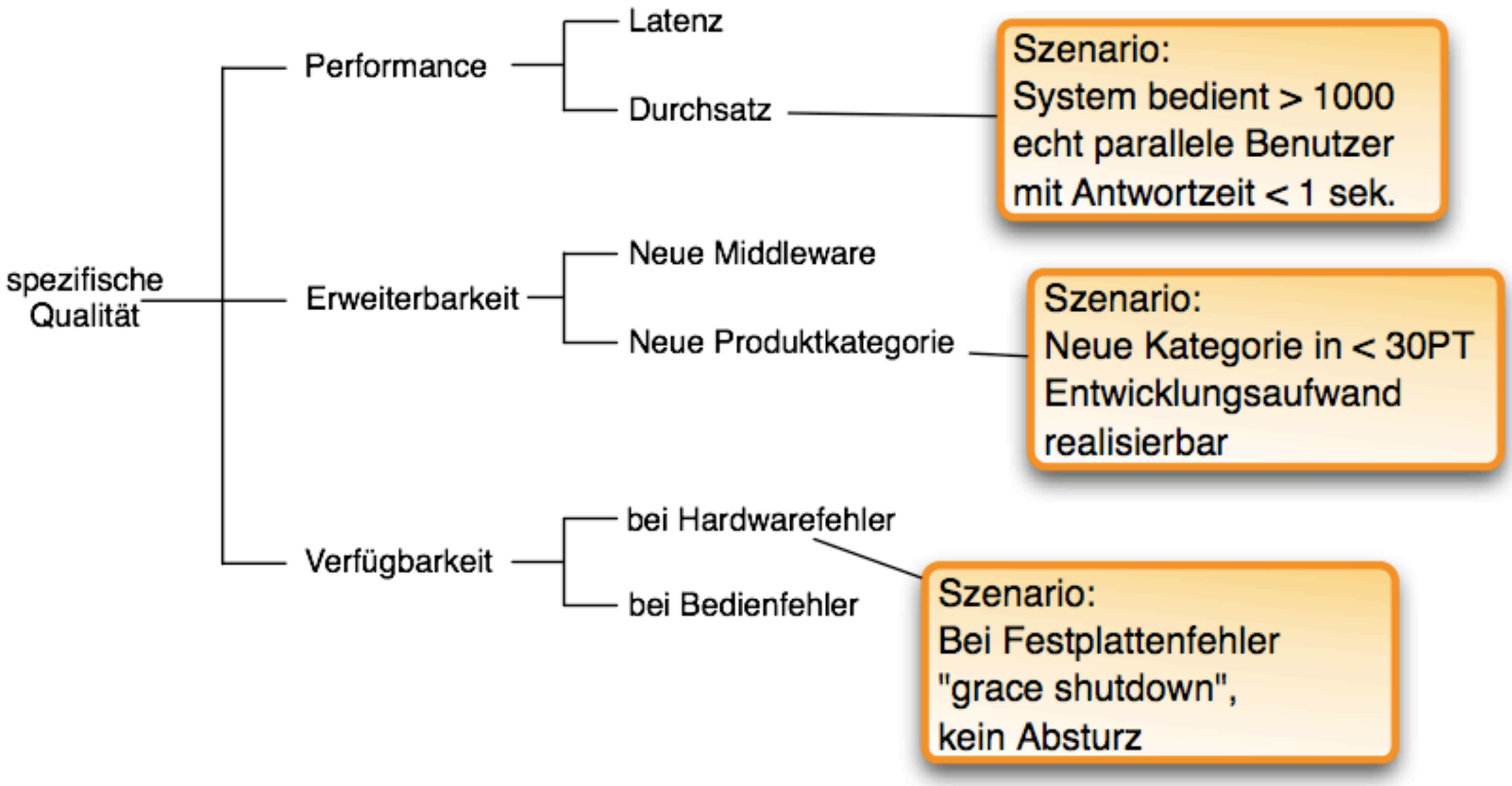
**Qualitative Analysis**

- (similar to) ATAM
- established methodology, published by SEI
- industry-proven

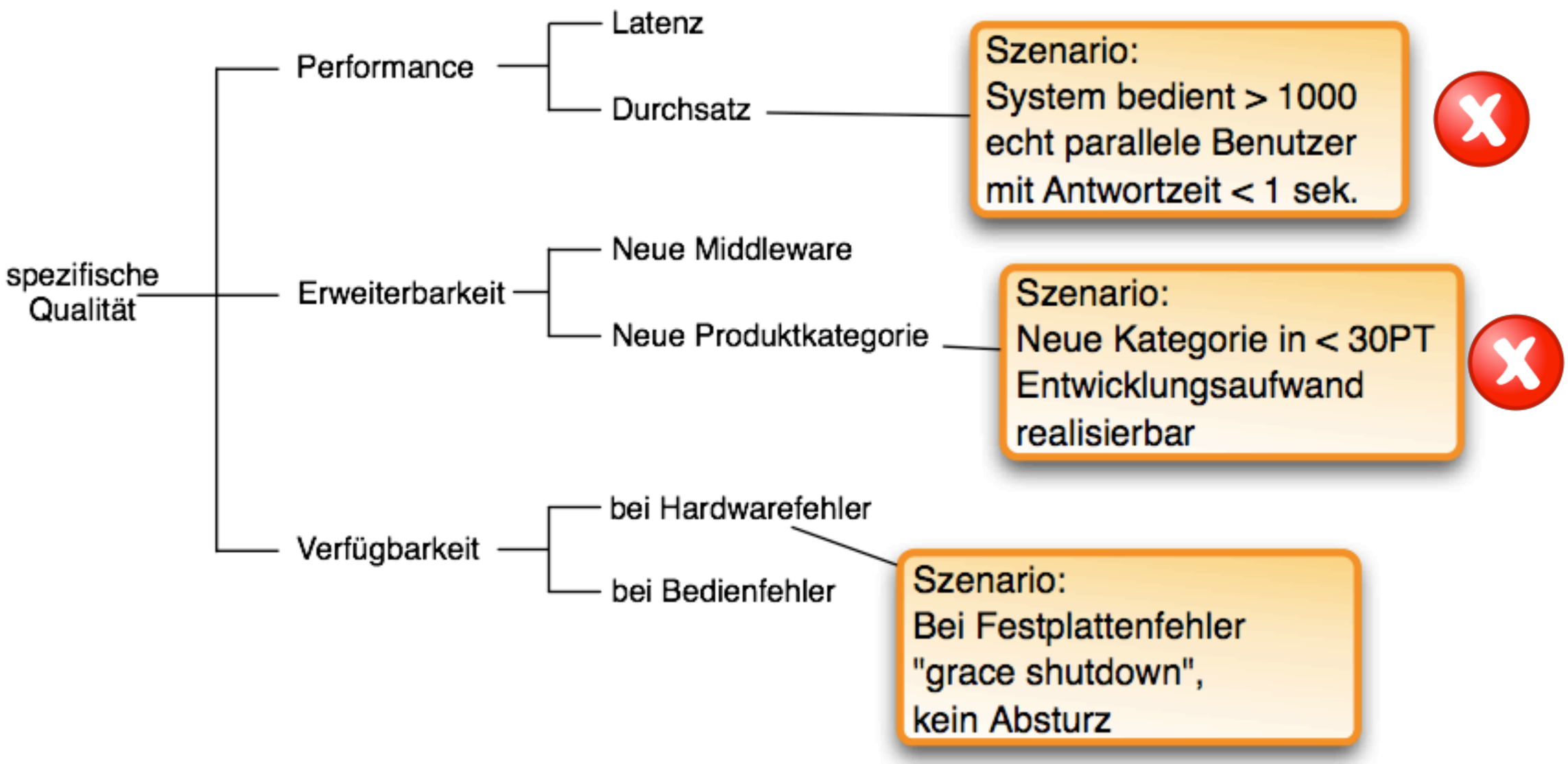
- basic idea: nominal-actual-comparison

1. determine explicit & specific quality goals
2. determine architecture approaches taken
3. analyze if 2. is sufficient for 1.

# Hierarchical Quality Model (1)



# Hierarchical Quality Model (II)




# Qualitative Analysis: Compare Goal / Solution

## Qualitätsziele

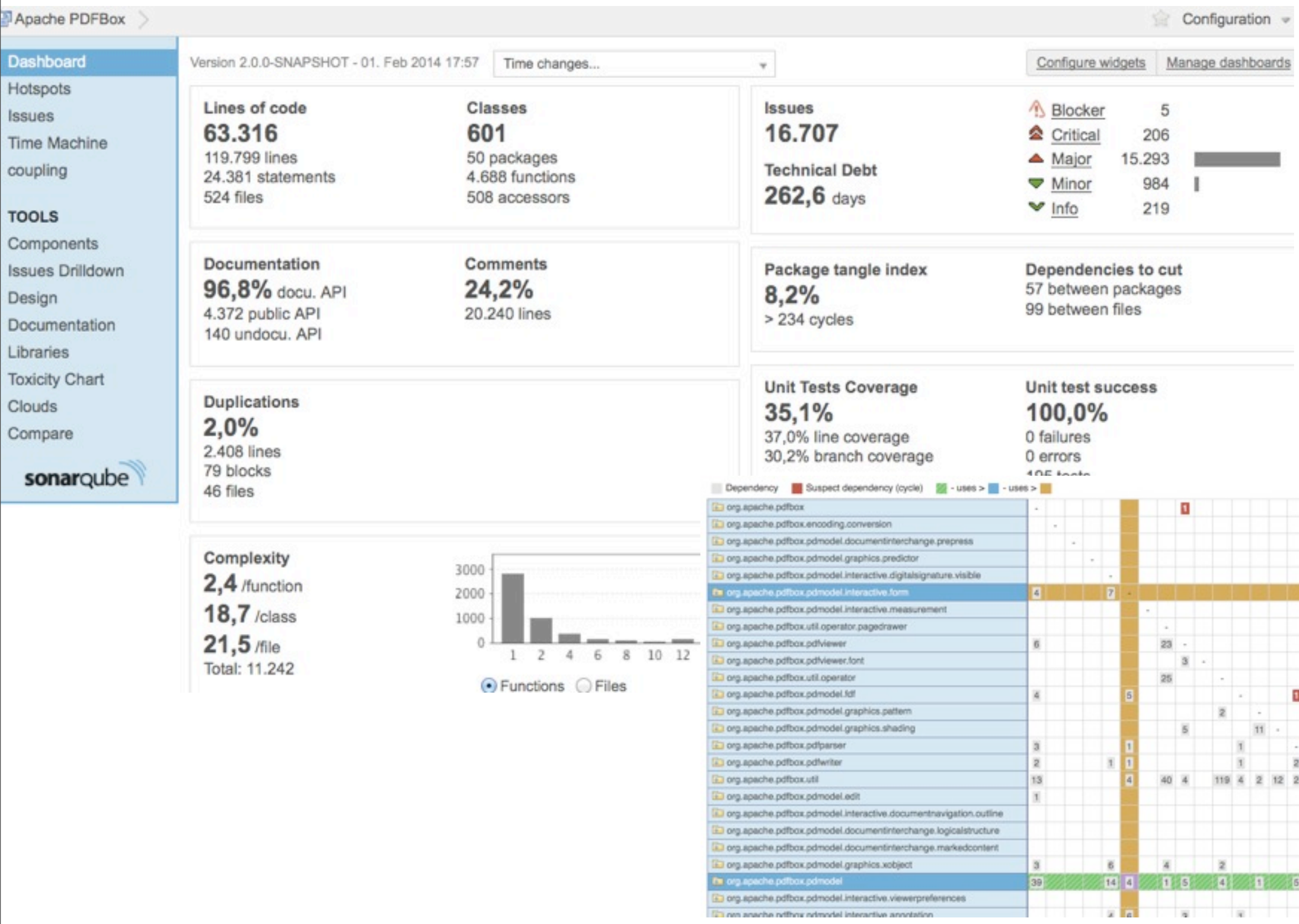
| Q-Ziel | Bedeutung / Szenarien |
|---|---|
| Flexibilität | • Neues csv-Importformat in <4h konfigurierbar |
| Last / Performance | • 250.000 eingelieferte Fotos innerhalb von 24h prozessiert |
| Sicherheit | • Mandant kann niemals Zugriff auf Daten anderer Mandanten erhalten |

Risiken?

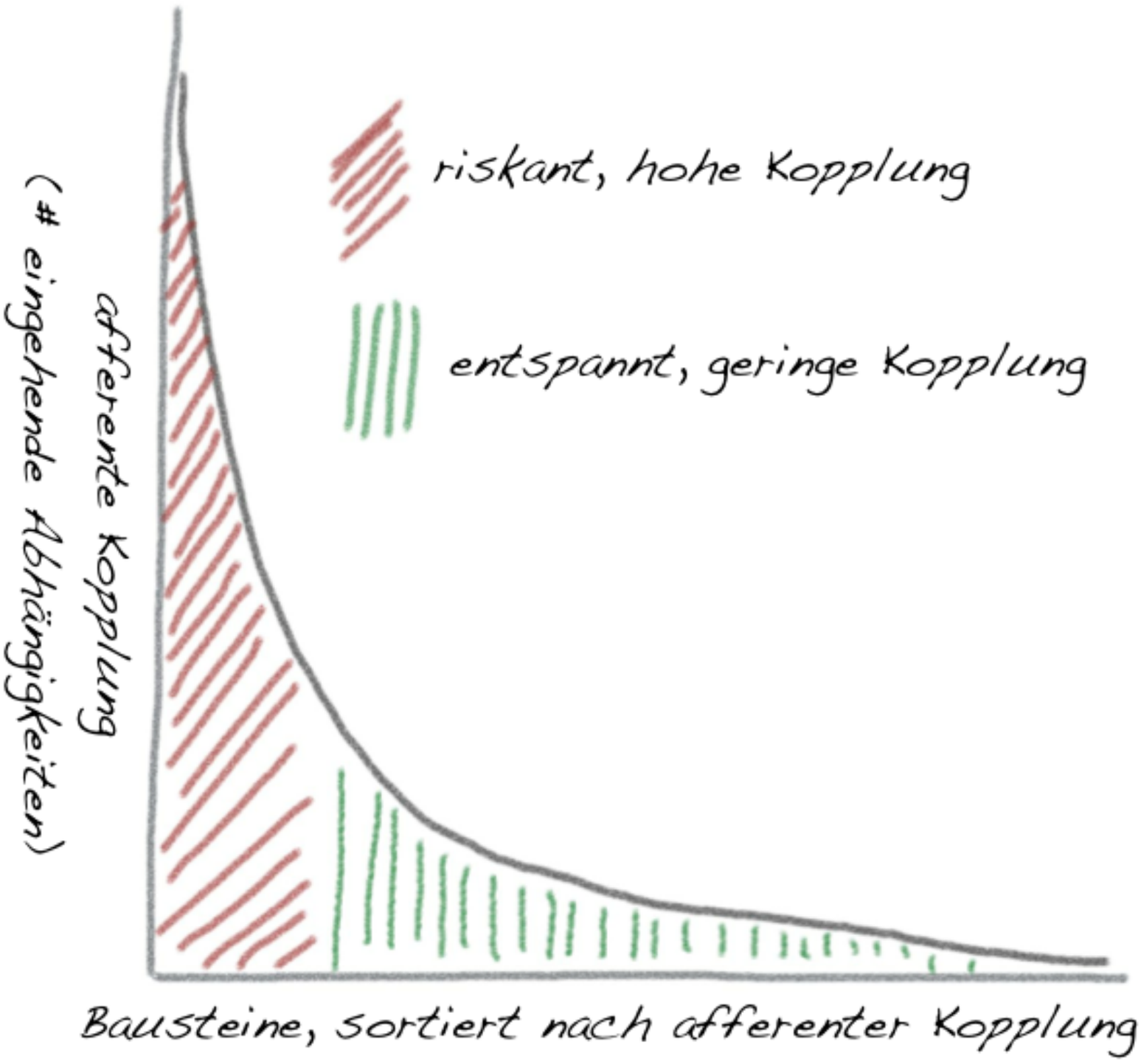
## Lösungs-ansätze

| Architektur-/Lösungsansatz |
|---|
| • Konfigurationssprache für CSV-Parser (Import), auf Basis ANTLR<br>• Syntaxgesteuerter Editor für die Sprache |
| • Bilder als Dateien speichern, Links in DB<br>• Lasttests im DailyBuild<br>• Generator für (Massen-)Testdaten |
| • Mandantenspezifische Daten grundsätzlich in (eigener) VM<br>• Datenlieferungen grundsätzlich in mandantenspezifische Verzeichnisse (ftp-Server)<br>• Unix-Kennungen spezifisch für Mandanten |

# Static Code Analysis
# (here: SonarQube dashboard / Apache PDFbox)



# Static Code Analysis (here: afferent coupling)




- understand
  - root-causes of problems

- determine „value“ of:
  - problems / risks / issues
  - their remedies

- prioritize


Goal:

# Convince Stakeholder

(in their language)


# The Patterns...

- Failure Mode And Effect Analysis
- Impact Analysis
- Root Cause Analysis
- Separate Cause From Effect
- Software Archeology
- View Based Understanding

- Determine Problem Cost
- Determine Feature Value
- Estimate Remedy Cost


| | |
|---|---|
| Issue | „finding“ from the analysis-phase: symptom, problems, risks etc. |
| Reason/Cause | What‘s the (technical, structural...) reason behind the issue |
| Cost / time | what negative impact (in terms of money) does this issue have per time (i.e. per month)? |
| Remedy | what tactics, strategies or actions can resolve this issue, who need to be involved, prerequisites? |
| Cost of remedy | what will the remedies and actions cost? |
| Impact & risk | If remedy is taken what risks might follow, what impact might the remedy have? |


| Prio | Issue / Cause | Cost / time | Remedy | Cost of remedy | Impact & Risk |
|---|---|---|---|---|---|
| 1. | data loss under high load | damaged reputation, sales loss: €50.000/yr | • replace in-memory data transfer by MOM<br>• introduce transaction concept<br>• apply standard integration patterns | • 150 FTE-days<br>• maybe: license cost | • (-) higher complexity of implementation<br>• (+) cleaner architecture<br>• (+) modern middleware |
| 2. | high effort to configure to customer corporate identity | cannot bill total effort to customer, €2.000/ customer | • re-architect UI styling, introduce templating mechanism<br>• re-implement pdf generation | • 30 FTE-days | • existing customers might require additional effort to migrate |
| 3. | pdf generator is rarely used, requires | 25.000€/yr | • migrate from iText-pdf to Apache pdfbox | • 20 FTE-days | • pdfbox is currently not well-documented<br>• existing iTextPdf know-how not immediately useable<br>• learning-curve |

# Experience Report (Logistics)

**Analyze**:

- ✓ ATAM & qualitative analysis
- ✓ Stakeholder interviews

**Evaluate**:

- ✓ Prioritized issues
- ✓ Top-3 as „tasks“ to maintenance team


**solve issues, refactor,**
**re-architect, re-structure,**
**renew, migrate, cleanup**


# The Patterns...

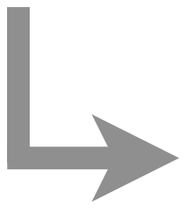
- Automated Tests
- Change by Extension
- Change by Copy
- Branch-for-Improvement
- Refactoring
- Refactoring-Plan
- Refactor-Data-Structures
- Migrate Data
- Keep-Data-Toss-Code
- Improve Code Layout
- Handle-If-Else-Chains
- Isolate Changes
- Extract-Reusable-Component
- Group Improvement Actions
- Schedule Improvement Work

# Change-by-Extension

## Close for change

# Change-by-Extension

Close for change

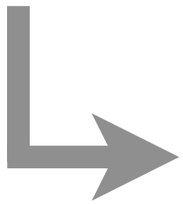


Enable integrateability

(auth/auth, navigation)

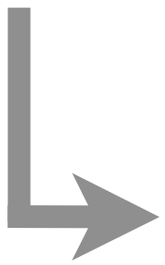
# Change-by-Extension

Close for change



Enable integrateability

(auth/auth, navigation)



Create new system
for new features

# Change-by-Extension

Close for change

↳ Enable integrateability

(auth/auth, navigation)

↳ Create new system for new features → Integrate and/or replace part

# Change-by-Extension

Close for change

→ Enable integrateability

(auth/auth, navigation)

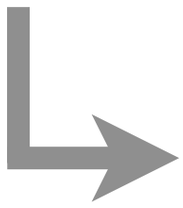
→ Create new system for new features → Integrate and/or replace part

# Change-by-Copy

Close for change

# Change-by-Copy

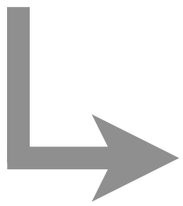
Close for change



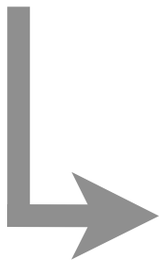
Enable integrateability

# Change-by-Copy

Close for change



Enable integrateability



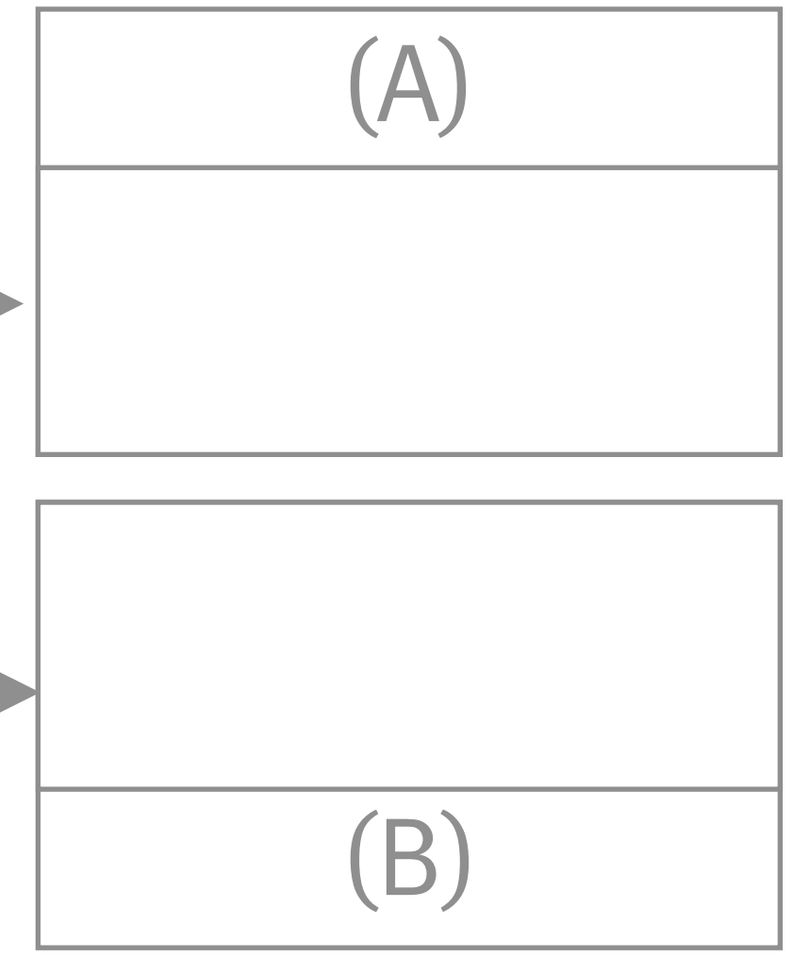
Copy System
to A, B

# Change-by-Copy

Close for change

Enable integrateability

Copy System
to A, B

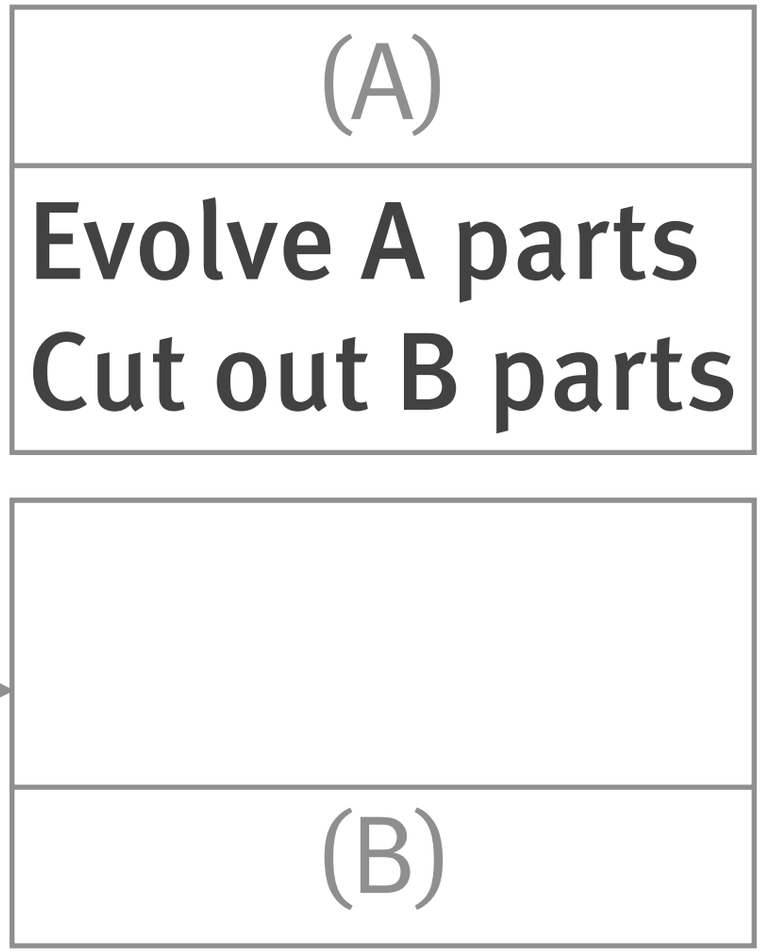


# Change-by-Copy

Close for change

Enable integrateability

Copy System
to A, B



# Change-by-Copy

Close for change

Enable integrateability

Copy System
to A, B

(A)

Evolve A parts
Cut out B parts

Cut out A parts
Evolve B parts

(B)

# Change-by-Copy

Close for change

Enable integrateability

Copy System
to A, B

(A)

Evolve A parts
Cut out B parts

Cut out A parts
Evolve B parts

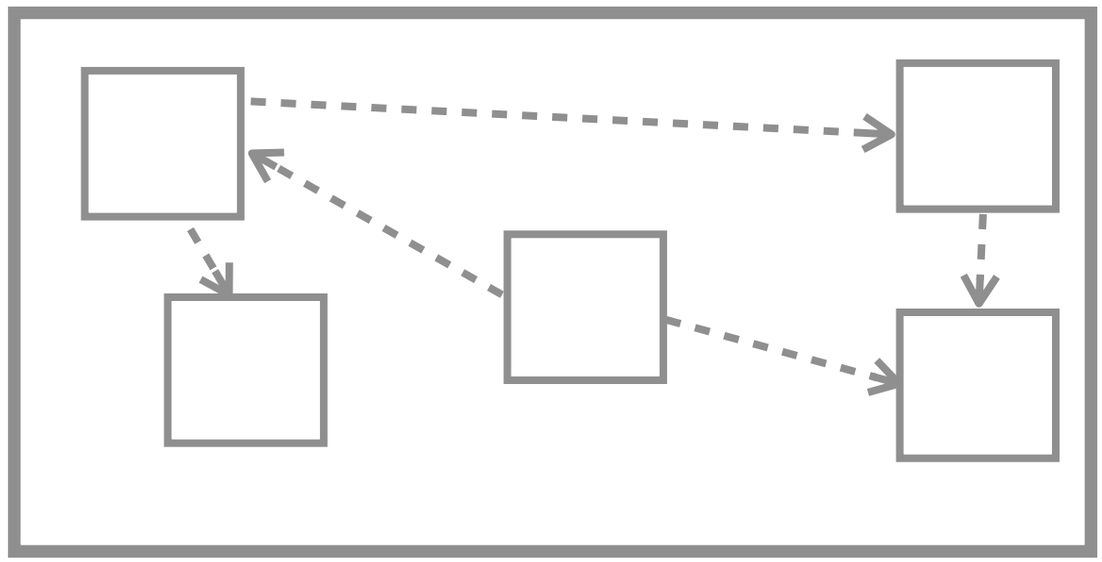
(B)

# Outside-in-Interfaces

Define external, “ideal” interface

Adapt to existing codebase

Transform system to match interfaces

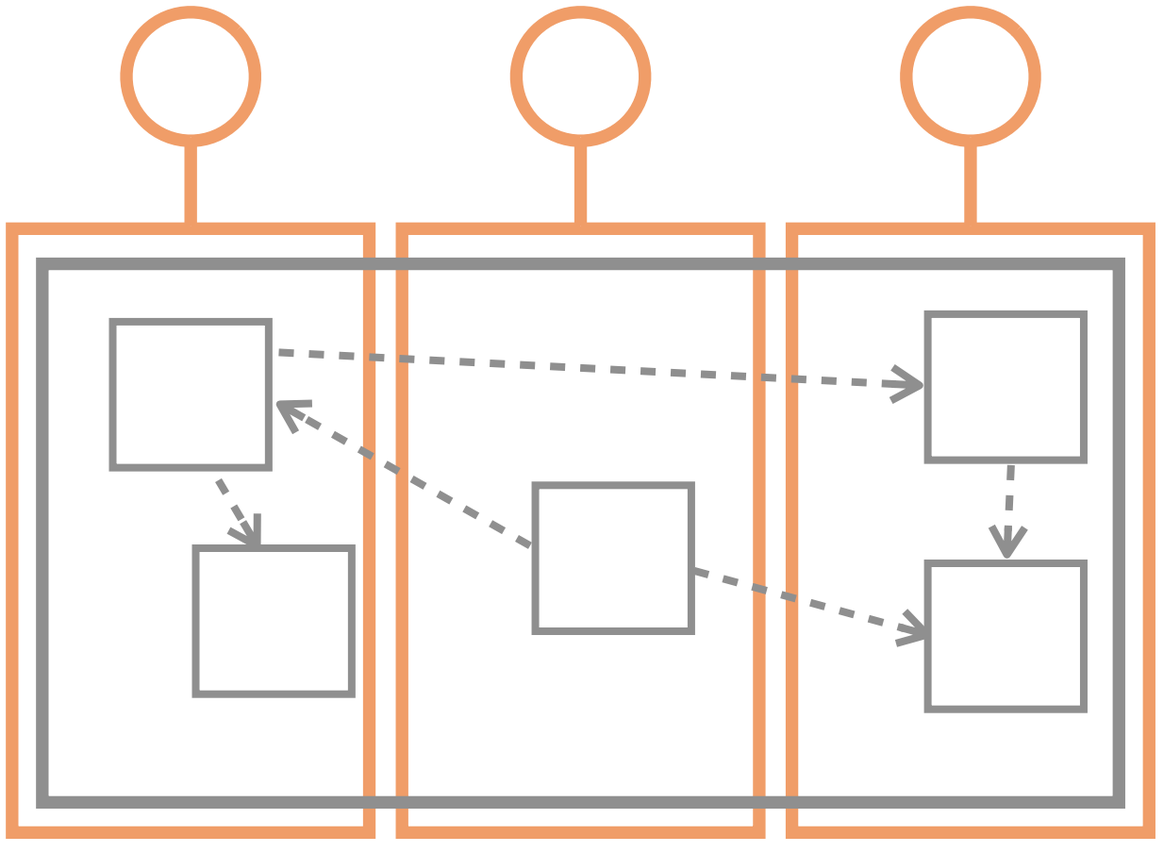


# Outside-in-Interfaces

Define external, “ideal” interface

Adapt to existing codebase

Transform system to match interfaces

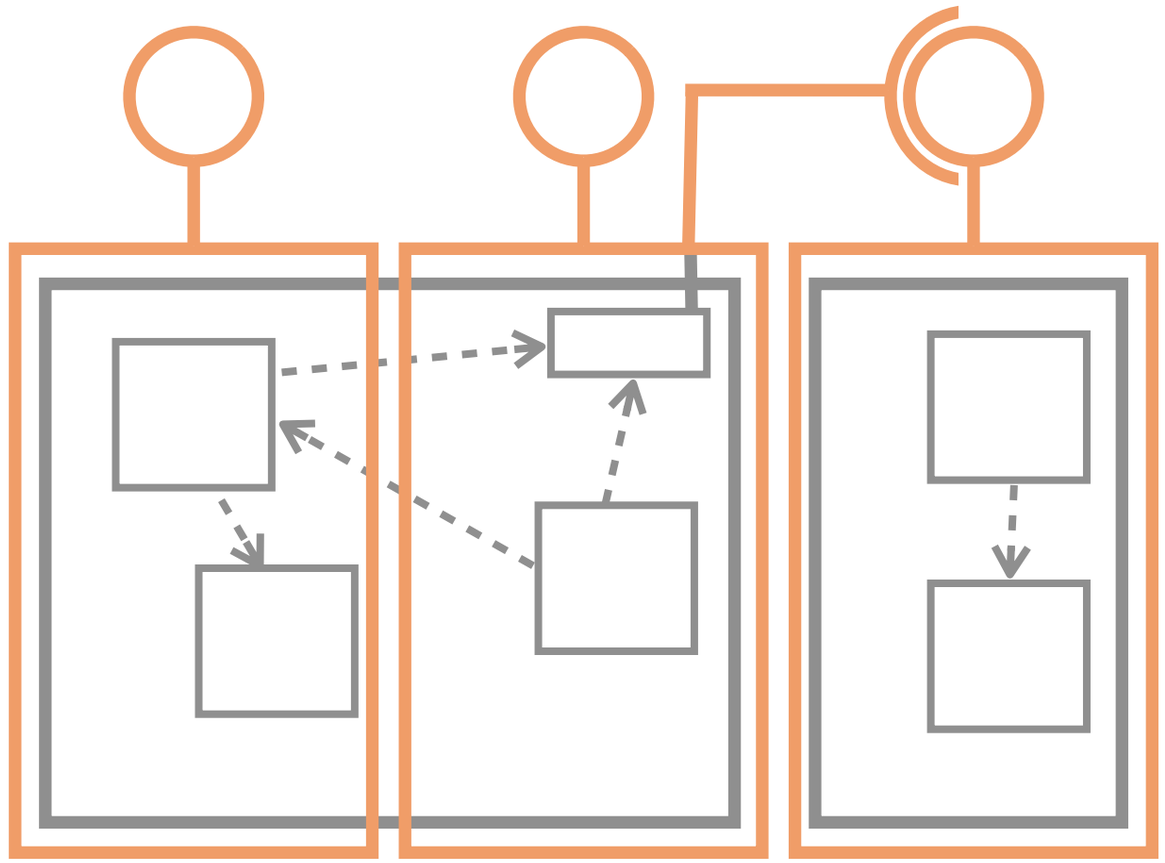


# Outside-in-Interfaces

Define external, “ideal” interface

Adapt to existing codebase

Transform system to match interfaces



# Limit Feature by Client

Add new logic gradually

Enable for specific client (user) groups only

Allow for co-existence

# Remove Flexibility

Analyze customizations by user/client kind

Identify unused/rarely used paths

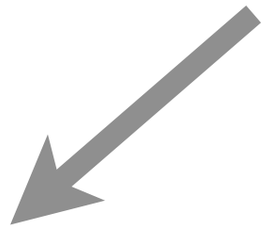
Replace customizable parts with hard-coded alternatives

# Natural Death

Separate old and new use cases + products

# Natural Death

Separate old and new use cases + products



Implement new logic in new parts

# Natural Death

Separate old and new use cases + products

Implement new logic in new parts

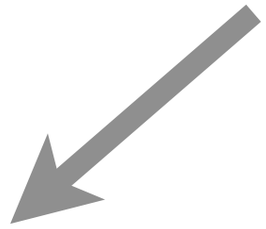
Retire old parts once data is “dead”

# Architecture Backlog

Treat issues as (special) features
after analysis/evaluation!

# Architecture Backlog

Treat issues as (special) features
after analysis/evaluation!



involve
“Product owner“

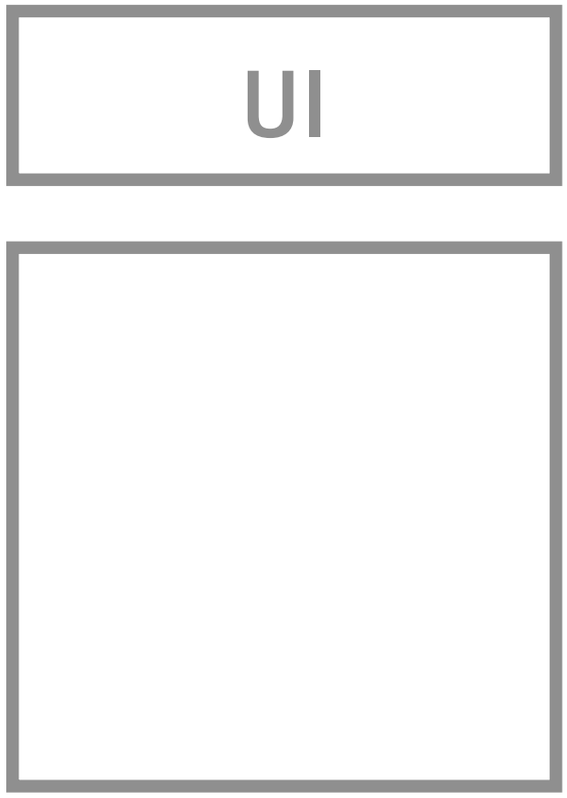
# Architecture Backlog

Treat issues as (special) features
after analysis/evaluation!

involve
“Product owner“

involve
“Scrum master“

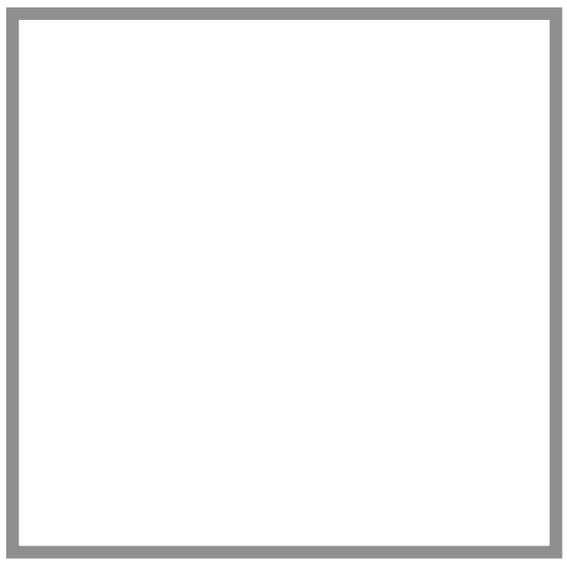
# Front-end switch




Create new surface area (UI)

Gradually replace backend parts

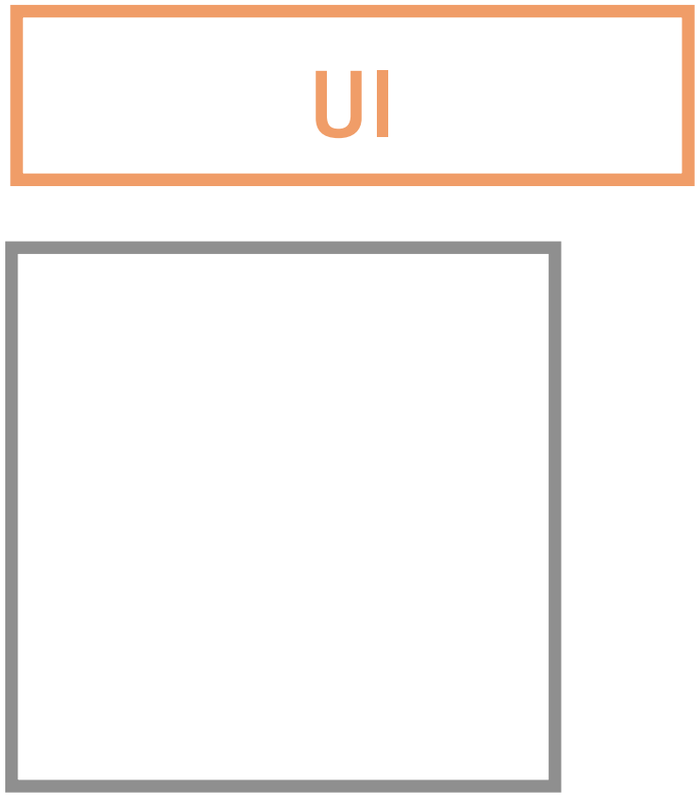
# Front-end switch



Create new surface area (UI)

Gradually replace backend parts

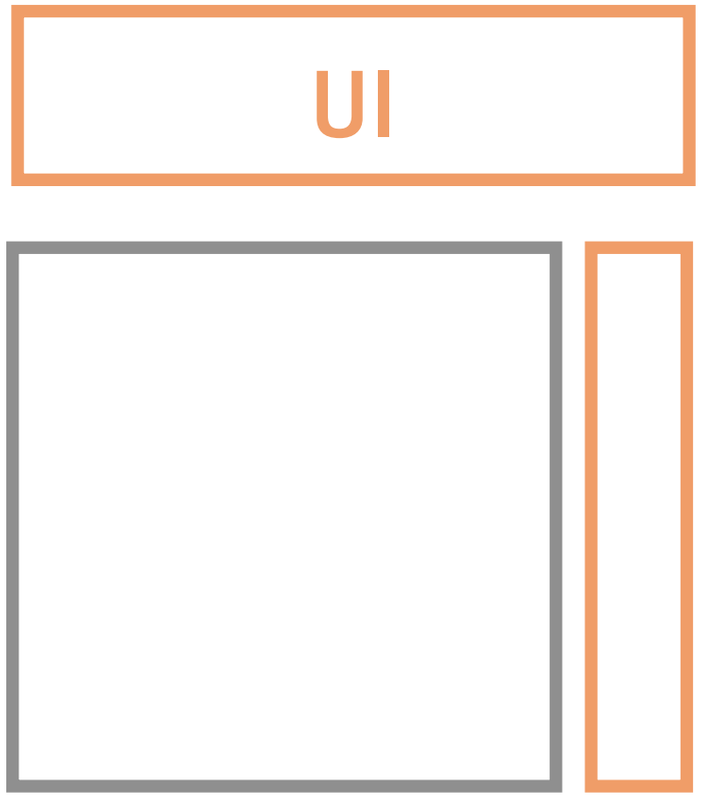
# Front-end switch




Create new surface area (UI)

Gradually replace backend parts

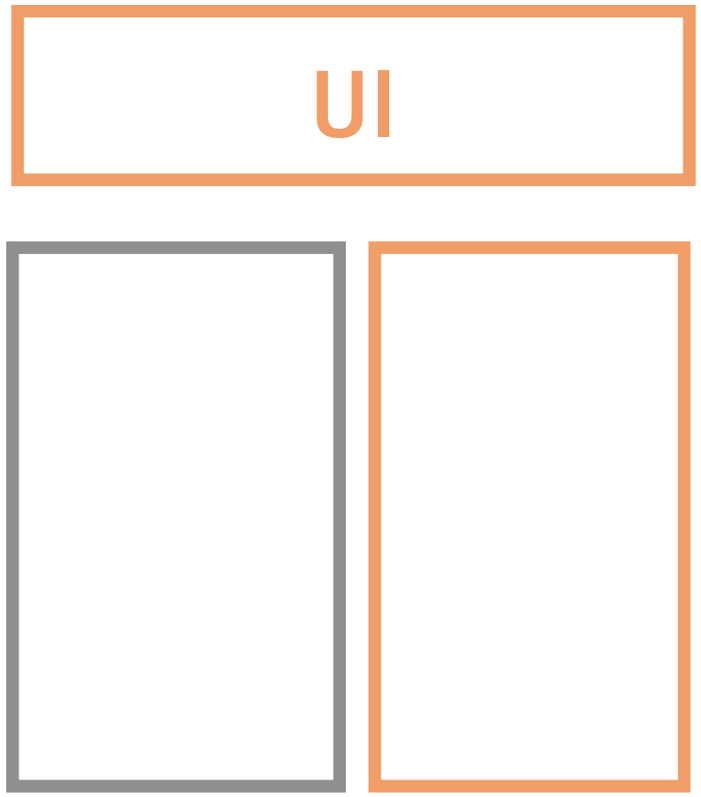
# Front-end switch




Create new surface area (UI)

Gradually replace backend parts

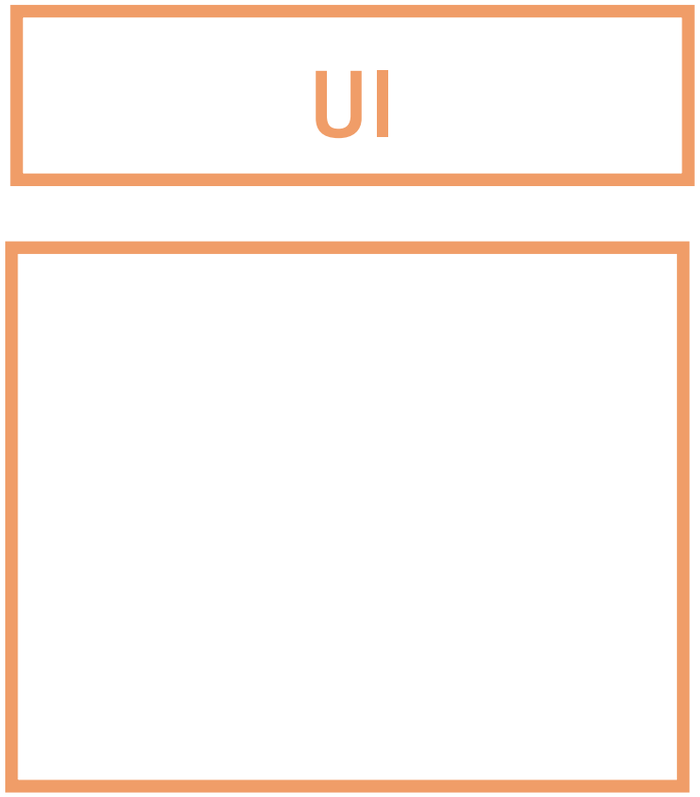
# Front-end switch




Create new surface area (UI)

Gradually replace backend parts

# Front-end switch




Create new surface area (UI)

Gradually replace backend parts

# «Your Pattern or Practice Here »

Contributions welcome!

http://aim42.org

# Fazit

These:

# Practices ersetzen Erfahrung nicht

These:

# Erprobte Vorgehensweisen erleichtern Fokus aufs Wesentliche

These:

# Grundsätzlich können wir ändern

aim 42

These:

# Techies müssen Notwendigkeit von Änderungen verkaufen

aim42

These:

Logisches

# Denken trotz Pattern- und Practice-Katalog erforderlich

These:

# Spezifische Sprache erleichtert Kommunikation

# Questions?
# Comments?


Dr. Gernot Starke, @gernotstarke
gs@gernotstarke.de
http://gernotstarke.de

Stefan Tilkov, @stilkov
stefan.tilkov@innoq.com
http://www.innoq.com/blog/st/



innoQ

www.innoq.com

**innoQ Deutschland GmbH**

Krischerstr. 100
40789 Monheim am Rhein
Germany
Phone: +49 2173 3366-0

Ohlauer Straße 43
10999 Berlin
Germany
Phone: +49 2173 3366-0

Robert-Bosch-Straße 7
64293 Darmstadt
Germany
Phone: +49 2173 3366-0

Radlkoferstraße 2
D-81373 München
Germany
Telefon +49 (0) 89 741185-270

**innoQ Schweiz GmbH**

Gewerbestr. 11
CH-6330 Cham
Switzerland
Phone: +41 41 743 0116

**info@innoq.com**

# References (excerpt)

## Books:

- ✓ M.Lippert, S.Roock: Refactoring in Large Software Projects: Performing Complex Restructurings Successfully (2006)
- ✓ M. Fowler: Refactoring
- ✓ M. Feathers: Working Effectively with Legacy Code.

## Research:

- ✓SERIOUS: „Software Evolution, Refactoring, Improvement of Operational & Usable Systems", ITEA-Eureka Project (2008)
  http://www.hitech-projects.com/euprojects/serious/deliverables.htm
- ✓ ATAM: Architecture Tradeoff Analysis Method, Software Engineering Institute, http://www.sei.cmu.edu/architecture/tools/evaluate/atam.cfm

## Online:

- ✓practices

**see http://aim42.org for more**

# Experience Report (Industry)

**Analyze**:

- ✓ ATAM & qualitative analysis
- ✓ (extensive) stakeholder interviews
- ✓ Software Archeology

approx 50 issues, problems found

**Evaluate**:

- ✓ Identifies ~5 issues as „safety critical production risk“
- ✓ Identified architectural & code causes
- ✓ Cost of those issues: >100T€

**Improve**:

- ✓ Isolate-changes, introduce (internal) interfaces
- ✓ removal became high-priority for developers