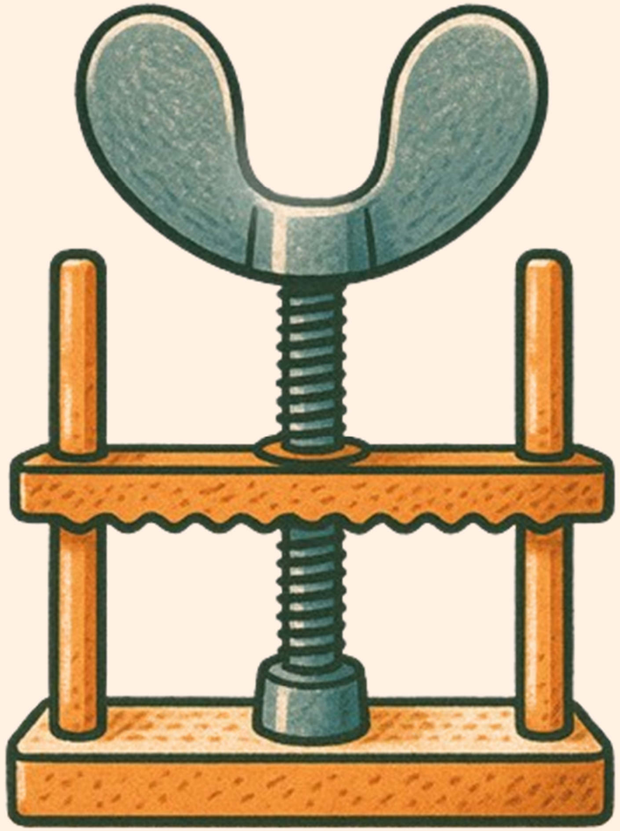


INNOQ Technology Day 2025, 20.11.2015, online

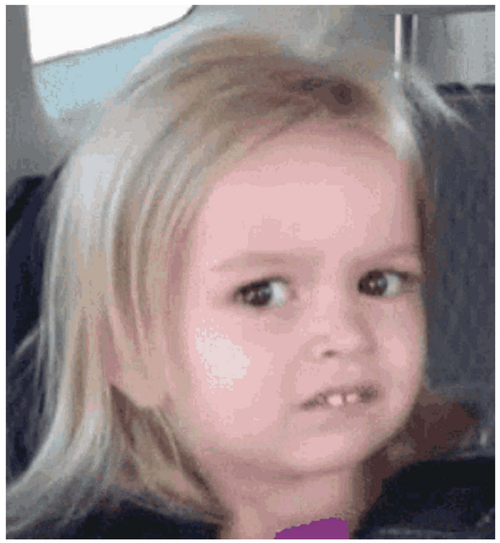
Architektur-Governance: Daumenschrauben für Softwareentwickelnde

Markus Harrer

Senior Consultant

INNOQ

Architektur-Governance?

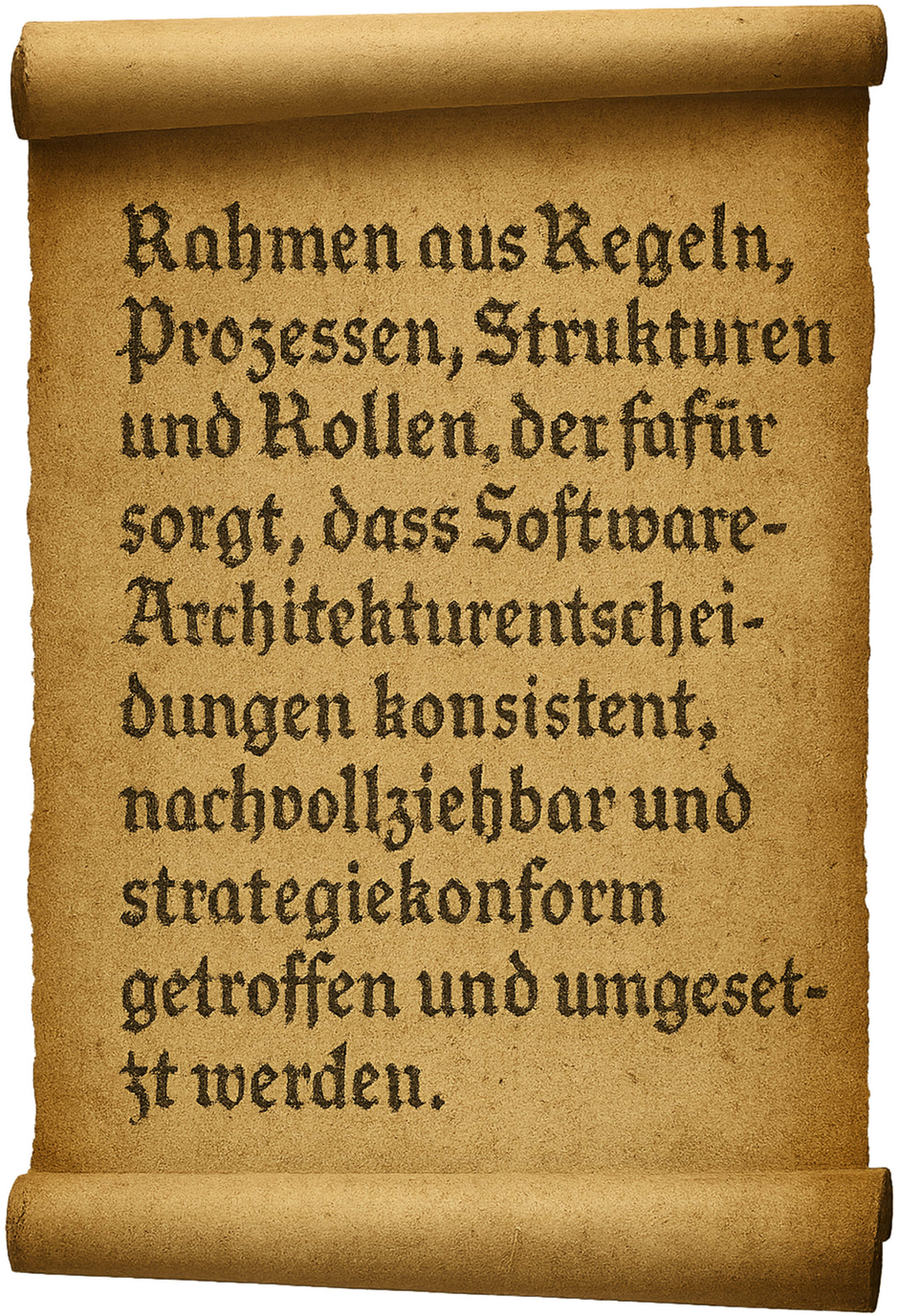


Regeln?

Prozesse?

Strukturen?

Rollen?



Regel A

Ach ja, noch ne Regel

Regel B

IT

= Department Of No

naysayers

Information Security Officer

Enterprise Architects

Softwarearchitekten

Regel dies

Regel das

Regel bla

Regeln? WtF!

Als Entwickler:in möchte ich doch kreativ, frei und selbstbestimmt arbeiten

WTF: Welch’ törichte Frage

Kubernetes

Authentifizierung

Continuous Delivery

Gitflow

Monitoring

Domain Driven Design

Code Reviews

API-Dokumentation

Alerting

Microservices

Code-Doku

AI

Input Validation

Commit Messages

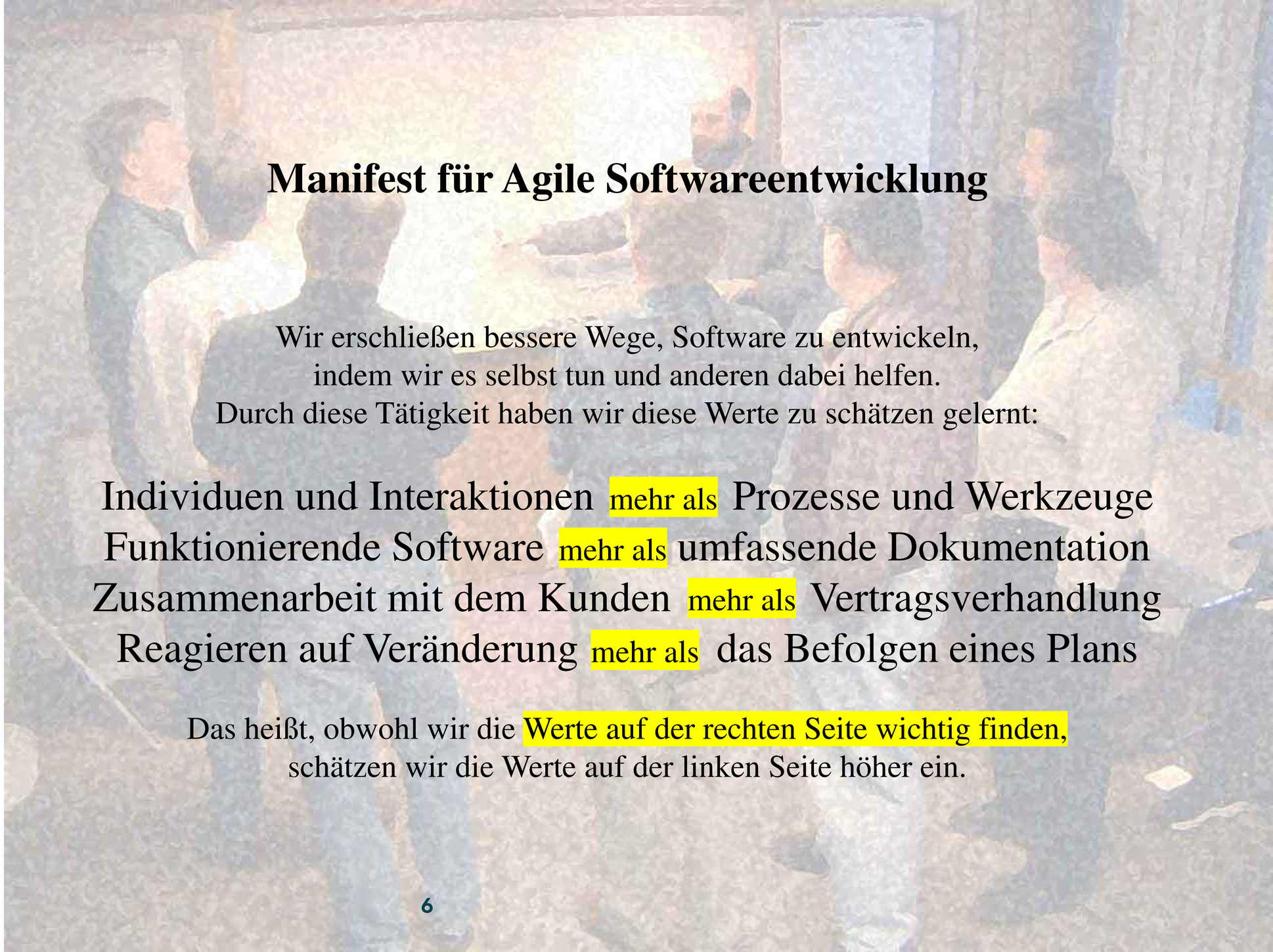
RESTful APIs

Logging

Datenverschlüsselung

Performance-Tests

Hallo, agil?



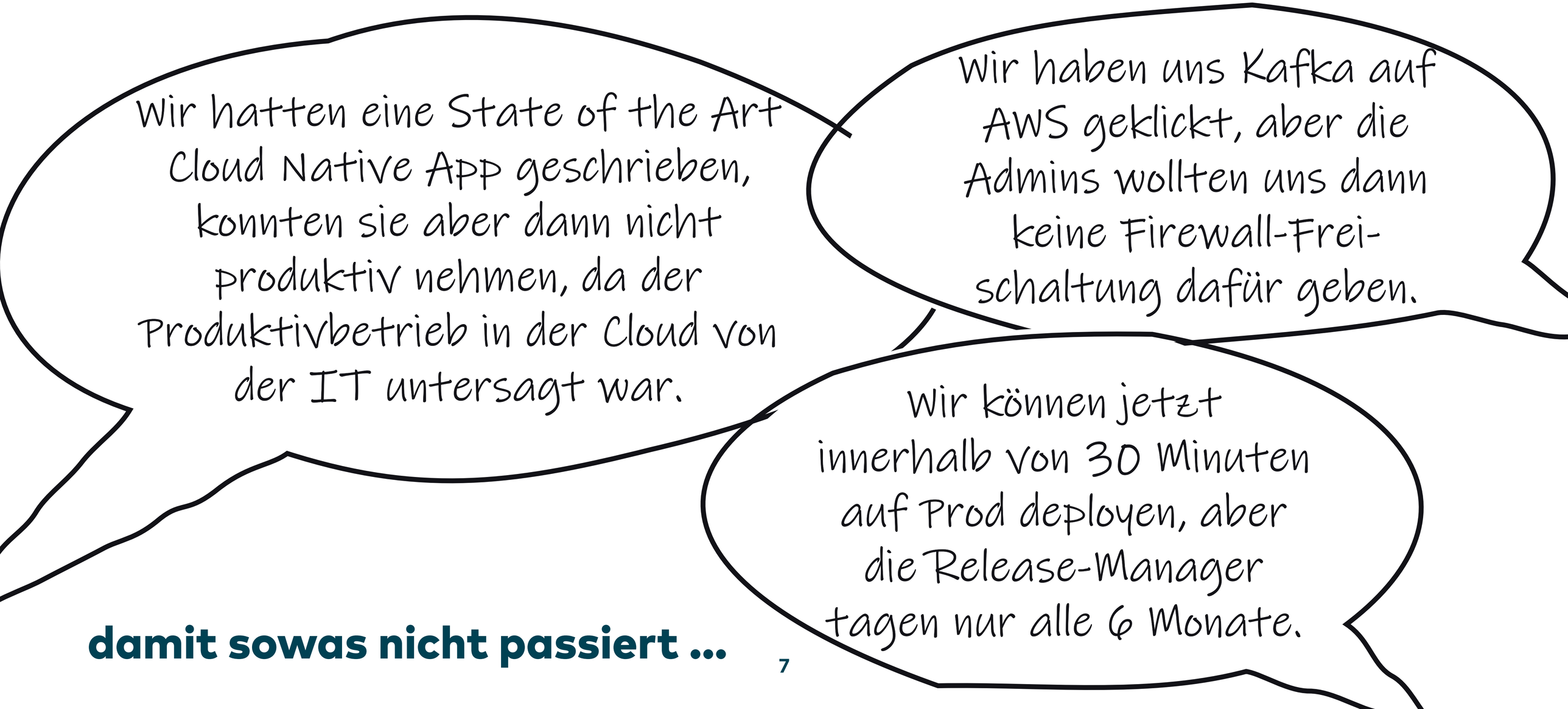
Manifest für Agile Softwareentwicklung

Wir erschließen bessere Wege, Software zu entwickeln,
indem wir es selbst tun und anderen dabei helfen.
Durch diese Tätigkeit haben wir diese Werte zu schätzen gelernt:

Individuen und Interaktionen mehr als Prozesse und Werkzeuge
Funktionierende Software mehr als umfassende Dokumentation
Zusammenarbeit mit dem Kunden mehr als Vertragsverhandlung
Reagieren auf Veränderung mehr als das Befolgen eines Plans

Das heißt, obwohl wir die Werte auf der rechten Seite wichtig finden,
schätzen wir die Werte auf der linken Seite höher ein.

Architektur-Governance ist wichtig



damit sowas nicht passiert ...

Wo dürfen wir regieren?

Umstellung des Hosting Providers

Einführung einer neuen Programmiersprache im Unternehmen

Anpassungen im eigenen Datenbank-Schema

Konkrete Umsetzung von Features im Code

Änderungen an der Public API

Auswahl einer Open-Source-Bibliothek

Kleine Experimente mit einer neuen Programmiersprache

Entscheidungsband

Wer darf was entscheiden?

Irreversibel

- Globaler Konsens notwendig
- (Fast) Unmöglich zu ändernde Festlegungen
- Gut durchdachte Lösungen notwendig

Reversibel

- Lokale Entscheidung
- Niedrige Kosten beim Zurücknehmen einer Entscheidung
- Ad-Hoc-Entscheidungs-findung akzeptabel

Adaptiert von Sam Newman: Monoliths to Microservices

Wie wird entschieden bei uns?

Business-Monarchie

Gruppe von Vorständen oder einzelner Vorstand der Geschäftsbereiche (ohne die IT)

IT-Monarchie

Gruppen IT-Manager oder einzelner IT-Managern (ohne die Manager der Geschäftsseite)

Feudal

Manager von Geschäftseinheiten, Manager von Schlüsselprozessen oder ihre Delegierten

Umstellung des Hosting Providers

Auswahl einer Open-Source-Bibliothek

Kleine Experimente mit einer neuen Programmiersprache

Föderal

Manager von Geschäftsseite und IT (auch aus verschiedenen Ebenen)

IT-Duopol

IT-Manager und eine von Gruppe Manager auf hohe Geschäftsebene

Anarchie

Von jedem Menschen selbst. Jeder Mensch entscheidet für sich selbst.

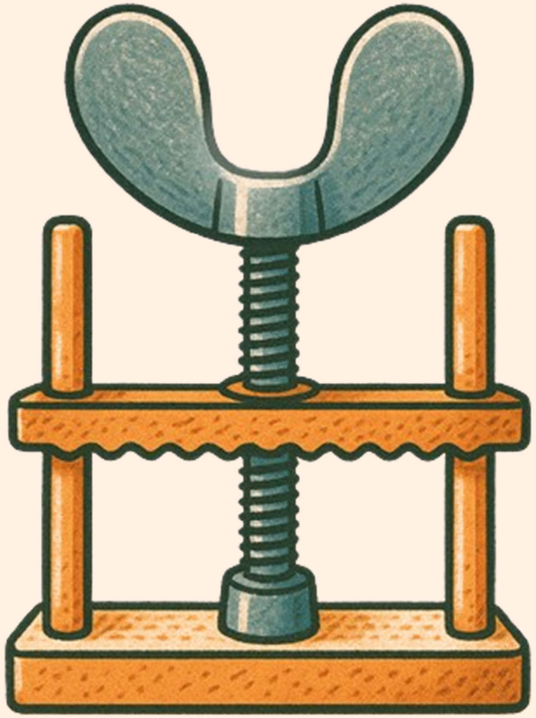
Von dem Artikel „IT Governance on One Page“

Ziel dieses Vortrags

Das richtige Drehmoment für die Daumenschrauben finden

„Geregeltes Quälen“ das nicht nervt

Spoiler: kann weh tun



Mehr Infos: https://wissen.schloesserland-sachsen.de/blog/detail/sprichwoertlich-die-daumenschrauben-anlegen/

Woher kommen denn diese ganzen Regeln?

Mögliche Einflüsse von außen

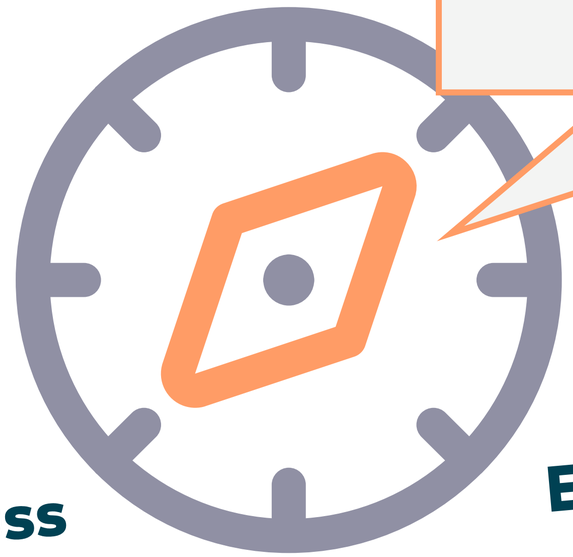
- Gesetzgebung
- Normen / Standards
- Regulatorische Anforderungen (z. B. DSGVO, BaFin, HIPAA)
- Sicherheitsrichtlinien (z. B. ISO/IEC 27001, OWASP, BSI)
- Branchenspezifische Vorschriften (z. B. BAIT/VAIT, Medizinproduktegesetz, Luftfahrtstandards)
- Industrie-Konsortien und -Allianzen (z. B. W3C, GAIA-X)
- …

Mögliche Einflüsse von innen

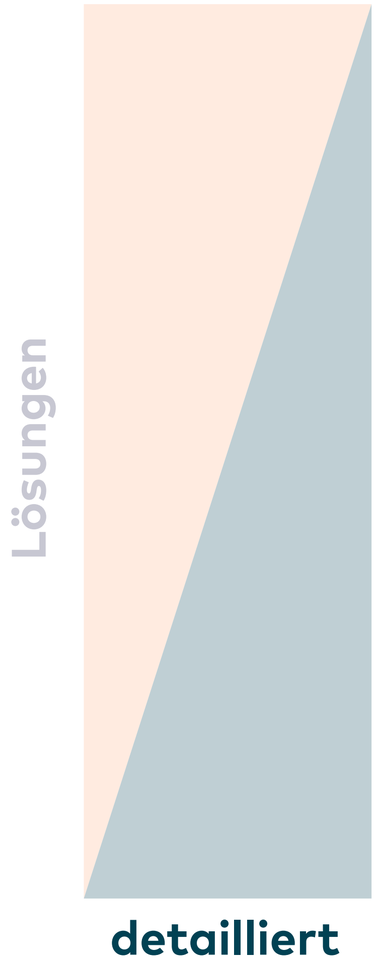
- Wachstumspläne (z. B. Internationalisierung, neue Märkte)
- Digitalisierungsstrategie / Cloud Transformation
- Erfahrung und Skills im Team
- Verfügbare Kapazitäten (FTEs, Zeit, Geld, ...)
- Erwartungen von Fachabteilungen, Controlling, Legal, ...
- Vorhandene andere Systeme, Plattformen und Infrastruktur
- ...

Architektur-Governance

Das adressieren wir in unseren Softwaresystemen jetzt so und so!



→ Hierarchische Einschränkung des Lösungsraums



Ebene	Entscheidung (die für mich bereits getroffen wurde)
Investoren	„Schaut, dass ihr +10% Marktanteile bekommt!“
Einschränkung	
Geschäftsstrategie	„Wenige Partner für maximale Effektivität“
Einschränkung	
IT-Strategie	„Oracle ist unser strategischer Partner“
Einschränkung	
Enterprise Architecture	„Buy before Build“, „Best of Suite“
Einschränkung	
Softwarearchitektur	Oracle JVM, Oracle Database, Oracle Cloud Infrastructure

JVM: Java Virtual Machine

Architektur-Governance ist Mindset!

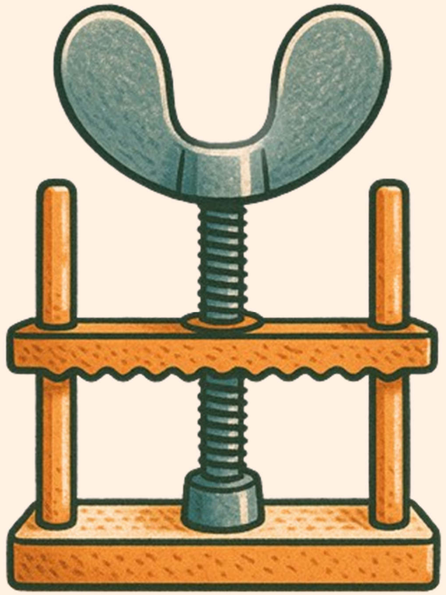
Negative Sicht

Es gibt ganz viel Kram,
der auf die Nerven geht!

Positive Sicht

Es gibt bereits viele Entscheidungen,
die mir bereits abgenommen wurden.

Was bedeutet das für uns?



Wir müssen uns damit leider auseinandersetzen!

Stile der Architektur-Governance

bestimmend

begleitend

"to **rule** without sovereign power and usually **without having** the **authority** to determine basic policy"

"to **manipulate**"

"to **control**"

Verschiedene Auslegungen

"to **direct**, or **strongly influence** the **actions** and conduct of"

"to **serve** as a precedent or **deciding principle** for"

Definitionen: Merriam Webster

bestimmend

begleitend

Extrinsisch motiviert

Ehre

Ruhm

Geld

Verschiedene Motivatoren

Mastery

Purpose

Autonomy

Intrinsisch motiviert

bestimmend

begleitend

hart & detailliert

Strafen

Verfahren

Regeln

Feedback

Verschiedene Umsetzungen

Vorgaben

Praktiken

Prinzipien

Werte

Weich & offen

Stile der Architektur-Governance

bestimmend

Tayloristisch

Zerlegung von Arbeit in kleinste, standardisierte Einzelschritte

Trennung von Denken (Planung) und Handeln (Ausführung)

Vorgaben und Kontrolle durch Management

begleitend

Agile / Lean

Ganzheitliche Aufgaben im Team, Verantwortung über Features oder User Stories

Integrierte **Verantwortung** (Team denkt und handelt selbst)

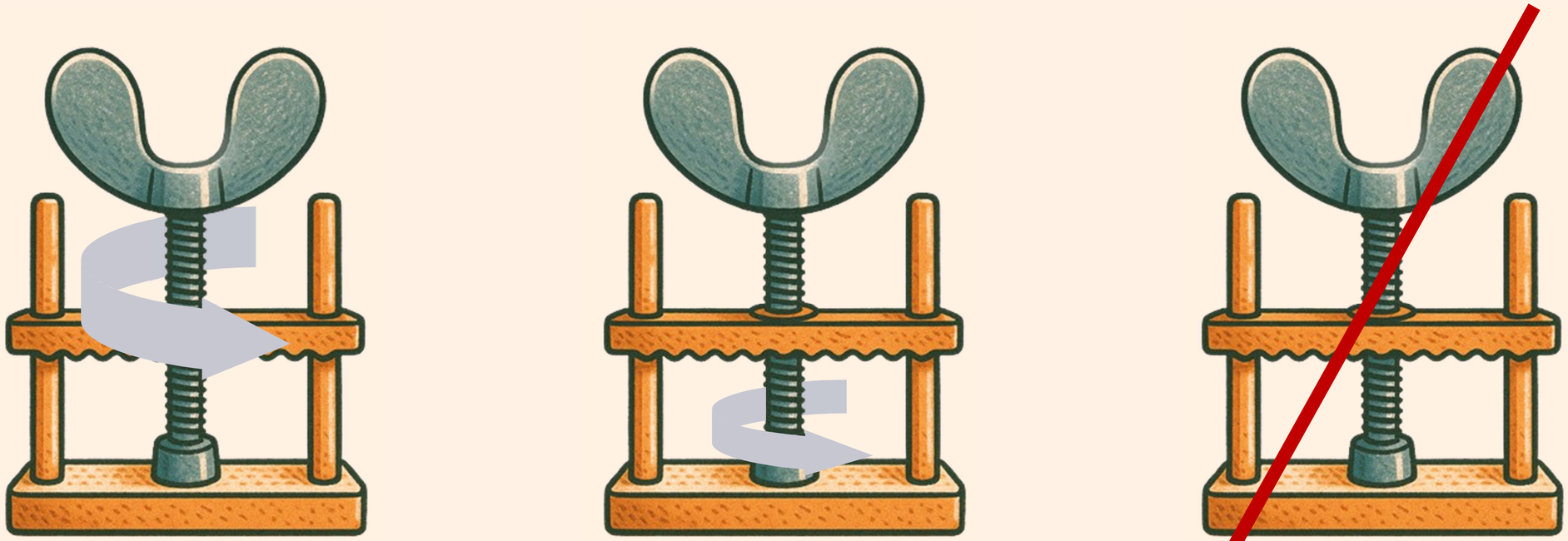
Selbstorganisation des Teams, über Prinzipien, Ziele, Vertrauen

Ziel: **Maximale Effizienz und Produktivität**

Dreyfus-Modell / Kompetenzstufen

Angelehnt an https://www.brainbok.com/guide/pm-fundamentals/interpersonal-and-team-skills/dreyfus-model-of-skill-acquisition/

Was bedeutet das für uns?



Neuling	fortgeschrittener Anfänger	Gewandter	Kompetenter	Experte

Das richtige Drehmoment hängt von Fähigkeiten ab!

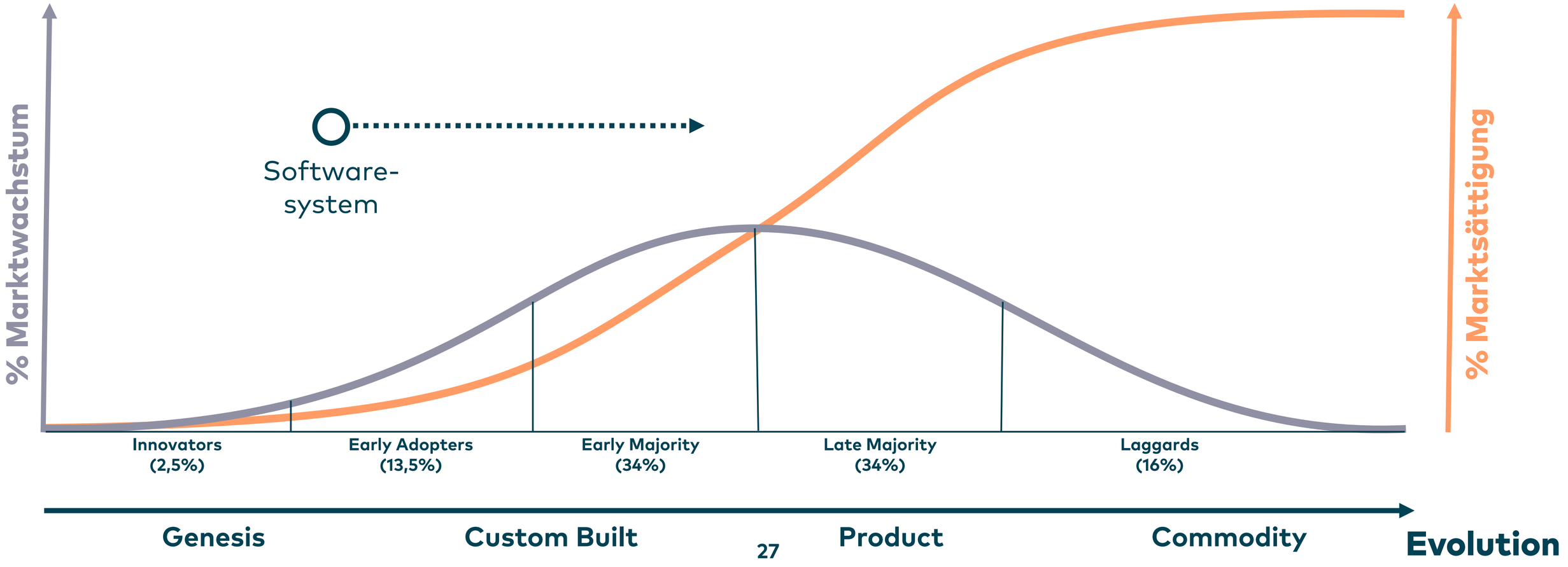
Evolution der Governance

Falls Software erfolgreich ist …

„Alles entwickelt sich durch den Wettbewerb von Angebot und Nachfrage“ – **Simon Wardley**

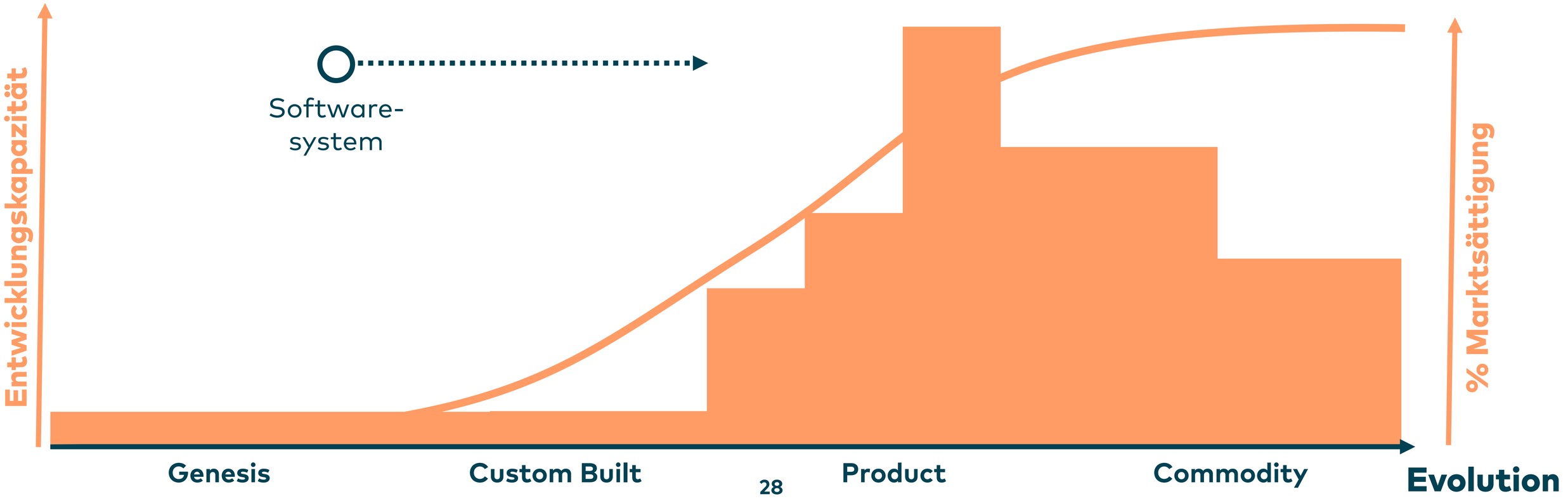
Wer will unsere Software?

Nachfragewettbewerb (oder: Technology Adoption Curve FTW)

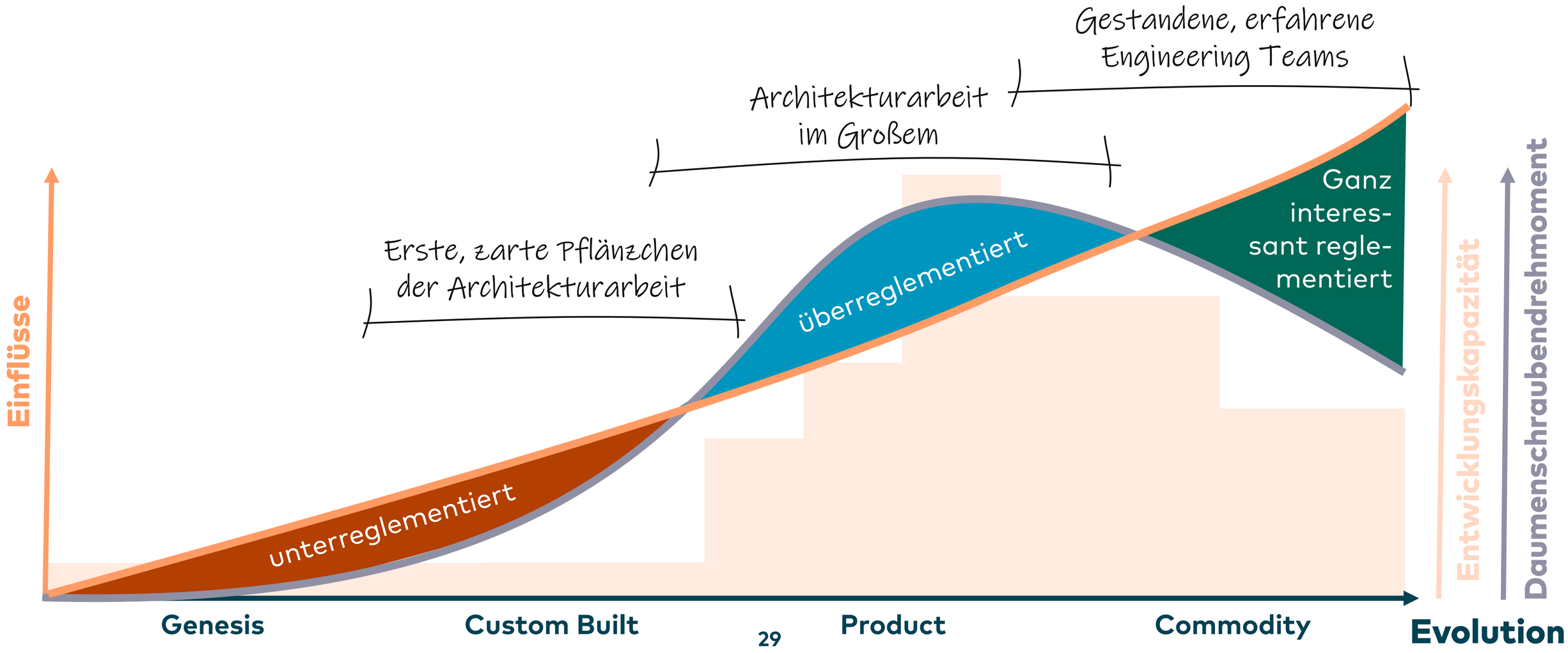


Wer muss mitarbeiten?

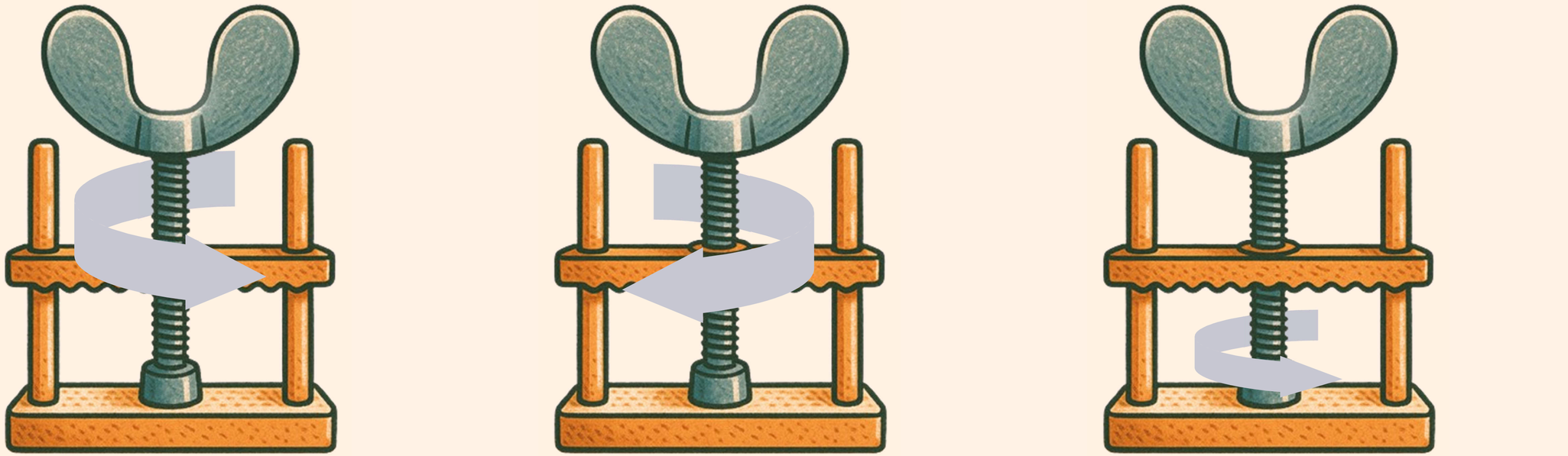
Angebotswettbewerb: Lieferfertigkeit muss nachziehen



Wie stark müssen wir regeln?



Was bedeutet das für uns?



Das richtige Drehmoment für die Daumenschrauben zu finden, ist eine kontinuierliche Aufgabe!

Strategien der Governance

Governance-Strategien

zentral

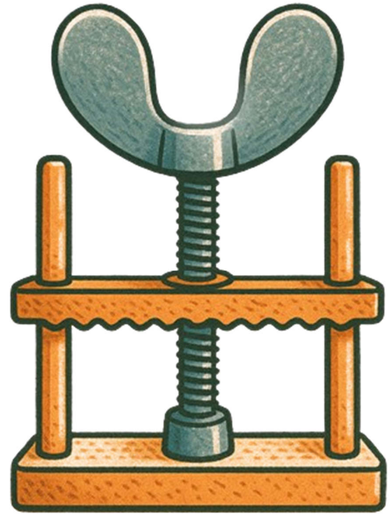
dezentral

Fixe Vorgaben
Bis ins Detail durchentschiedene Lösung

Paved Roads
Angebot von alternativen Lösungen

Multi Level
Verteilte, hierarchische Standards

API Mandate
Individuelle Standards, aber standardisierte Schnittstellen



Stark angelehnt an Mark Richards “Software Architecture Monday, Lesson 62 - Enterprise Architecture Strategies”

Governance-Strategien

zentral

dezentral

Fixe Vorgaben

Bis ins Detail durchentschiedene Lösung

Paved Roads

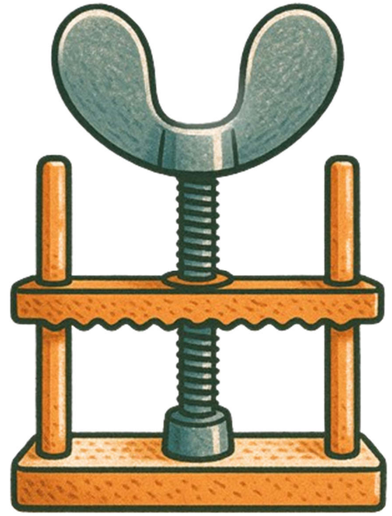
Angebot von alternativen Lösungen

Multi Level

Verteilte, hierarchische Standards

API Mandate

Individuelle Standards, aber standardisierte Schnittstellen



Stark angelehnt an Mark Richards “Software Architecture Monday, Lesson 62 - Enterprise Architecture Strategies”

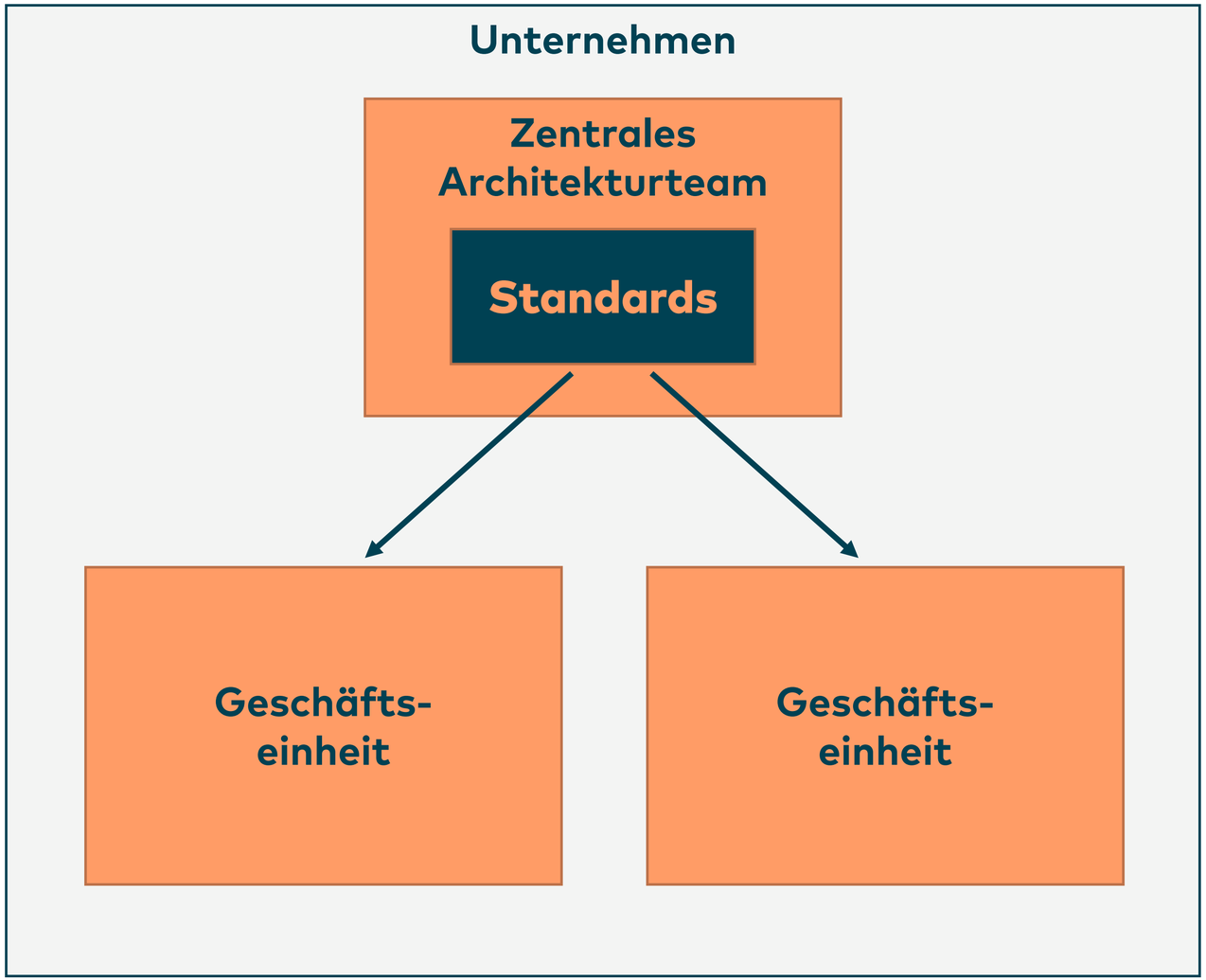
Fixe Vorgaben

zentral

+ reduziert Komplexität
+ weniger Zeitbedarf bei Entscheidungen
+ hohe Wiederverwendung
+ kostengünstig

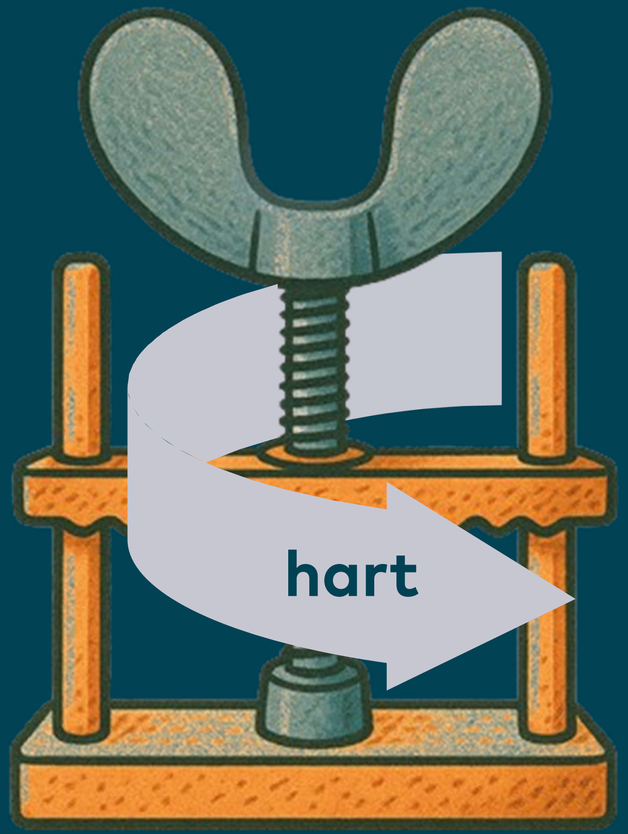
- passt evtl. nicht überall
- schwer, Akzeptanz zu schaffen
- nicht jede(r) ist glücklich
- starke Kontrolle notwendig

Alles festgelegt



Fixe Vorgaben

zentral



Ziel | Vorgabe | Team-Lösung

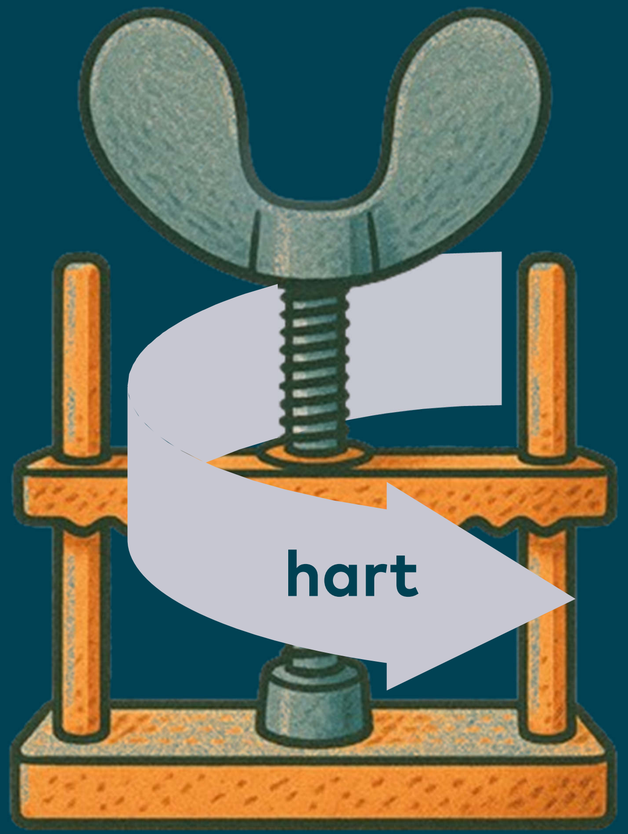
Alle Entwicklungsteams müssen Oracle JDK 21.0.6 verwenden

Flexible Entwicklungs-planung

Standards

Fixe Vorgaben

zentral



Ziel | Vorgabe | Team-Lösung

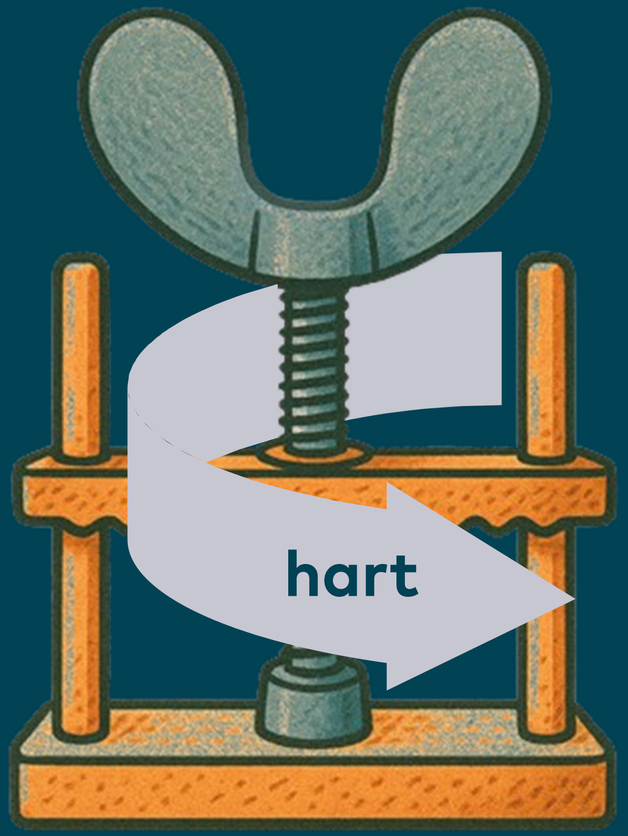
Der Default-Regelsatz von SonarQube meldet keine Major und Critical Findings

Hohe Code-Qualität

Standards

Fixe Vorgaben

zentral



Ziel | Vorgabe | Team-Lösung

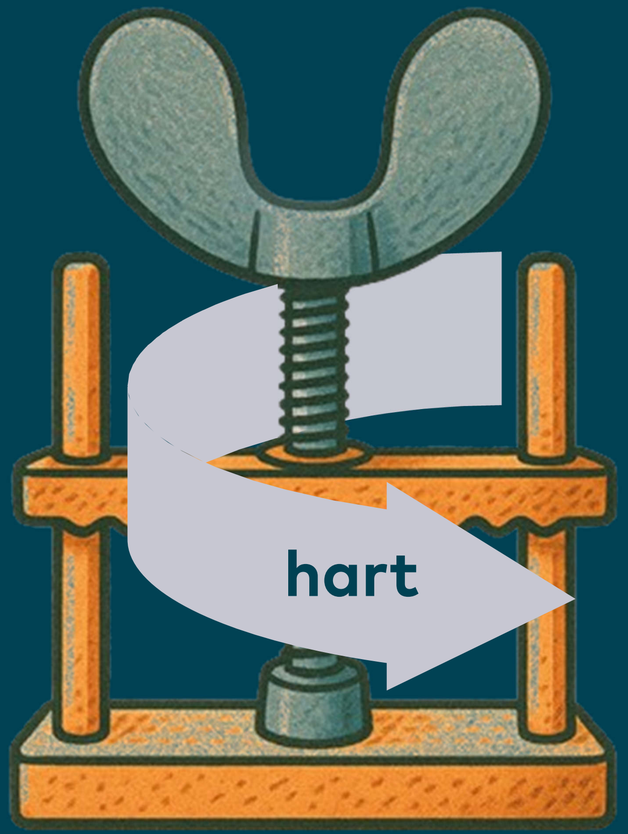
Die Test-Coverage der Unit-Tests muss min. 70 % betragen.

Funktio-nierende Software

Standards

Fixe Vorgaben

zentral



Was soll schon schief gehen?

Teams könnten anfangen zu cheaten, wenn der Sinn dahinter nicht verstanden wurde!

Foto: Kantenflimmern CC BY-SA 2.0
https://www.flickr.com/photos/kantenflimmern/3078108108/

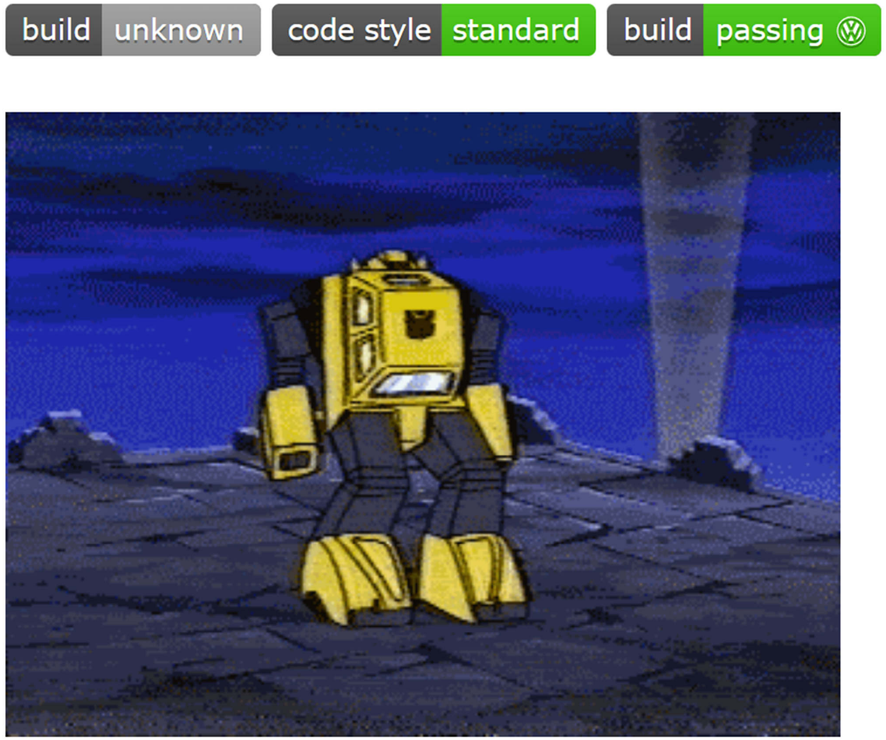
```
protected void processRequest(HttpServletRequest request,
  HttpServletResponse response) throws Exception {
    locale = LocaleResolver.getLocale(request);
    EventCRFBean ecb =
      (EventCRFBean)request.getAttribute(INPUT_EVENT_CRF);
    SectionBean sb = (SectionBean)request.getAttribute(SECTION_BEAN);
    <495 Lines of Code>
    processRequest2(request, response);
}

protected void processRequest2(HttpServletRequest request,
  HttpServletResponse response) throws Exception {
  FormProcessor fp = new FormProcessor(request);
    Boolean b = (Boolean)
      request.getAttribute(INPUT_IGNORE_PARAMETERS);
    isSubmitted = fp.isSubmitted() && b == null;
    int eventDefinitionCRFId = 0;
    ...
```

Methoden müssen < 500 Zeilen lang sein

volkswagen

Volkswagen detects when your tests are being run in a CI server, and makes them pass.



https://github.com/auchenberg/volkswagen

Fixe Vorgaben

zentral

Tipps

- Das „Warum“ deutlich machen
- Ausgeschlossene Varianten kennzeichnen (ADRs)
- Regelmäßig Feedback einholen (oder Entscheidungen selbst ausbaden)
- Bei Bedarf anpassen

ADR: Architecture Decision Record

Governance-Strategien

zentral

dezentral

Fixe Vorgaben

Bis ins Detail durchentschiedene Lösung

Paved Roads

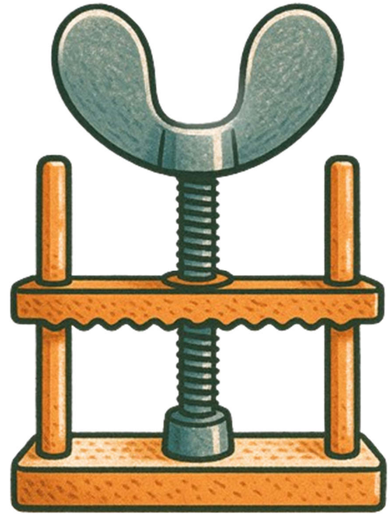
Angebot von alternativen Lösungen

Multi Level

Verteilte, hierarchische Standards

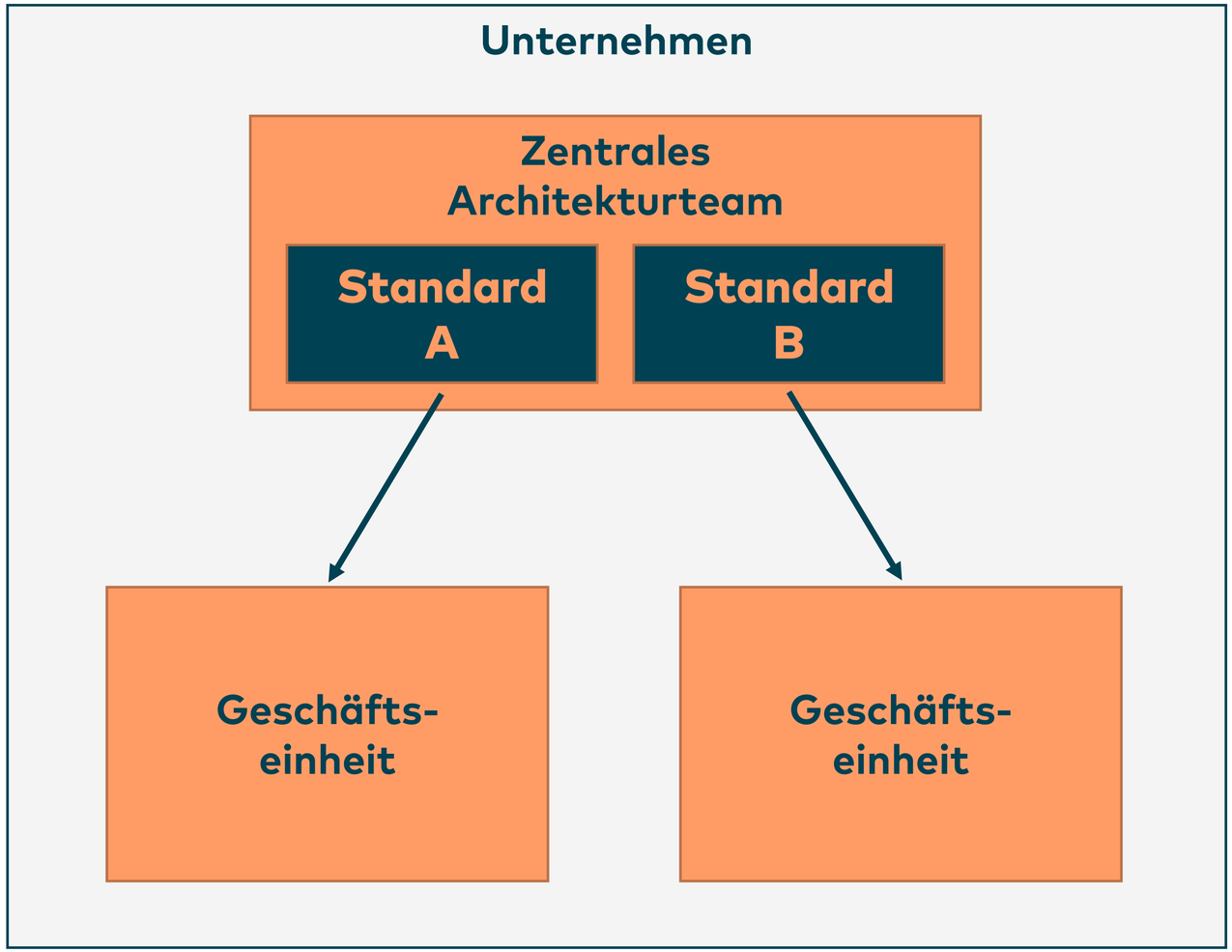
API Mandate

Individuelle Standards, aber standardisierte Schnittstellen



Stark angelehnt an Mark Richards “Software Architecture Monday, Lesson 62 - Enterprise Architecture Strategies”

Standardisierte Alternativen



Paved Roads

eher zentral

+ richtiges Werkzeug für den Job
+ mehr Kontrolle der Auswahl
+ bessere Zufriedenheit

- braucht Zeit für Auswahl
- Wahl / Zusammensetzung könnte nicht die richtige sein
- höhere Kosten

Ziel | Vorgabe | Team-Lösung

Wir entwickeln unsere Enterprise-Applikationen mit Java und unsere Datenanalysen mit Python / Pandas

Flexible Entwicklungs-planung

Standards

Paved Roads

eher zentral

Mögliche Paved Roads

Attraktive Defaults statt harter Vorgaben

Infrastruktur ✓

Tooling ✓

Automatisierung ✓

… ✓

CI/CD-Pipeline ✓

Guidelines ✓

Tipps

- Alle Alternativen gleichwertig behandeln und weiterpflegen
- Ausnahmeregelungen dennoch zulassen
- Feedback-Loop einbauen

Paved Roads

eher zentral

Governance-Strategien

zentral

dezentral

Fixe Vorgaben

Bis ins Detail durchentschiedene Lösung

Paved Roads

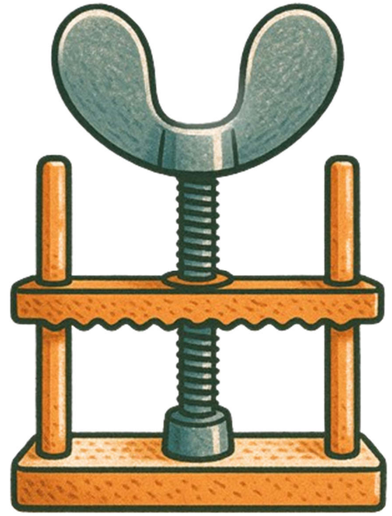
Angebot von alternativen Lösungen

Multi Level

Verteilte, hierarchische Standards

API Mandate

Individuelle Standards, aber standardisierte Schnittstellen



Stark angelehnt an Mark Richards “Software Architecture Monday, Lesson 62 - Enterprise Architecture Strategies”

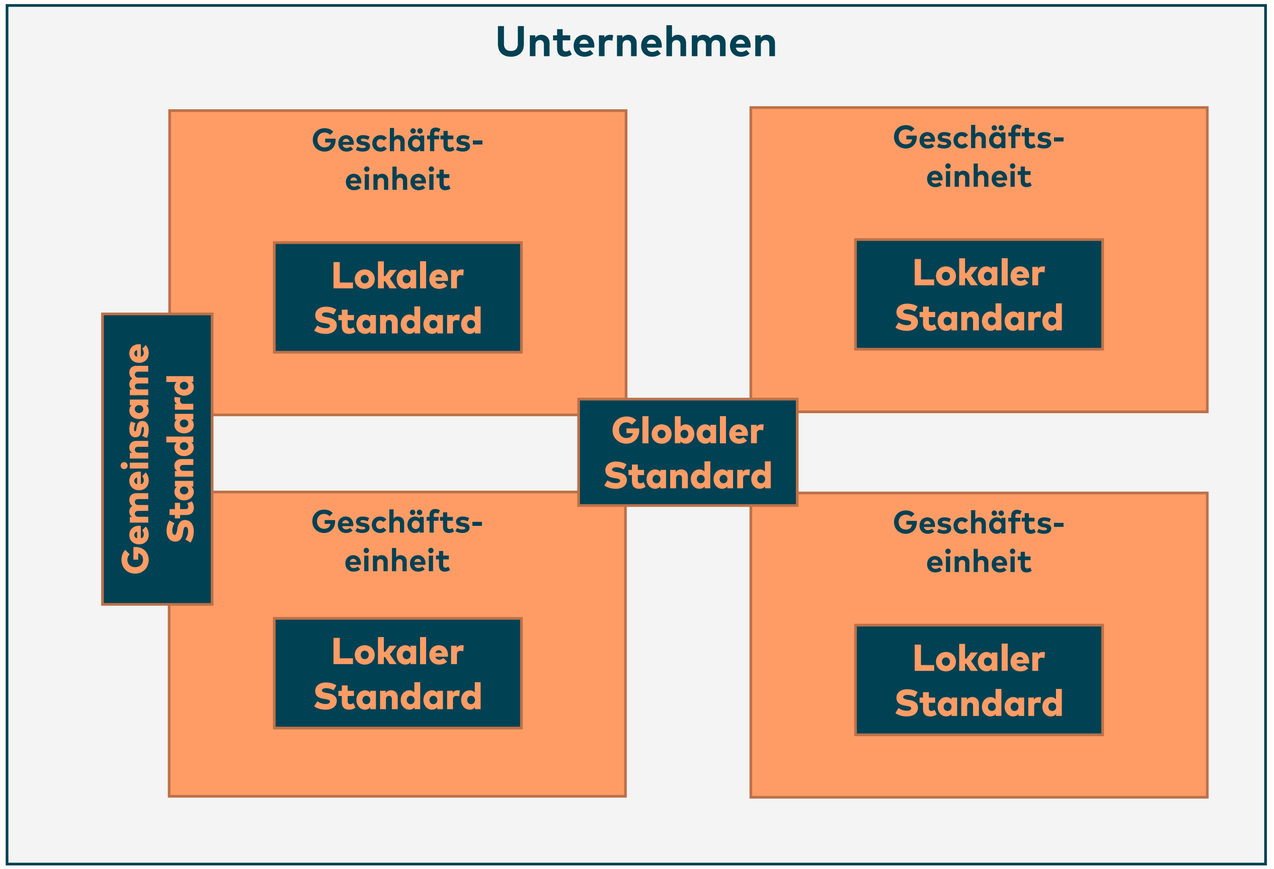
Multi Level

eher dezentral

+ richtiges Werkzeug für den Job
+ Geschäftseinheit hat Kontrolle
+ minimale, zentrale Vorgaben
+ bessere Gesamtzufriedenheit

- fehlende Abstimmungen
- hohe Kosten
- erschwerte Kostenkontrolle
- schwer global zu steuern

Verteilte, hierarchische Standardisierung



Leitplanken

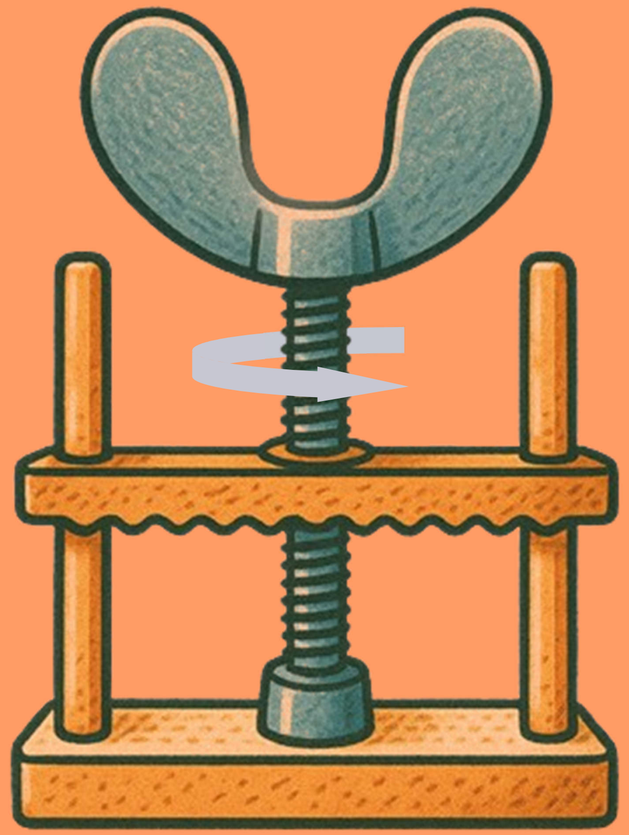
= **Universell nützliche High-Level-Prinzipien**

- Legen die wichtigsten Prioritäten fest
- Helfen dem Team, auf Kurs zu bleiben

→ Vermeidet auch, dass es wider-sprüchliche Interessen gibt

Multi Level

eher dezentral



Flexible Entwicklungs-planung

Ziel | Vorgabe | Team-Lösung

High level

Uns ist wichtig, dass wir Devs schnell auf alle anderen Projekte einarbeiten können

Mid level

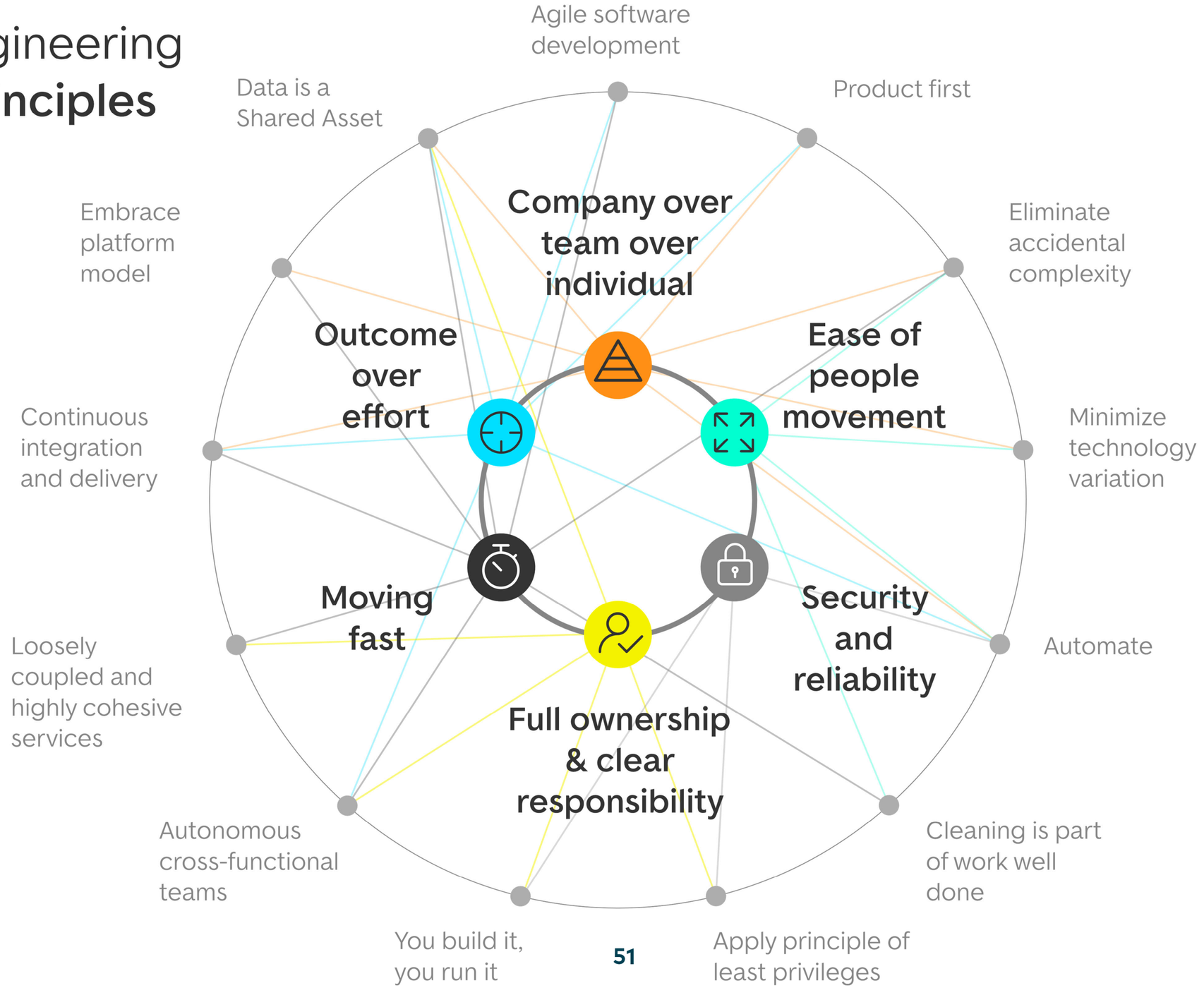
Alles auf der JVM ist OK

Standards

Lass mal Java nehmen

Low level

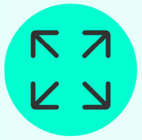
Scout24 Engineering
Values & Principles



Values

Company over team over individual

We value decisions that have a positive impact for the entire tech organization even if they have a negative impact on our own team. We rationally assess the impact and acceptance of both new and current technologies on other teams. We look for opportunities to help other teams benefit from what we are building.



Ease of people movement

The movement of people into teams with complex, high priority work is an important part of how Scout24 will continue to grow as a business. To reduce onboarding barriers, we value standardization of technologies and processes across Scout24. We value engineers who can work across multiple stacks.

https://github.com/Scout24/scout24-engineering-values-and-principles/

Principles

Agile software development

We listen to user needs and release value often so we can learn from successes and failures and adjust our approach quickly.

Continuous integration and delivery

We, as team members, continuously integrate our software changes with those of our teammates and push these changes to production automatically.

Minimize technology variation

We deliberately minimize the variety of technologies in our system to drive efficiency, and, at the same time, have a managed program of innovation.

Automate

We automate repetitive activities to achieve better predictability, efficiency, reproducibility, resiliency, and security.

Loosely coupled and highly cohesive services

We strive to minimize the coupling of services and maximize their cohesion in order to, among other things, limit failure propagation, reduce response times and limit the scope of software changes.

(Architektur-)Prinzip

„Ein Architekturprinzip ist eine deklarative Aussage, die mit der Absicht getroffen wird, architektonische Designentscheidungen zu leiten, um eine oder mehrere Qualitäten eines Systems zu erreichen.“

- Eoin Woods

Twelve Factor App

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

https://12factor.net/

Initial erstellt von Heroku, einem Anbieter einer Cloud-Plattform as a Service (PaaS)

The Frugal Architect

PHASE: DESIGN

I. Make Cost a Non-functional Requirement
II. Systems that Last Align Cost to Business
III. Architecting is a Series of Trade-offs

PHASE: MEASURE

IV. Unobserved Systems Lead to Unknown Costs
V. Cost Aware Architectures Implement Cost Controls

PHASE: OBSERVE

VI. Cost Optimization is Incremental
VII. Unchallenged Success Leads to Assumptions

https://www.thefrugalarchitect.com/

Boring Software Manifesto

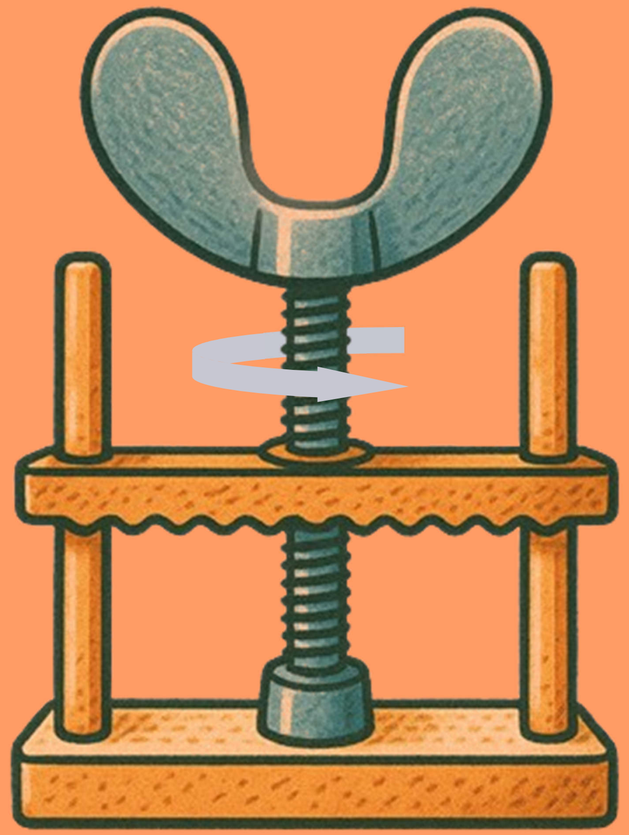
Anti-Hype-Driven-Development

- Not only working software,
  - but also well-crafted software,
  - built exclusively with popular and proven tools.
- Not only responding to change,
  - but also steadily adding value,
  - while reducing the dependencies and complexity.
- Not only individuals and interactions,
  - but also a community of professionals,
  - that share best practices with verifiable claims.
- Not only customer collaboration,
  - but also productive partnerships,
  - to reduce the scope of the software we build.

https://tqdev.com/2018-the-boring-software-manifesto

Multi Level

eher dezentral



Ziel | Vorgabe | Team-Lösung

Funktionierende Software

High level

Design für Testbarkeit

Standards

Wir setzen für kritische Komponenten Mutation Testing ein

Low level

Beispiel Architekturprinzip 1/2

Prinzip: Design für Testbarkeit

Die Lösungen sollten so konzipiert - und der Code so strukturiert - sein, dass die Ausführung der Tests einfacher und schneller erfolgt.

Begründung

Lösungen, die unter Berücksichtigung von Testaspekten entwickelt werden, ermöglichen ein schnelleres Feedback und können letztendlich häufiger und sicherer an die Kunden weitergegeben werden. Eine testorientierte Entwicklung führt automatisch zu einer besseren Verständlichkeit und Evolvierbarkeit.

Beispiel Architekturprinzip 2/2

Auswirkungen

- Strukturiere deinen Code so, dass die Komponenten isoliert ausgeführt werden können, um ihr Verhalten bei der Überprüfung und beim Testen zu beobachten.
- Stelle sicher, dass die Komponenten eines Systems in klar definierte Verantwortlichkeiten aufgeteilt sind.
- Die Komponenten eines Systems sollten leicht zu verstehen sein, d.h. der Code sollte so geschrieben sein, dass er sich selbst erklärt und durch seine Tests oder, falls nötig, durch eine separate Dokumentation dokumentiert wird.

Weitere Informationen

- Testability blog post von Michael Bolton.
- Team Guide to Software Testability von Ash Winter and Rob Meaney.

Übersetzt in Auszügen übernommen von https://engineering-principles.jlp.engineering/

Architecture Decision Record (ADR)

Entwurfsentscheidungen nachvollziehbar dokumentieren

- **Titel**: Um welche Entscheidung geht es?
- **Kontext**: Was waren die Rahmenbedingungen?
- **Entscheidung**: Was war die Entscheidung, was waren die Alternativen?
- **Status**: Wie steht es um das ADR?
  - hier auch abgelehnte oder überholte ADRs aufführen
- **Konsequenzen**: Mit welchen Vor- und Nachteilen wollen wir jetzt leben?

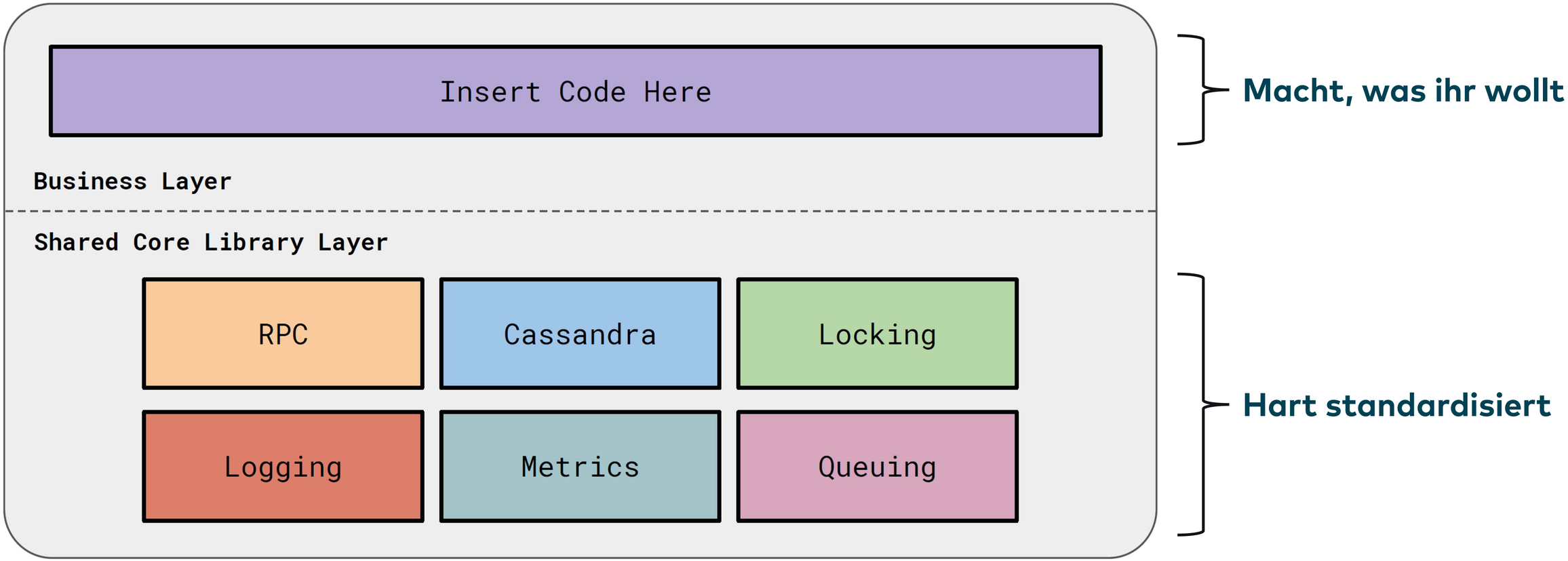
Beispiele unter https://github.com/arachne-framework/architecture

Beispiel ADR (skizziert)

Titel: Einsatz von Ping-Mechanismen

- Kontext: Beteiligte Systeme der Tagesendverarbeitung müssen verfügbar sein.
- Entscheidung: Wir setzen die Architekturtaktik „Ping“ ein, um die Bereitschaft der Systeme während der Tagesendverarbeitung vor der Nutzung zu kontrollieren. Die Alternative eines „Heartbeats“ schlossen wir aus, da wir dies derzeit für zu komplex für unsere Anwendung halten.
- Status: Beschlossen
- Konsequenzen:
  - Alle Systeme implementieren Ping-Mechanismen in den Systemen
  - Risiko „Ping Flooding“ durch bösartige Systeme oder Unachtsamkeit
  - Nachteile „Performance-Overhead“ und „weniger einfache Tests“

Beispiel Monzo (Bank aus UK)



Aus dem Vortrag "Modern Banking in ~~1500~~ 1600 Microservices"
https://www.infoq.com/presentations/monzo-microservices/

Beispiel Monzo (Bank aus UK)

Engineering Principles

1. Ship it and iterate.
2. Make changes small, make them often.
3. Technical debt is a useful tool.
4. Solve problems at the root.
5. Do not accept deviant system behaviour.
6. Write code to be read.
7. Write code to be debugged.
8. If you can't show it's a bottleneck, don't optimise it.
9. Unblock others whenever you can.
10. Leave the codebase better than you found it.

https://monzo.com/blog/2018/06/29/engineering-principles

The Principles of Craftsmanship

Well-Crafted Software

Steadily Adding Value

A Community of Professionals

Productive Partnerships

"We at 8th Light are principled; and these are the principles we follow."

https://blog.cleancoder.com/uncle-bob/2013/02/10/ThePrinciplesOfCraftsmanship.html

Governance-Strategien

zentral

dezentral

Fixe Vorgaben

Bis ins Detail durchentschiedene Lösung

Paved Roads

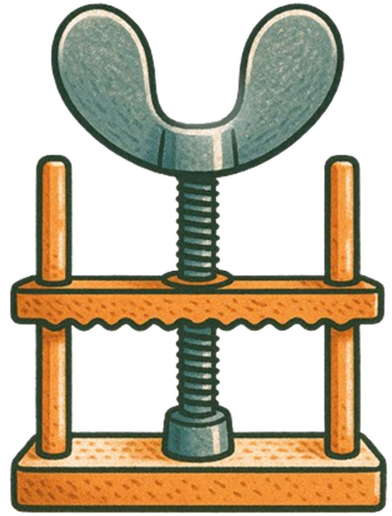
Angebot von alternativen Lösungen

Multi Level

Verteilte, hierarchische Standards

API Mandate

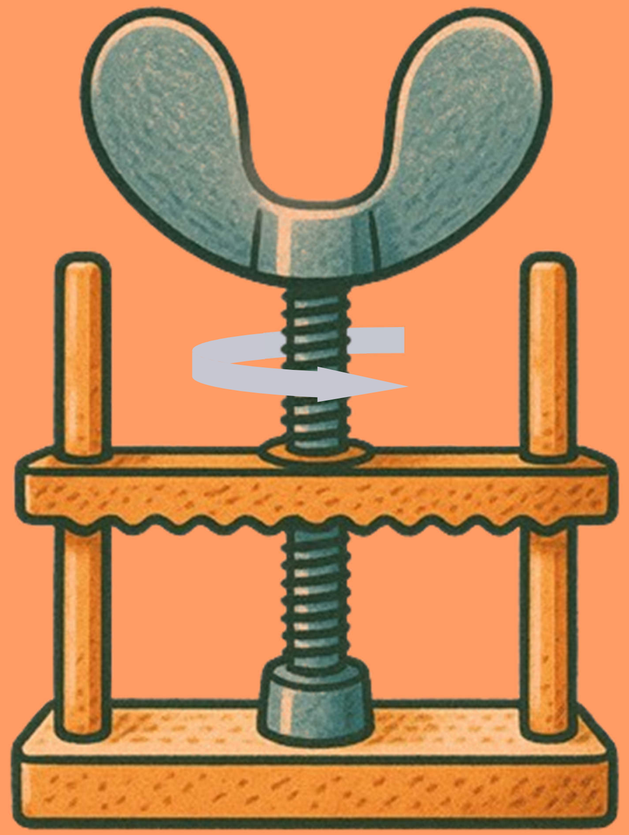
Individuelle Standards, aber standardisierte Schnittstellen



Stark angelehnt an Mark Richards “Software Architecture Monday, Lesson 62 - Enterprise Architecture Strategies”

Multi Level

eher dezentral

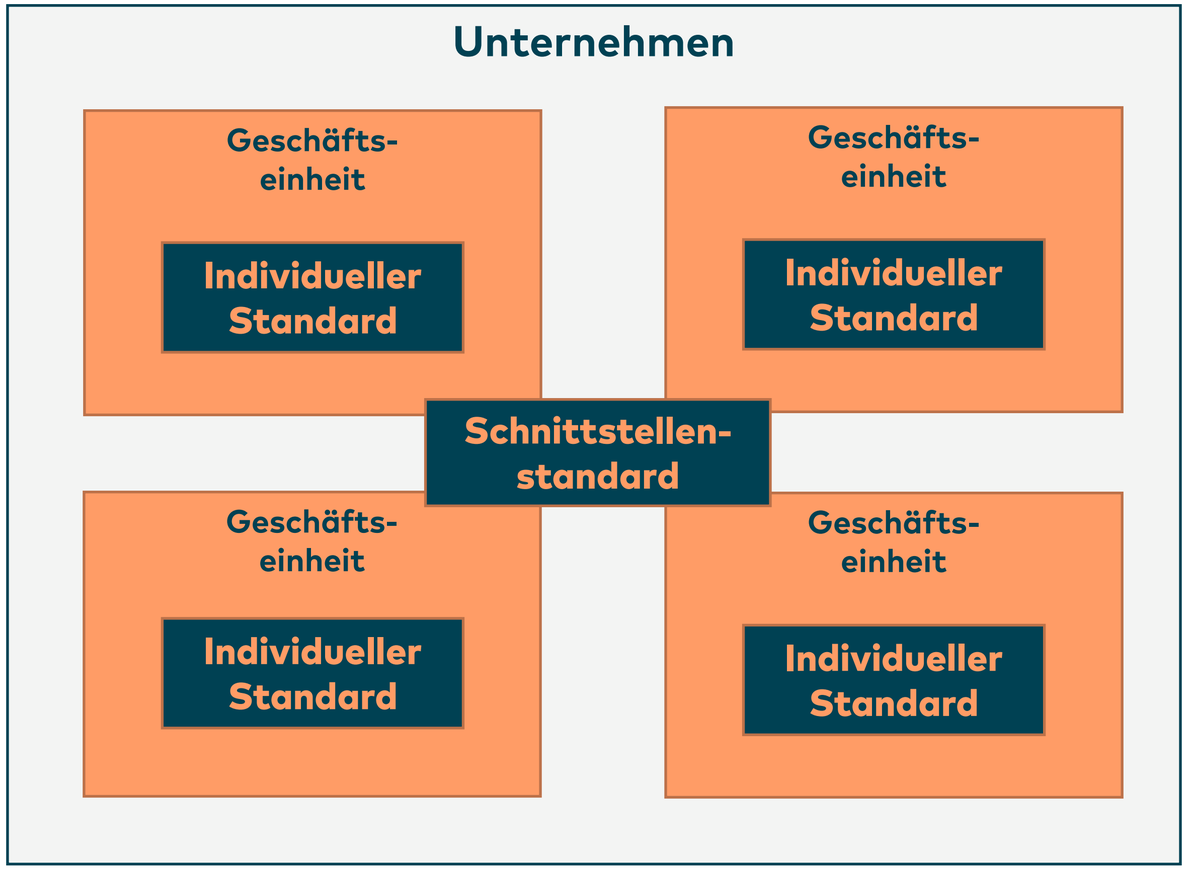


Tipps

- Rote Fäden ziehen: Was wollen wir erreichen? Warum?
- Konkret beschreiben / kein Blabla
- Lokale Entscheidungen/ Best Practices kommunizieren
- Austauschformate einsetzen (CoP)
- Spezialistinnen und Spezialisten benennen als Ansprechpartner

CoP: Community of Practice

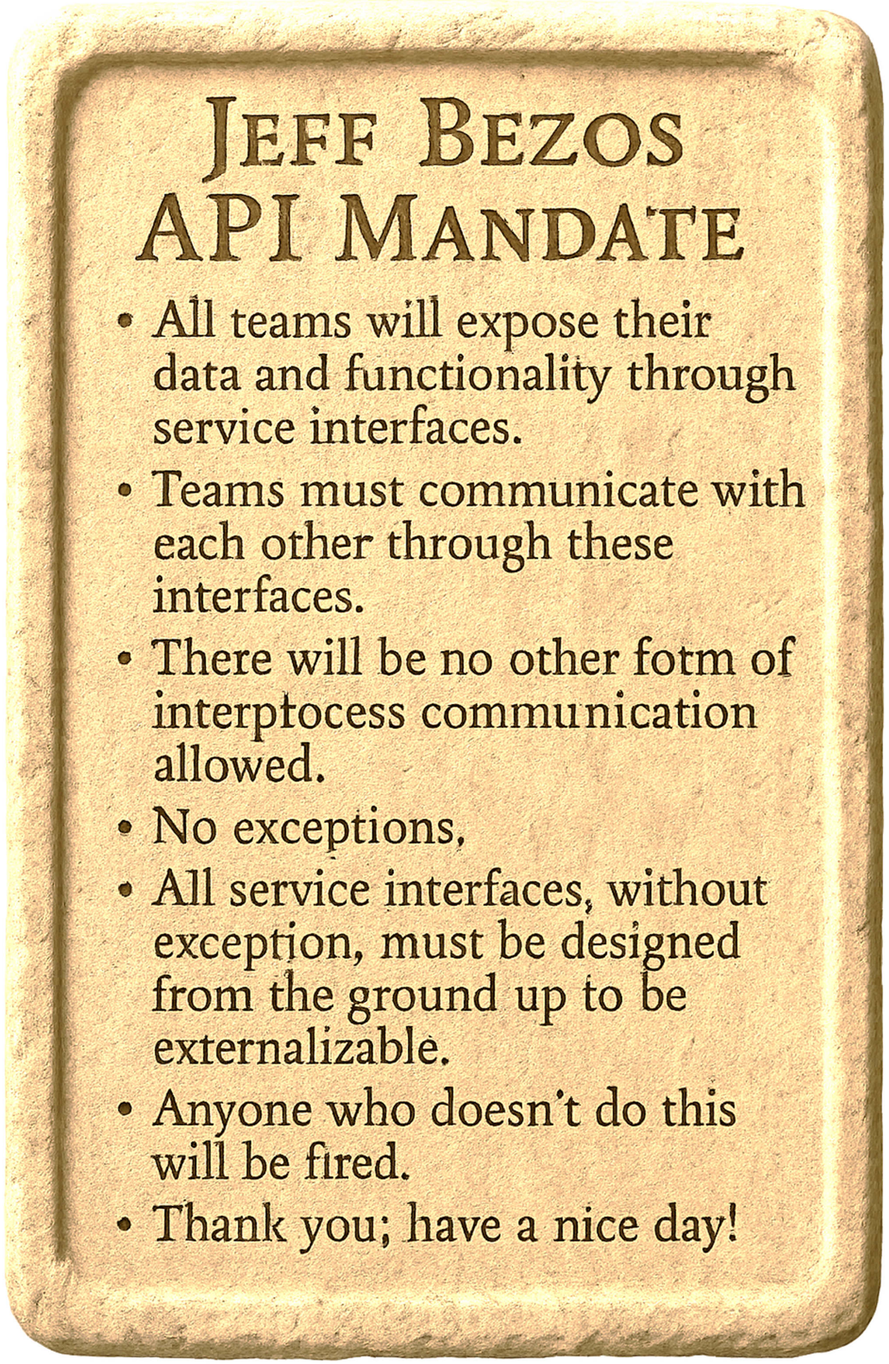
Individuelle Standards mit definierten Schnittstellen



API Mandate

dezentral

+ richtiges Werkzeug für den Job
+ Geschäftseinheit hat Kontrolle
+ Synergien zwischen Einheiten
+ bessere Zufriedenheit
- braucht Zeit für Auswahl
- höchste Kosten
- schwerste Kostenkontrolle



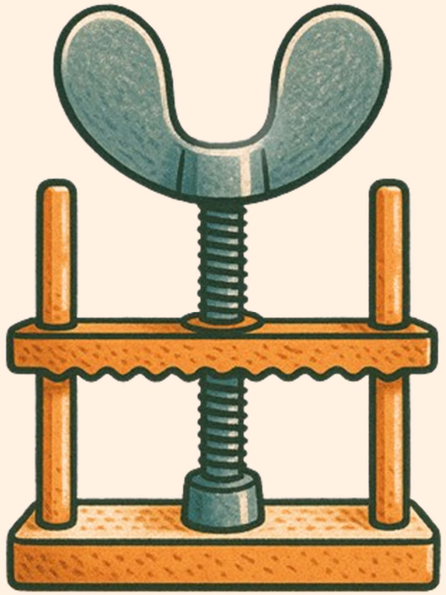
API Mandate

dezentral

“With great power comes great responsibility”

Aus dem Vortrag “From Gates to Guardrails - Alternate Approaches to Product Security” von Jason Chan
https://www.youtube.com/watch?v=geumLjxtc54

Was bedeutet das für uns?



Daumenschrauben lassen sich auch so anziehen, so dass man sie gar nicht bemerkt.

Bonus

6 Tipps

zur Gestaltung einer angenehmen Architektur-Governance

Moderne Governance-Meta-Prinzipien

1. Weg vom Gesetz, hin zu Vision und Prinzipien (und Leitplanken)
2. Compliance automatisieren und delegieren
3. Gatekeeper zu Mitarbeitende machen
4. Gepflasterte Straßen bereitstellen
5. Informationen offen teilen / Kommunikationsmuster überdenken
6. An Evolution gewöhnen

Quelle: https://www.thoughtworks.com/insights/articles/lightweight-technology-governance

Beispiel

Compliance automatisieren und delegieren

Schlaue Architekturdokumentation

Business modules

Salespoint ships with a set of business modules that interact with each other. The following component diagram shows the overall structure of the setup. The relationships are used as follows:

- *uses* — The module has a Spring component dependency and actively interacts with the target module. Implies a type dependency into the target module as well.
- *listens to* — The module contains an event listener for events the target modules publishes. Implies a type dependency into the target module as well.
- *depends on* — The module has a general type dependency to the target module, i.e. it uses the target module as library.

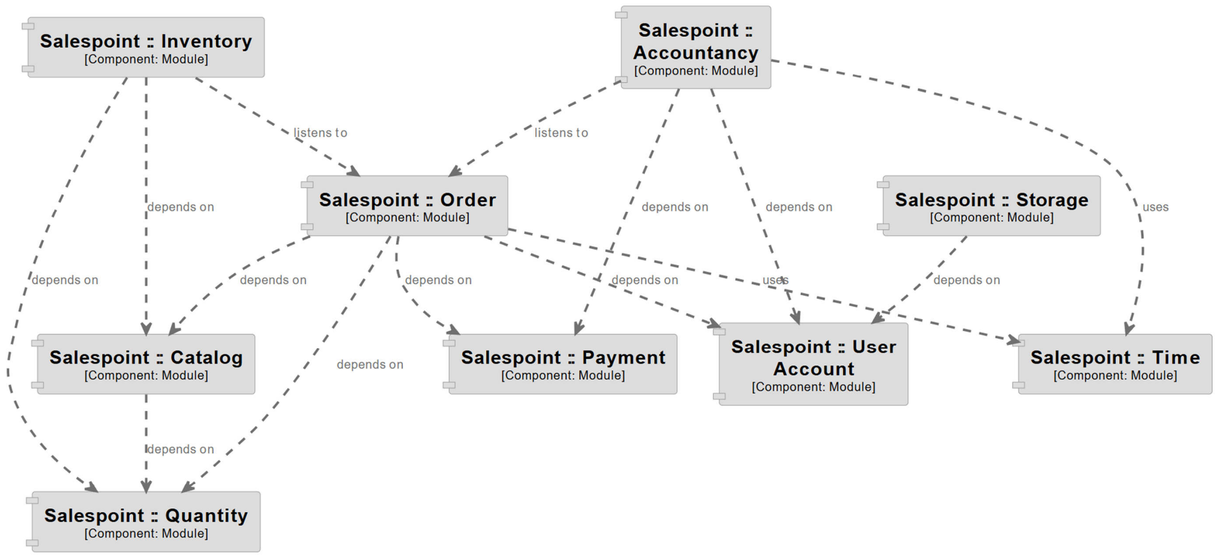


Figure 1. Salespoint component overview

Direkt aus dem Code erzeugte Doku

https://github.com/st-tu-dresden/salespoint

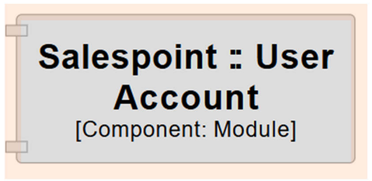
Schlaue Architekturdokumentation

User accounts

Figure 2. User account component

Base package	`org.salespointframework.useraccount`
Spring components	*Services* • `o.s.u.UserAccountManagement` (via `o.s.u.PersistentUserAccountManagement`) *Others* • `o.s.u.AuthenticationManagement` (via `o.s.u.SpringSecurityAuthenticationManagement`)
Aggregate roots	• `o.s.u.UserAccount`
Value types	• `o.s.u.EncryptedPassword` • `o.s.u.UnencryptedPassword` • `o.s.u.Role`
Published events	• `o.s.u.UserAccountCreated` created by: ◦ `o.s.u.UserAccount.onCreate()`
Properties	• `salespoint.authentication.login-via-email` — `java.lang.Boolean`, default `false`. Enables the login procedure to use the email address to lookup a user instead of their username. Defaults to `false`.

Schlaue Architekturdokumentation




```java
@Entity
@NoArgsConstructor(access = AccessLevel.PRIVATE)
public class UserAccount extends AbstractAggregateRoot<UserAccountIdentifier> {
```

```java
/**
 * User account management.
 *
 * @see org.salespointframework.useraccount.UserAccountManagement
 */
@org.springframework.lang.NonNullApi
@org.springframework.modulith.ApplicationModule(displayName = "Salespoint :: User
package org.salespointframework.useraccount;
```

ArchUnit – Left Shifting Governance

Eine entwicklungsnahe Architekturvalidierung

✓Erstellung von Architekturregeln mittels eigener domänenspezifischer Sprache (DSL)

✓Prüfung von Entwurfsregeln zur Testzeit

```
public class MyArchitectureTest {

@Test public void some_architecture_rule() {

        JavaClasses importedClasses = new ClassFileImporter().importPackages("com.myapp");
        ArchRule rule = classes()...; }
}
```


Mehr Informationen unter https://www.archunit.org/

ArchUnit Code-/API-Beispiel

```
classes().that().
```

```
areAnnotatedWith(String annotationTypeNa… GivenClassesConjunction
areAnnotatedWith(Class<? extends Annotation> annotationType)  GivenClassesConjunc
areAnnotatedWith(DescribedPredicate<? su… GivenClassesConjunction
haveNameMatching(String regex)            GivenClassesConjunction
areNamed(String name)                     GivenClassesConjunction
areNotAnnotatedWith(String annotationTyp… GivenClassesConjunction
areNotAnnotatedWith(Class<? extends Anno… GivenClassesConjunction
areNotAnnotatedWith(DescribedPredicate<?… GivenClassesConjunction
resideInAPackage(String packageIdentifie… GivenClassesConjunction
areAssignableFrom(Class<?> type)          GivenClassesConjunction
areAssignableFrom(String typeName)        GivenClassesConjunction
areAssignableFrom(DescribedPredicate<? s… GivenClassesConjunction
```

```
classes().that().areAnnotatedWith(Service.class)
        .or().haveNameMatching( regex: ".*Service")
        .should().resideInAPackage( packageIdentifier: "..service..")
        .orShould().beAnnotatedWith(LegacyBridge.class);
```

Fazit

Was bedeutet das für uns?

Geschicktes Daumenschrauben muss nicht nerven!

Vielen Dank!

Fragen, Diskussionen, Vorschläge ?

Networking →

Markus Harrer

Bildnachweise

- Bitmap-Grafiken wurden mit OpenAI GPT 4o (March 25, 2025 Release) erzeugt
- Vektorgrafiken kommen von Streamline

INNOQ

UNSER ANGEBOT

Produktkonzeption & Design
Software-Entwicklung & -Architektur
Technologie-Beratung
Infrastruktur & Betrieb
Wissenstransfer, Coaching & Trainings

FAKTEN

~160 Mitarbeitende
1998 gegründet
9 Standorte in
D & CH

FOKUS

Webapplikationen
SaaS
IoT
Produktentwicklung
ML/AI
Blockchain

TECHNOLOGIEN

(Auswahl)

Java/Spring
Ruby/Rails
Scala
AWS
Kubernetes
Azure
JavaScript
Python
C#
ML/AI
Blockchain

KLIENTEN

Finance • Telko • Logistik • E-Commerce • Fortune 500 • KMUs • Startups

AXA breuninger ebay 83 mobile.de SACAC VORWERK