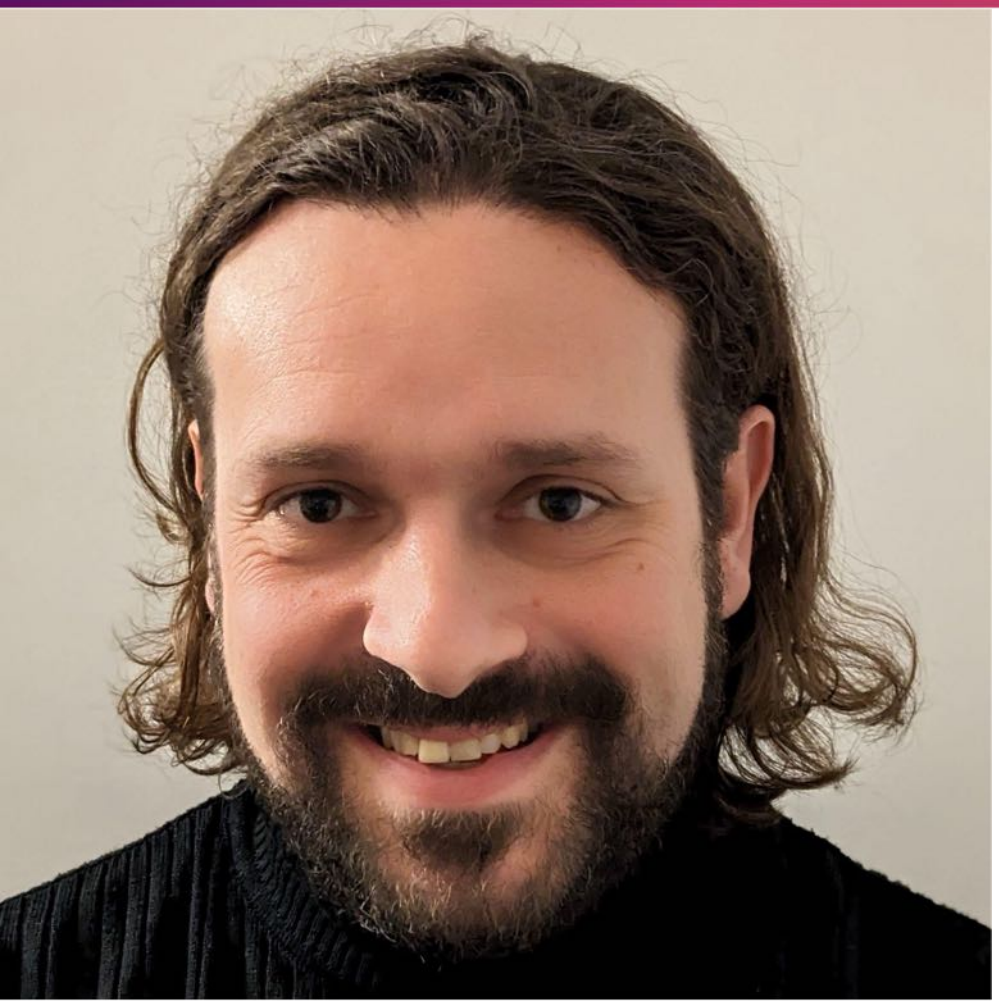
Mythen und Legenden der Softwarearchitektur

Busted!

Wer sind wir nicht:



Architekt, Developer, Trainer, Berater

DDD, Strategie, Socio-Technical Systems

iSAQB Member, AG Advanced Level

Zuhause in Heppenheim

Founder von tangible concepts

tom.asel@tangible-concepts.de
https://bsky.app/profile/asel.io
https://linkedin.com/in/tom-asel

Tom Asel

iSAQB ACCREDITED Trainer

iSAQB ACCREDITED Training Provider

tangible concepts

Falk Sippach

- Softwarearchitekt, Berater, Trainer bei **embarc**
- früher bei Orientation in Objects (OIO), Trivadis

Schwerpunkte

- Architekturberatung und -bewertung
- Cloud- und Java-Technologien

iSAQB International Software Architecture Qualification Board

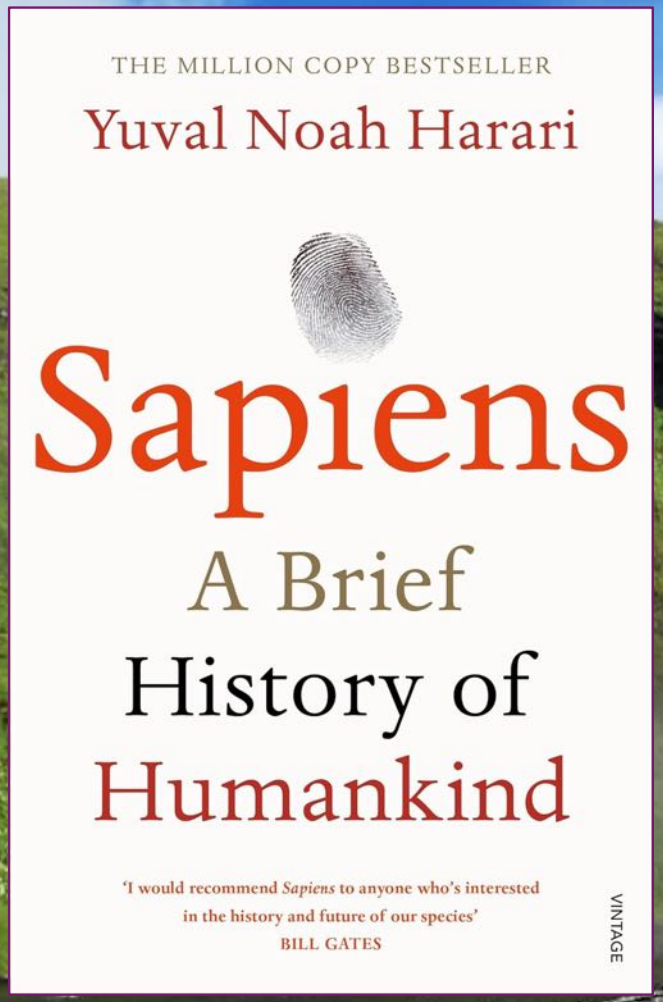
Java Champions

Cyber LAND

JUG DARMSTADT

embarc

Mythen bringen uns weiter



"As far as we know, only Sapiens can talk about entire kinds of entities that they have never seen, touched, or smelled.

[...] This ability to speak about fictions is the most unique feature of Sapiens' language ...

Large numbers of strangers can cooperate successfully by believing in common myths.

— Yuval Noah Harari

Mythen können Indikatoren sein

- Ungleich verteiltes Wissen
  - domänenspezifische Begriffe
  - relevante technische Konzepte
  - Unzugängliche Doku
  - falsche Adressaten / Teilnehmerkreise
  - Fehlendes Grundlagenwissen
- Kultureller Mismatch
- **“Inertia” -** Erfolgsfaktoren aus der Vergangenheit, die heute hinderlich sind

Was busten wir heute?

#1	Qualität ist uns wichtig!
#2	Wir treffen Entscheidungen auf Basis von Fakten
#3	Onion/Clean/Hexagonal … das ist doch alles das Selbe!
#4	Architekturentwurf muss Upfront erfolgen!
#5	Agile Teams brauchen keinen Architekten
#6	Microservices sind moderner, Monolithen sind technisch rückständig
#7	Architekturdokumentation ist überbewertet

Mythos #1

Qualität ist uns wichtig!

Qualität ist uns wichtig!

Ok, was sind unsere wichtigsten Qualitätsziele?

Dokumentieren wir am Ende. Wenn noch Zeit ist.

Worauf basieren unsere Entscheidungen bis dahin?

Was ist das Problem?

Wenn wir unsere Qualitätsziele nicht kennen …

… entsteht trotzdem eine Architektur!

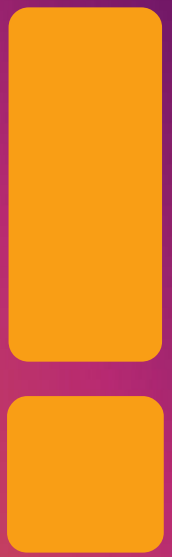
- Aber ist diese auch optimal?

… worauf basieren die Architekturentscheidungen?

- Eignung / Vergleichbarkeit / Tradeoffs von Optionen

… woher wissen wir dann:

- ob sie erreichbar sind
- wann sie erreicht sind
- wie wir sie erreichen
- wie groß das Delta ist



Was ist das Problem?

- Qualitätsanforderungen prägen Architektur stärker als funktionale Anforderungen
  - Treiber für Architekturentscheidungen
- Architekturentscheidungen:
  - langlebig
  - schwer zu ändern
    - Aufwendig / Teuer
    - → Risiko

Symptome

- Keine **gemeinsame Sprache** für Qualität
- **Priorität** von Qualitätszielen unklar
- **Metriken** nicht definiert
- **Kriterien** für Architekturentscheidungen bleiben **implizit**
- Architekturdokumentation:
  - im **Nachhinein** statt **kontinuierlich**
  - Doku an **Einzelperson** delegiert statt **Teamaufgabe**
  - Motivation: **Pflichterfüllung** statt **konkretem Nutzen**
  - Zielgruppe: **„Die anderen“** statt dem **eigenen Team**

Wie begegnen wir dem Mythos?

- Qualitätsziele **frühzeitig** diskutieren, festhalten und priorisieren
  - **Informelle** Beschreibungen sind ok
  - Qualitätsmodelle als **Vorlage** verwenden
    - Quality 42
    - ISO 25010
- **Kontinuierlich** betrachten, anpassen, erweitern
- Top Qualitätsziele und Merkmale bei **Architekturentscheidungen** berücksichtgen
  - in Workshops
  - Whiteboard-Skizzen

Qualitätsziele und Qualitätsmerkmale

Qualitätsziele:

- Was ist wichtig?
- Priorisierung

→ Architektur muss wesentlich zur Zielerfüllung beitragen

Qualitätsmerkmale:

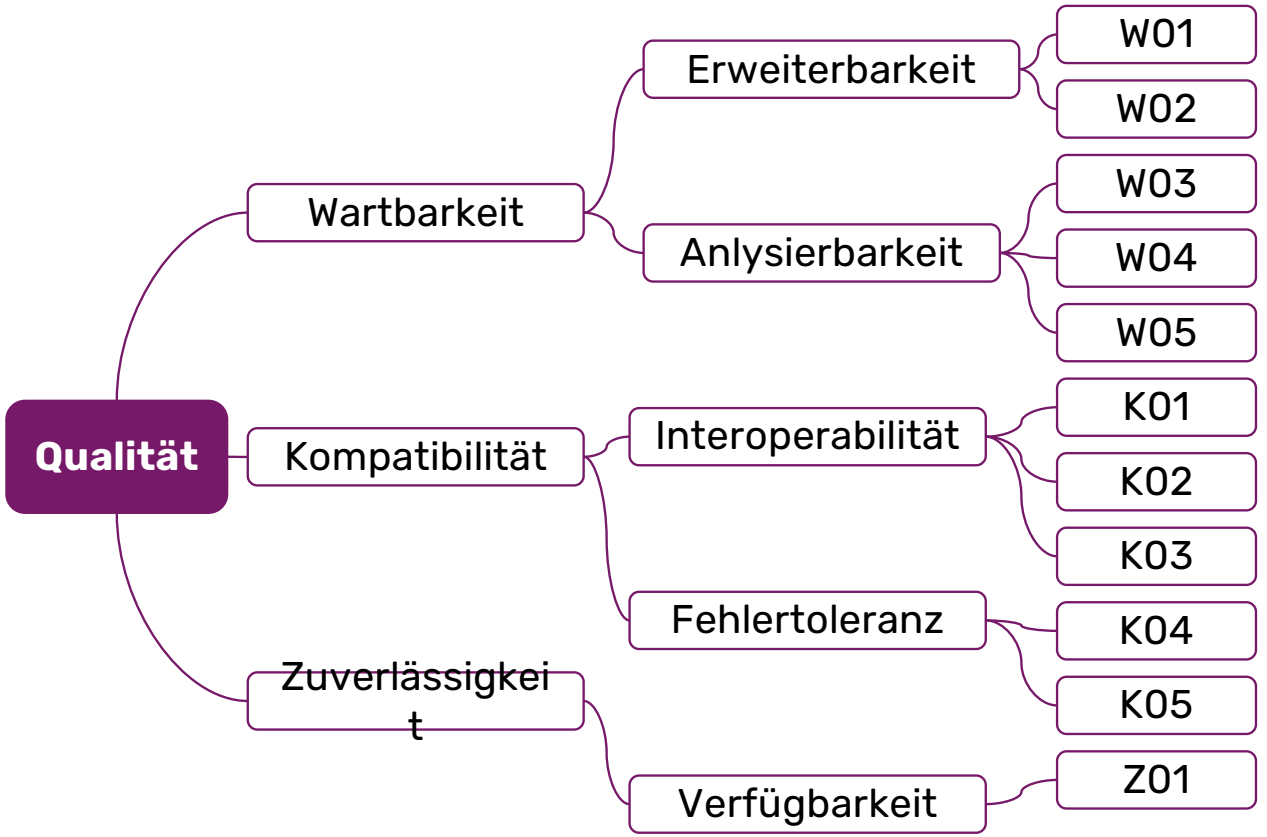
- Wie kann Qualität beschrieben werden?
- Objektiv überprüfbar
- Metriken

→ Architektur muss objektiv überprüfbare Merkmale aufweisen

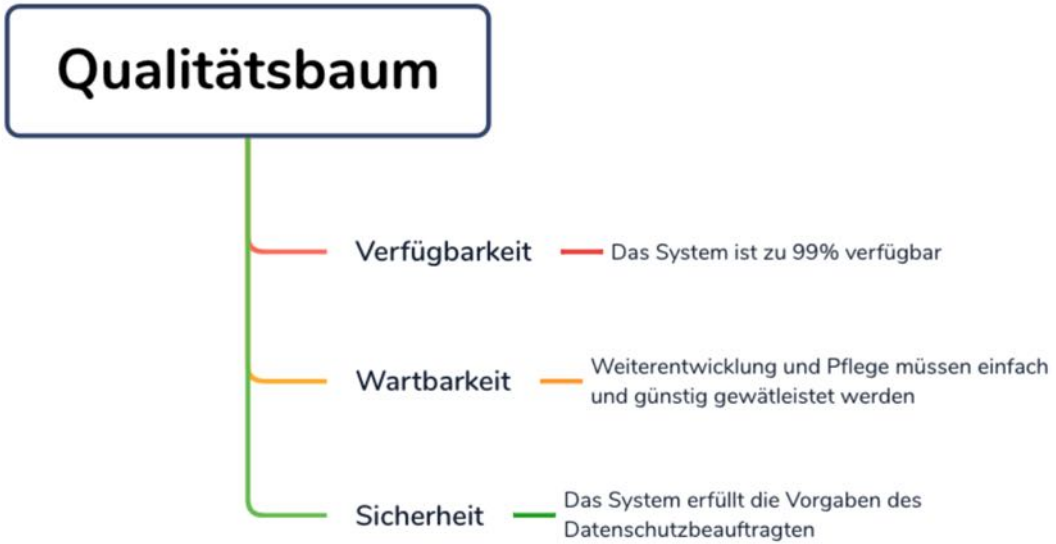
Qualitätsbaum

Beispiel !!!

- Beschreibt wichtigste Qualitätsmerkmale
- Jedes Merkmal ist objektiv überprüfbar
- Merkmale sind mit Szenarien beschrieben
  - Auslöser
  - Betroffener Systemteil
  - Erwartetes Verhalten
  - Antwortmetrik



Die traurige Realität



Mythos #2

Wir treffen Entscheidungen auf Basis von Fakten

Ich entscheide immer auf Basis von Fakten!

Was soll ich sonst machen?

Ich bin der Architekt, ich treffe die Entscheidungen!

Tust du nicht. Du entscheidest auf Basis deiner Annahmen

Deine Annahmen hinterfragen lassen.

Ok, aber triff sie nicht alleine!

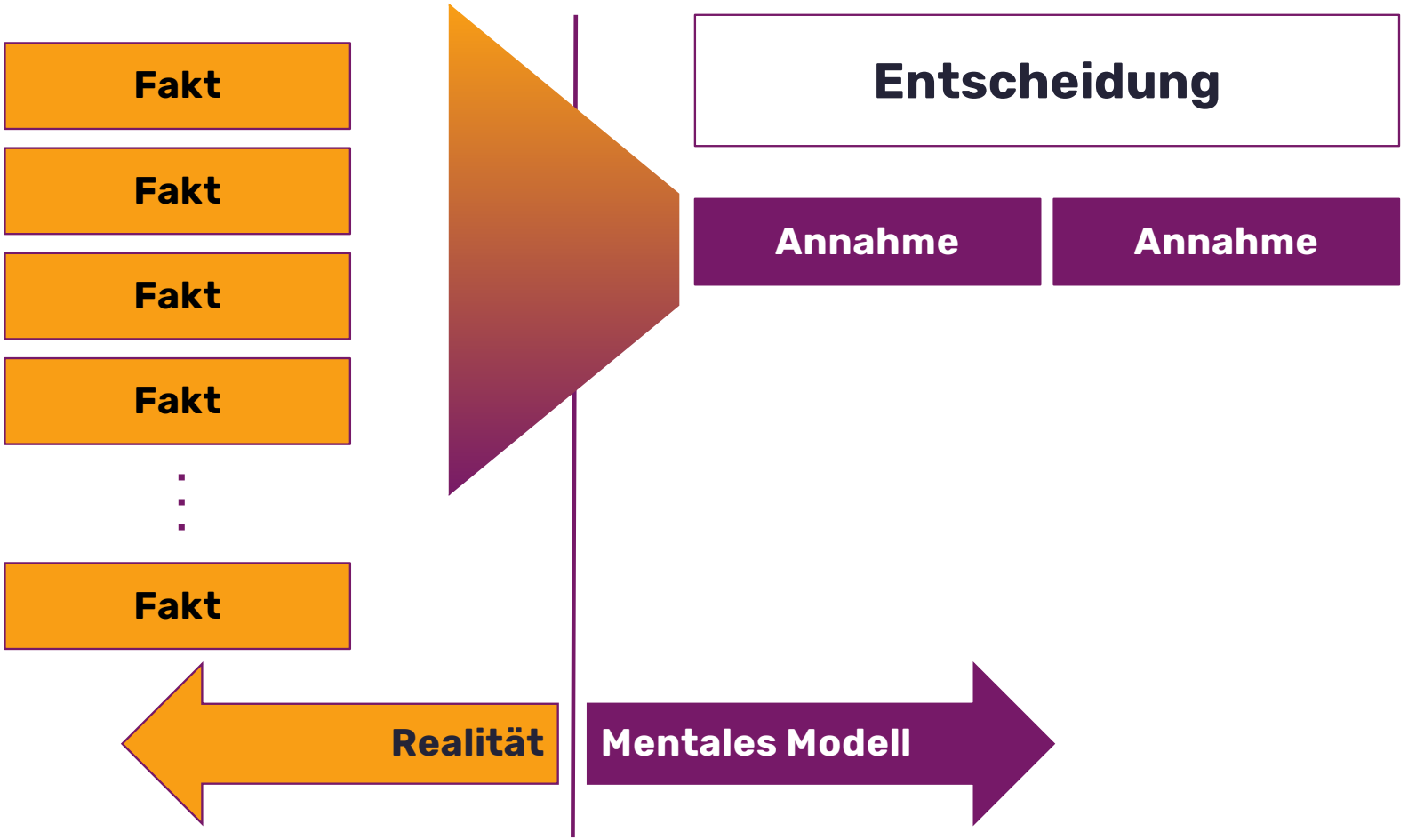
Was ist das Problem?

“Wir treffen Entscheidungen auf Basis von Fakten”

Quelle: Wir alle, zeitlos

Fakten vs. Annahmen

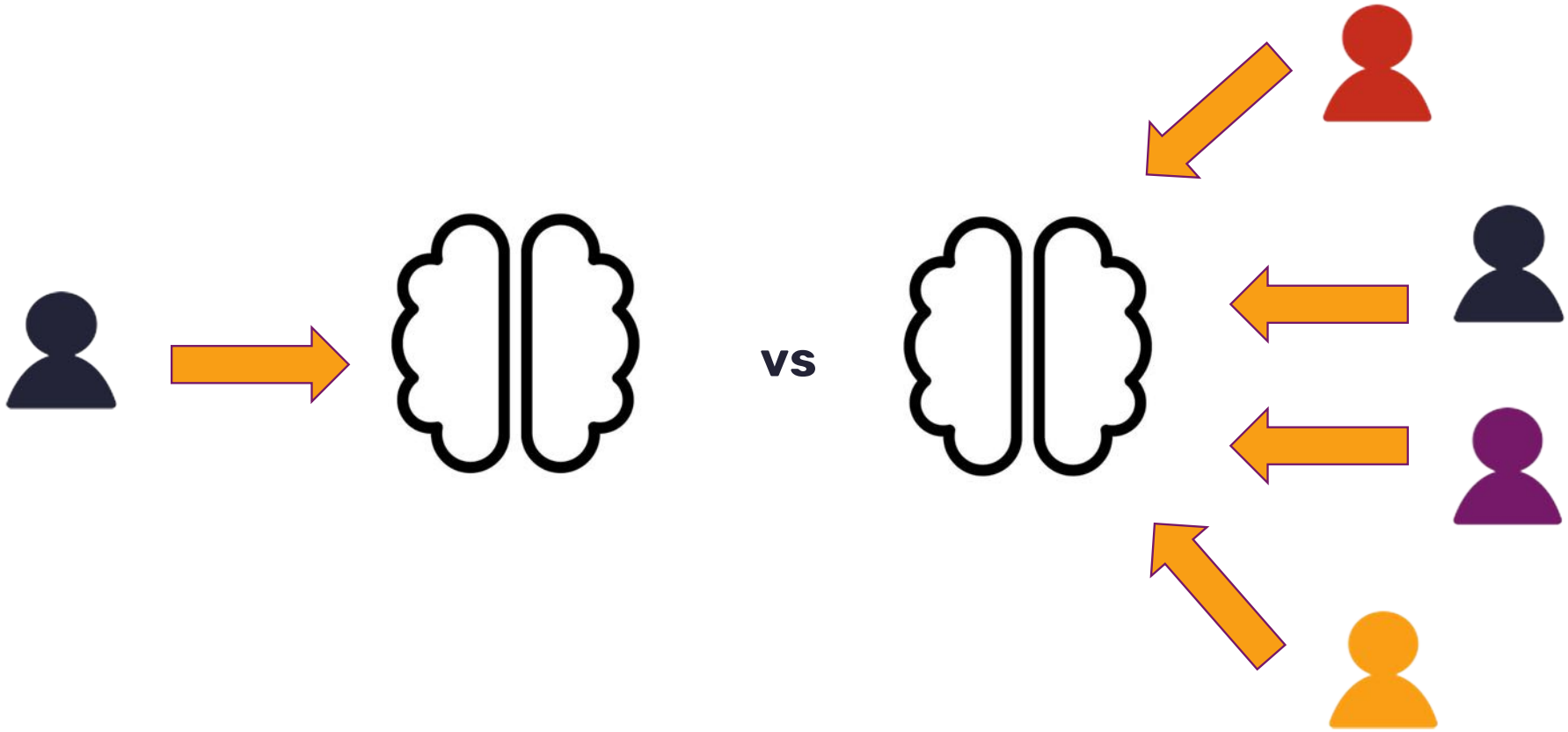
Entscheidungen basieren auf Annahmen



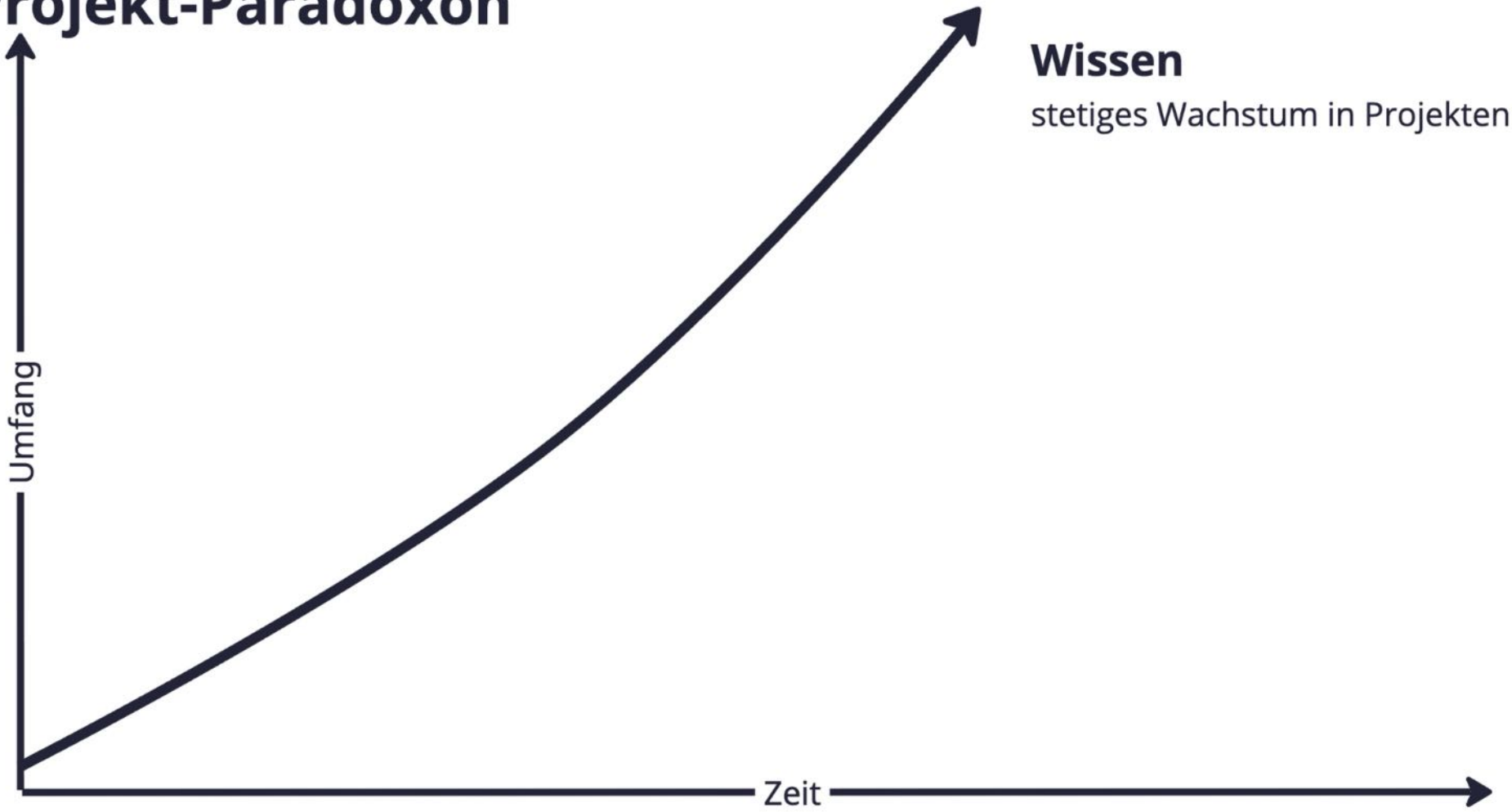
Wie begegnen wir dem Mythos?

Einzelverantwortung

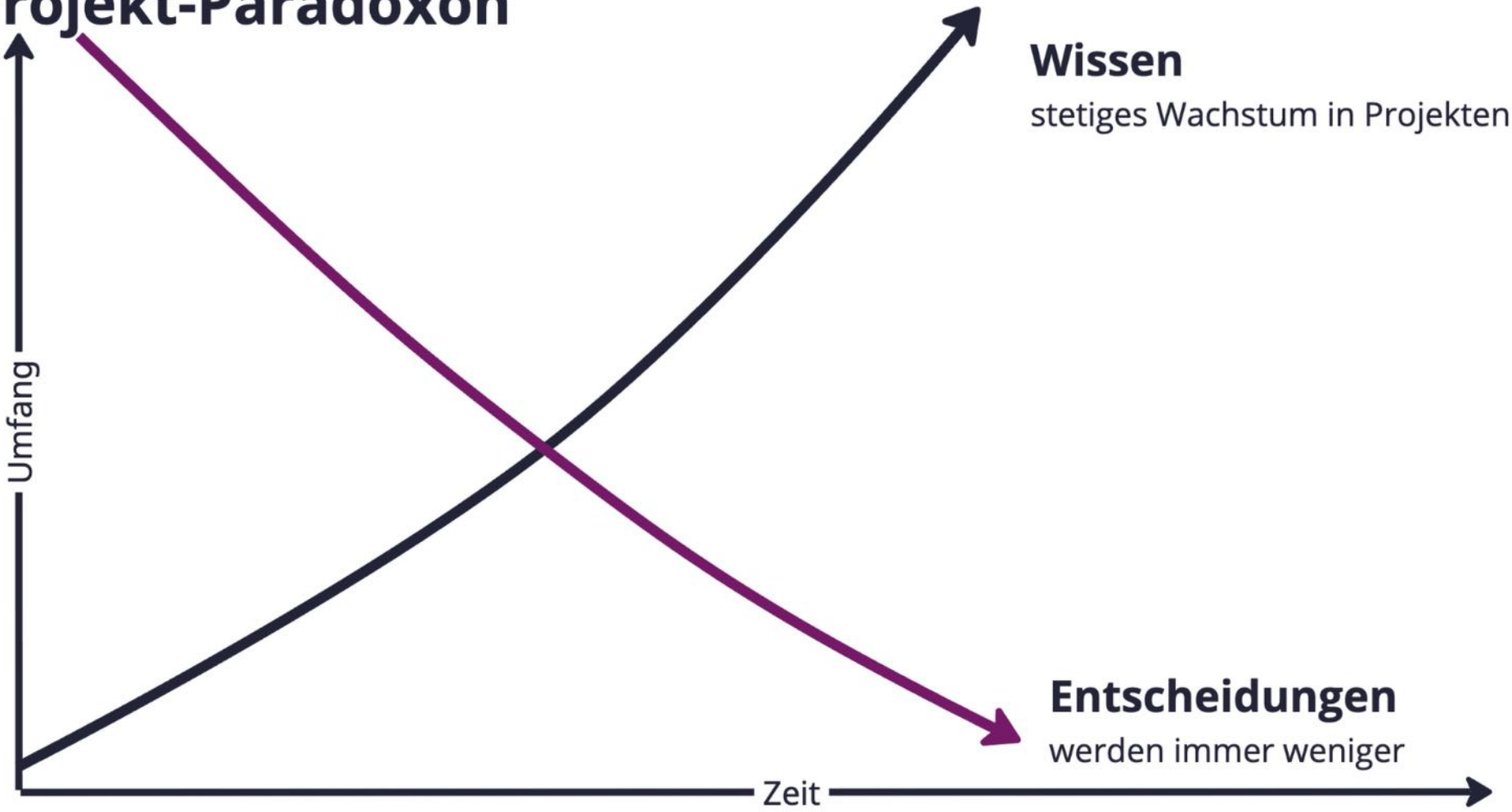
Kollaborative Architekturarbeit



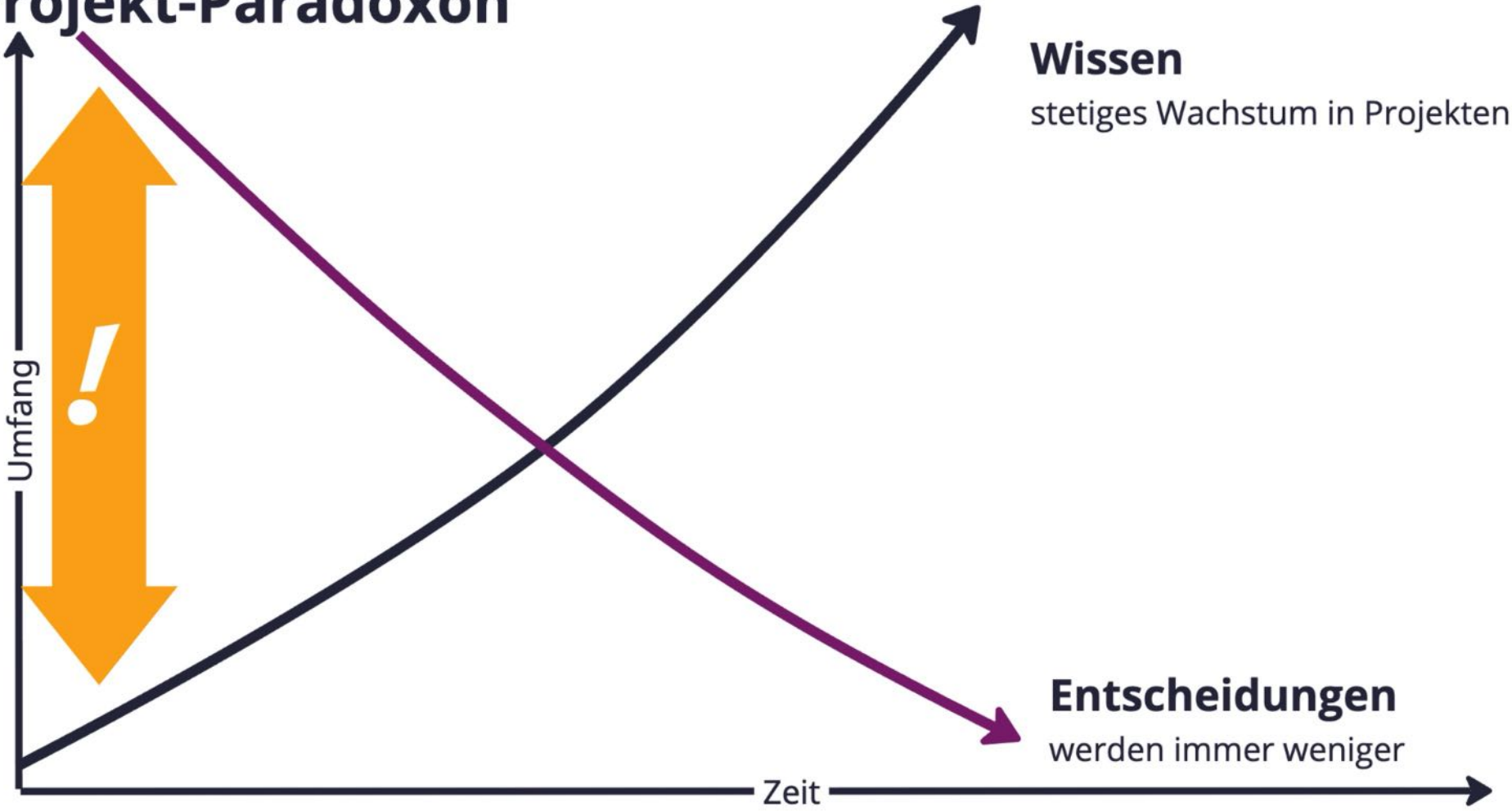
Projekt-Paradoxon



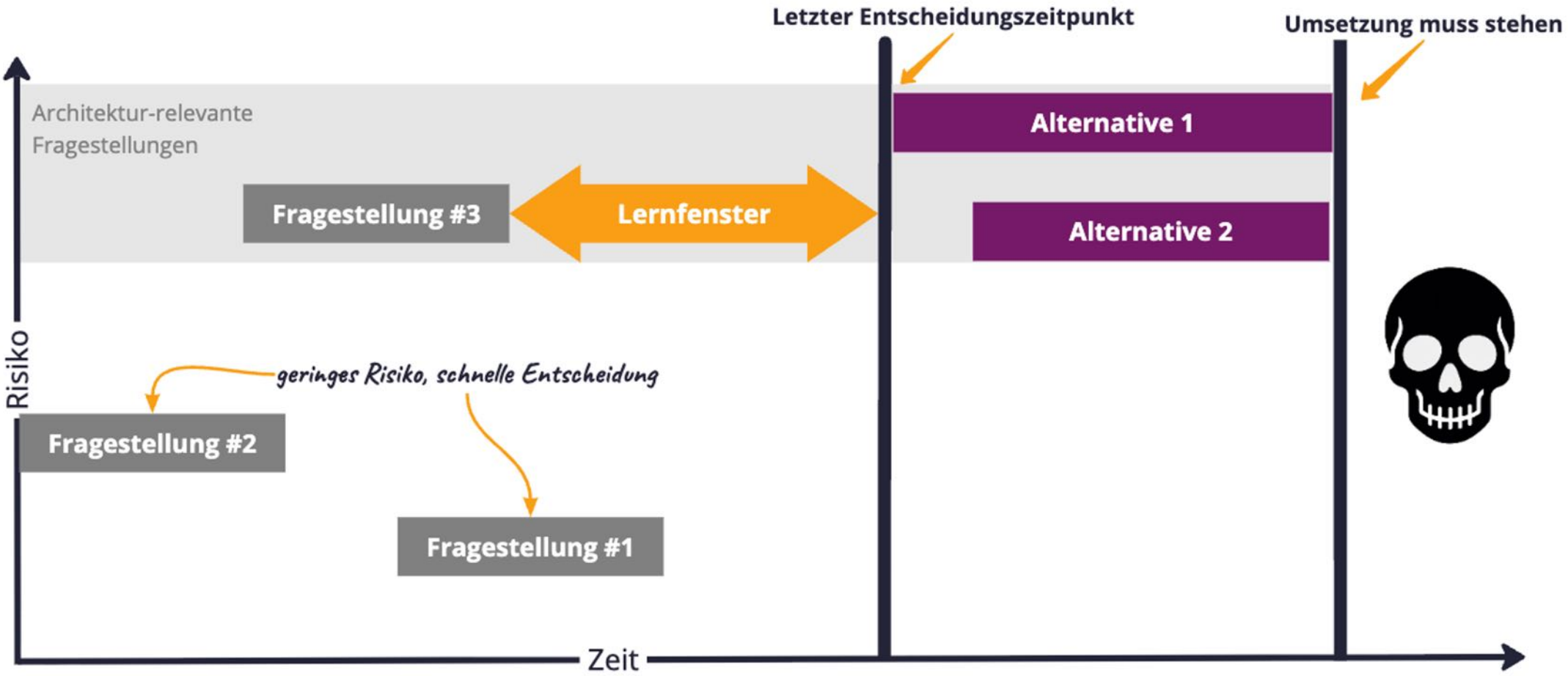
Projekt-Paradoxon



Projekt-Paradoxon



Last responsible Moment



Dem Mythos entkommen

Architekturentscheidungen

- langlebig
- schwer zu revidieren
- basieren auf Annahmen
- führen Risiken ein
- Alternativen
- Tradeoffs

Praktiken

- Kollaborative Architekturarbeit
- Annahmen hinterfragbar machen
- Lernfenster nutzen (“Last Responsible Moment”)
- Grundlagen und Alternativen dokumentieren (z.B.: ADR’s)

Mythos #3

Onion/Clean/Hexagonal
… das ist doch alles das Selbe!

… alles das Gleiche!

Die Ansätze haben viele Gemeinsamkeiten, aber auch Unterschiede.

Die Unterschiede sind in der Praxis nicht relevant.

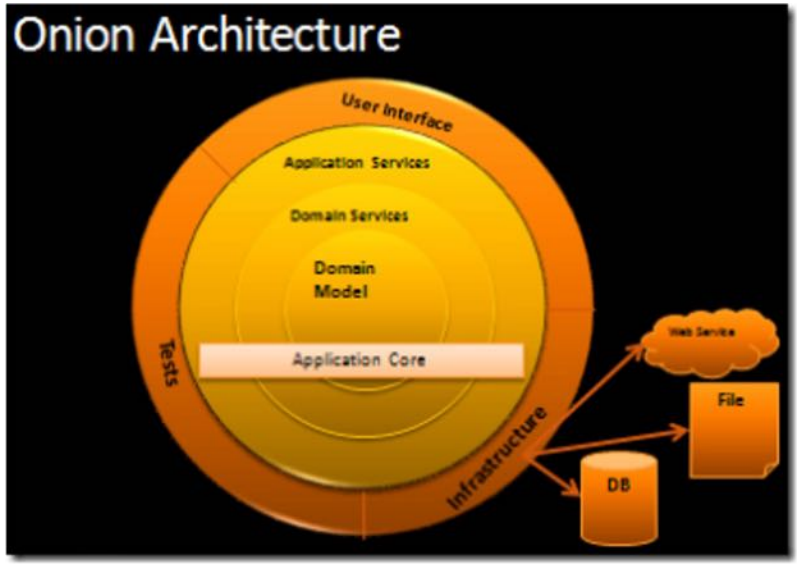
Das kommt drauf an …

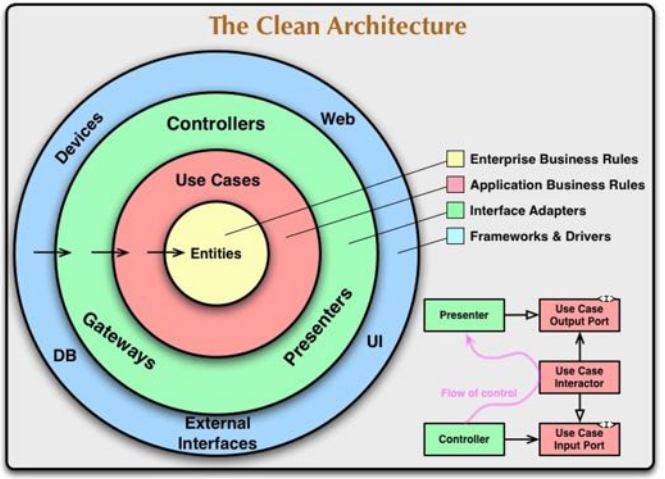
Am Ende ist das alles wieder nur eine Schichten-architektur

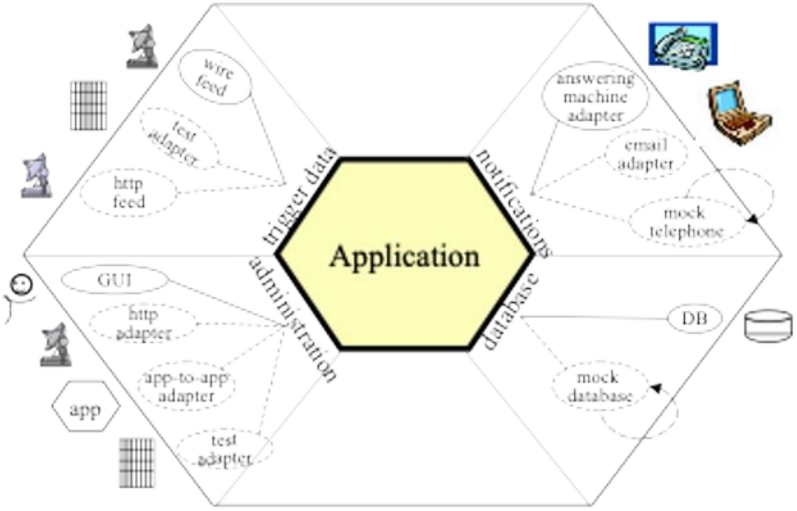
Guter Punkt.

Was ist das Problem ?

Ähnliche Ansätze, scheinbar in Konkurrenz

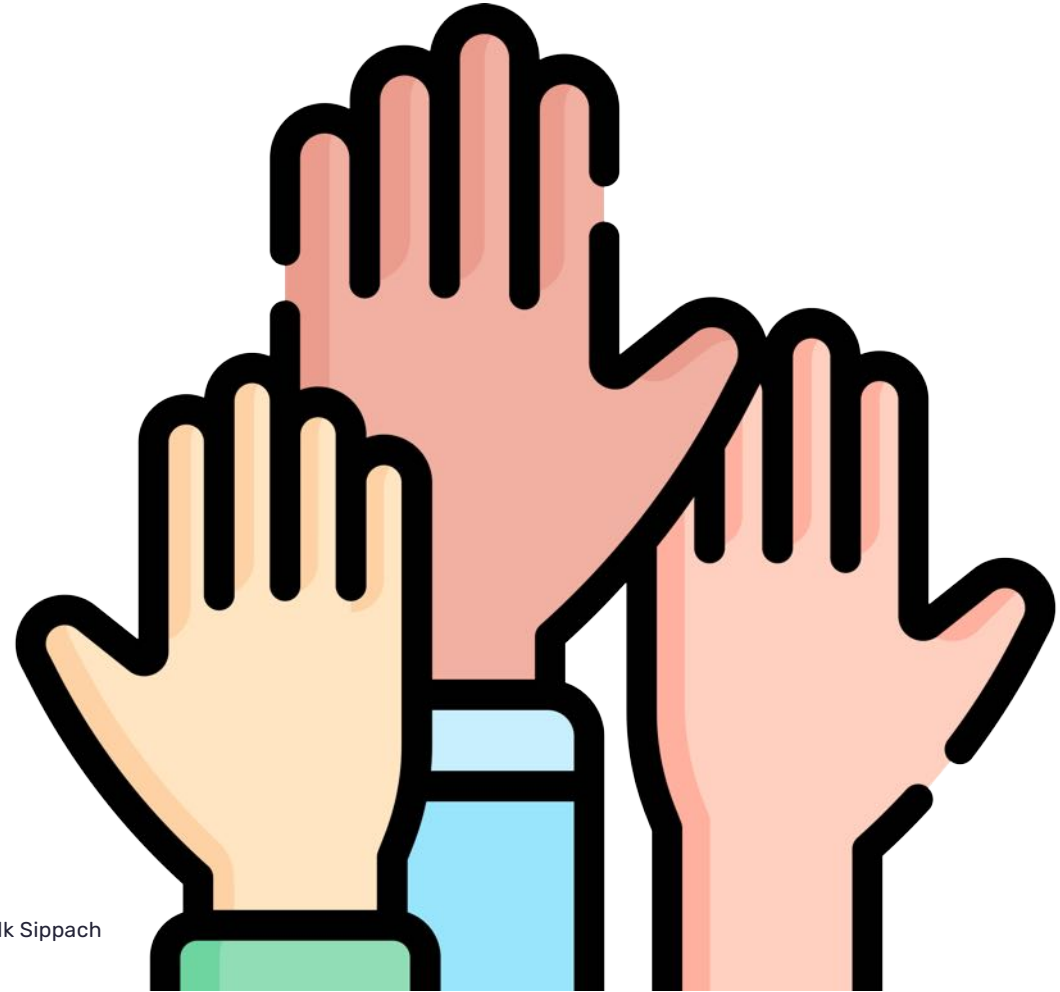






Eure Meinung ist gefragt

Ist das alles das Selbe?



Symptome

- Verunsicherung
- Subotpimales Design
- endlose Diskussionen

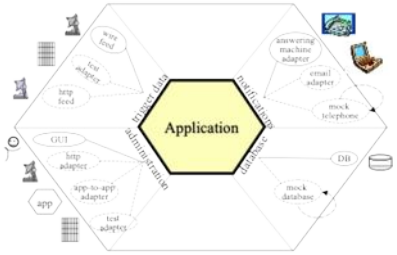
Historie

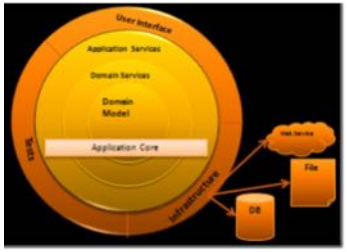
“The Hexagonal (Ports & Adapters) Architecture”, Alistair Cockburn

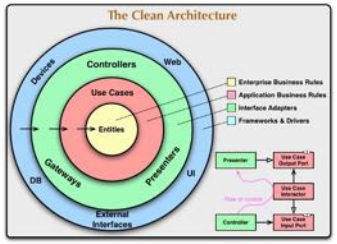
“The Onion Architecture”, Jeffrey Palermo

“The Clean Architecture”, Robert C. Martin

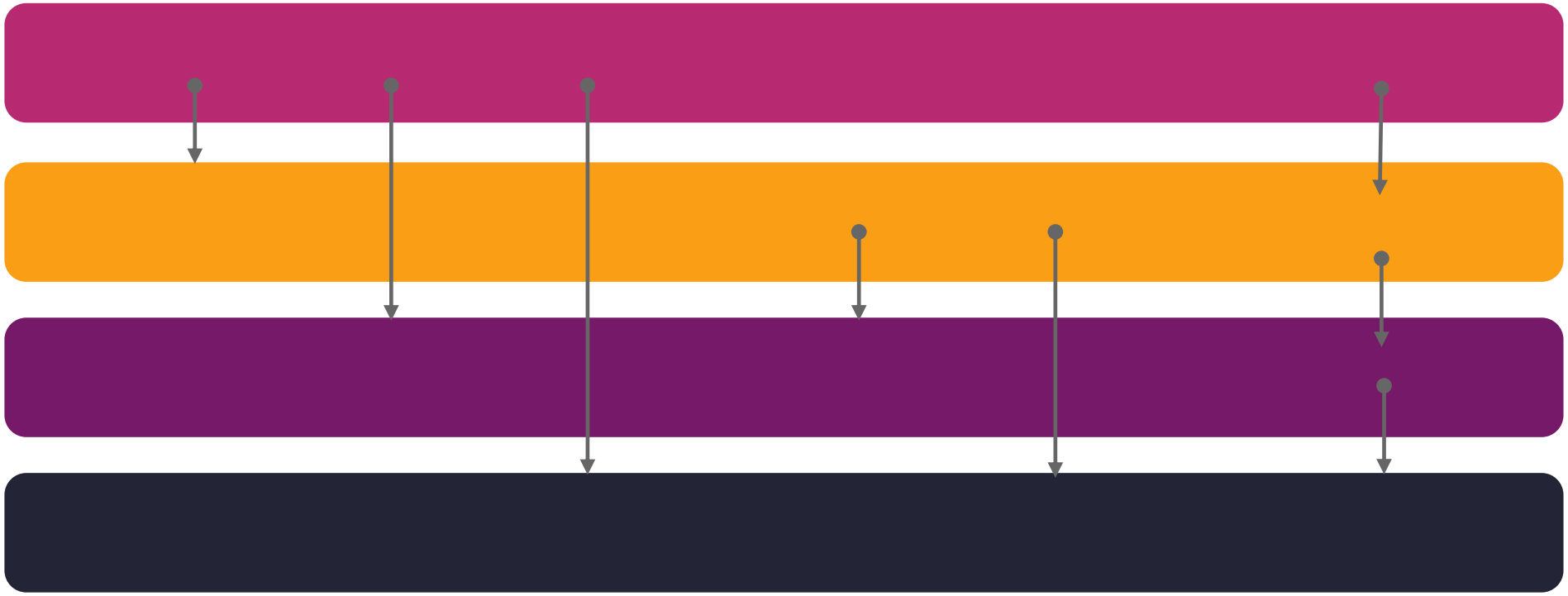
2005 — **2008** — **2012** — **HEUTE**





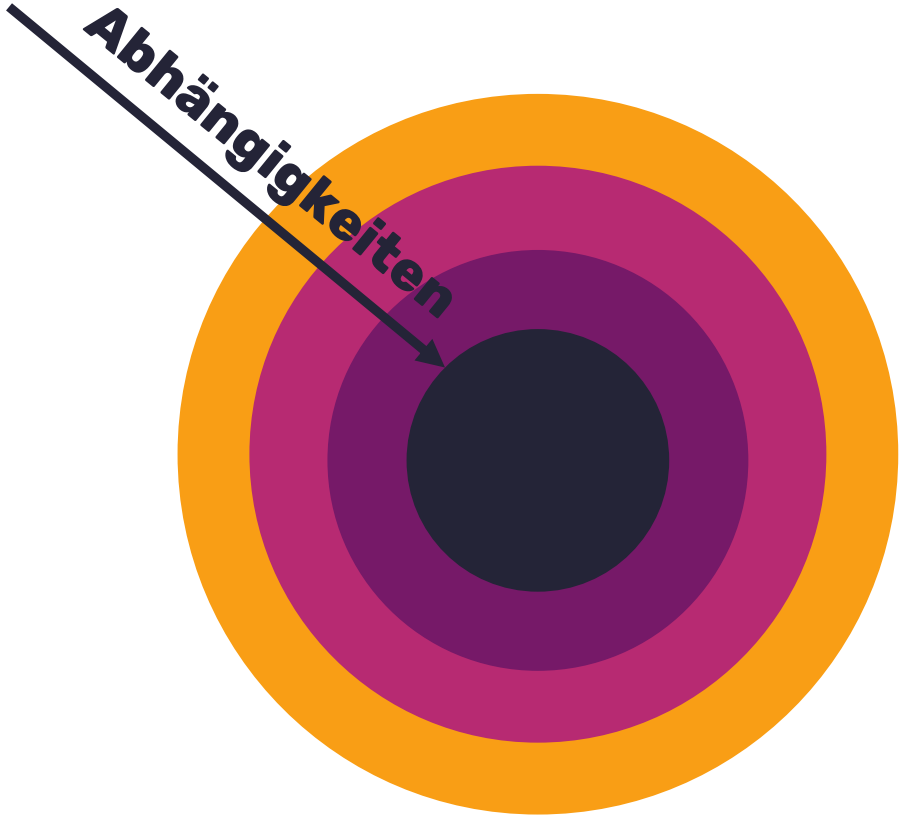


Relaxed Layer Architecture



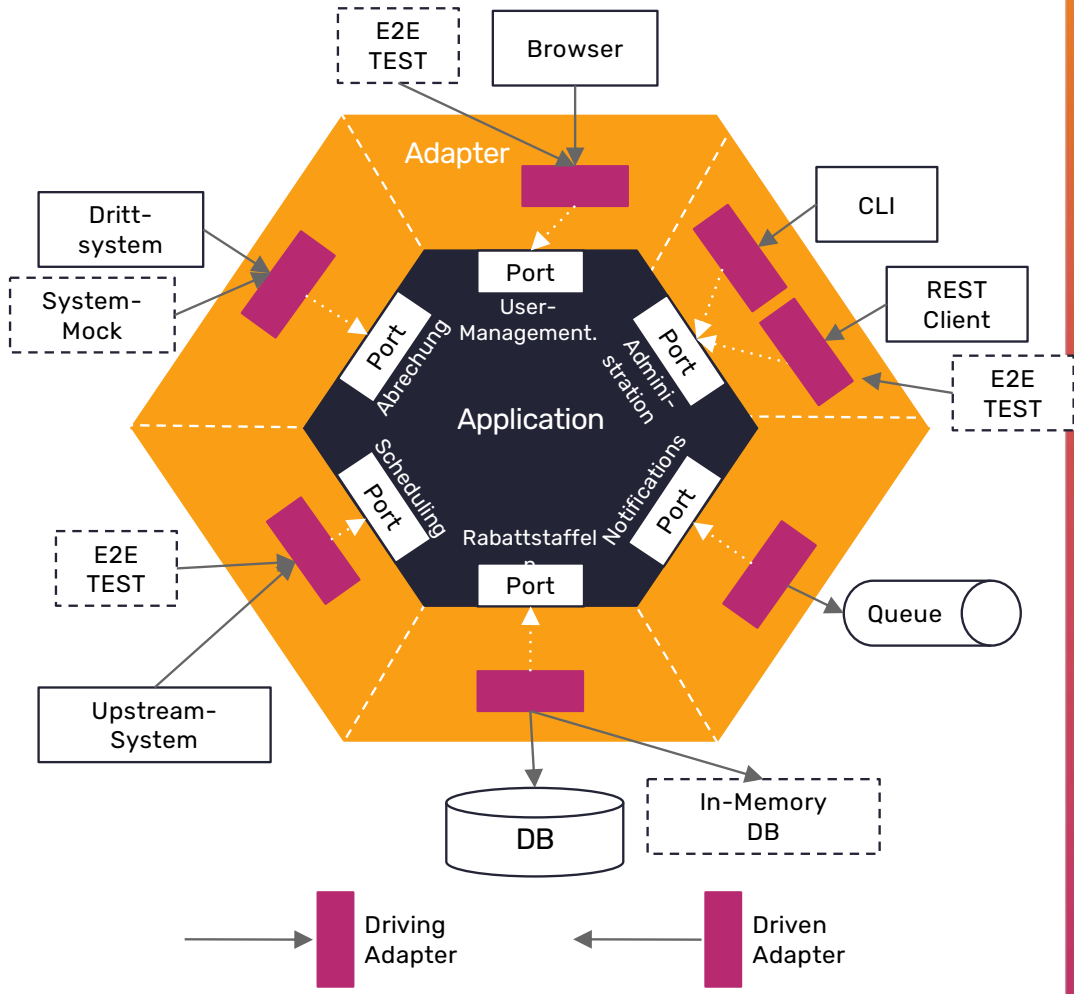
Dependency Rule

- Quellcode-Abhängigkeiten zeigen nur nach innen
- Innere Schichten wissen nichts von äußeren
  - Konzepte, Datenstrukturen, Variablen, Zustände, …
- Abstraktionsgrad nimmt nach innen zu
  - außen: “Technik”
  - innen: “Business”



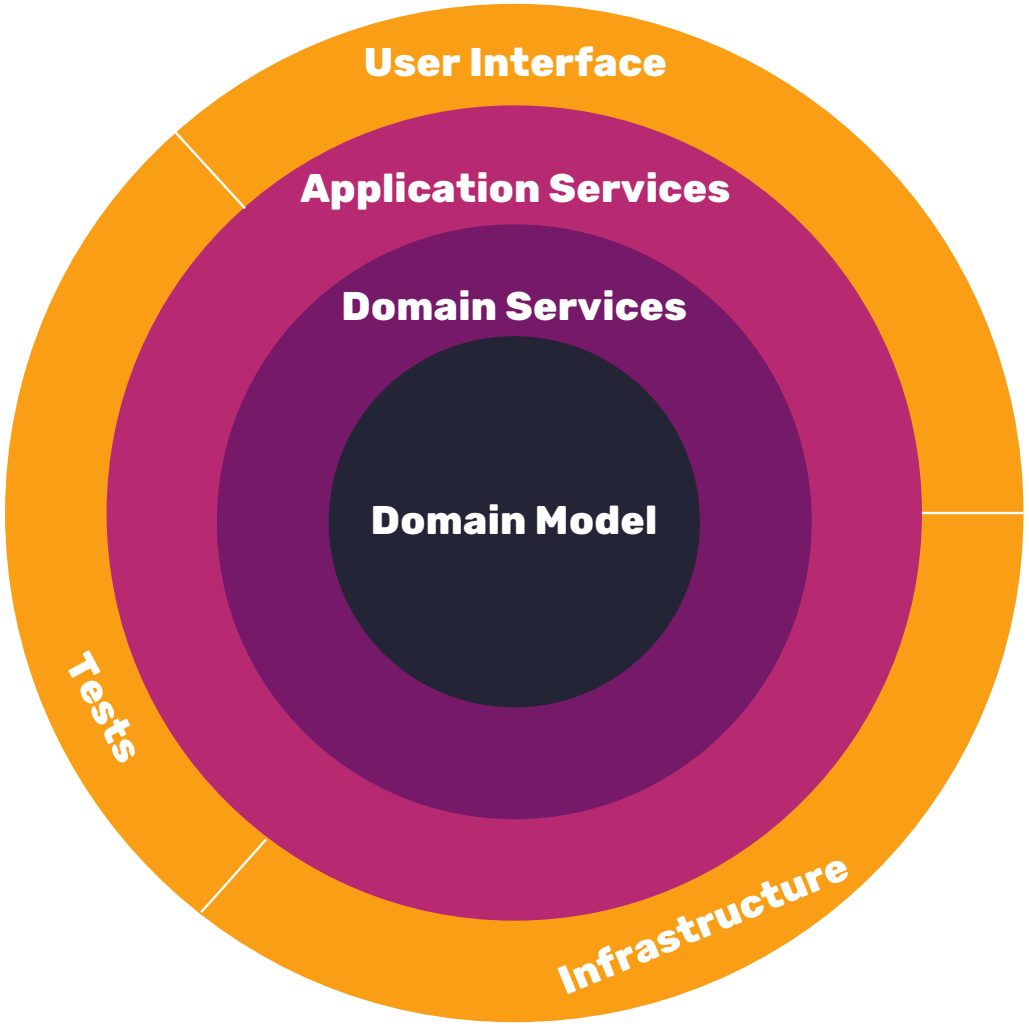
Ports & Adapters (“Hexagonal Architecture”)

- nur “**innen**” (Application) und “**außen**” (Adapter)
- **Application** von **Technik** entkoppelt
- **Ports:** fachlich motiviert
- **Adapter:** technische **Anbindung**
  - Driving
  - Driven



Onion Architecture

- Teilt wichtigste Charakteristiken von hexagonaler Architektur
- macht zusätzlich Aussagen über “das Innere”
- Kritikpunkte
  - “Layeritis”: “too much” für kleine Projekte
  - DDD: Domain Services getrennt vom Model?



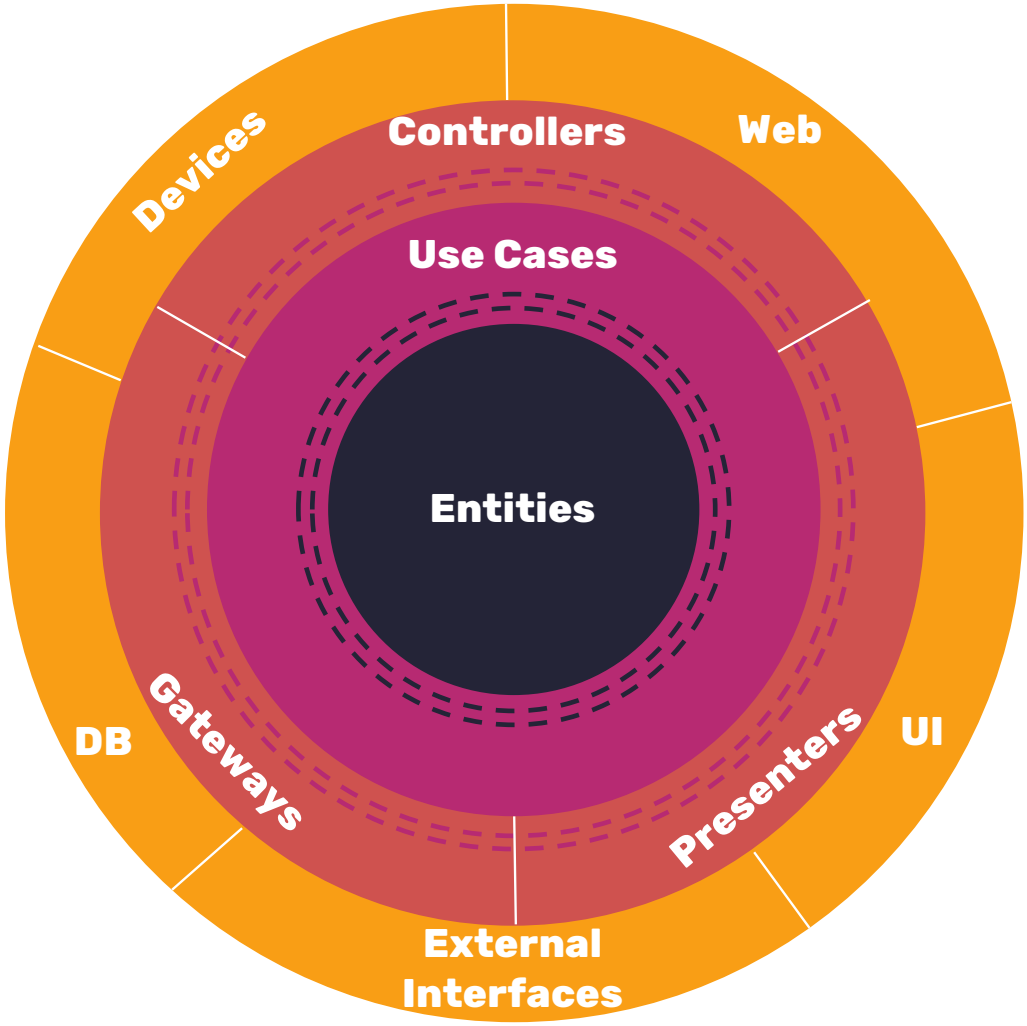
Clean Architecture

Konsolidierung ähnlicher Ansätze:

- BCE (Jacobsen, 1992)
- Ports & Adapters (Cockburn, 2000)
- Onion Architecture (Palermo, 2008)
- DCI (Coplien, Reenskaug, 2010)

Explizite Hinweise zur Umsetzung, z.B:

- Anzahl der Schichten nicht fest vorgegeben(!)
- Nur **simple Datenstrukturen** sollten Schichtengrenzen überschreiten
- Austauschbarkeit von Schichten / Segmenten, wenn diese obsolete werden → **Evolution**



Ok, … und jetzt?

- Motivation für Abstraktionen kennen, zB.
  - kleine Systeme: Hexagonal mit einfacher innerer Struktur reicht vielleicht aus
  - Benefit von Abstraktion durch weitere Schichten?
    - Ist “Schichten austauschen können” ein relevantes Qualitätsziel?
- Kenne die Qualitätsziele deiner Architektur

Mythos #4

Architekturentwurf muss Upfront erfolgen

tangible concepts

Wir planen die Architektur jetzt komplett durch, sonst wird das Projekt zum Chaos.

Chaos kriegen wir eher, wenn wir versuchen, Hellseher zu spielen.

Aber wir müssen doch einmal alles sauber definieren!

Oder wir definieren erst das, was wir wirklich verstehen — und lernen den Rest unterwegs.

Du meinst, Architektur entsteht nicht auf einmal?

Sie entsteht genauso wie gute Software: Schritt für Schritt.

Was ist das Problem?

Zu Beginn: Unbekannte Anforderungen

Vollständiges Architekturdesign: Trügerische Sicherheit

Teure Fehl-entscheidungen

Fehlende Lernschleifen

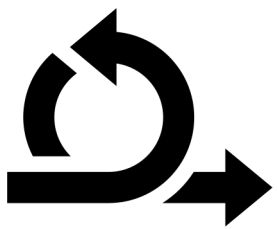
Risiko der Architektur-Erosion

Symptome

- Überdimensionierte Architektur
- Wenig echte Lernschleifen
- Hohe Änderungsängste im Team
- Frühe Festlegung auf ungeeignete Technologien
- Architekturmodelle passen nicht zur Realität
- „Planned Architecture“ vs. „Implemented Architecture“ driften auseinander
- Langsame Entscheidungen, weil alles „endgültig“ sein muss
- Prototypen und Experimente fehlen oder gelten als Zeitverschwendung
- Teams klammern sich an den ursprünglichen Plan
- Hohe Reibung, sobald neue Anforderungen kommen

Wie begegnen wir dem Mythos?

Architekturaufwand dem Problem angemessen



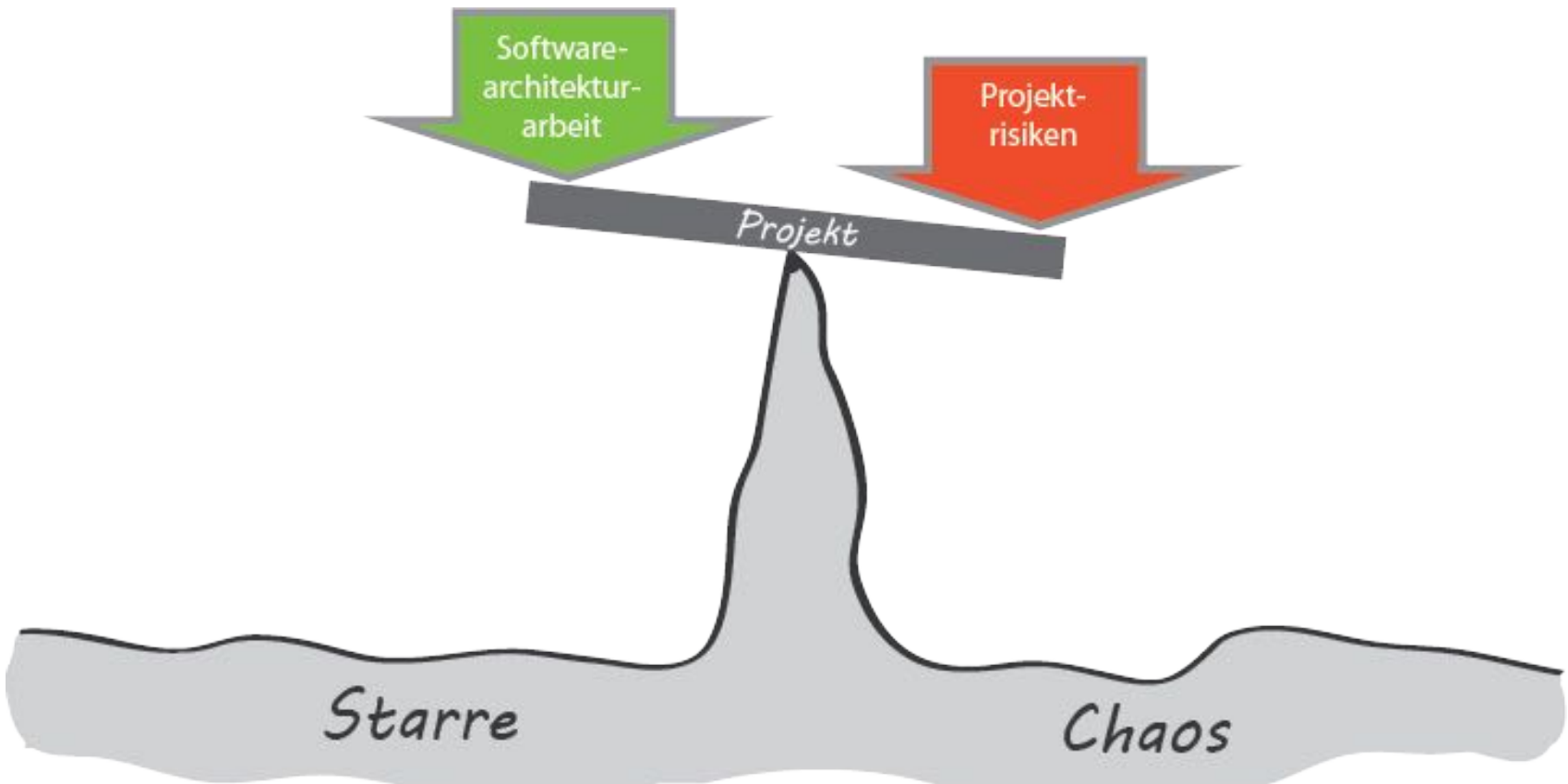
Architektur entsteht iterativ

LVM

Architektur-vision

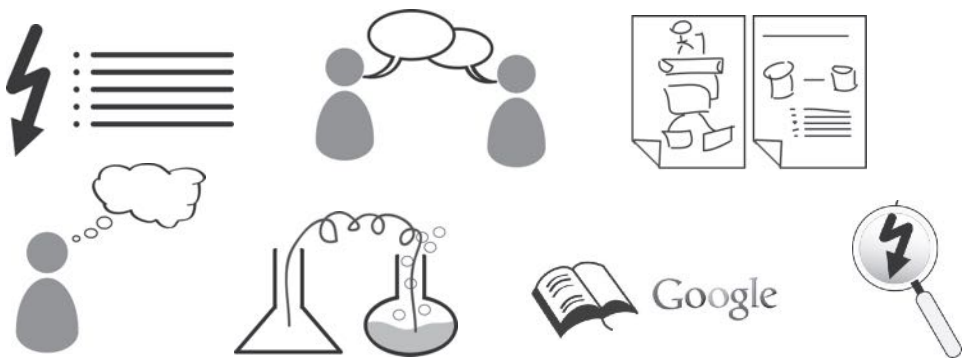
Walking Skeleton

Generell...



Toth, Stefan: "Vorgehensmuster für Softwarearchitektur", Carl Hanser Verlag, 2025

Architekturarbeit ist …



- Aufwand
- Kein Code
- Kein direkter Kundennutzen

Risikominderungs-maßnahme

für nicht triviale Problemstellungen

Gute Prinzipien zum Umgang mit Risiken

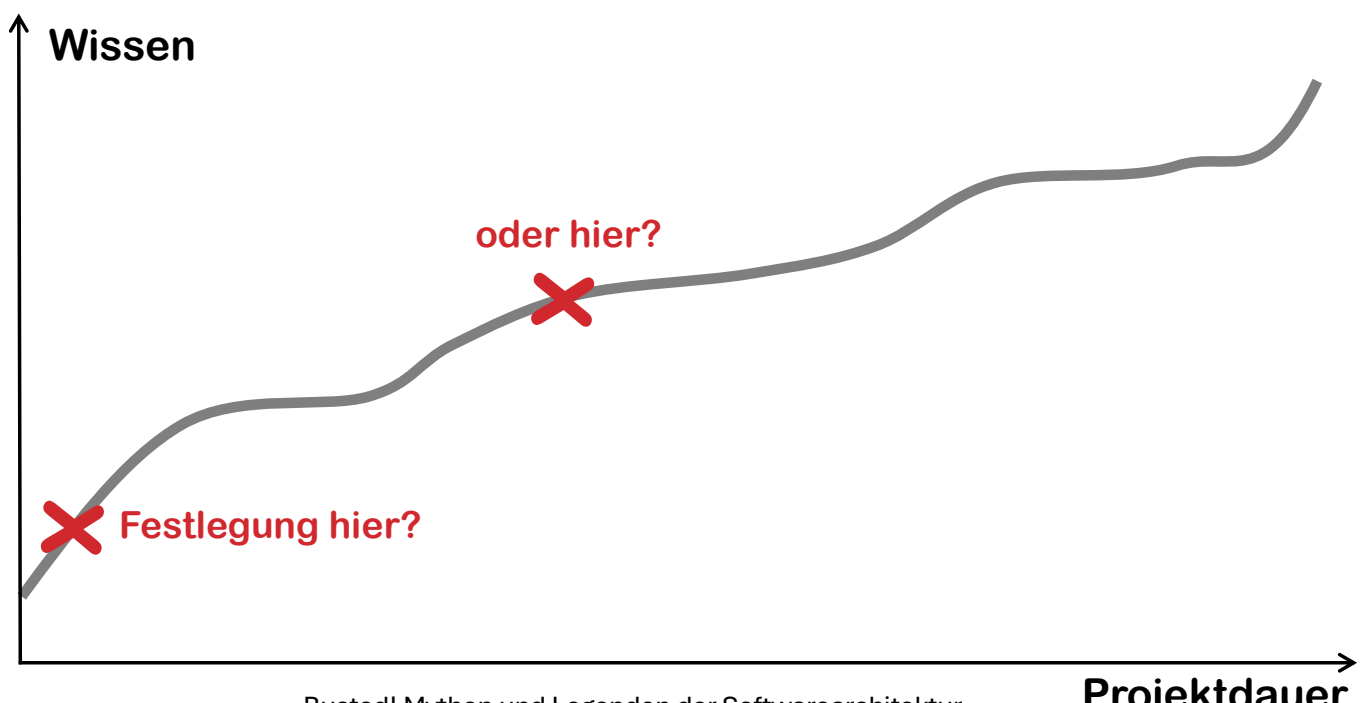
- Fragestellungen **früh erkennen** und analysieren
- Möglichst **spät entscheiden** (bei mehr als einer sinnvollen Alternative)

Das schafft ein großes "Lernfenster"

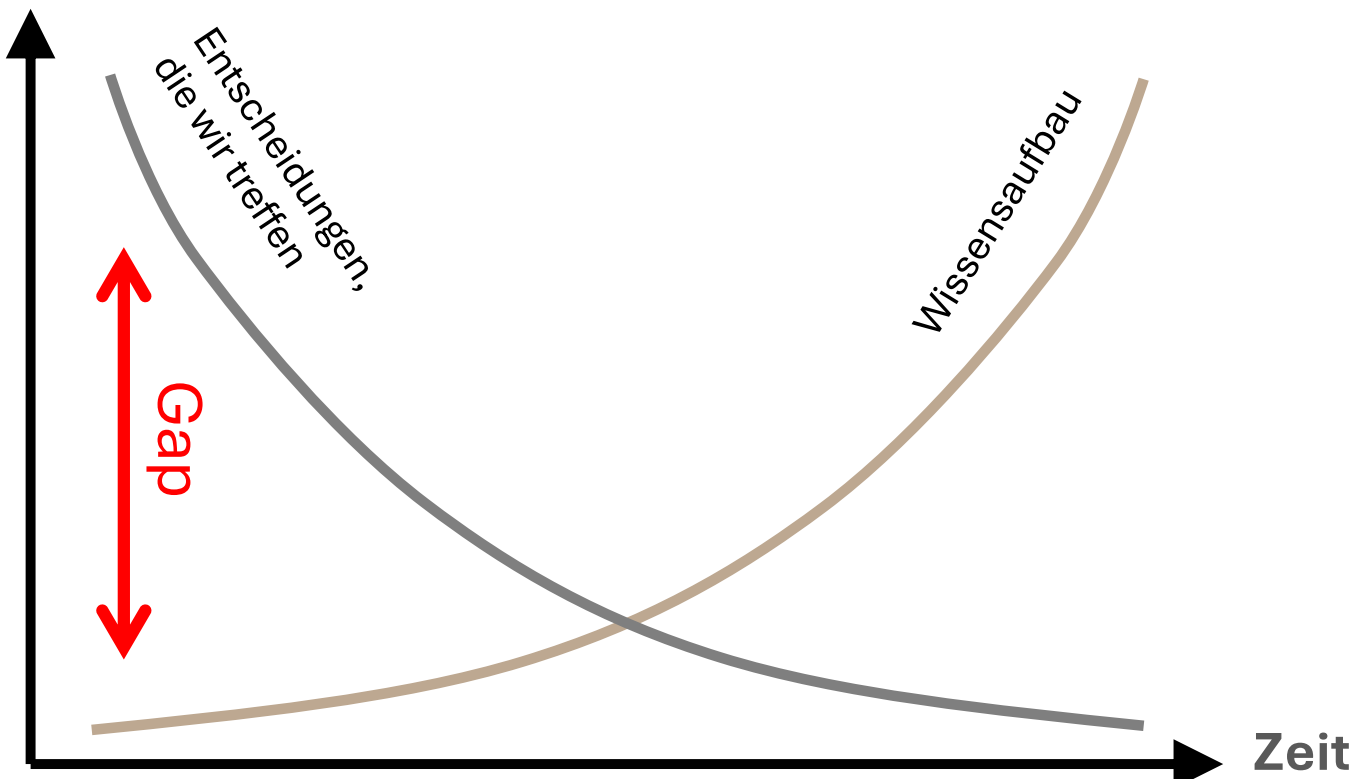
(bzw. Zeit zur Risikominderung)

Fragestellungen wann beantworten?

“Architecture is about the important stuff” (Martin Fowler)

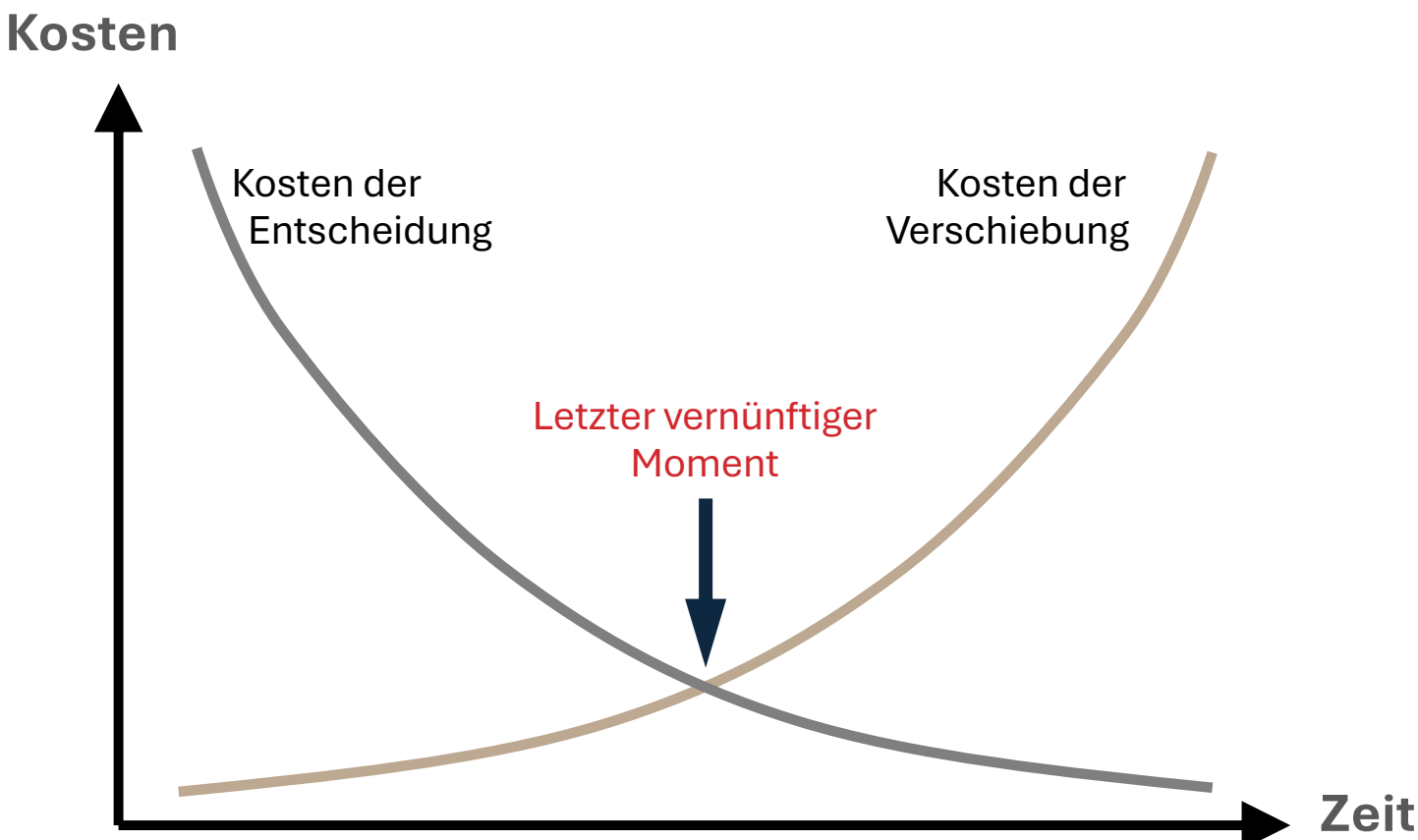


Getroffene Entscheidungen vs. aufgebautes Wissen

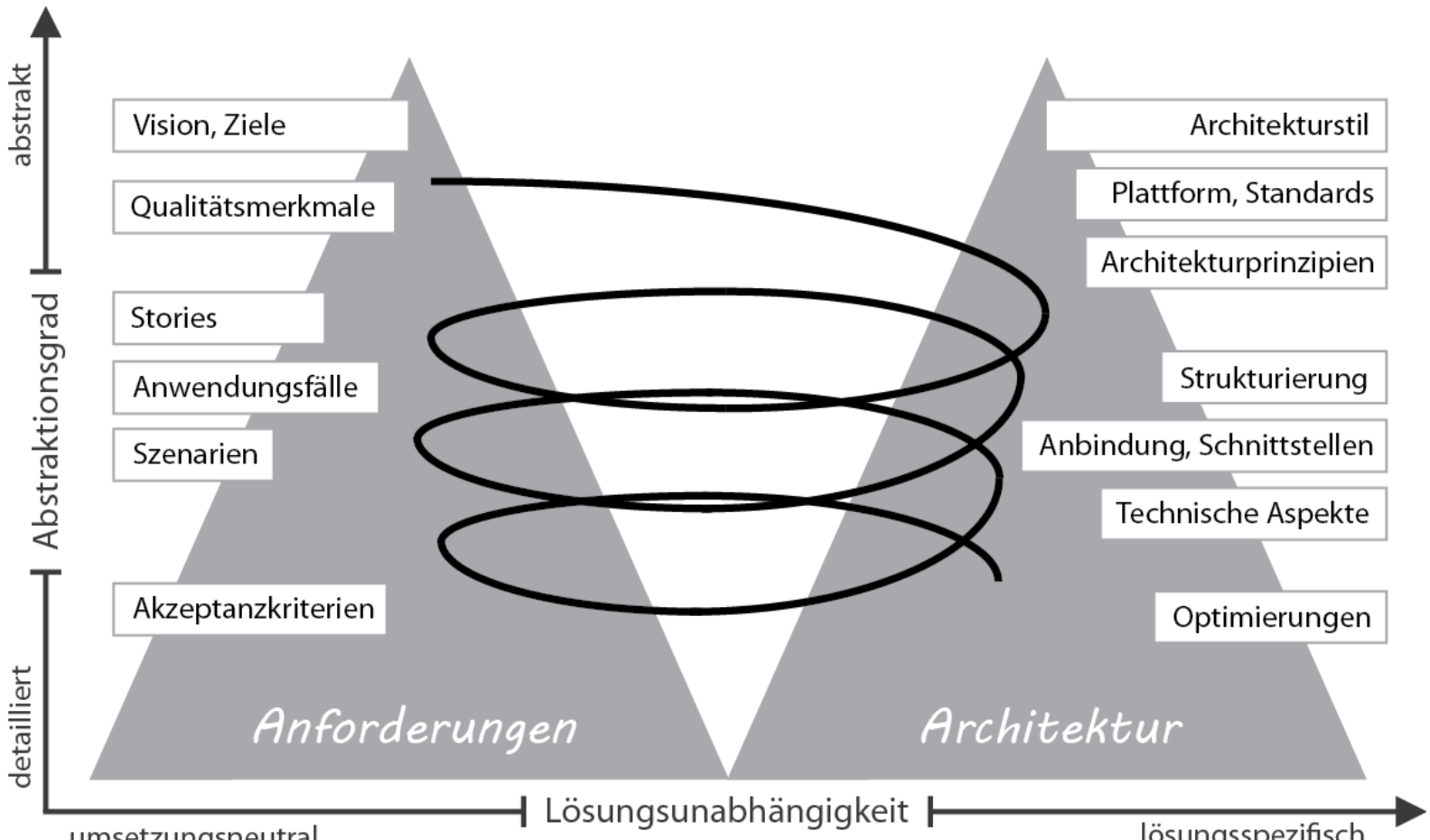


Der letzte vernünftige Moment

Der letzte vernünftige Moment



Iterative Architekturarbeit



Toth, Stefan: "Vorgehensmuster für Softwarearchitektur", Carl Hanser Verlag, 2025

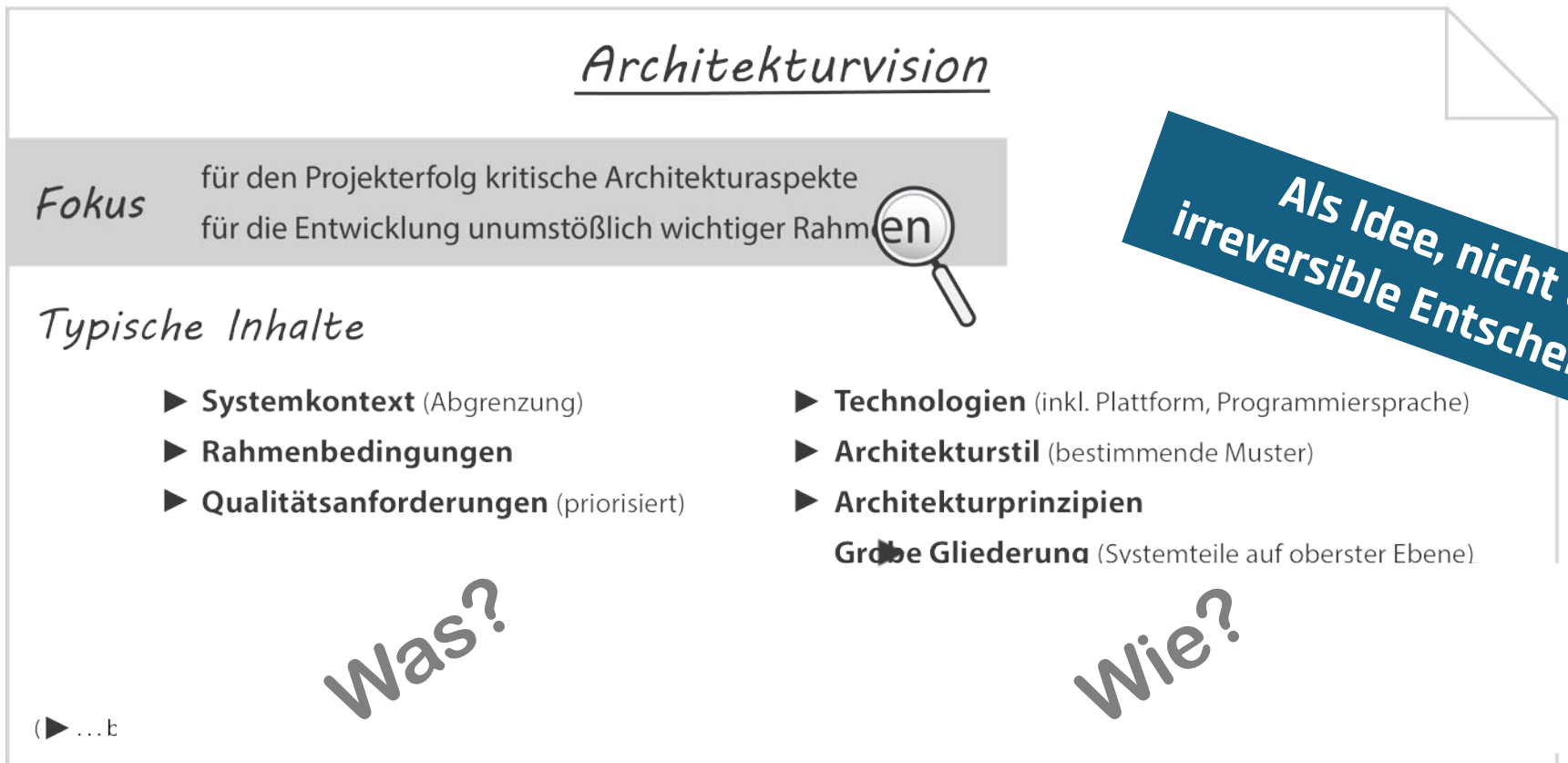
statt **Big Design Up Front**

Eine Architekturvision ist eine schlanke, sich stetig weiterentwickelnde Übersicht zur aktuellen Architekturidee und deren Motivation.

Stefan Toth

Architekturvision statt BUFD

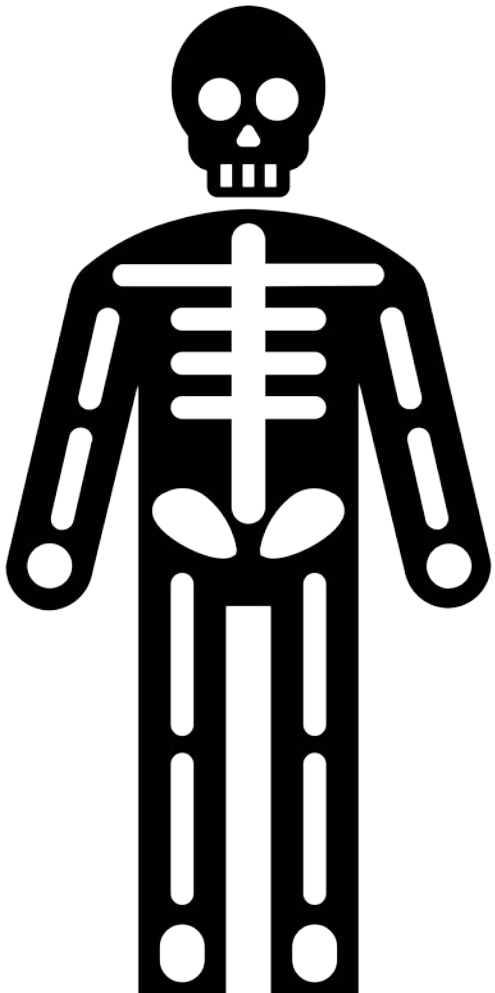
Big Design
Up Front



Als Idee, nicht als irreversible Entscheidung!

Toth, Stefan: "Vorgehensmuster für Softwarearchitektur", Carl Hanser Verlag, 2025

2004



Frühe **End-to-End-Funktionalität** mit minimalem Umfang

Ein schlankes, aber lauffähiges System als **technisches Fundament**

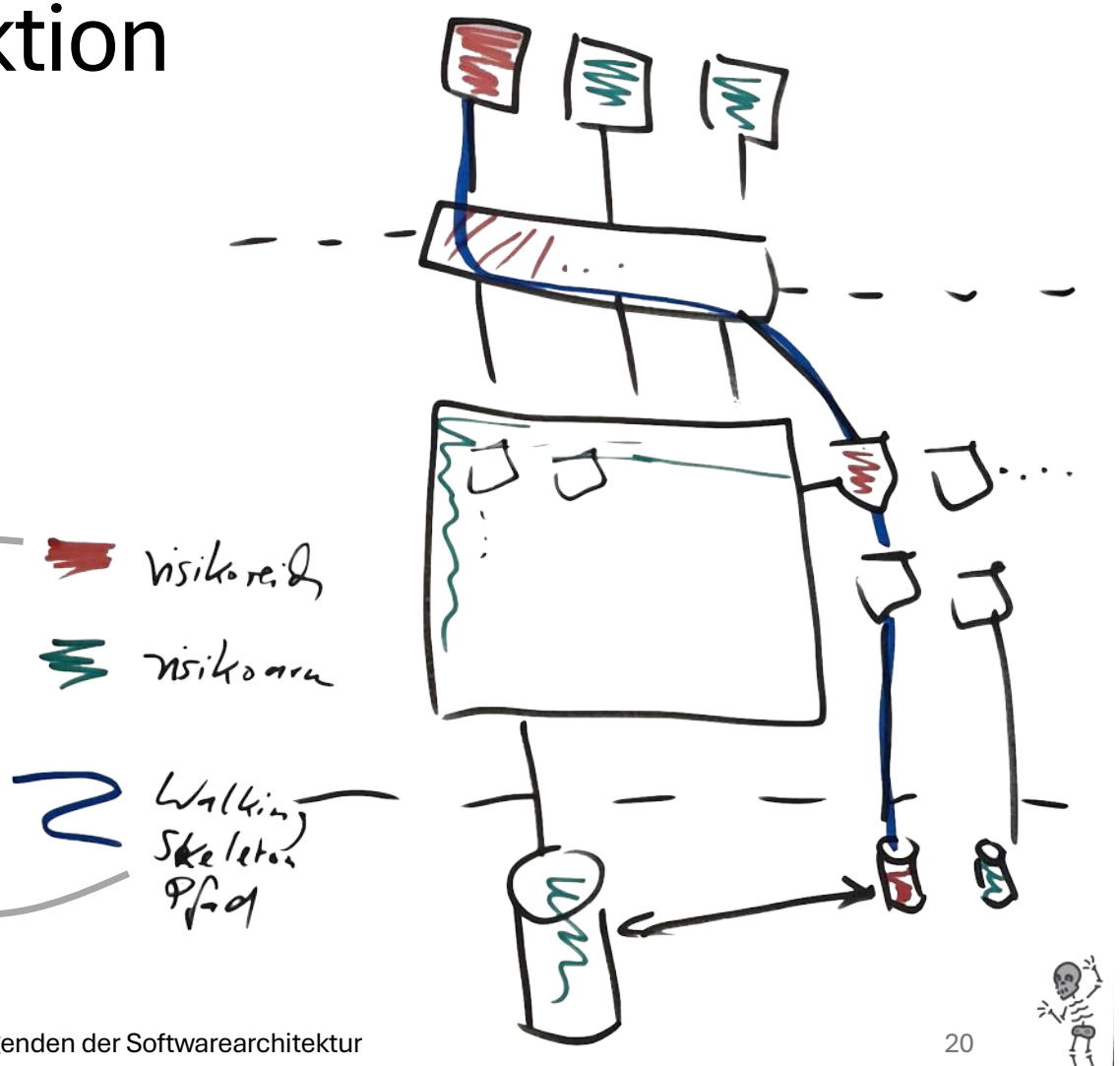
Identifiziert **Risiken**, **etabliert Architektur & CI/CD** von Anfang an

Walking Skeleton in Aktion

Die **momentan beste Idee** möglichst **früh widerlegen!**

Risiken
früh angehen!

kein MVP,
kein Durchstich/Prototyp



Mythos #5

Agile Teams brauchen keinen Architekten

Architekten bremsen uns nur aus.

Chaos bremst euch viel mehr.

Wir entscheiden einfach alles im Team.

Und wer sorgt dafür, dass es zusammen-passt?

Das ergibt sich schon.

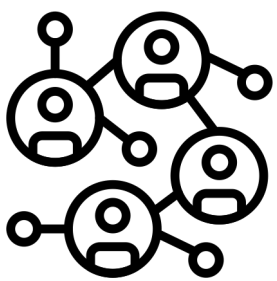
Unwahrscheinlich, Struktur entsteht nicht zufällig.

Was ist das Problem?

Fehlende technische Ausrichtung

Inkonsistente Entscheidungen

Zufällige Architektur

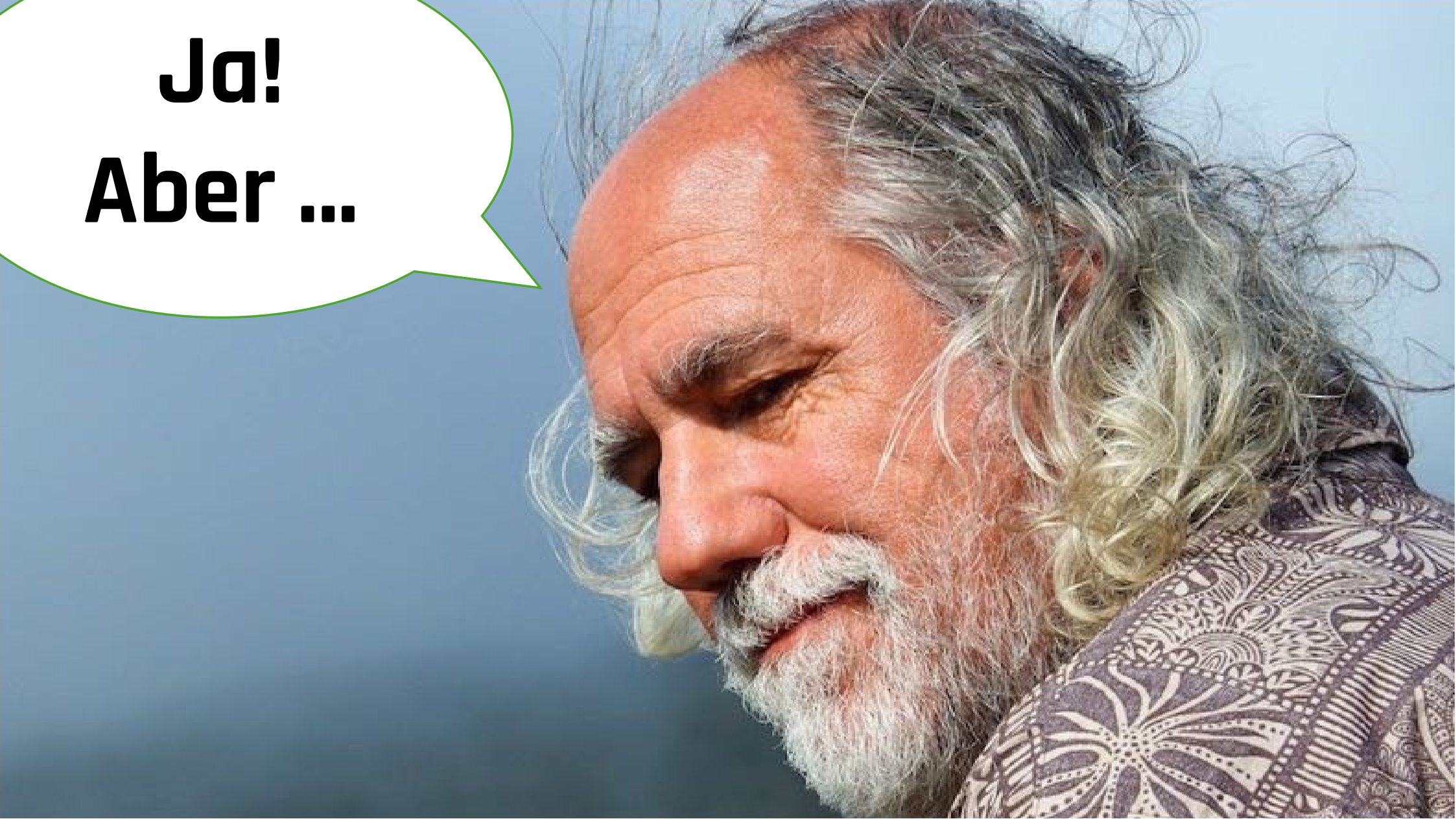


Koordinationsprobleme zwischen Teams

Fehlende Qualitäts-kriterien & Leitplanken

Gibt es eigentlich immer eine Architektur?

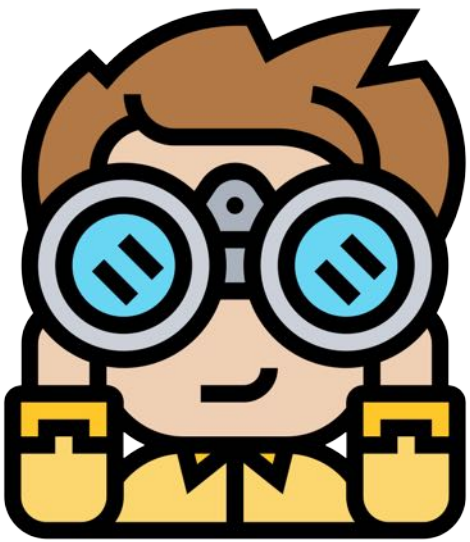
tangible concepts embarc


Ja!
Aber ...

Accidental Architecture

Grady Booch:
„The Accidental Architecture“
IEEE Software 2006

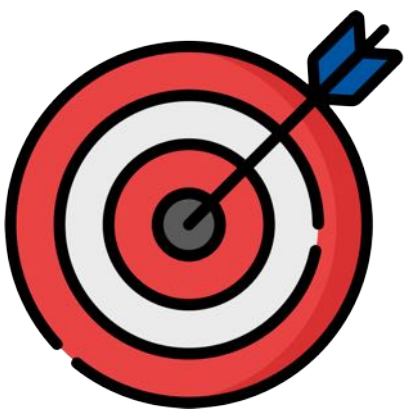
vs.



zufällig **gewollt**

https://www.inf.ed.ac.uk/teaching/courses/seoc/2006_2007/resources/Arc_Accidental.pdf

Bewusst oder unbewusst?

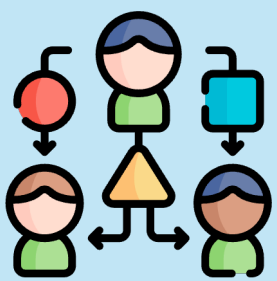


Anforderungen kennen!!!

Symptome

- Uneinheitliche technische Entscheidungen
- Wachsende Abhängigkeiten zwischen Teams
- Inkonsistente Schnittstellen & Modelle
- Zunehmende technische Schulden
- „Lokale Optimierungen“ statt Gesamtsicht
- Hoher Abstimmungsaufwand bei Änderungen
- Fehlende Ownership für Architekturfragen
- Architektur driftet mit der Zeit auseinander
- Teams blockieren sich gegenseitig
- Qualitätsziele sind unklar oder variieren stark

Wie begegnen wir dem Mythos?

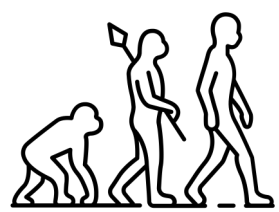


Architektur als Teamaufgabe

Moderation statt Vorgaben

Gemeinsame Architekturprinzipien

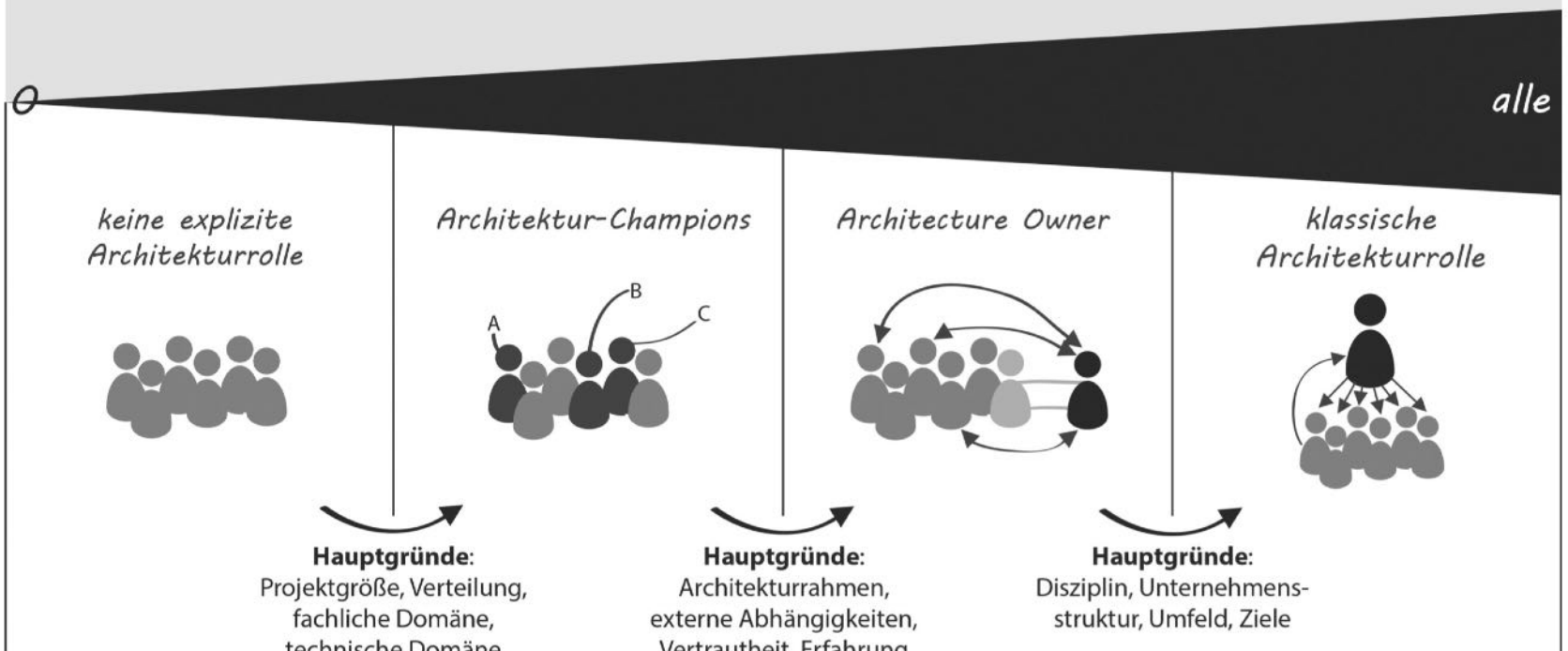
Transparenz



Evolutionäre Architektur fördern

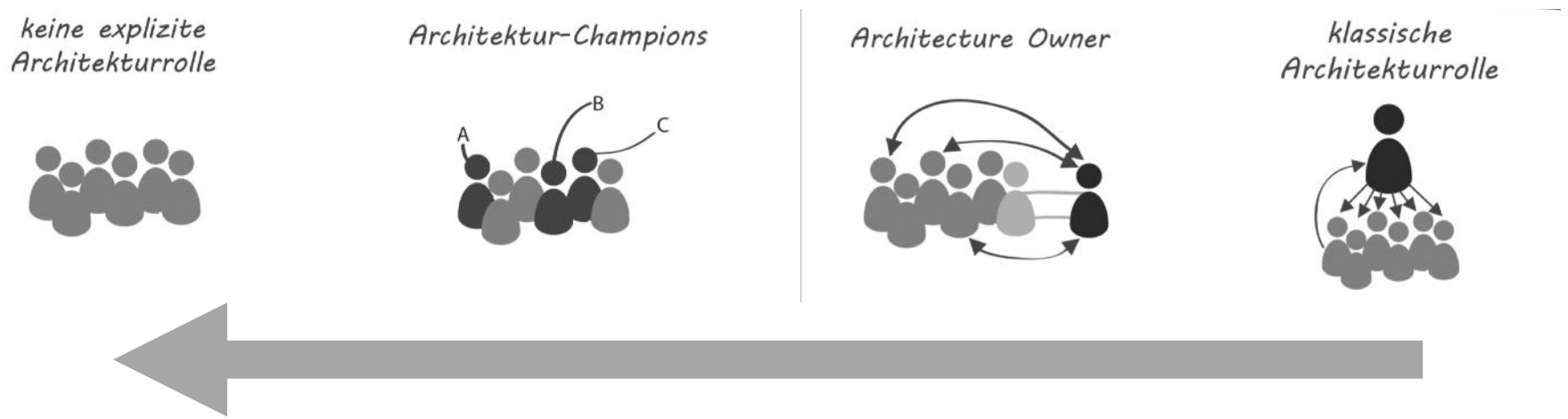
"Rollenfaktoren" für Architektur

Projektgröße:	mehrere Teams	**Vertrautheit**:	Erstes Vorhaben in dieser Zusammensetzung
Verteilung:	geografisch verteilt	**Erfahrung**:	Viele unerfahrene Teammitglieder
Fachliche Domäne:	komplex, neu	**Disziplin**:	Verantwortungsübernahme mangelhaft
Technische Domäne:	schwierig, herausfordernd, neu	**Unternehmensstruktur**:	stark hierarchisch
Architekturrahmen:	muss erst geschaffen werden	**Umfeld**:	reguliert oder von Standards bestimmt
Externe Abhängigkeiten:	hoch	**Ziele**:	Architekturziele in Konflikt (auch zu Projektzielen)



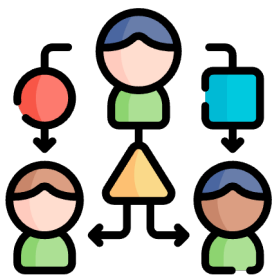
Toth, Stefan: "Vorgehensmuster für Softwarearchitektur", Carl Hanser Verlag, 2025

Praktiken für offenere Modelle



Architektur-Katas	Architektur-Prinzipien	Gemeinsame Entscheidungen	Communities of Practice
Architekturarbeit im Backlog	Qualitative, automatisierte Tests	Reflexions-Workshops	Architekturwand

Wie begegnen wir dem Mythos?

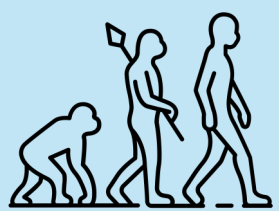


Architektur als Teamaufgabe

Moderation statt Vorgaben

Gemeinsame Architekturprinzipien

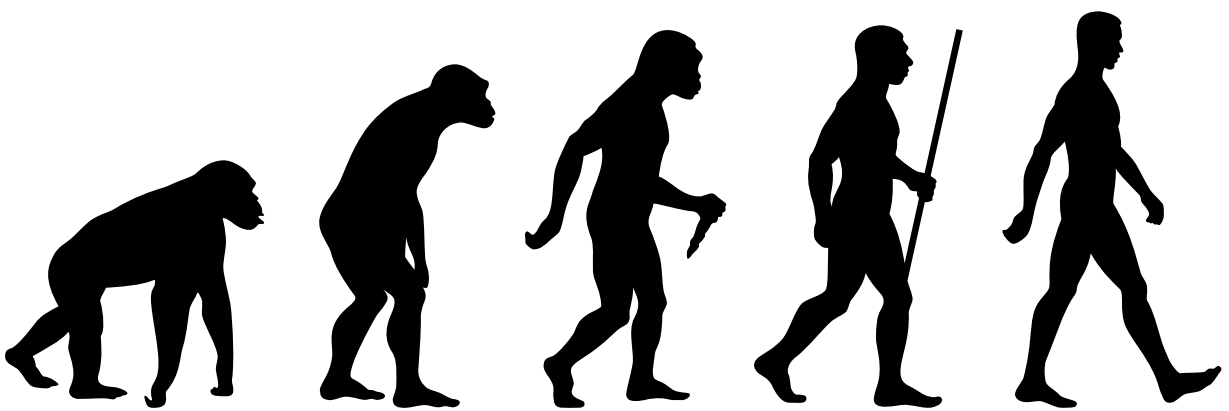
Transparenz



Evolutionäre Architektur fördern

Die Natur hat Lösungen …

… für komplexe Probleme …



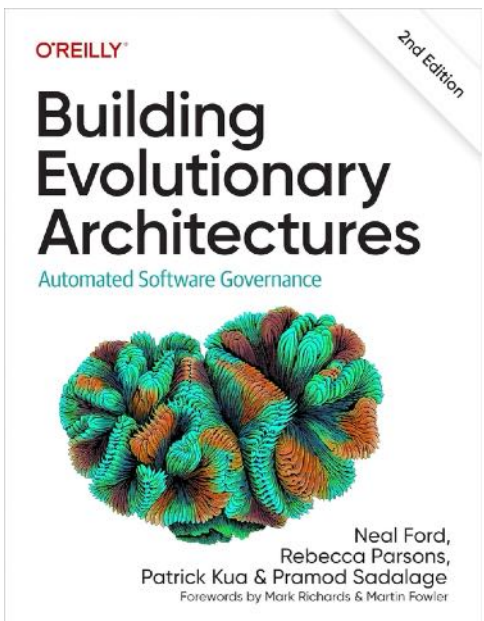
… durch kleinteilige, stetige Anpassung

tangible concepts embarc

Evolutionäre Algorithmen

Natürliches Vorbild	Evolutionärer Algorithmus	Beispiel
Organismus	**Lösungskandidat**	Flugzeugflügel
Fortpflanzungserfolg	**Wert der Fitnessfunktion**	Strömungswiderstand
Natürliche Mutation	**Mutation**	Änderung der Form

Übertragung auf Softwarearchitektur



*„Eine **evolutionäre Architektur** unterstützt **geleitete, inkrementelle Veränderungen** über **mehrere Dimensionen** hinweg.“**

N. Ford, R. Parsons, P. Kua, P. Sadalage:
Building Evolutionary Architectures: Automated Software Governance
O'Reilly 2022

* Engl.: “An evolutionary architecture supports guided, incremental change across multiple dimensions.”

Mythos #6

Microservices sind moderner,

Monolithen sind technisch rückständig

Monolithen sind doch längst überholt.

Komisch, viele davon laufen stabil seit Jahren.

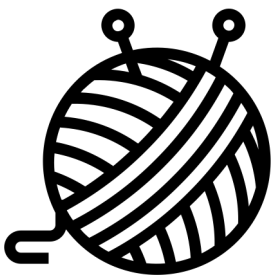
Microservices sind viel moderner.

Aber nur, wenn die Probleme dazu passen.

Ein Monolith bremst uns aus.

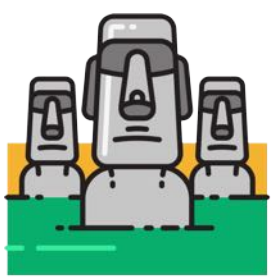
Ein modularer Monolith bremst niemanden aus.

Was ist das Problem?



Distributed Big Ball of Mud

Komplexität wird unterschätzt



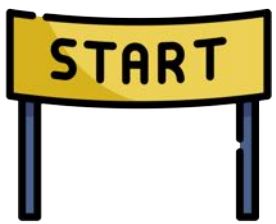
Pauschal: "Monolithen sind per se schlecht"

Falsche Motivation für Microservices

Symptome

- Microservices als Default-Lösung
- Verteilung ohne klaren fachlichen Schnitt
- Hoher Overhead durch viele Services
- Wachsende Komplexität im Betrieb
- Inkonsistente Schnittstellen & Datenmodelle
- Teams blockieren sich durch Abhängigkeiten
- Monolith wird nicht modularisiert, sondern verteufelt
- Architektur-Drift zwischen Services
- Steigende Betriebs- und Infrastrukturkosten
- Fehlende Observability & Monitoring-Probleme

Wie begegnen wir den Mythos?

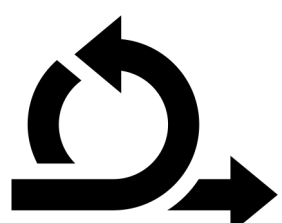


Modularität first

Modulith als Default

Organisation und Architektur zusammen denken

Ehrliche Einschätzung operative Reife



Iterative Modernisierungen

Erkenntnisse nach dem Hype

>> Microservices lösen bestimmte Probleme brillant – und schaffen gleichzeitig neue. <<

Microservices in kurz …

*“In short, the microservice architectural style is an approach to developing a **single application** as a suite of **small services**, each running in its own process and communicating with lightweight mechanisms…”*

(James Lewis, Martin Fowler, 2014)

➔ https://martinfowler.com/articles/microservices.html

Charakteristische Eigenschaften

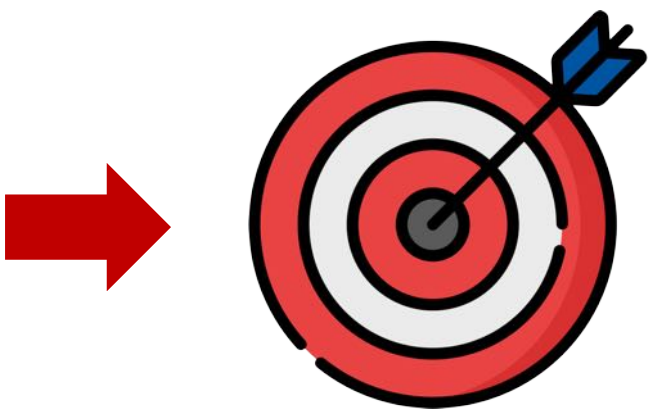
- Zerlegung in relativ kleine (fachliche) Services
- Services sehr lose gekoppelt
- Services einzeln installierbar und upgradebar
- Dezentrale Datenhaltung
- Hoher Freiheitsgrad bei Technologieauswahl

Hauptsache gut geschnitten?

"If you can't build a well-structured monolith, what makes you think you can build a well-structured set of microservices?"

Simon Brown

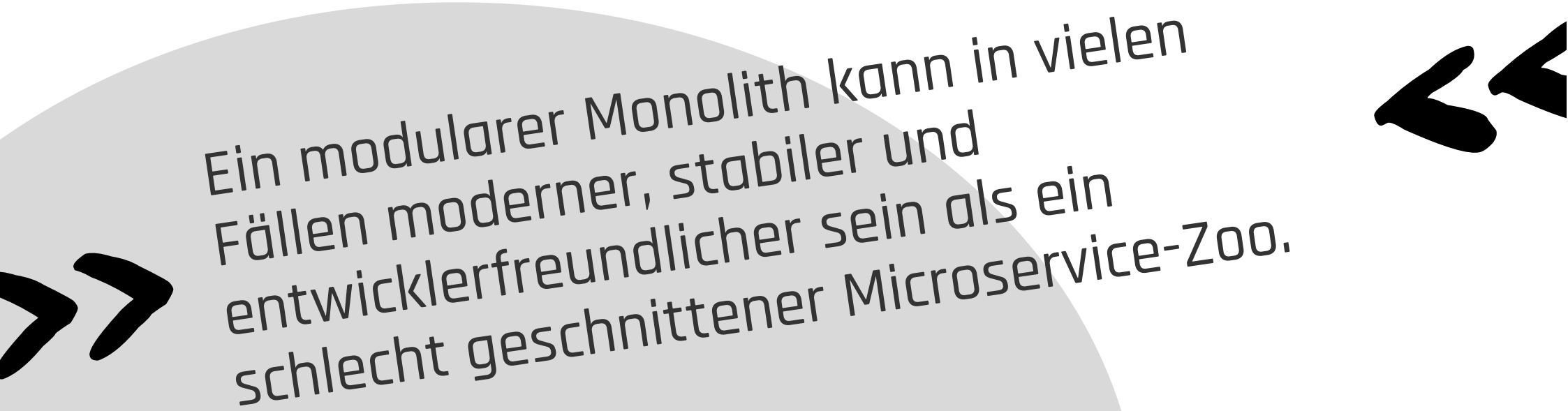
Bewusst oder unbewusst?

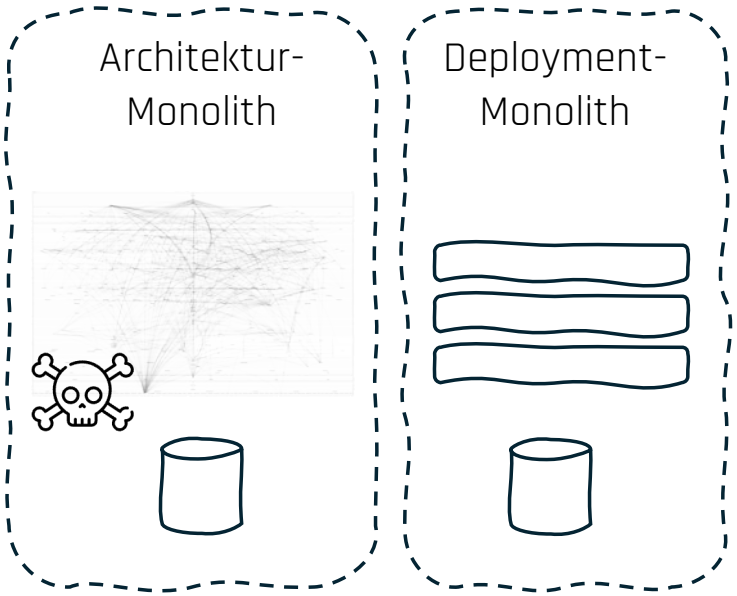


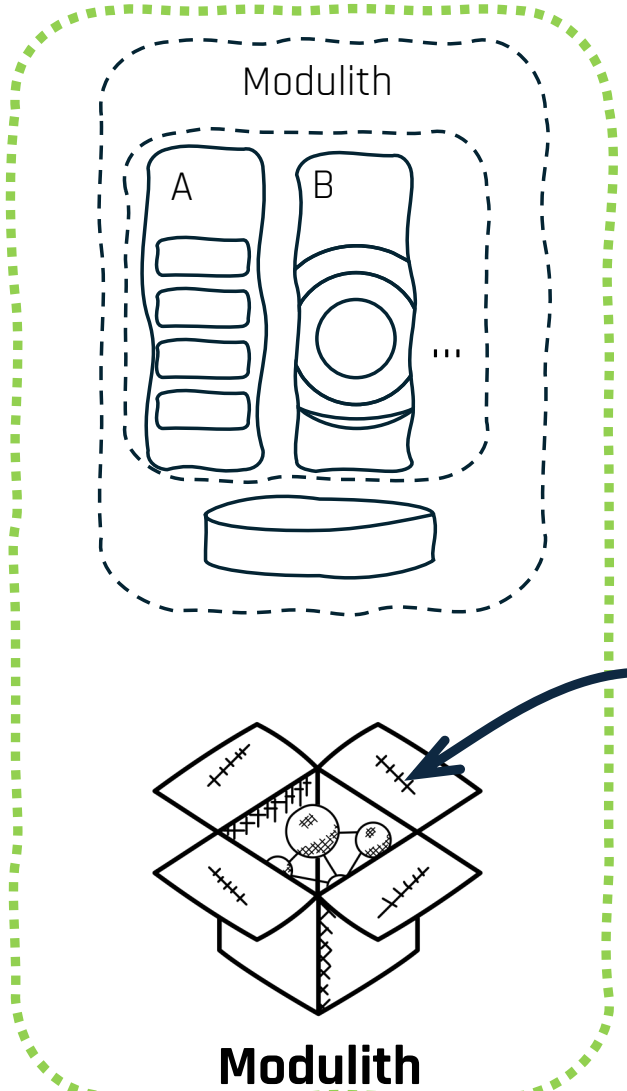
Anforderungen kennen!!!

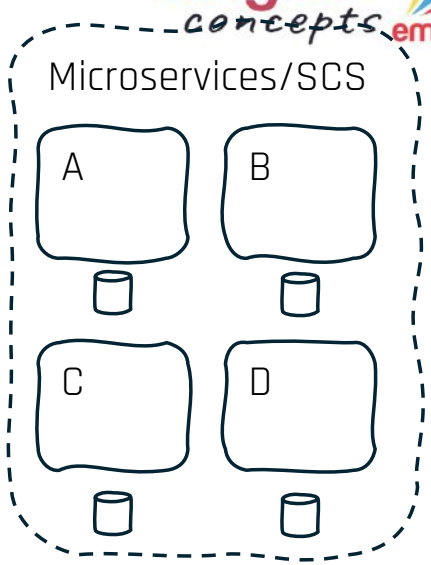
Tom Asel, Falk Sippach https://entwicklertag.de/karlsruhe/2021/sites/entwicklertag.de.karlsruhe.2021/files/folien/Wenn_Microservices_die_Anwort_sind.p

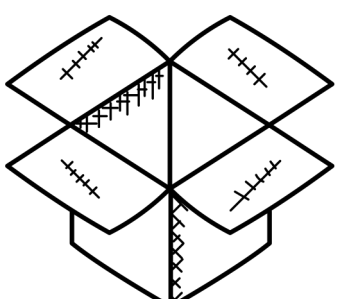
Modulith als Alternative





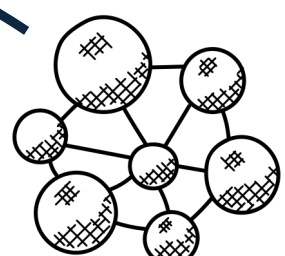






Monolith

Modulith



Microservices

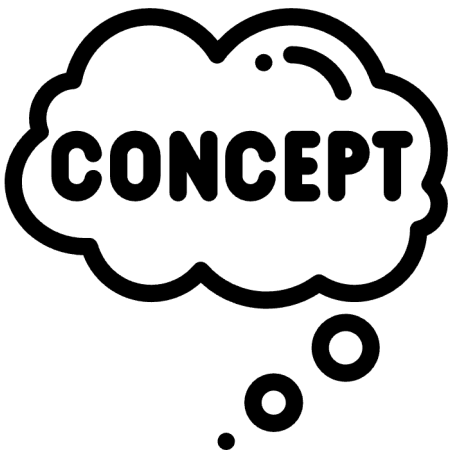
Spring Modulith

- unterstützt Entwickler bei der Implementierung logischer Module (über Package Konventionen und Meta-Informationen)

- erzeugt intern ein Modell (über Spring Komponenten-Modell, Meta-Informationen, jMolecules, ...)

https://spring.io/projects/spring-modulith

Validierung der Architekturregeln



Explizite Architektur-konzepte im Code

Transparenz durch Dokumentation

Mythos #7

Architekturdokumentation ist überbewertet

tangible concepts


Wir müssen über ein ernstes Thema reden:
Dokumentation

Dokumentation? Die liest doch keiner.“

Nur, wenn sie schlecht ist.

Ich mache das lieber im Kopf.

Blöd, dass dein Kopf nicht versioniert ist.

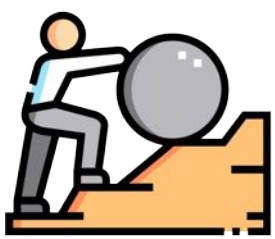
Außerdem reicht der Code als Dokumentation.

Nur für Details – nicht für Zusammenhänge.

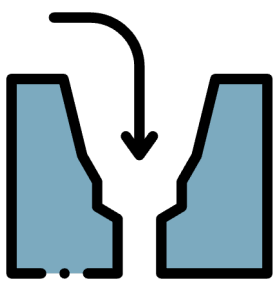
Was ist das Problem?

Implizites Wissen, Missverständnisse

Lange Einarbeitungs-zeiten von neuen Kollegen



Hohe Abstimmungs-aufwände



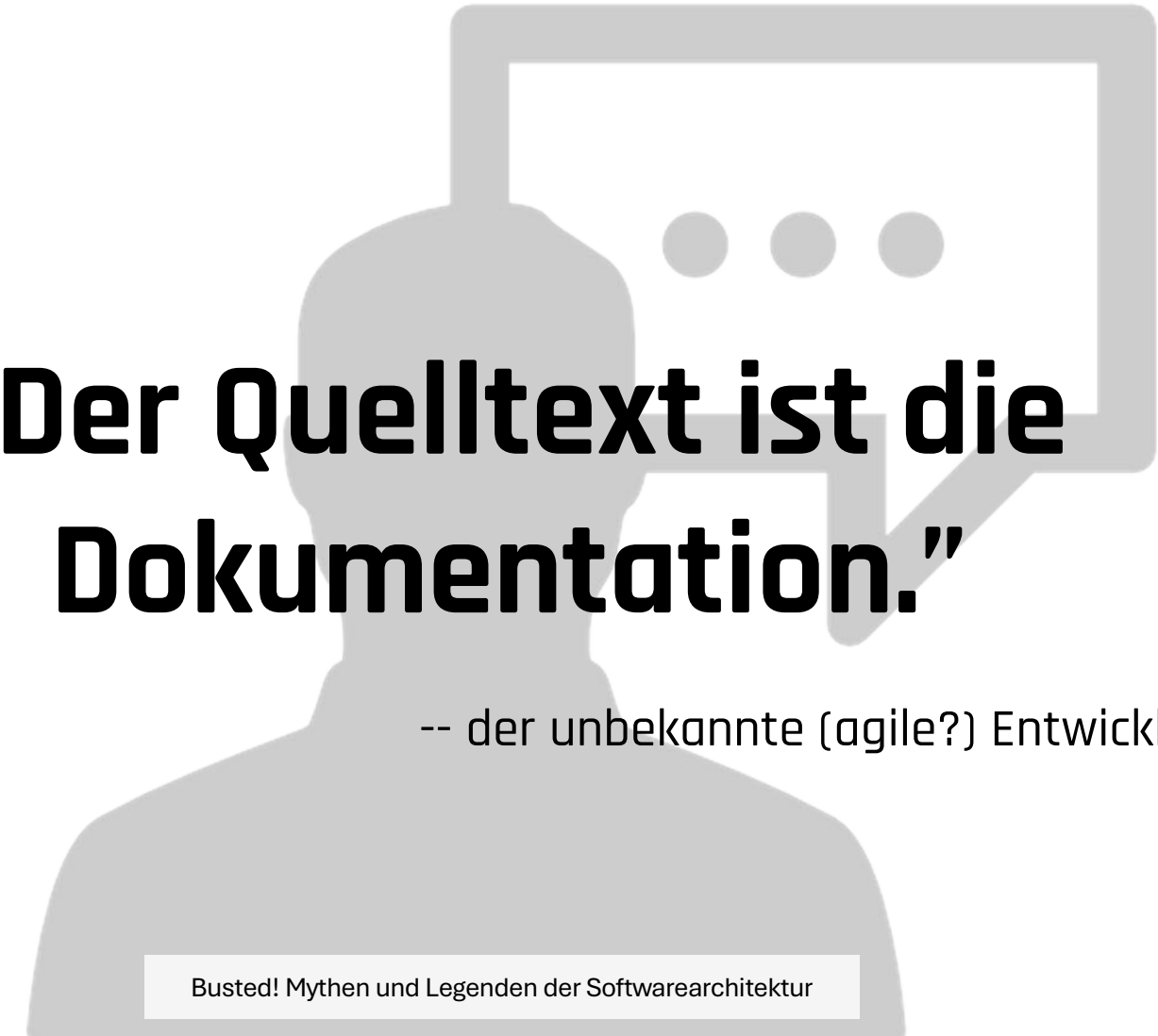
Soll-Ist Gap

Entscheidungen nicht nachvollziehbar

Das agile Manifest

Individuen und Interaktion	**vor**	Prozessen und Werkzeugen
Funktionierende Software	**vor**	Umfassender Dokumentation
Zusammenarbeit mit dem Kunden	**vor**	Vertragsverhandlungen
Reagieren auf Veränderung	**vor**	Befolgen eines Plans

Zitat



“Der Quelltext ist die Dokumentation.”

-- der unbekannte (agile?) Entwickler

Replik

“Der Quelltext erzählt nicht die ganze Geschichte.”

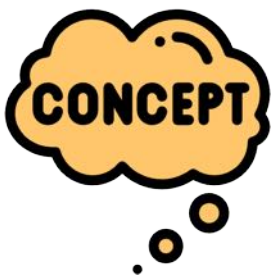
-- Simon Brown

Das agile Manifest umgedreht

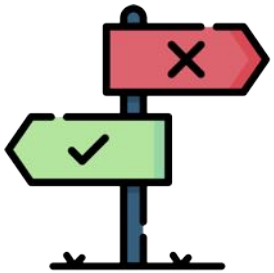
Prozesse und Werkzeug	**für**	**Individuen und Interaktion**
Nützliche Dokumentation	**für**	**Funktionierende Software**
Vertragsverhandlungen	**für**	**Zusammenarbeit mit dem Kunden**
Ausreichend Planung	**für**	**Reagieren auf Veränderung**

QuellCode != Dokumentation

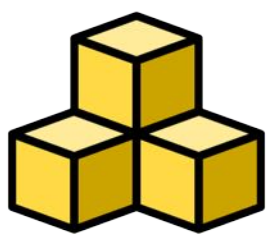
Sichtbar-machen von ...?



Architektur-konzepten



Architektur-entscheidungen



fachliche & technische Modulstrukturen

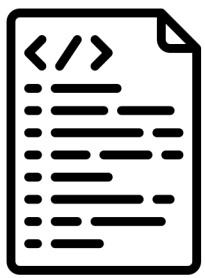
Rollen von Bausteinen (DTO, ...) und Verortung

Symptome

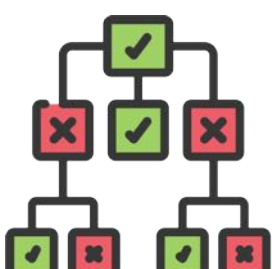
- Hoher Abstimmungsaufwand im Alltag
- Viele Rückfragen zu grundlegenden Strukturen
- Neuzugänge brauchen lange zum Einstieg
- Widersprüchliche Annahmen über die Architektur
- Schnittstellen werden unterschiedlich interpretiert
- Implementierung weicht zunehmend vom Zielbild ab
- Entscheidungen werden mehrfach getroffen
- „Das wusste ich nicht“-Momente häufen sich
- Teams erzeugen versehentlich neue Abhängigkeiten
- Qualitätsziele sind unklar oder variieren stark

Wie begegnen wir dem Mythos?

Zielgerichtete, Dokumentation



Leichtgewichtig, Docs-as-Code

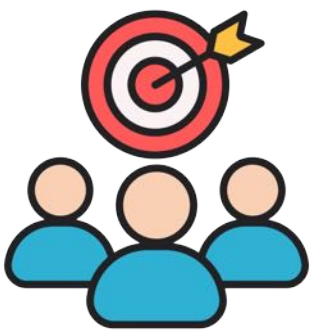


Transparente Architekturentscheidungen

Continuous Documentation

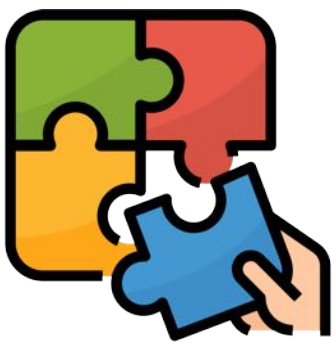
Lebendige Dokumentation

Pragmatisch dokumentieren



An Zielgruppen orientiert

Nicht zu viel, stets aktuell halten



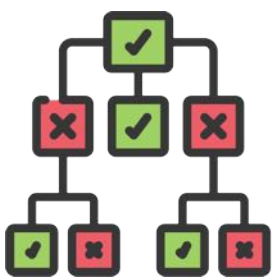
Modularisieren, Inhalte referenzieren/generieren

Feedback einholen, lebendige Dokumentation

Wie begegnen wir dem Mythos?

Zielgerichtete, Dokumentation

Leichtgewichtig, Docs-as-Code



Transparente Architekturentscheidungen

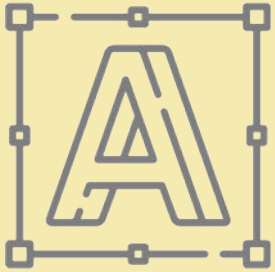
Continuous Documentation

Lebendige Dokumentation

Maturity Model

Docs-as-Code






Mit Implementierung in Einklang bringen

Architekturkonzepte im Quellcode explizit (**sichtbar**) machen, daraus **Architekturregeln** ableiten, festhalten und aus der Dokumentation regelmäßig als Tests automatisiert verifizieren.

Dokumentation kontinuierlich verbessern

Die unterschiedlichen Zielgruppen sowie ihre Anforderungen adressieren und regelmäßig **Feedback** einarbeiten. Erstellung und Wartung der Dokumentation in **Team-Prozesse** integrieren.

Werkzeuge einsetzen und integrieren

Unnötige Kontextwechsel vermeiden und Dokumentation in den typischen **Entwicklungswerkzeugen** erstellen. **Automatisierung** nutzen und Resultate regelmäßig in CI/CD-Pipeline erzeugen.

Mit Grafiken effizient umgehen

Den übertriebenen Einsatz von UML-Werkzeugen sowie kommerziellen Grafikprogrammen vermeiden und die **Einstiegshürde** durch Einsatz schlanker, freier Tools **verringern**.

Grundlage für Docs-as-Code legen

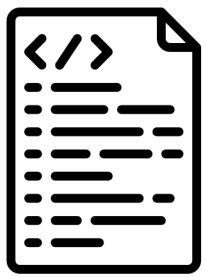
Mit **Markup-Sprachen** starten, Dokumentation modularisieren und auf Zielgruppen zugeschnittene Ergebnisse erstellen. Die Inhalte in der **Versionsverwaltung** sammeln.

Konventionell dokumentieren

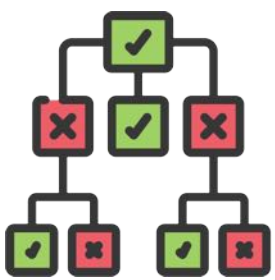
Standardstrukturen wie **arc42, C4 und ADRs** kommen zum Einsatz, aber die Verwendung kon- ventioneller Werkzeuge demotiviert und **bremst beim Dokumentieren** sogar aus.

Wie begegnen wir dem Mythos?

Zielgerichtete, Dokumentation



Leichtgewichtig, Docs-as-Code



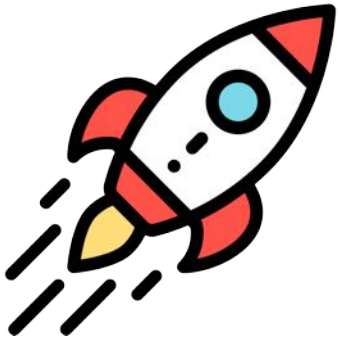
Transparente Architekturentscheidungen

Continuous Documentation

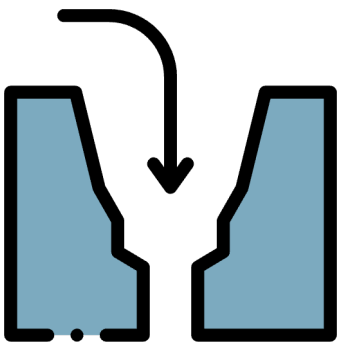
Lebendige Dokumentation

Dokumentation –
Das notwendige Übel

~~Dokumentation –~~
~~Classic Style~~

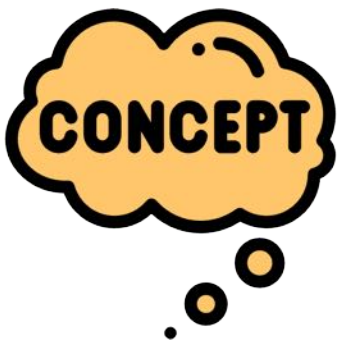


Dokumentation –
State of the Art



Mind the Gap

Architektur-
Validierung



Explizite Architektur-
konzepte im Code

Vielen Dank für Euer Aufmerksamkeit!

Fragen?

Teilt Eure Mythen mit uns!

Tom Asel
tom.asel@tangible-concepts.de
https://www.linkedin.com/in/tom-asel

Falk Sippach
falk.sippach@embarc.de
https://www.linkedin.com/in/falk-sippach

tangible concepts embarc