SOCREATORY
The Software Creators' Academy

Flexible Architekturen

... nur wie?

Eberhard Wolff (SWAGLab), Falk Sippach (embarc)

Flexible Architekturen – nur wie?

Der Traum jeder Architekt:in: ein System, das sich mühelos an neue Anforderungen anpassen lässt. In der Praxis ist das jedoch natürlich oft schwer umzusetzen. In diesem Vortrag erfahrt ihr, welche Schritte nötig sind, um eine wirklich flexible Architektur zu gestalten.

Dabei betrachten wir nicht nur die fachliche Aufteilung mit Domain-driven Design, sondern auch die konkreten Vorteile moderner Ansätze wie Microservices oder Self-contained Systems. Praktische Beispiele zeigen, wie sich Flexibilität in der Praxis tatsächlich steigern lässt.

Wer sind wir?

Eberhard Wolff (SWAGLab)

eberhard.wolff@swaglab.rocks

ewolff.com

software-architektur.tv

SWAGLab

Falk Sippach (embarc)

falk.sippach@embarc.de

LinkedIn: /in/falk-sippach

embarc

iSAQB

Agenda

1. Fachliche vs. technische Flexibilität 
2. Modularisierung 
3. Fachliche Flexibilität 
4. Technische Flexibilität 
5. Fazit & Ausblick 

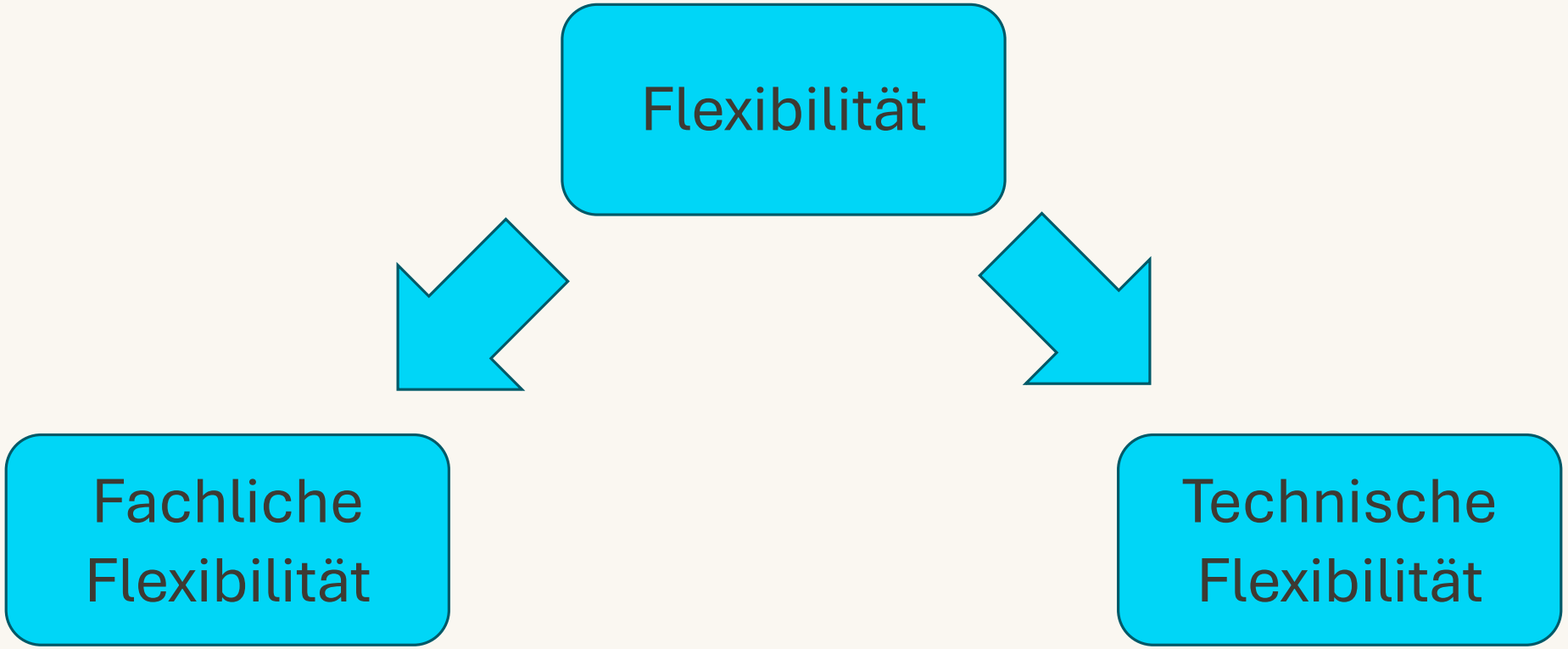
Intro: Fachliche vs. technische Flexibilität

Flexibilität

- Software-Entwicklung erfolgt in Iterationen.
- Änderungen und Erweiterungen sind die Norm.

- Requirements ändern sich.
- Ungeplante Änderungen sind typisch.

- Flexibilität = einfache Änderbarkeit
- Die wichtigste Eigenschaft von Software-Architektur?


Flexibilität
Fachliche Flexibilität
Technische Flexibilität

Fachliche Flexibilität

Fachliche Flexibilität

- Einfache Änderbarkeit bei neuen fachlichen Anforderungen.
- Software wird zum Lösen fachlicher Herausforderungen implementiert.
- Wert der Software: Wie gut ist die Fachlichkeit?
- Daher: Fachliche Flexibilität sehr wichtig.

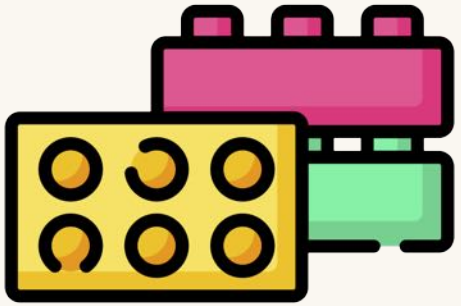
Technische Flexibilität

- Einfach Änderbarkeit der Technologien

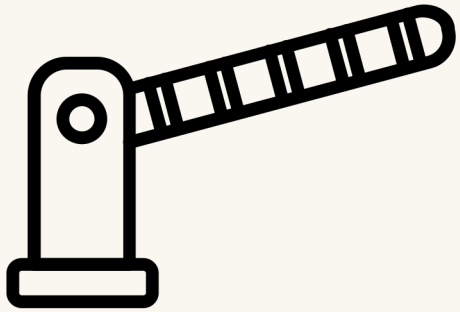
- Neue Technologien helfen bei nicht-funktionalen Anforderungen / Qualitäten.
- Technologie ohne Sicherheitsupdate ersetzen
- Performance durch andere Datenbank / Persistenz-Technologie
- Mangelnde Wartbarkeit wegen veralteter Technologie
- Generell: Technische Flexibilität weniger wertvoller für Kunden als fachliche Flexibilität (?)

Technische Flexibilität

Modularisierung

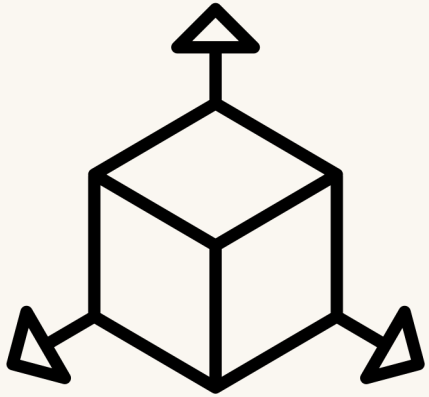


Modularisierung — warum und wie?



Grenzen setzen

Änderungen
lokal halten



Viele
Dimensionen

Beispiel OOP

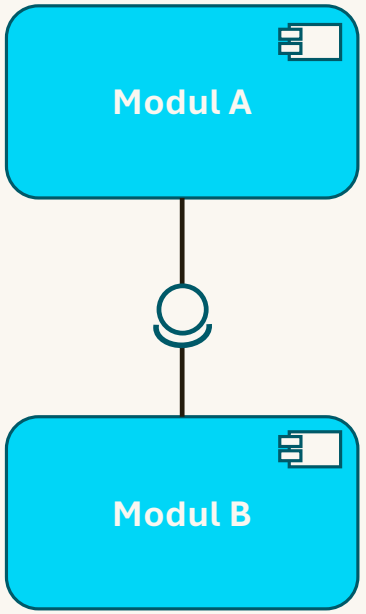
> *OOP encourages us to model complex entities and processes using objects, which combine state and behavior. OOP is at its best when it is defining and defending boundaries.*

https://www.infoq.com/articles/data-oriented-programming-java/

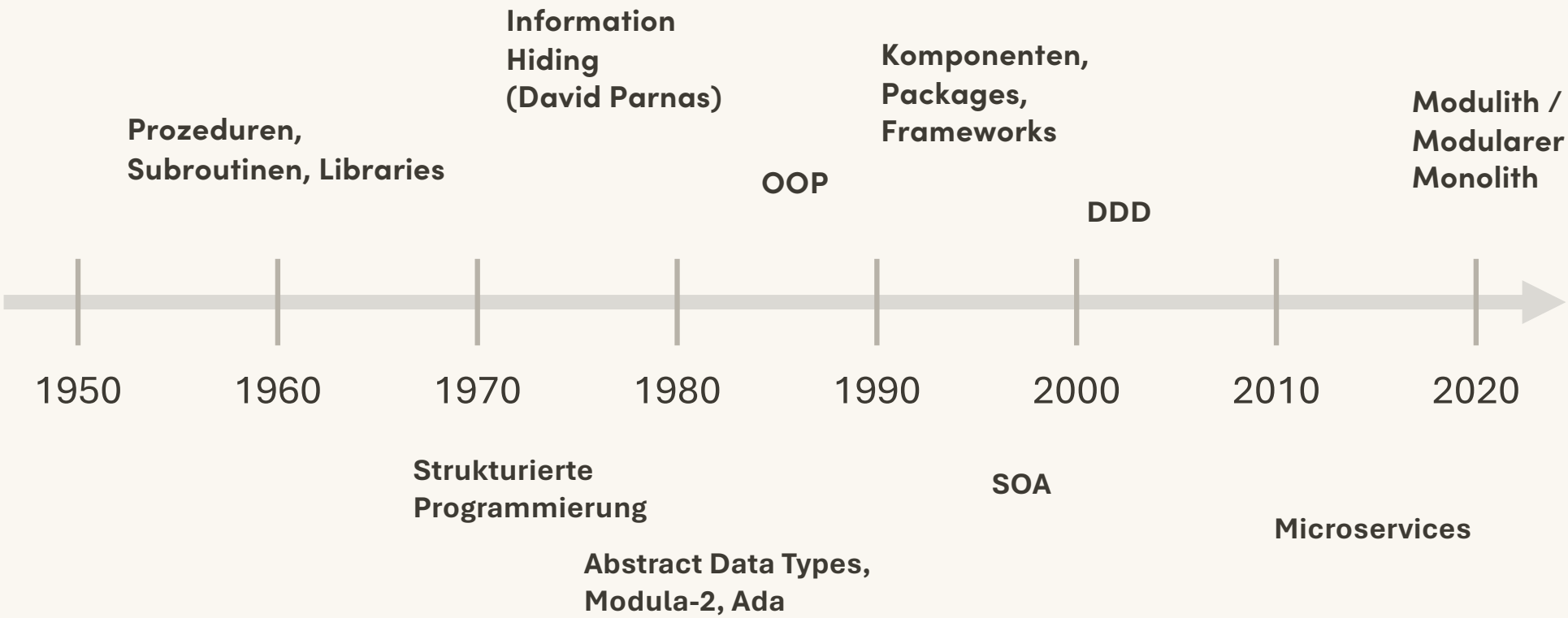
Brian Goetz

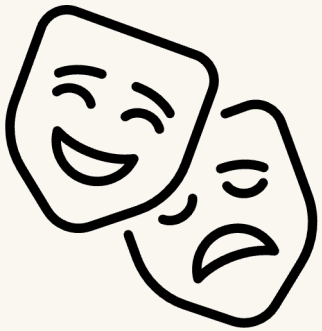
Was ist ein Modul?

- kapselt Funktionalität
- bietet Schnittstellen an und garantiert diese im Sinne eines Vertrages
- konsumiert Schnittstellen anderer Bausteine
- kann durch anderen Baustein ersetzt werden, welcher gleichartige Schnittstellen zur Verfügung stellt
- stellt einen Teil eines Gesamtsystems dar.

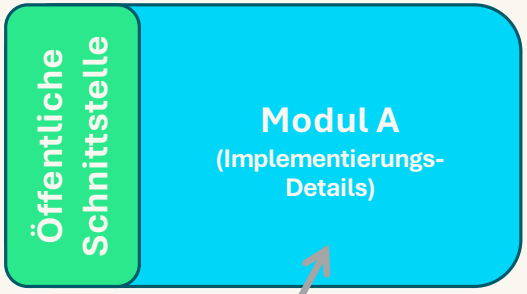


Historie

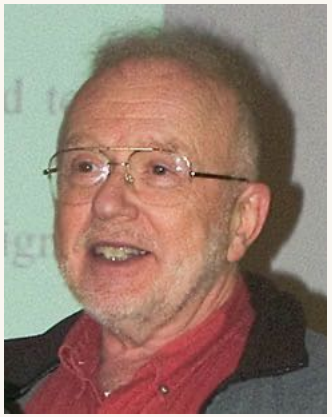

Prozeduren,
Subroutinen, Libraries
Information
Hiding
(David Parnas)
OOP
Komponenten,
Packages,
Frameworks
DDD
Modulith /
Modularer
Monolith
1950
1960
1970
1980
1990
2000
2010
2020
Strukturierte
Programmierung
Abstract Data Types,
Modula-2, Ada
SOA
Microservices



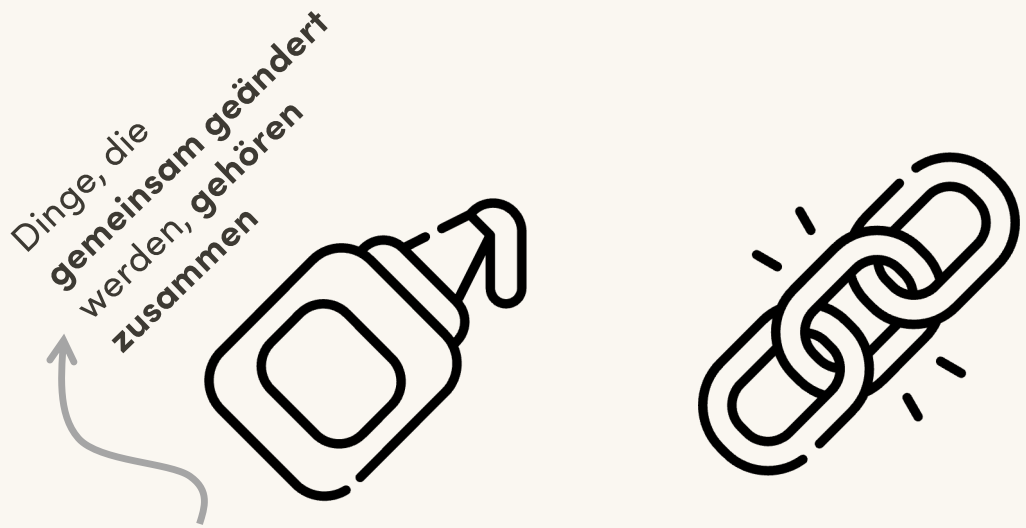
Gute Module **verstecken** die **Entscheidungen**, die **sich** (häufig) **ändern** (können).



Information Hiding



David Parnas



Kohäsion & Kopplung

Kopplung

- Lose Kopplung: Dinge, die **unabhängig geändert** werden sollen, sollten **nicht** unnötig **voneinander abhängen**
- Kopplung ist nicht grundsätzlich schlecht, aber sie muss bewusst eingesetzt werden
- Flexibilität leidet besonders unter **unbewusster, transistiver und zyklischer Kopplung**

Arten von Kopplung

- fachlich
- Daten
- zeitlich
- technologisch
- Deployment
- Team/Organisation
- ...

Modularisierung ist mehrdimensional

- **Code**: Packages, Namespaces, Komponenten
- **Build**: Libraries, Module, Abhängigkeiten
- **Daten**: Tabellen, Schemas, Datenhoheit
- **Laufzeit**: Prozesse, Services, Kommunikation
- **Deployment**: ein Artefakt vs. mehrere Artefakte
- **Organisation**: Teamverantwortung, Ownership

Diese Grenzen **können zusammenfallen**, **müssen** es **aber nicht**.

Fachliche Flexibilität

Fachliche Flexibilität: Wie?

- Notwendige fachliche Änderungen sind nicht vorhersehbar.
- Änderungsschwerpunkte werden nur retrospektiv offensichtlich.

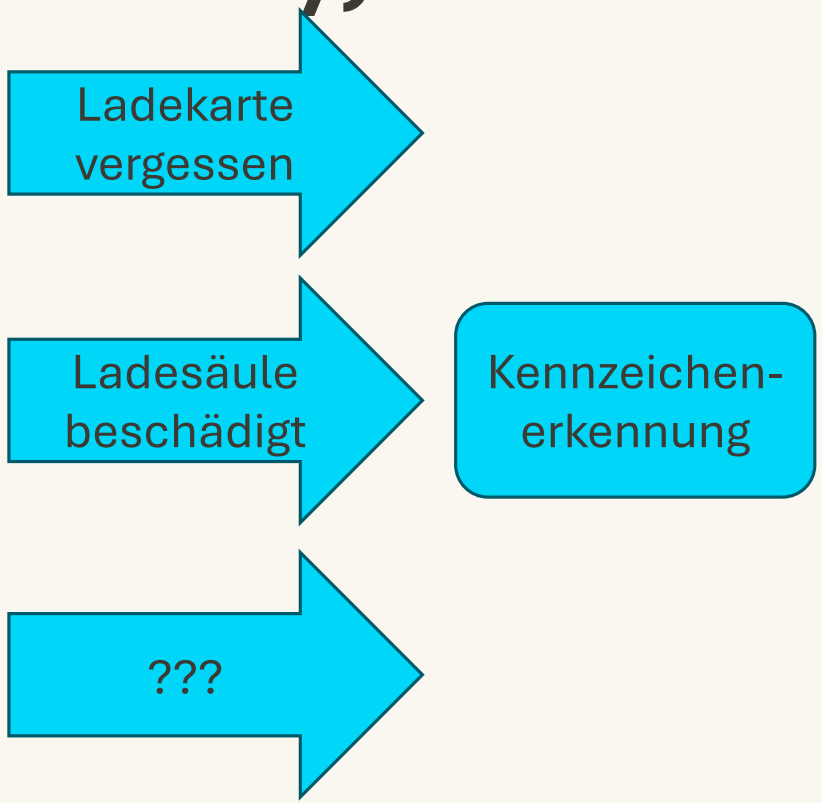
- Helfen historische Änderungsschwerpunkte bei der Vorhersage zukünftiger Schwerpunkte?
- Kann man zukünftige Änderungsschwerpunkte vorhersagen?
- Sollte man historische Änderungsschwerpunkte ignorieren?

Residuality Theory (Barry O'Reilly)

- Änderungsszenario = Stressoren gegen die Architektur
- Behauptung: Eine Architekturentscheidung kann die Antwort auf eine Menge an Stressoren sein.

Residuality Theory (Barry O'Reilly)

- Beispiel: Ladesäulen für Elektroautos
- Stressor: Ladekarte vergessen
- Lösung: Kennzeichenerkennung
- Stressor: Ladesäule beschädigt (Unfall / Vandalismus)
- Lösung: Kennzeichenerkennung
- Kennzeichenerkennung: Antwort auf auch unbekannte Stressoren?



Fachliche Flexibilität

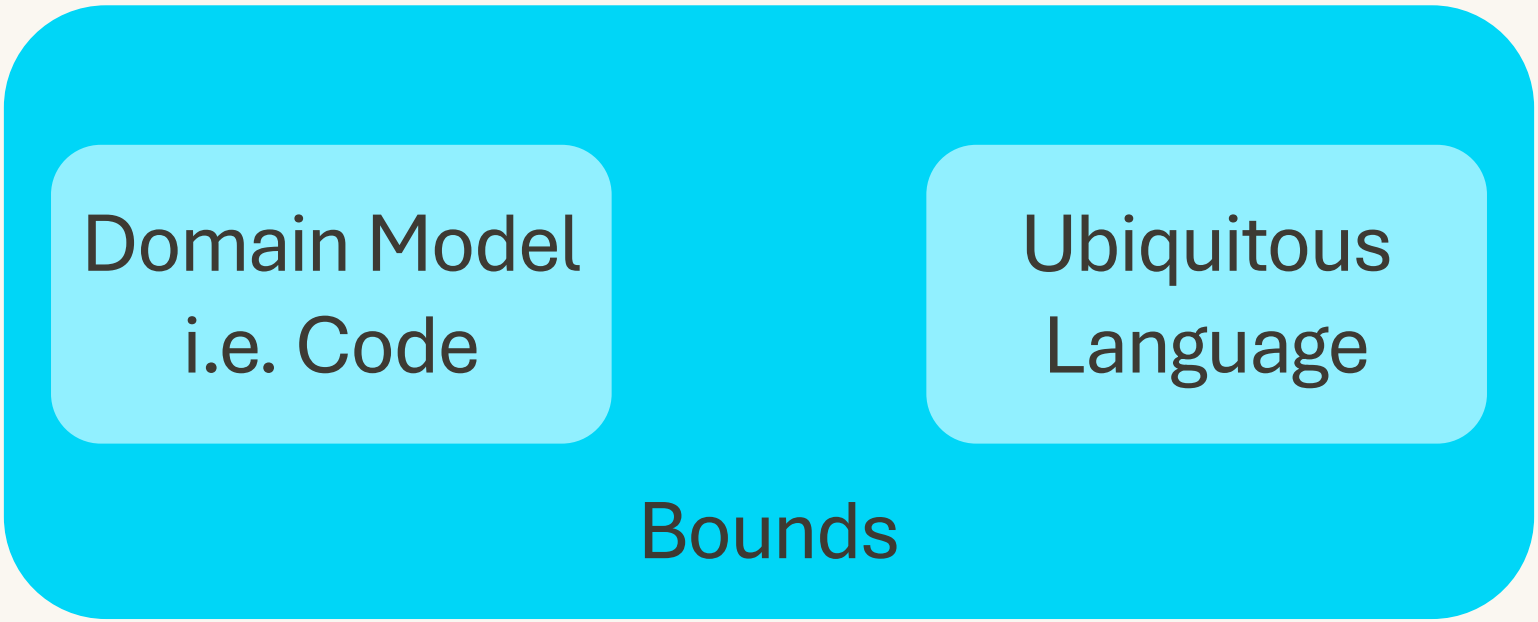
- Eine möglichst gute fachliche Modellierung macht fachliche Änderungen einfach.
- Die Domäne sollte das Design treiben.
- Domain-driven Design


Let the Domain
Drive the Design

Fachliche Aufteilung

- Wie kommen wir zu einer guten fachlichen Aufteilung?
- Zentrale Frage in der Architektur
- Verschiedene Heuristiken

Bounded Context



Bounded Context Beispiel

Rechnungslegung

Rechnung schreiben

Mehrwertsteuer

Lieferung

Tracking

Verpacken und versenden

Bestellprozess

Einkaufswagen

Bestellung akzeptieren

Bounded Context Beispiel

Rechnungslegung

Kunde Rechnungsadresse

Produkt Preis

Lieferung

Kunde Lieferadresse

Produkt z.B. Größe / Gewicht

Bestellprozess

Kunde Produktpräferenzen

Produkte Marketing-Infos

Unterschiedliche Modelle / Sprache

Rechnungslegung

Kunde Socreatory GmbH

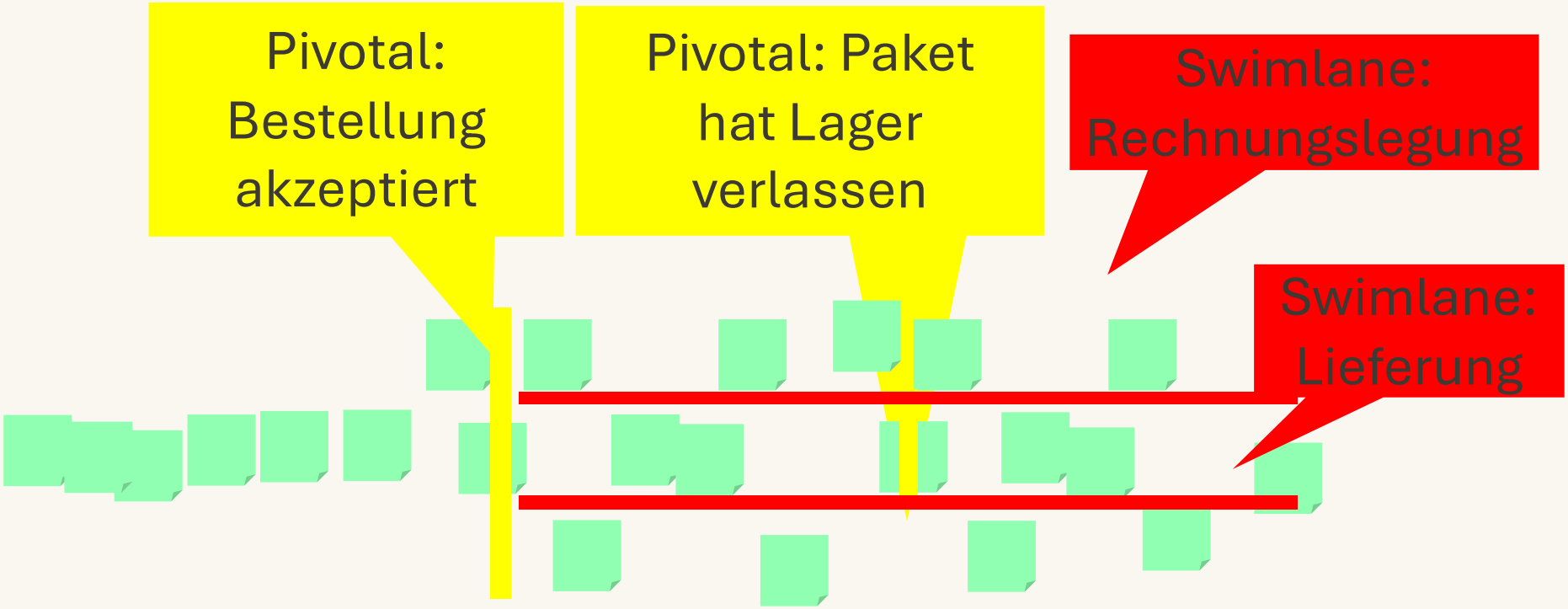
Lieferung

Kunde Eberhard Wolff

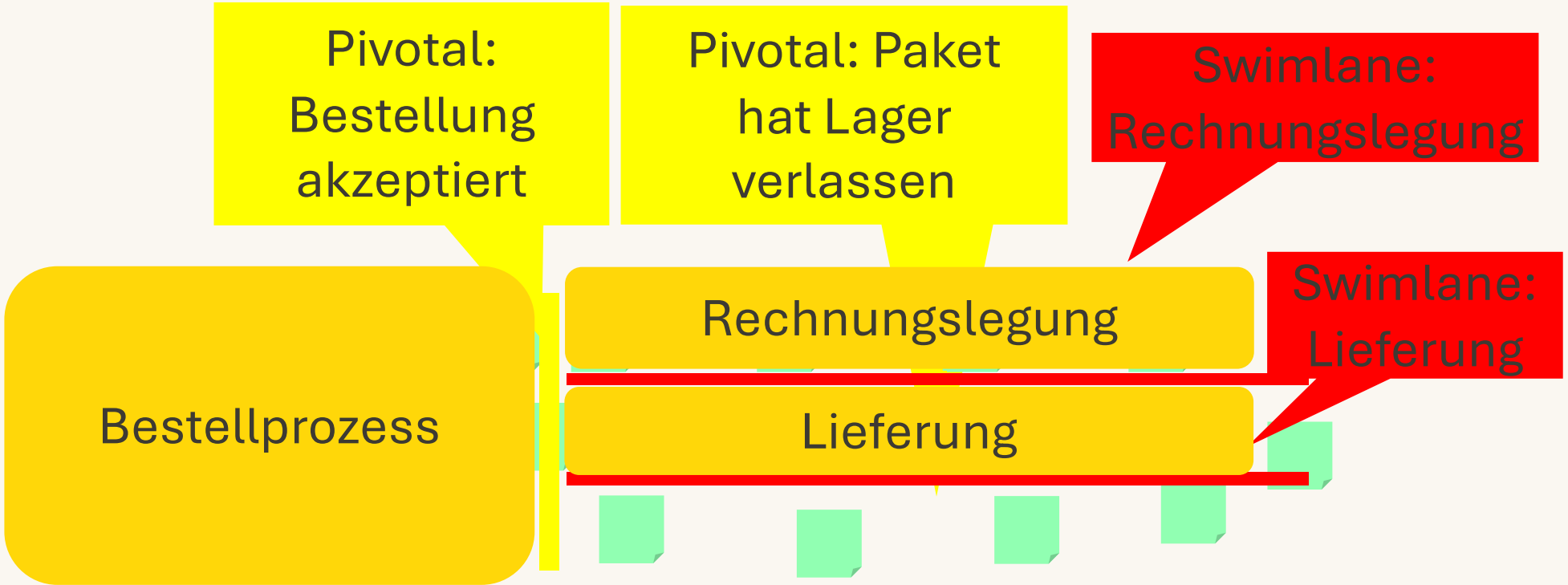
Andere Heuristik: Event Storming

- Pivotal Events: Etwas ändert sich grundlegend
- Swim Lanes: Parallele Aktivitäten
- Kandidat für Module: Bereich zwischen Pivotal Events und Swim Lanes

Andere Heuristik: Event Storming



Andere Heuristik: Event Storming



Andere Heuristik: Independent Service Heuristics

- 10 Heuristiken
- Beispiel: Lieferung
- 2 – Marke (Brand)

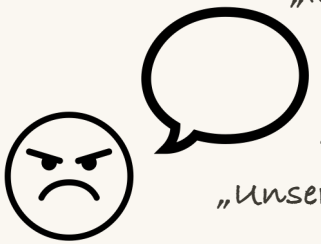
Ist es denkbar, dieses Ding als Public-Cloud-Service unter einer eigenen Marke angeboten wird?

- 10 – Produktentscheidungen

Könnte das Team, das an diesem Ding arbeitet, die Verantwortung für die eigene Produkt-Roadmap und die Produktausrichtung selbst übernehmen?

Technische Flexibilität

Wann ist ein System unflexibel?



„Ich muss ständig **mit anderen Teams reden** um meine Features umzusetzen"

„Ich kann keine Testumgebung für das System erstellen, weil es **zu komplex** ist"

„Diese DSGVO-Regeln einzubauen nervt …"

„Unser **Technologie-Stack ist veraltet**"

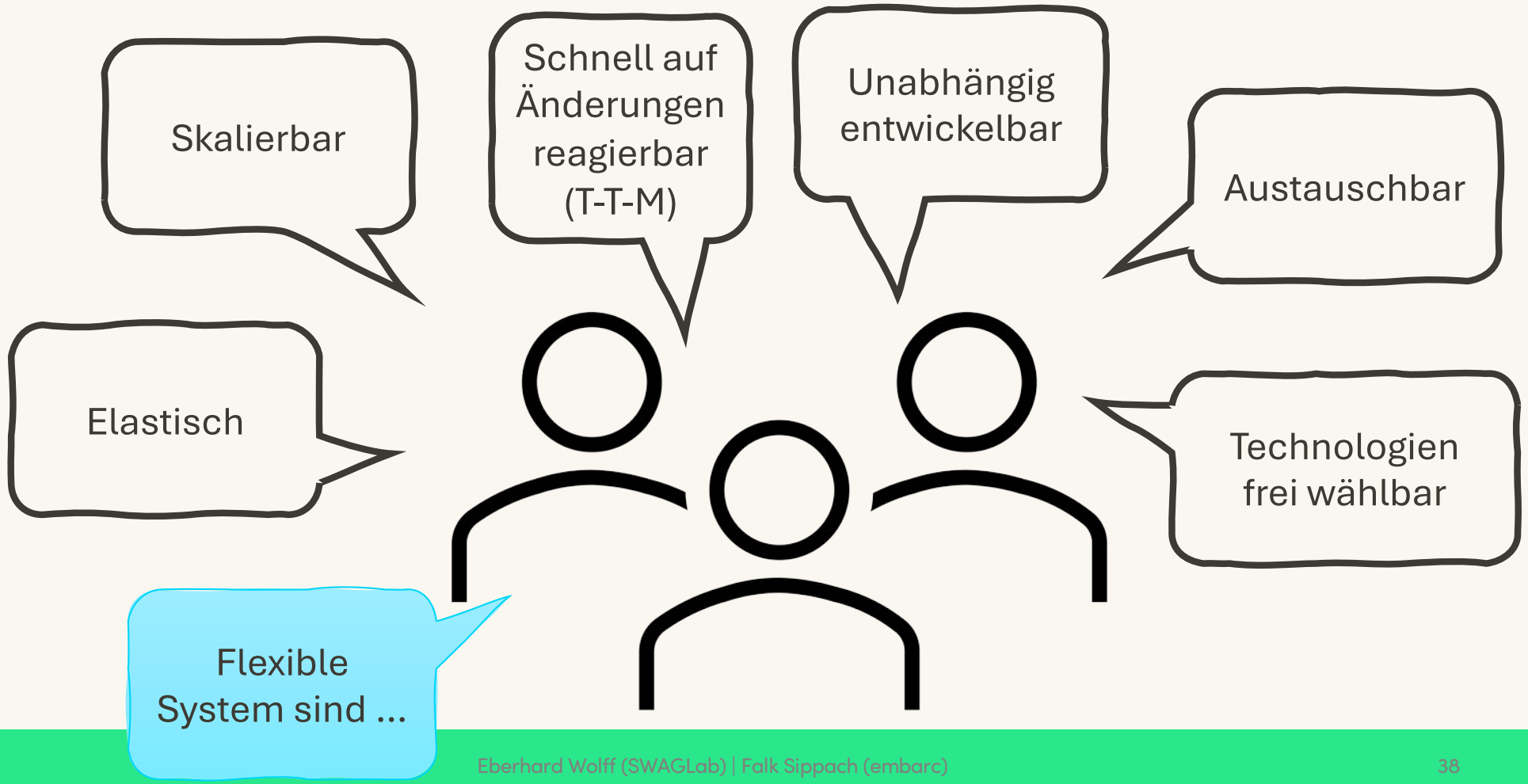
„Bei Lastspitzen ist das **System echt langsam**"

„Dieser **Fehler** taucht schon **seit Monaten** auf"

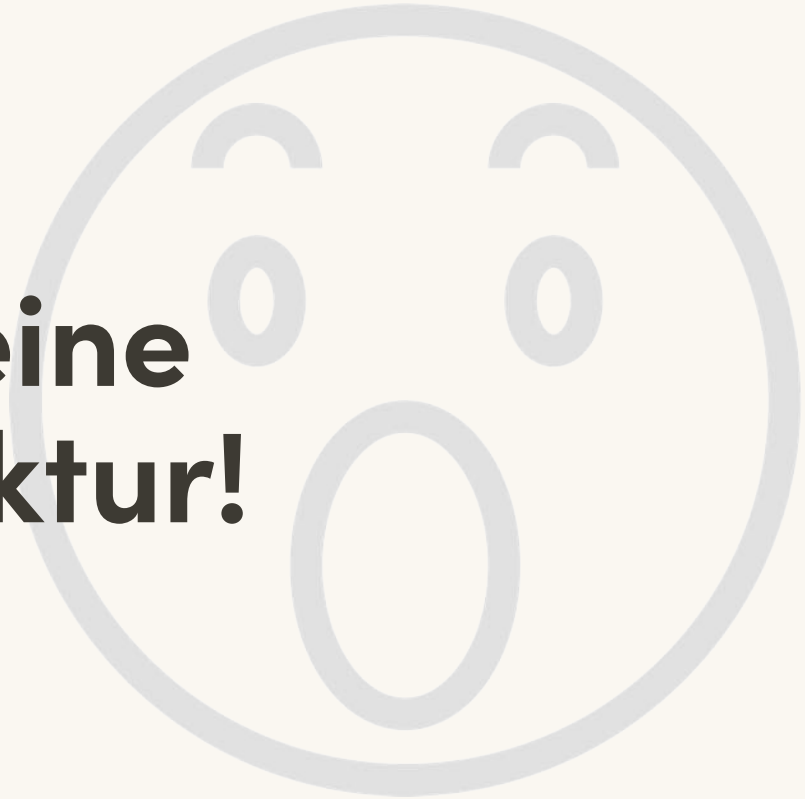
„Feature X wäre echt toll, der Konkurrent bietet das schon an"

„Entwickler:innen **brauchen so lang**"

„Mit neuen Feature-Ideen muss ich zu ganz vielen Leuten gehen und **niemand** kann mir **eine Einschätzung geben**"

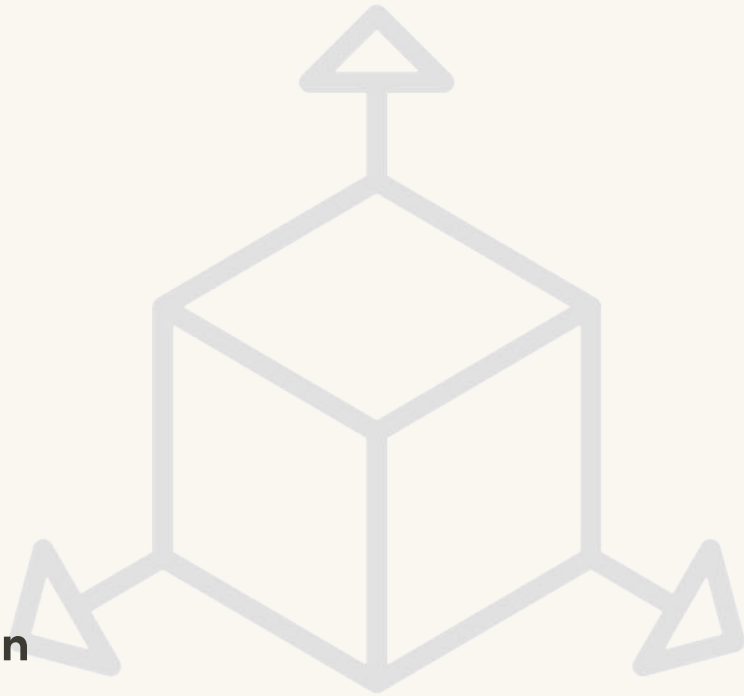

Skalierbar
Schnell auf Änderungen reagierbar (T-T-M)
Unabhängig entwickelbar
Austauschbar
Elastisch
Technologien frei wählbar
Flexible System sind ...

Es gibt nicht die eine "flexible" Architektur!



Viele Ansatzpunkte ...

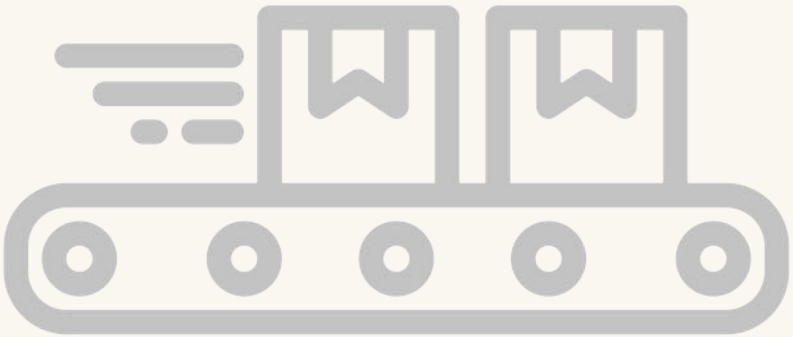
- technische **Grenzen** setzen
- **Deployment** & Release ohne Hindernisse
- effizienter **Betrieb** und **Überwachung**
- Entkopplung der **Daten** und **Persistenz**
- einfache **Integration** und flexible **Schnittstellen**
- ...



Deployment Flexibilität

- Rolling Updates / Blue-Green / Canary
- Versionierung und Kompatibilität
  - Rückrollbarkeit
  - Feature Toggles
- Automatisierung in CI/CD Pipeline
- DevOps Prinzipien

Änderungen **unabhängig** und **risikoarm** ausliefern!



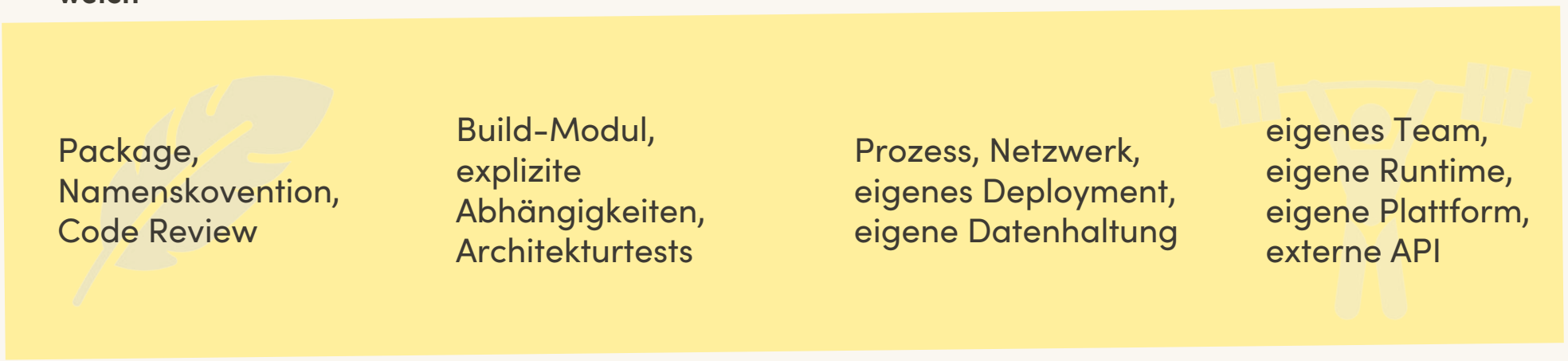
Betriebsflexibilität

- Monitoring, Logging, Tracing, Alerting
- Metriken auf Modul-Ebene
- Resilienzmechanismen
- Chaos Engineering
- Fitness Functions
- Disaster Recovery

Flexibilität endet nicht beim Deployment, sie muss **im Betrieb abgesichert** werden!

Unterschiedliche harte Grenzen

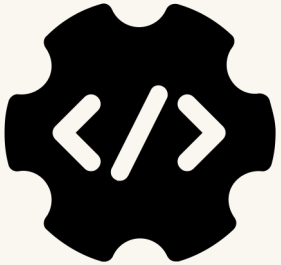
weich **sehr hart**



Je härter die Grenze, desto **stärker** die **Entkopplung**,
aber auch desto **höher** die **Integrationskosten**.

Grenzen technisch setzen: Feedback

Frühes Feedback → **Spätes** Feedback



Compiler-/IDE-Unterstützung

- Packages, Sichtbarkeit
- Modulsysteme (JPMS, OSGi, ...)

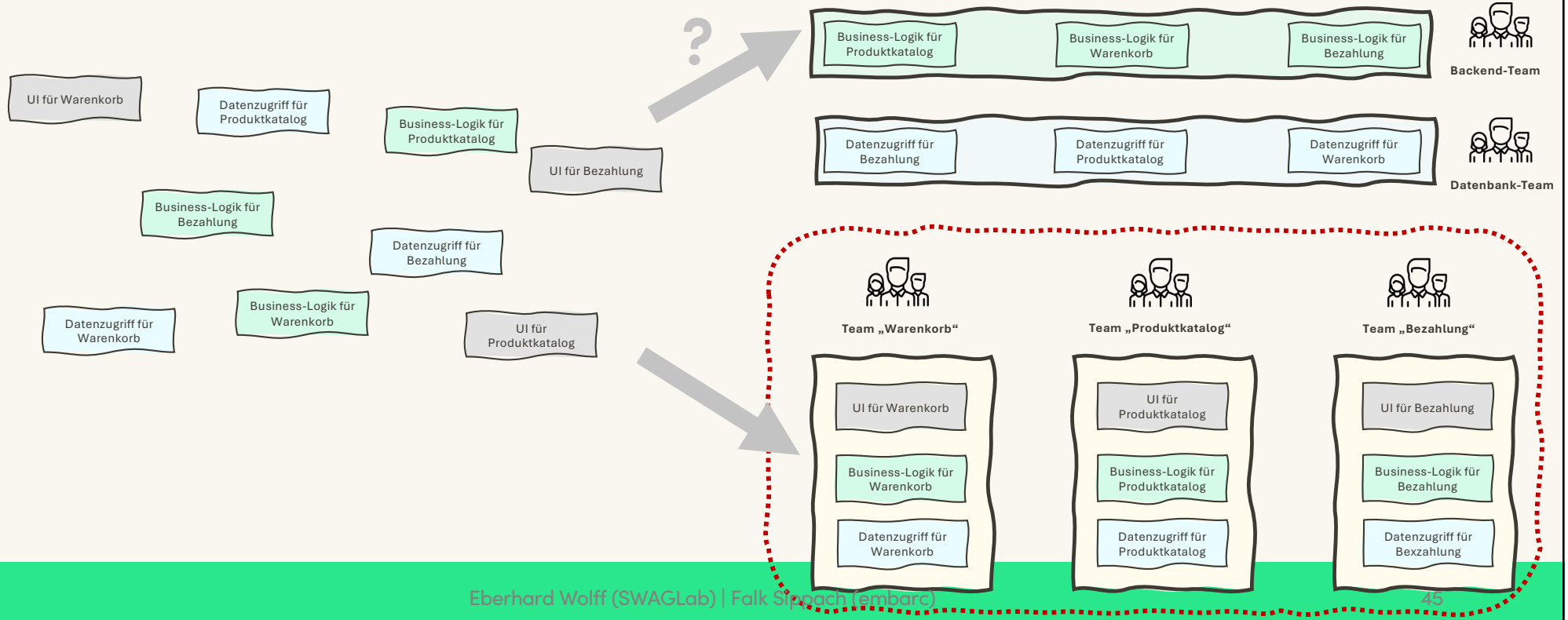
Build-management

Validierung zur Laufzeit

- ArchUnit
- jQAssistant
- ...

Ein Software-System

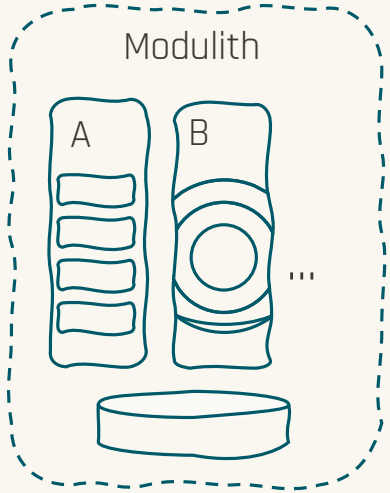
... bestehend aus Warenkorb, Produktkatalog und Bezahlung

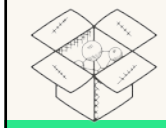


Modulith

Modulith

- fachliche Module innerhalb einer Anwendung
- **klare Modulgrenzen** und explizite Schnittstellen
- ein Deployment-Artefakt
- gemeinsame Laufzeit
- oft gemeinsame Datenbank,
  aber idealerweise klare Datenhoheit
- **keine Remote-Kommunikation** zwischen Modulen
  und damit **kein Overhead verteilter Systeme**





Ein Modulith bleibt nur flexibel, wenn Modulgrenzen sichtbar und überprüfbar sind.

Validierung der Architekturregeln

Explizite Architektur-konzepte im Code

Transparenz durch Dokumentation

Spring Modulith

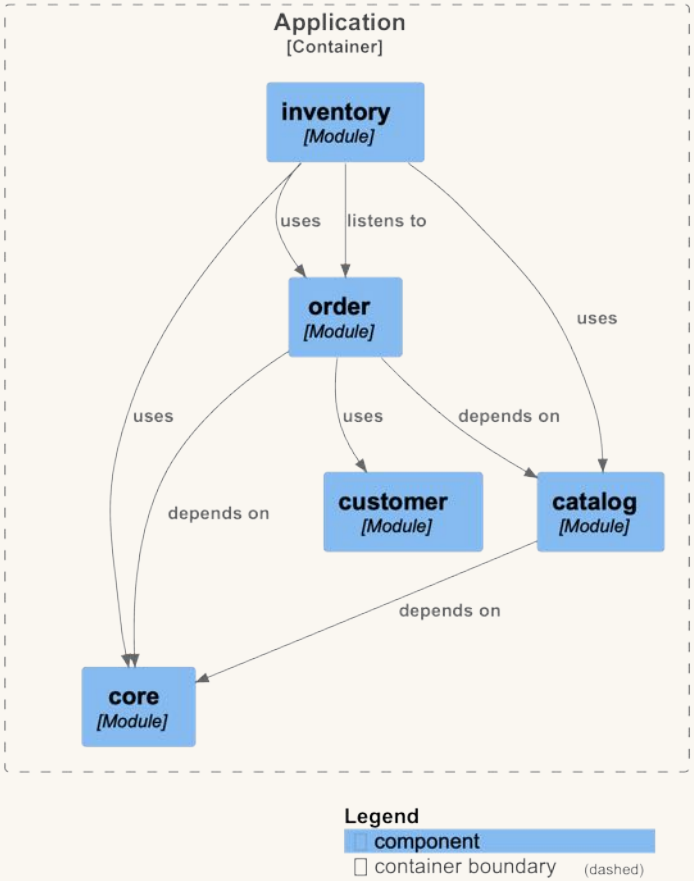
> ... allows developers to build well-structured Spring Boot applications and guides developers in finding and working with application modules driven by the domain. It supports the **verification of such modular arrangements, integration testing individual modules, observing the application's behavior on the module level and creating documentation snippets** based on the arrangement created.
>
> (https://spring.io/projects/spring-modulith)

Spring Modulith – Technische Struktur

- unterstützt Entwickler bei der Implementierung logischer Module (über Package Konventionen und Meta-Informationen)
- erzeugt intern ein Modell (über Spring Komponenten-Modell, Meta-Informationen, jMolecules, ...)

Spring Modulith – Validierung + Dokumentation

- kann **Verifikationen/Fitness Functions** laufen lassen (ArchUnit)
- erzeugt **Dokumentations-Snippets** und -diagramme mit AsciiDoc und PlantUML



Spring Modulith - Entkopplung

- **@ApplicationModuleTest**
  – Hochfahren von einem Modul und den Abhängigkeiten

- **@ApplicationModuleListener**
  – asynchrone, persistierte Events – Vorbereitung für EDA in verteilten Systemen

Microservices

Microservices

- Module als Container implementiert

- Größe: variiert
- Kleine Wegwerf-Microservices
- Aufgabe für ein Team

- Kommunikation: Asynchron oder synchron?

Microservices: Vorteile

- Unabhängiges Deployment
- Unabhängige Skalierbarkeit
- Bessere Isolation bzgl Sicherheit
- Unabhängige Technologie-Wahl

- Vor allem: Technische Flexibilität
- Alternativen mit denselben Vorteilen?

Microservices: Nachteile

- Höhere technische Komplexität der Infrastruktur
- Begrenzte Vorteil z.B. von unabhängigem Deployment, wenn die fachliche Aufteilung nicht gut ist.

Self-contained Systems

Self-contained Systems

- Microservices
+ Web-Oberfläche
+ Integration in der Oberfläche (Links, Transklusion)

- Größe: Ein Bounded Context
- Organisation: Ein SCS gehört ein Team

- Idee: Funktionalität inkl. Oberfläche in einem Team und einem SCS implementieren
- Sehr hohe fachliche und technische Autonomie (self-contained)

Fazit & Ausblick


Fehlende Modulgrenzen machen aus kleinen Änderungen schnell einen Flächenbrand.
Eberhard Wolff (SWAGLab) | Falk Sippach (embarc)


Brandschutzstreifen (Modulgrenzen)
verhindern die unkontrollierte Ausbreitung.

SOCREATORY

Meine Trainings bei socreatory

- iSAQB® CPSA-A® FLEX
  - 29.09. - 01.10.2026 in Hannover
- Architektur-Kickstart
- Team Topologies Deep-Dive
- DDD saniert Legacy

Mein Trainer-Profil:

https://www.socreatory.com/trainers/eberhard-wolff

SOCREATORY

Meine Trainings bei socreatory

- iSAQB® CPSA® Foundation
- iSAQB® CPSA-A® AGILA
- iSAQB® CPSA-A® API
- iSAQB® CPSA-A® FLEX
- iSAQB® CPSA-A® IMPROVE

- Documentation-as-Code
- Spring Modulith
- Phantastische Diagramme

Mein Trainer-Profil:

https://www.socreatory.com/trainers/falk-sippach

Danke für die Aufmerksamkeit!

Folien: saf2026@ewolff.com

EMail wird durch Amazon Lambda bearbeitet und 14 Tage gespeichert. Typos in der EMail-Adresse werden manuell bearbeitet.