Modernisierung in der Praxis

Modulith statt Microservices?

Falk Sippach (+ Stefan Toth)

0

Abstract

Modernisierung in der Praxis: Modulith statt Microservices?

Heutzutage stehen viele Unternehmen vor der Herausforderung, ihre in die Jahre gekommenen Anwendungen zu modernisieren, flexibler zu machen und geeignet zu strukturieren. Während Microservices weit verbreitet sind, bietet ein Modulith – modular strukturierter Monolith – eine vielversprechende Alternative, die oft übersehen wird.

Dieser Vortrag liefert einen Einblick in die praktische Arbeit mit Modulithen und die Migration dort hin. Wir berichten auch von Erfahrungen mit passenden Technologien wie Spring Modulith, xMolecules, ArchUnit sowie jQAssistant und besprechen, worauf bei deren Einsatz zu achten ist.

Außerdem besprechen wir Vorteile und Herausforderungen, die sich gegenüber der Transformation hin zu Microservices ergeben. Wir betrachten Aspekte wie Implementierungsaufwand, Impact auf den vorhandenen Quellcode, grundsätzliche Risiken bei langlaufenden Ablösungsprojekten, Teststrategien, Cognitive Load, Abhängigkeiten auf fachlicher und technischer Ebene und die Auswirkungen, die all diese Themen in der Praxis haben.

Insgesamt bietet der Vortrag einen erfahrungsbasierten Überblick zu Modulithen und richtet sich an Softwarearchitekten, Entwickler und IT-Entscheider, die sich mit der Modernisierung ihrer bestehenden Systeme auseinandersetzen.

2

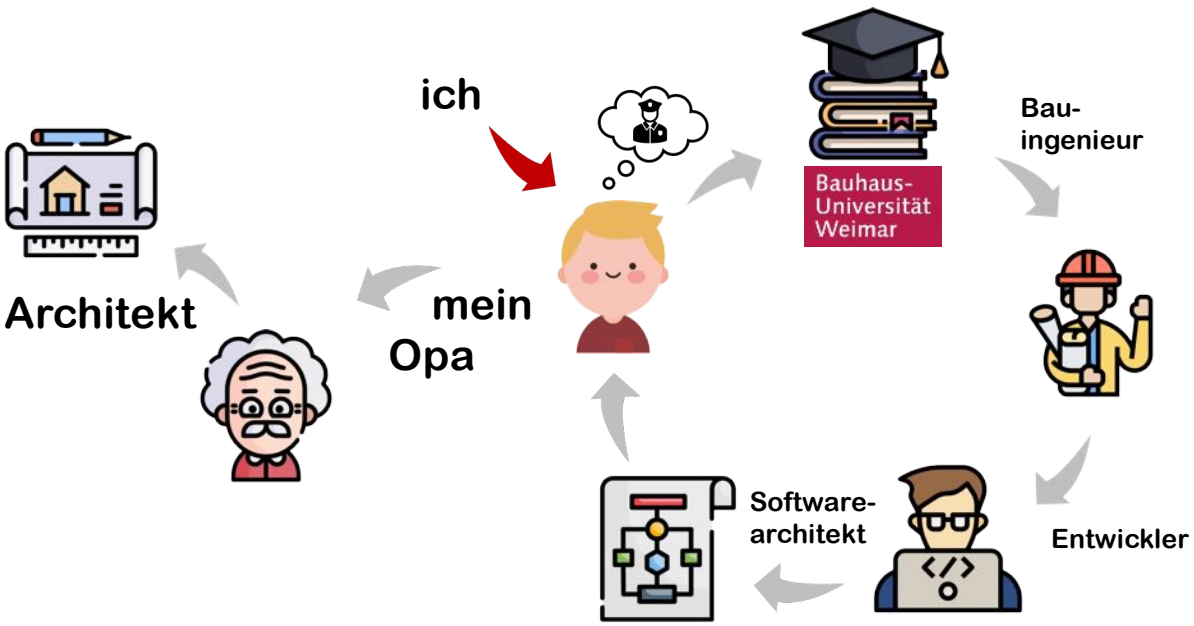
Falk Sippach

Softwarearchitekt, Berater,
Trainer

falk.sippach@embarc.de
@sippsack
linkedin.com/in/falk-sippach
www.embarc.de

embarc

3

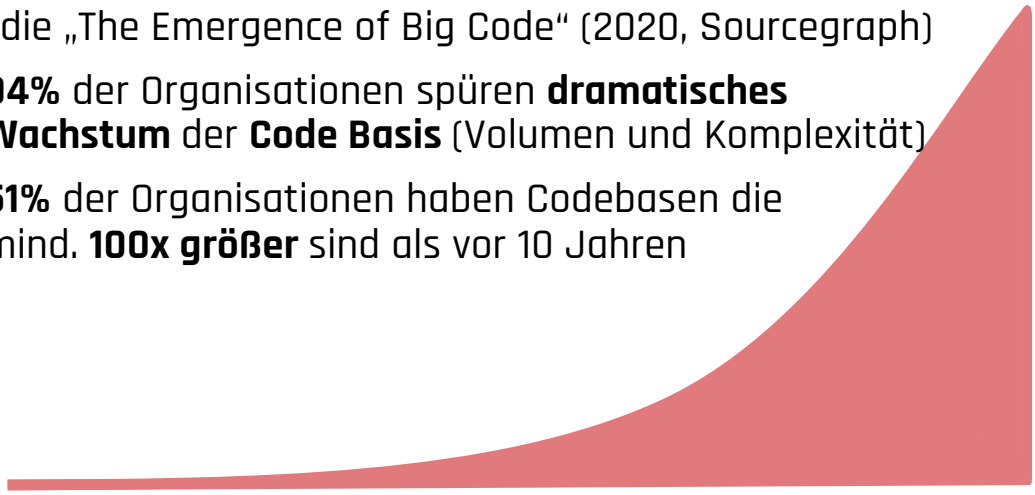


4

Softwaresysteme werden größer

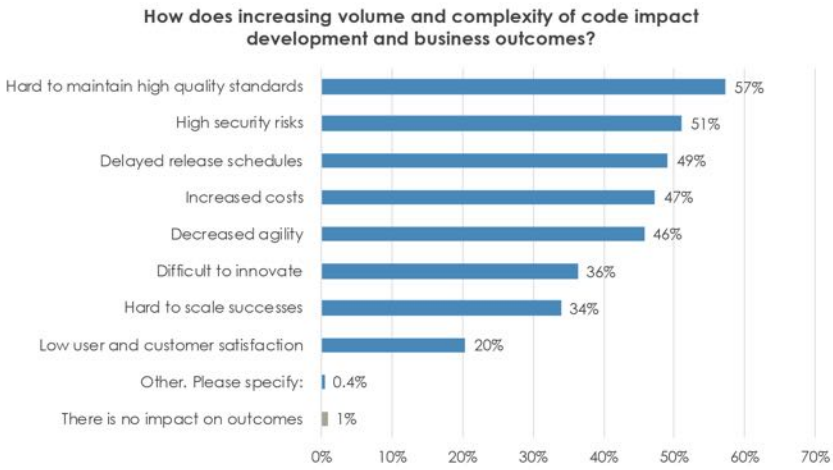
Studie „The Emergence of Big Code“ (2020, Sourcegraph)

- **94%** der Organisationen spüren **dramatisches Wachstum** der **Code Basis** (Volumen und Komplexität)
- **51%** der Organisationen haben Codebasen die mind. **100x größer** sind als vor 10 Jahren



7

Größe von Software und die Folgen…



https://info.sourcegraph.com/hubfs/CTA%20assets/sourcegraph-big-code-survey-report.pdf

8

Größe von Software und die Folgen…

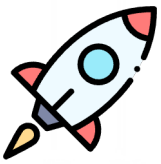
Schwer zu warten

Hohe Sicherheitsrisiken

Verspätete Auslieferung

Steigende Kosten

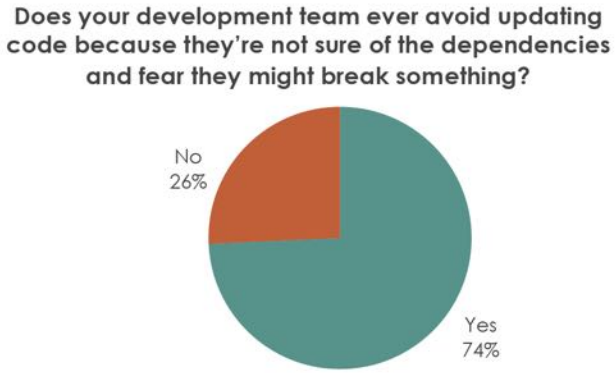
Verringerte Agilität



Kaum innovationsfähig

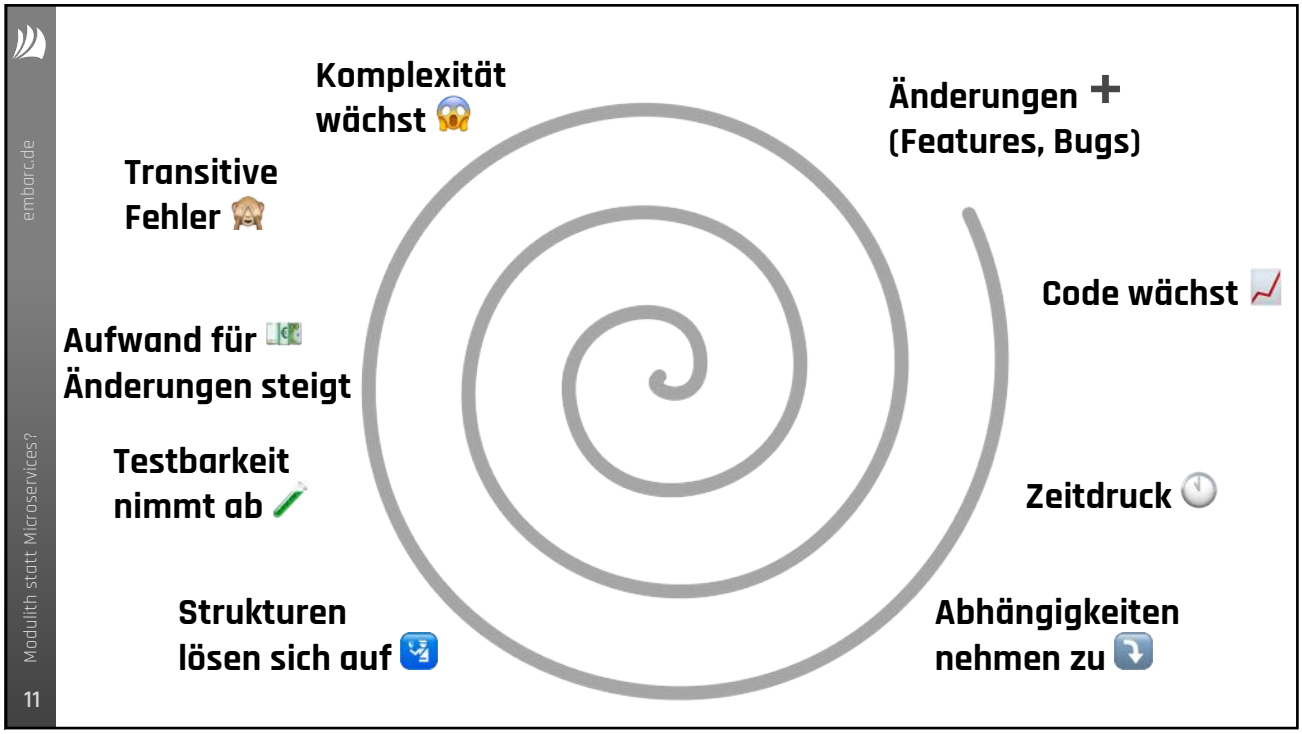
9

Größe von Software und die Folgen…



https://info.sourcegraph.com/hubfs/CTA%20assets/sourcegraph-big-code-survey-report.pdf

10


embarc.de
Modulith statt Microservices?
11
Komplexität wächst
Änderungen +
(Features, Bugs)
Transitive Fehler
Code wächst
Aufwand für Änderungen steigt
Testbarkeit nimmt ab
Zeitdruck
Strukturen lösen sich auf
Abhängigkeiten nehmen zu

11


Legacy Systeme sind
häufig monolithisch

12

Monolithen haben Grenzen

Wartbarkeit
Überraschende Abhängigkeiten,
Seiteneffekte bei Änderungen,
längere Deployment-Zyklen

Technologische Flexibilität
Einheitlicher technologischer Stack,
Keine lokalen Experimente in Prod

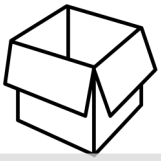
Skalierbarkeit
Technische Grenzen bei Skalierung,
Höherer Ressourcenverbrauch,
Fehler reißen alles mit

Team-Unabhängigkeit
Merge Conflicts,
Höhere kognitive Last,
Koordination bei Test/Deployment/...

13

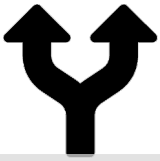
Monolithen ablösen: „Große“ Refactorings

Big Bang



White Box
Ablöse

Commercial
of the Shelf

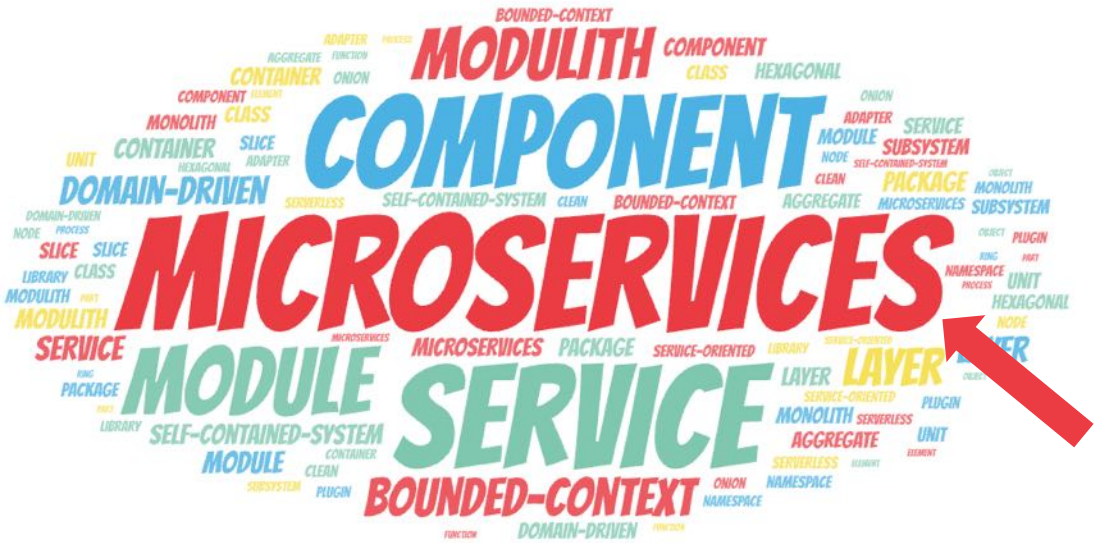


Change by
Split

Strangler
Approach

14

Strukturierung!



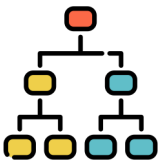
15

Modernisierungsstrategien

Big Bang **vs.** Schrittweise

Technische **vs.** fachliche Module

Monolith First **vs.** Microservices

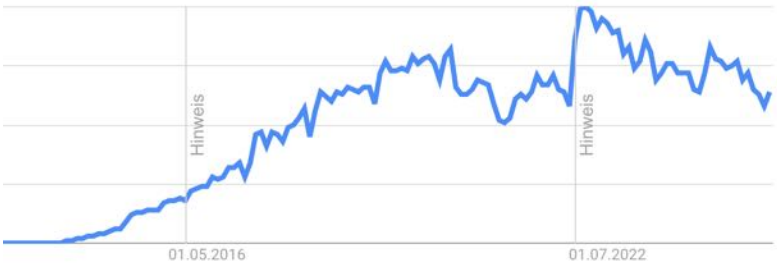


Anpassung der Team-Strukturen

16

Warum Alternativen zu Microservices?

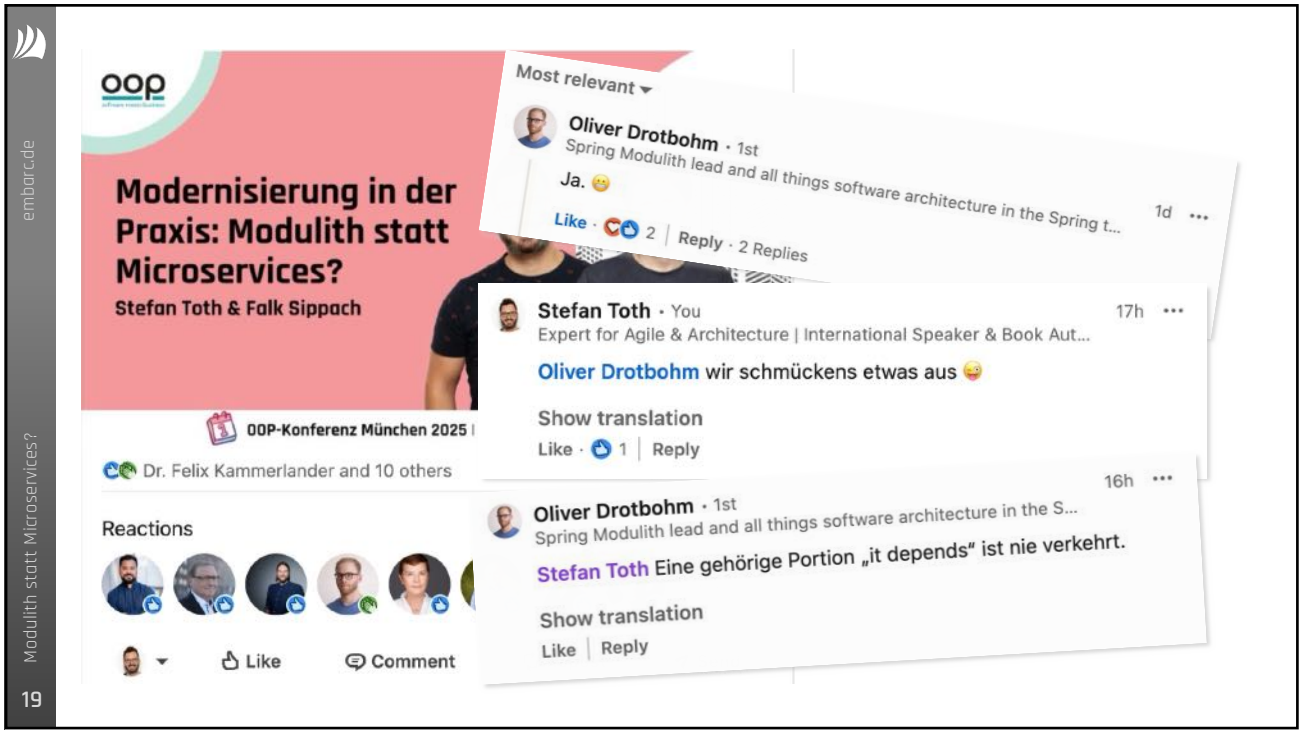
- **Gartner** prognostiziert, dass bis **2024** über **75 % der** globalen **Organisationen Microservices nutzen**
- Umfrage in **Nginx-Community:** Anteil der mit **Microservices** erstellten Anwendungen **in 2020 von 40 % auf 60 % gestiegen**.



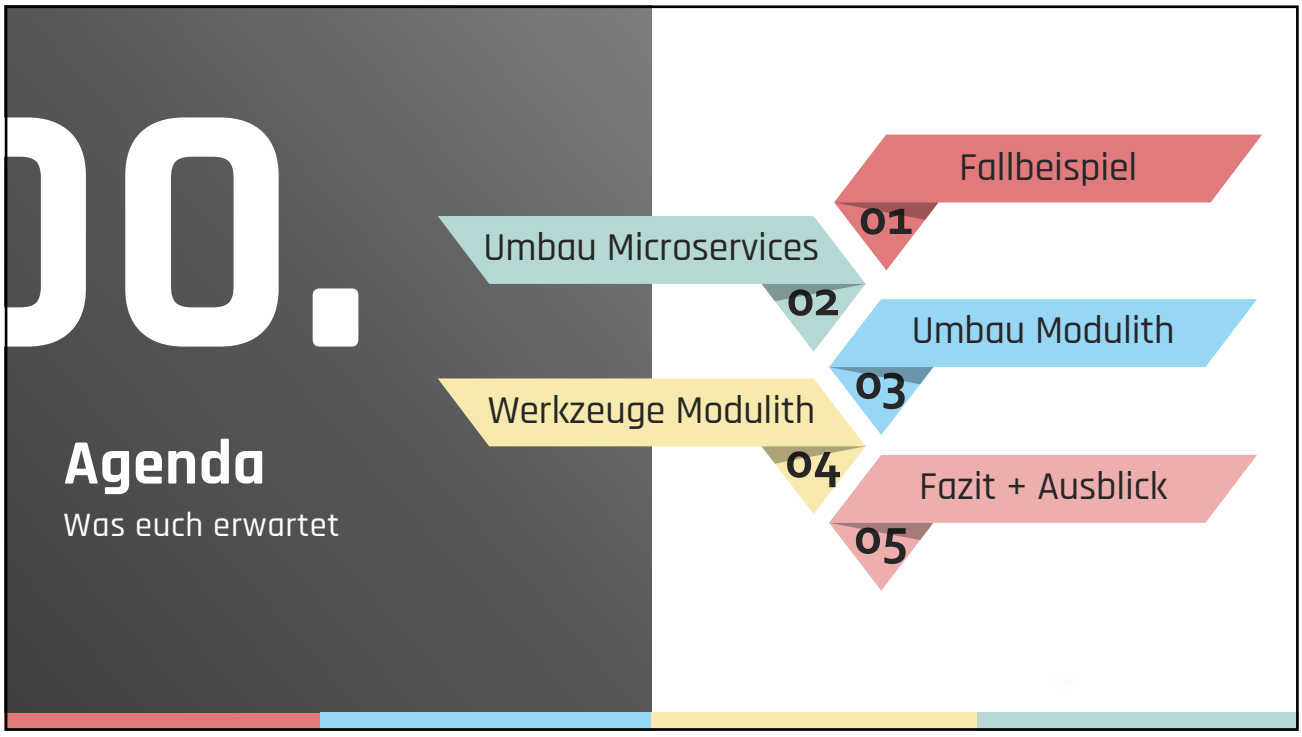
17

Strukturierung!

18


embarc.de
Modulith statt Microservices?
19
oop
Modernisierung in der Praxis: Modulith statt Microservices?
Stefan Toth & Falk Sippach
OOP-Konferenz München 2025
Dr. Felix Kammerlander and 10 others
Reactions
Like
Comment
Most relevant
Oliver Drotbohm · 1st
Spring Modulith lead and all things software architecture in the Spring t...
Ja.
Like · 2 | Reply · 2 Replies
1d
Stefan Toth · You
Expert for Agile & Architecture | International Speaker & Book Aut...
17h
Oliver Drotbohm wir schmückens etwas aus
Show translation
Like · 1 | Reply
Oliver Drotbohm · 1st
Spring Modulith lead and all things software architecture in the S...
16h
Stefan Toth Eine gehörige Portion „it depends" ist nie verkehrt.
Show translation
Like | Reply

19


00.
Agenda
Was euch erwartet
01 Fallbeispiel
02 Umbau Microservices
03 Umbau Modulith
04 Werkzeuge Modulith
05 Fazit + Ausblick

20

01.

Fallbeispiel

22

Ein Kundenprojekt

- **PayRight** - Die passgenaue **Rechnungsprüfung** für **Logistikdienstleister**.
- **Eingangsrechnungen** von Lieferanten und Sub-Auftragnehmer:
  - Viele Einzelpositionen
  - Leider oftmals fehlerhaft
- PayRight **prüft** auf Branchen-typische Regeln und spezifische **Plausibilitätsregeln** (Operational & Contractual Data)
- 8 Entwickler:innen

23

23

Zentrale NFR-Ziele der Software

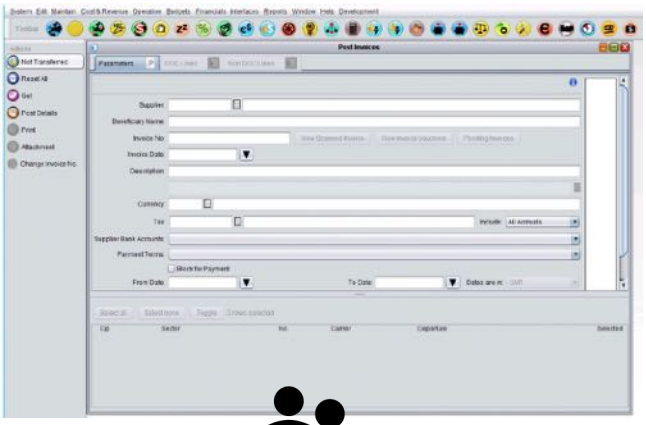
Flexibilität: Die Software ist auf Kundenbedürfnisse anpassbar und kann umfassend konfiguriert werden.

Skalierbarkeit: Die Software passt sich den Kunden-umgebungen und der Last des Kunden an.

Usability: Die Software bietet Sachbearbeitern alle relevanten Informationen an einer Stelle

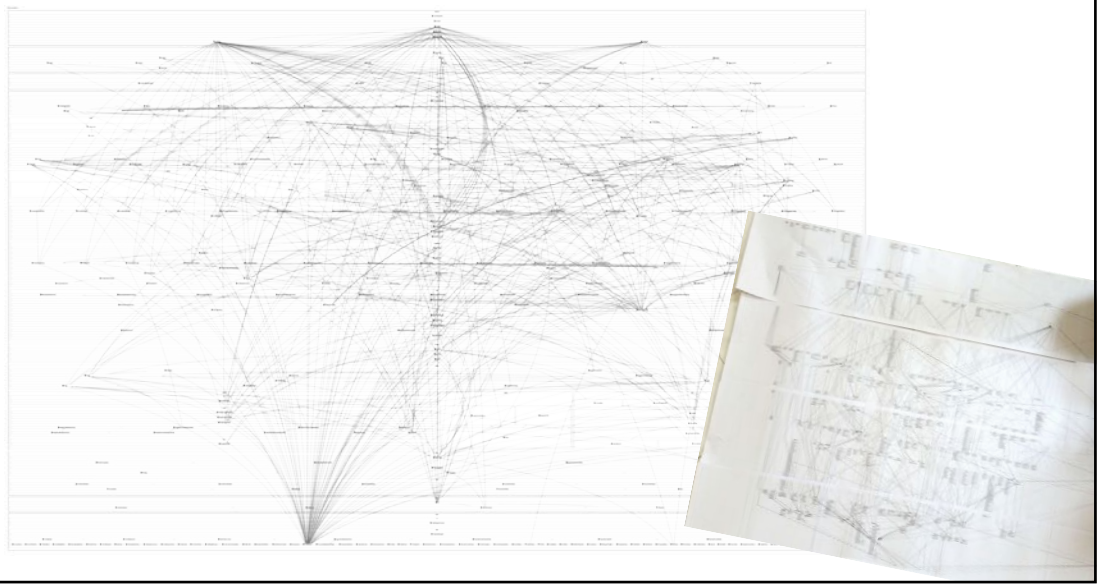
Funktionaler Überblick

- **Vertragsparameter-**verwaltung
- Konfiguration von **Kostenparametern**
- **Vorhersage** von Kosten
- **Rechnungsprüfung**
- **Reporting**
- **Budgetplanung**
- Integration in **SAP-**Abläufe



3-10 Nutzer pro Kunde

Abhängigkeiten: Entitäten / Tabellen

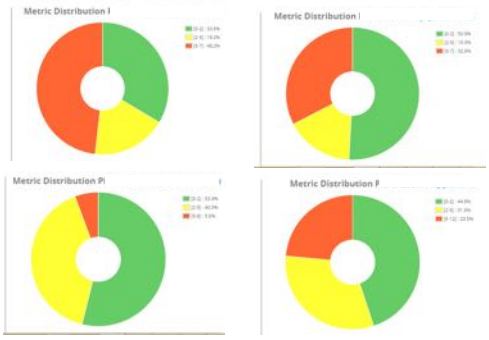


26

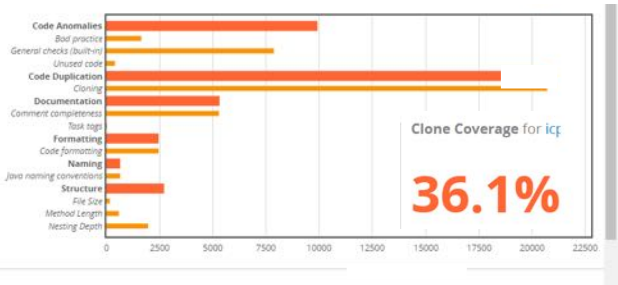
Codequalität... meh

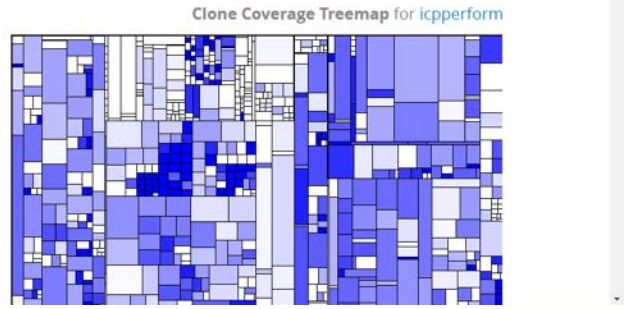
>50% der Codebasis

ist **schwer testbar**



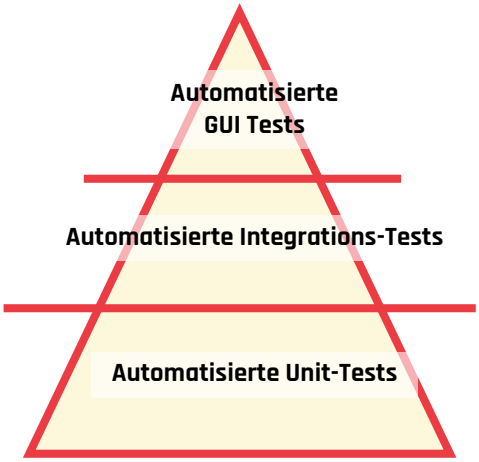
max nesting depth gelb >2 / rot >5





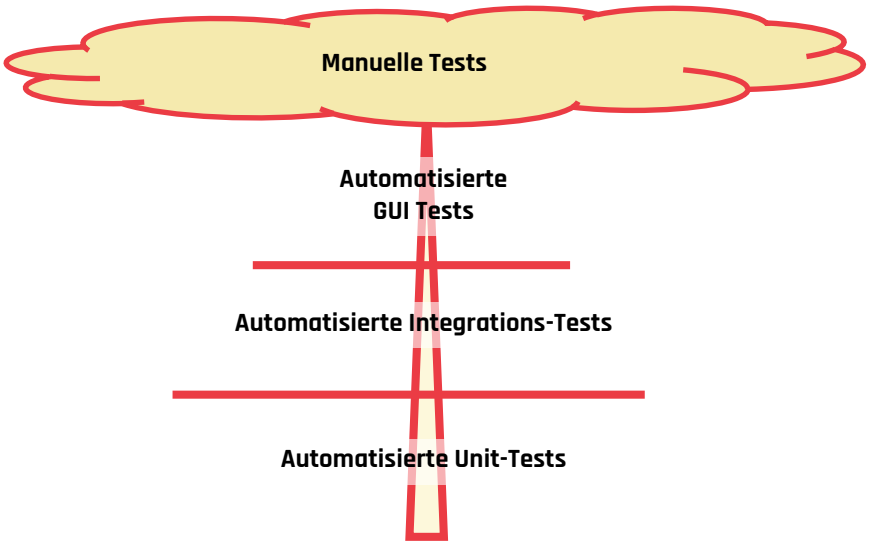
27

Die Test-Pyramide



28

Die Test-~~Pyramide~~ Pinie



29

Schwierigkeiten

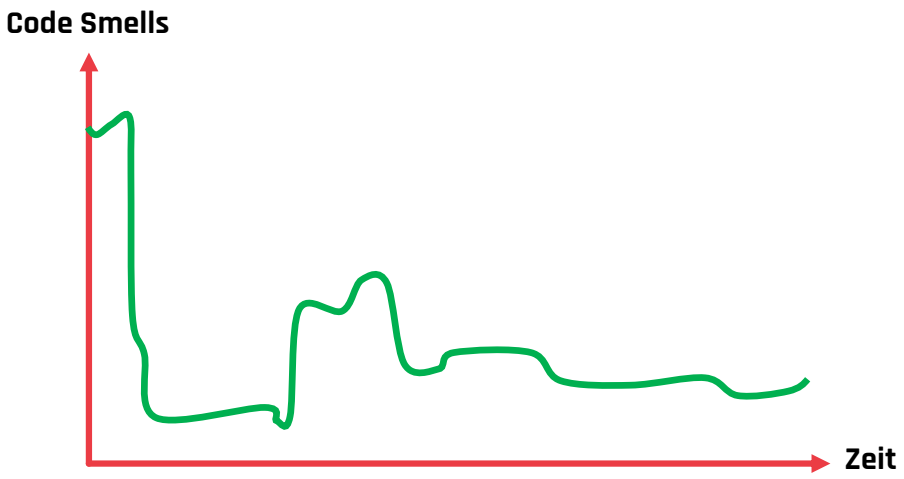
UATs:
62% failed / no run / not completed / blocked

	Operational Interface	Security User Admin	Master Data	Dom1	SAP Interfaces	Dom2	Ext.Sys	Calc 1	Reporting	Dom3	Dom4	
Failed	2		2	5		1		1		2	4	17
N/A	1	1	1	1	1	1	1	1	1	1	5	15
No Run	12		6	11	11	40	3	2	22		13	120
Not Completed	1		2	1	4			1		1		10
Passed	22		27	8	8	14	11	3			21	114
Passed with Remarks			2			1				1	2	6
Blocked		30					3	1			1	35
	38	31	40	26	24	57	18	9	23	5	46	317
Failed	5%	0%	5%	19%	0%	2%	0%	11%	0%	40%	9%	5%
N/A	3%	3%	3%	4%	4%	2%	6%	11%	4%	20%	11%	5%
No Run	32%	0%	15%	42%	46%	70%	17%	22%	96%	0%	28%	38%
Not Completed	3%	0%	5%	4%	17%	0%	0%	11%	0%	20%	0%	3%
Passed	58%	0%	68%	31%	33%	25%	61%	33%	0%	0%	46%	36%
Passed with Remarks	0%	0%	5%	0%	0%	2%	0%	0%	0%	20%	4%	2%
Blocked	0%	97%	0%	0%	0%	0%	17%	11%	0%	0%	2%	11%
	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

*"Das wichtigste für ein Tool zur **Rechnungsprüfung** ist, dass es funktioniert und **korrekt** arbeitet. Momentan haben wir hier eklatant Schwierigkeiten, Unsere **User Acceptance Test** Sessions sind von **vielen Fehlern** gekennzeichnet."*

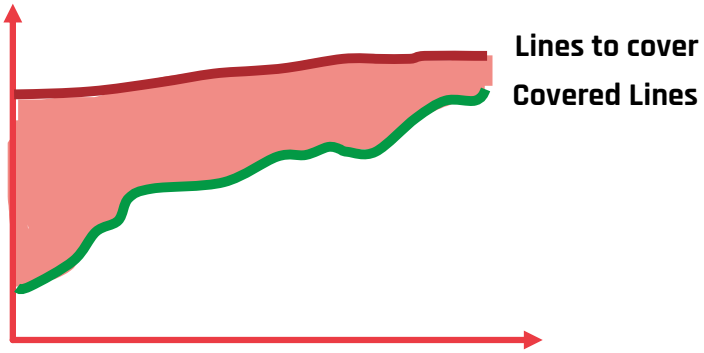
30

Qualitätsinitiative?



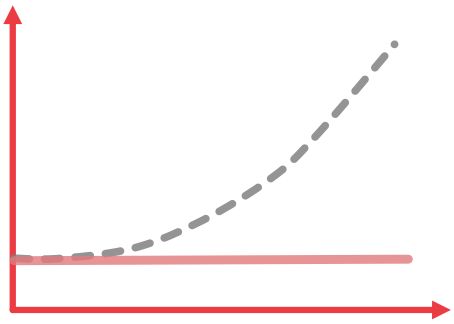
31

Coverage?

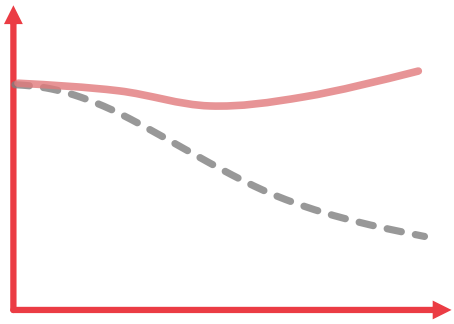


33

Typisches Ergebnis der Bemühungen...



Gefühlte **Qualität bei Kunden**
(Nutzerbefragung)



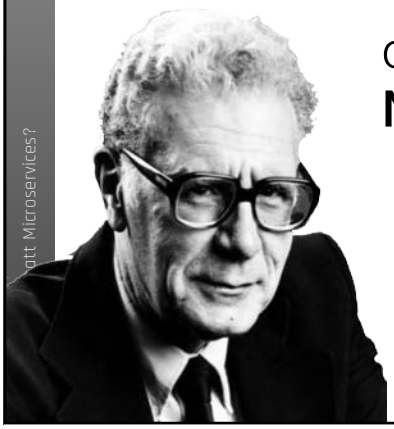
Fehler in Test und Prod
(Kumuliert)

erwartet
tatsächlich

34

Russel L. Ackoff

Auch wenn die einzelnen Bausteine **sauber getestet**, **performant** und **wartbar** sind

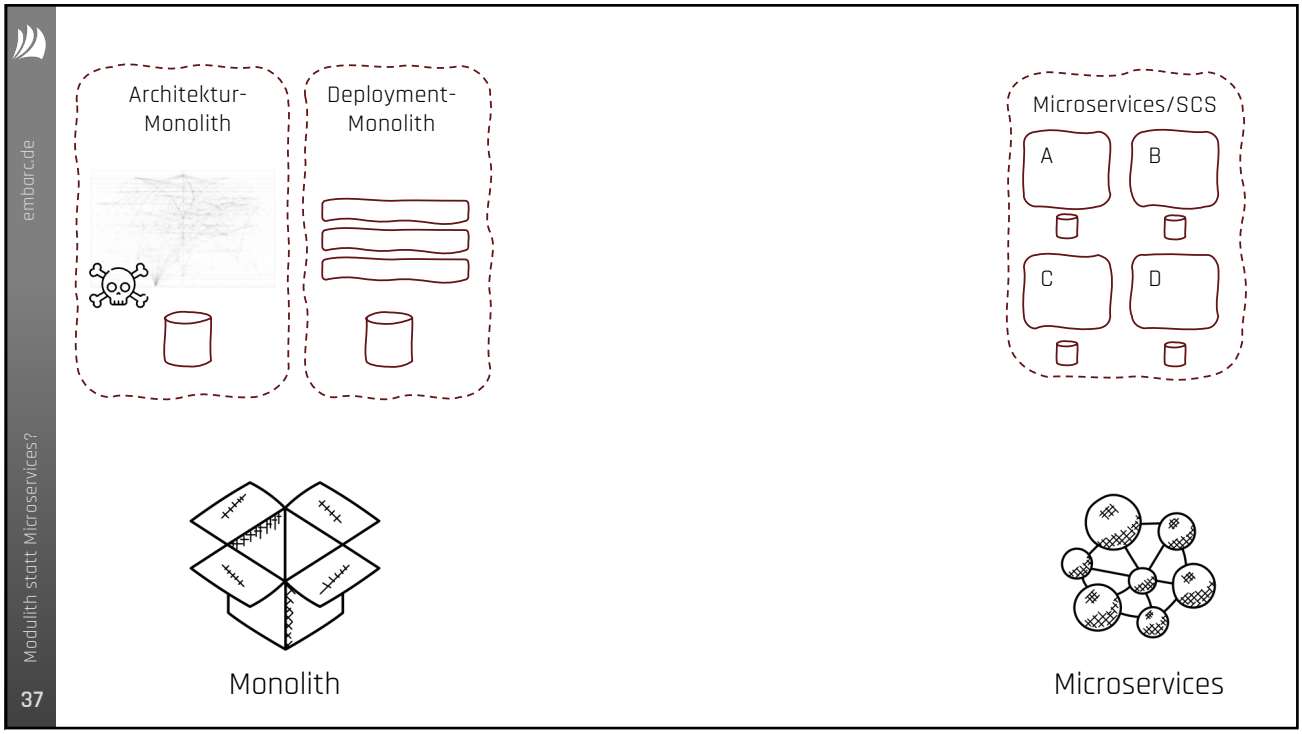


”The **quality of a system** depends on **how the parts fit**, **NOT** how they **act**, taken **seperately**”

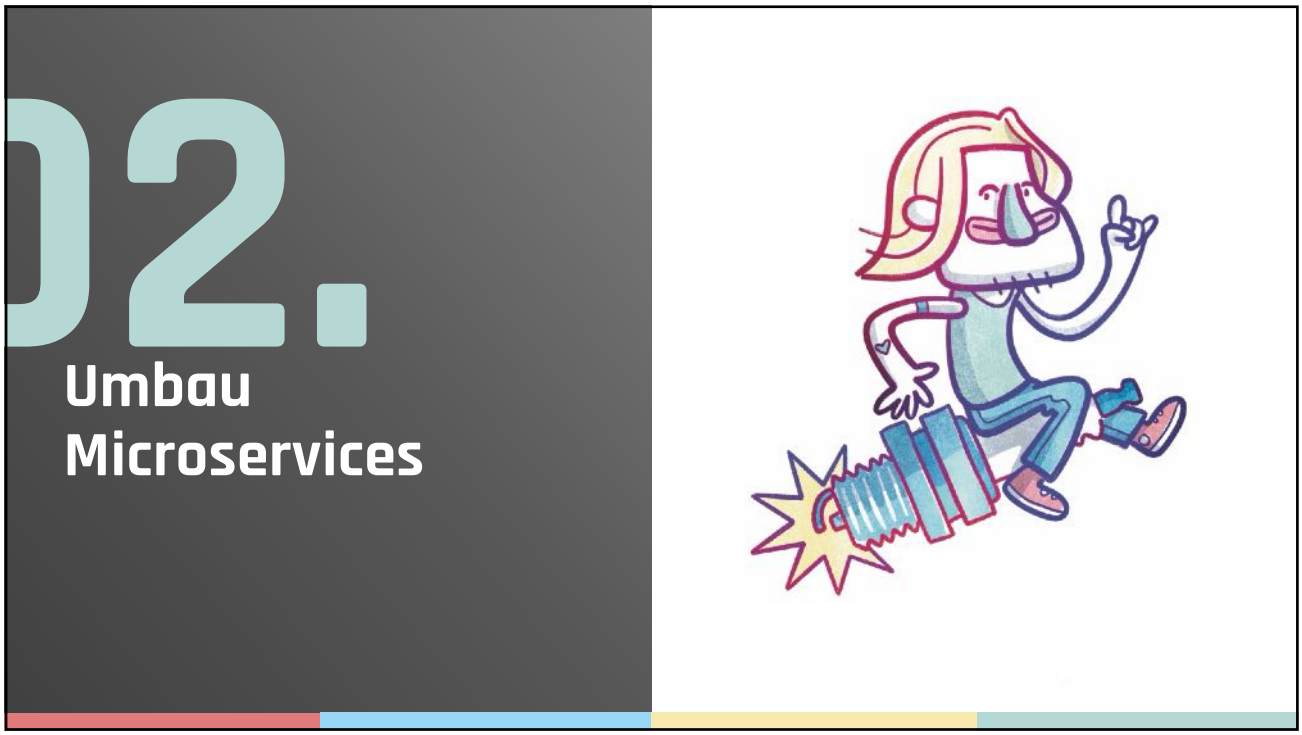
35

Strukturierung!

36


embarc.de
Modulith statt Microservices?
37
Architektur-
Monolith
Deployment-
Monolith
Microservices/SCS
A
B
C
D
Monolith
Microservices

37


02.
Umbau
Microservices

39

Microservices in kurz …

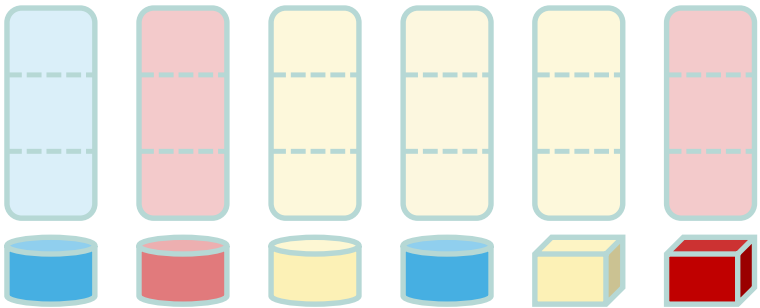
*"In short, the microservice architectural style is an approach to developing a **single application** as a suite of **small services**, each running in its own process and communicating with lightweight mechanisms…"*

(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

Microservices im Überblick



Zerlegung in Vertikalen

Oft genannte Argumente dafür:

- \+ Fördert Domänenorientierung
- \+ Funktionale Blöcke leicht auszutauschen und zu ergänzen
- \+ Technische Schulden besser kontrollierbar
- \+ Separate Skalierung, unabhängiges Deployment

Migrationsaspekte

- **Fachlicher Schnitt**
- **Technischer Umbau: Anwendung**
- **Technischer Umbau: Auslieferung / Betrieb**
- **Transition: Migrationsstrategien**

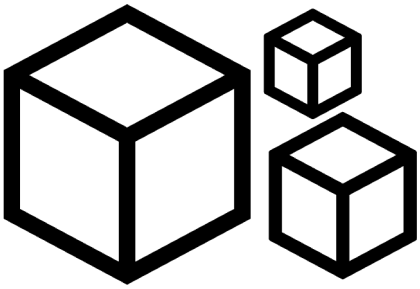
42

Migrationsaspekte

- **Fachlicher Schnitt**
- **Technischer Umbau: Anwendung**
- **Technischer Umbau: Auslieferung / Betrieb**
- **Transition: Migrationsstrategien**

43

Fachliche Grenzen finden



Granularität - Größe
von Microservices

46

Finden von Services ...

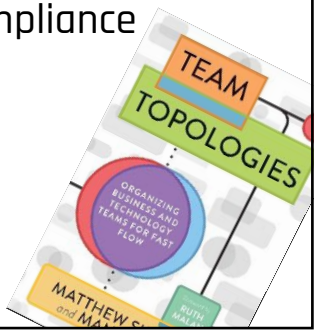
- Aus **Requirements**-Dokumenten (Sub-)Domänen ableiten
- Mit **Event-Storming**
- Aus **Code Strukturen**
- Aus **Datenfluss**
- ...

47

„Fracture Planes“ als weitere Überlegungen

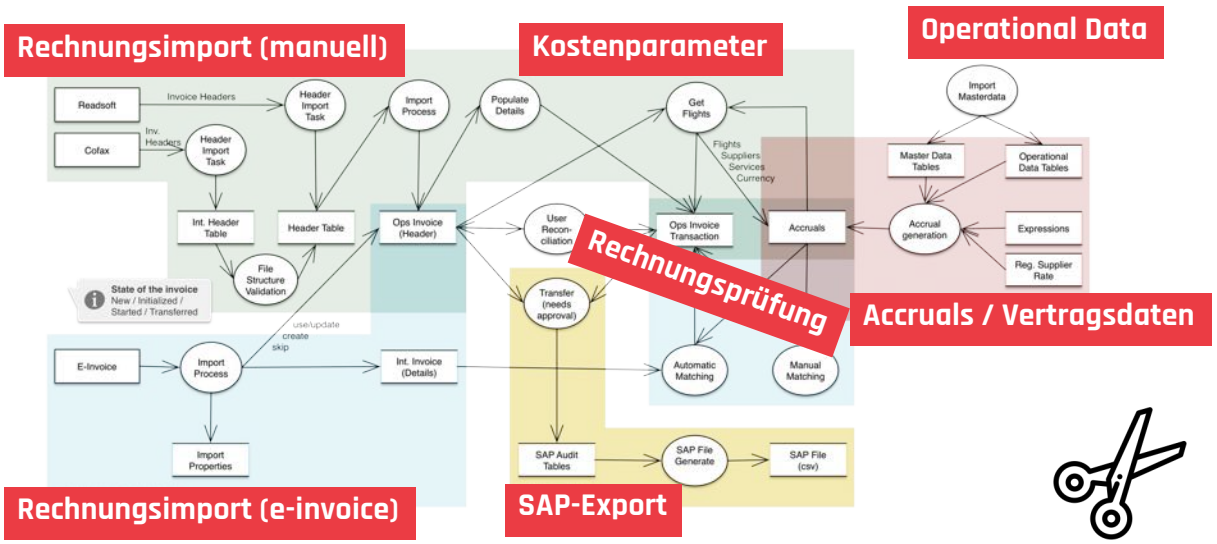
> “A **fracture plane** is a **natural seam** in the software system that allows the system to be **easily split into** two or more **parts.**”
>
> *M. Pais, M. Skelton*

- **Fachliche Grenzen (Bounded Contexts)**
- Performance isolation
- User personas
- Change cadence
- Regulatory compliance
- Team location
- Risk
- Technology



48

Fallbeispiel: Datenfluss bei „Prüfung“



49

Zerlegungidee

10 fachliche Hauptservices...

- Operational Data (Logistik-Sys. Datenimport)
- Cost Register
- Accruals (Vorhersagen / Abgrenzungs-posten)
- Contract Configuration
- Masterdata
- Invoice Reconcilliation (Rechnungs-prüfung)
- Invoice Import (manual)
- Invoice Import (electronic)
- Budget Planner
- Reporting

51

Migrationsaspekte

- Fachlicher Schnitt
- **Technischer Umbau: Anwendung**
- Technischer Umbau: Auslieferung / Betrieb
- Transition: Migrationsstrategien

55

Technische Aspekte: Anwendung

API-Gateway

Containerisierung

Service Discovery

Datenbankstrategie

UI-Strategie

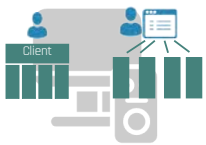
Technische Aspekte: Anwendung

API-Gateway

Containerisierung

Service Discovery

Datenbankstrategie



UI-Strategie

Typische Qualitätsziele der Anwendung

Sicherstellen von Resilienz

Konsistenz

Security

Performance

Interoperabilität

Skalierbarkeit

Typische Qualitätsziele der Anwendung

Circuit Breaker
Retry- und Timeout-Strategien

Sicherstellen von Resilienz

Eventual Consistency
Sagas/Choreographie-Patterns

Konsistenz

IP Netzwerk
User-Level & Application Level Auth

Security

Caching, Load Balancing
Optimierung Service-zu-Service-Kommunikation

Performance

API-Kompatibilität
Abwärts Kompatibilität
Versionsstrategien

Interoperabilität

Auto-Scaling, Workload-Orchestration
Sharding & Partitioning

Skalierbarkeit

Migrationsaspekte

Fachlicher Schnitt

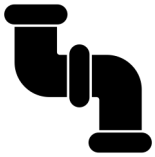
Technischer Umbau: Anwendung

Technischer Umbau: Auslieferung / Betrieb

Transition: Migrationsstrategien

74

Technische Aspekte: Auslieferung & Betrieb



CI/CD-Pipelines

Observability

Disaster Recovery

Test-Strategie

75

Technische Aspekte: Auslieferung & Betrieb

Deployment Automatisierung
Rolling Updates, Blue-Green Deployments

CI/CD-Pipelines

Zentrales Logging, Monitoring, Distributed Tracing

Observability

Failover Mechanismen
Organisatorische Aspekte: Blech-Ops vs. Teams

Disaster Recovery

Consumer-Driven Contracts (Pact)
Chaos-Engineering
und Fitness Functions

Test-Strategie

Migrationsaspekte

Fachlicher Schnitt

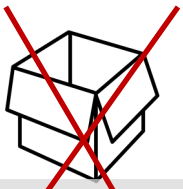
Technischer Umbau: Anwendung

Technischer Umbau: Auslieferung / Betrieb

Transition: Migrationsstrategien

„Große“ Refactorings

Big Bang



~~White Box Ablöse~~

Commercial of the Shelf

Change by Split

Strangler Approach

81

Big Bang Rewrite

- Anforderungen neu einsammeln, altes System als Spezifikation
- neue Features werden (a) abgelehnt, (b) aufgeschoben oder (c) doppelt entwickelt
- hohes Risiko bei Live-Gang/Migration, spätes Feedback

Big Bang

82

Strangler Approach

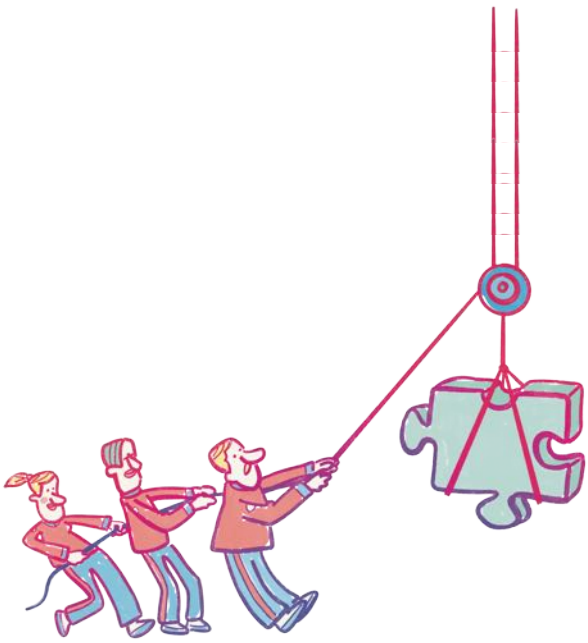
- modulare Strukturteile der neuen Architektur Schritt für Schritt extrahieren
- iteratives Vorgehen, relativ gefahrlos
- alte und neue Welt müssen miteinander integriert werden

Strangler Approach

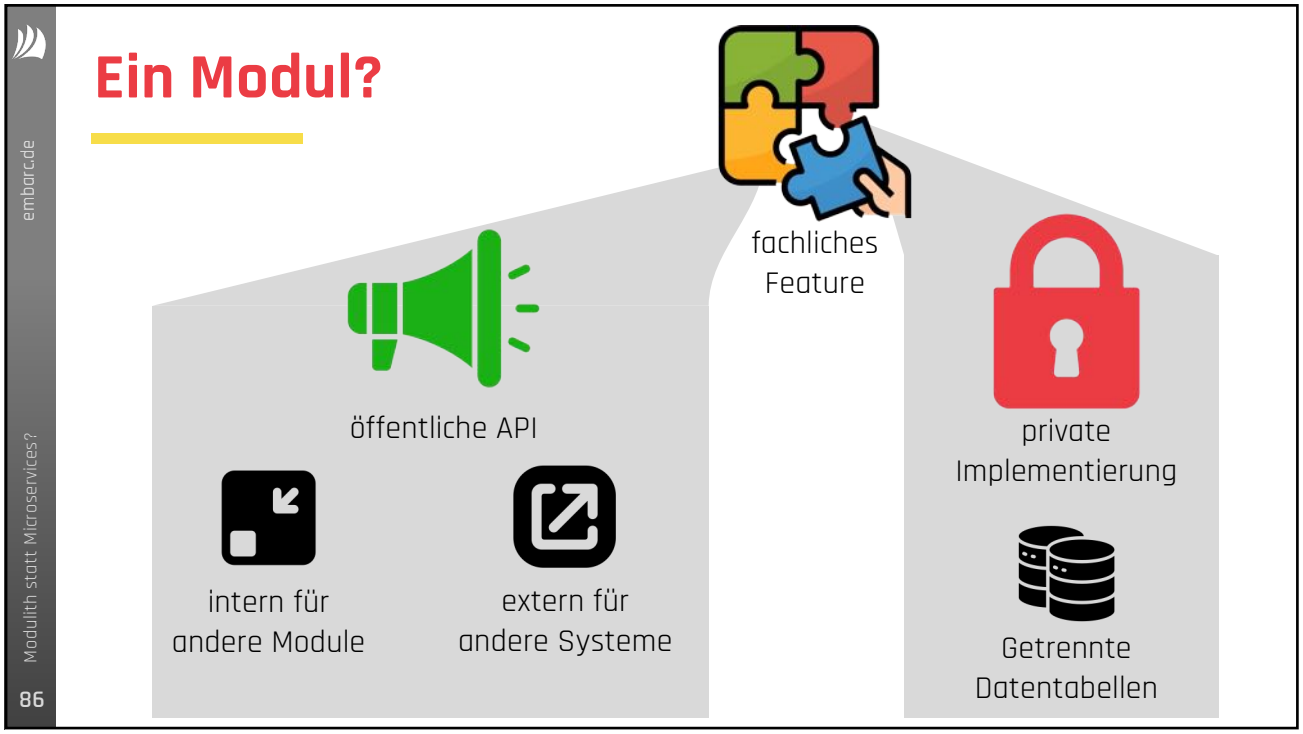
83

03.

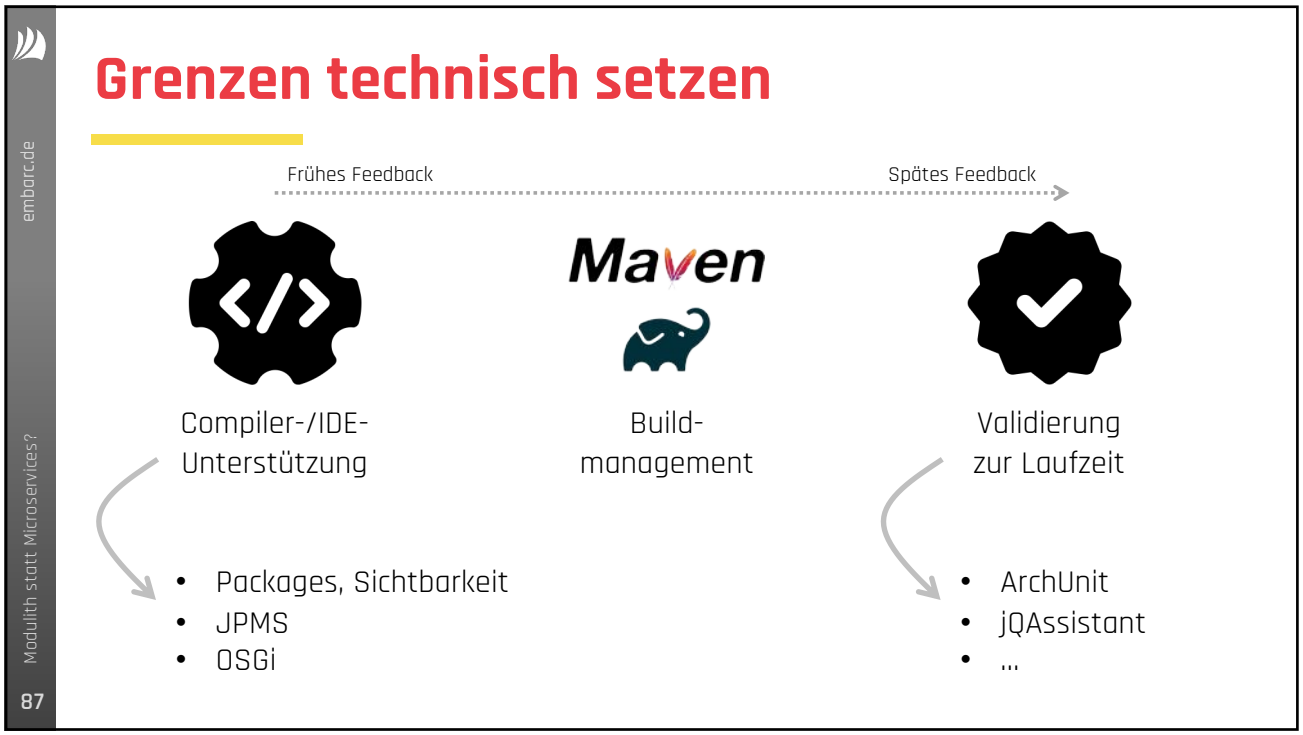
Umbau Modulith



85


Ein Modul?
fachliches Feature
öffentliche API
intern für andere Module
extern für andere Systeme
private Implementierung
Getrennte Datentabellen
embarc.de
Modulith statt Microservices?
86

86


Grenzen technisch setzen
Frühes Feedback
Spätes Feedback
Compiler-/IDE-Unterstützung
Build-management
Validierung zur Laufzeit
• Packages, Sichtbarkeit
• JPMS
• OSGi
• ArchUnit
• jQAssistant
• ...
embarc.de
Modulith statt Microservices?
87

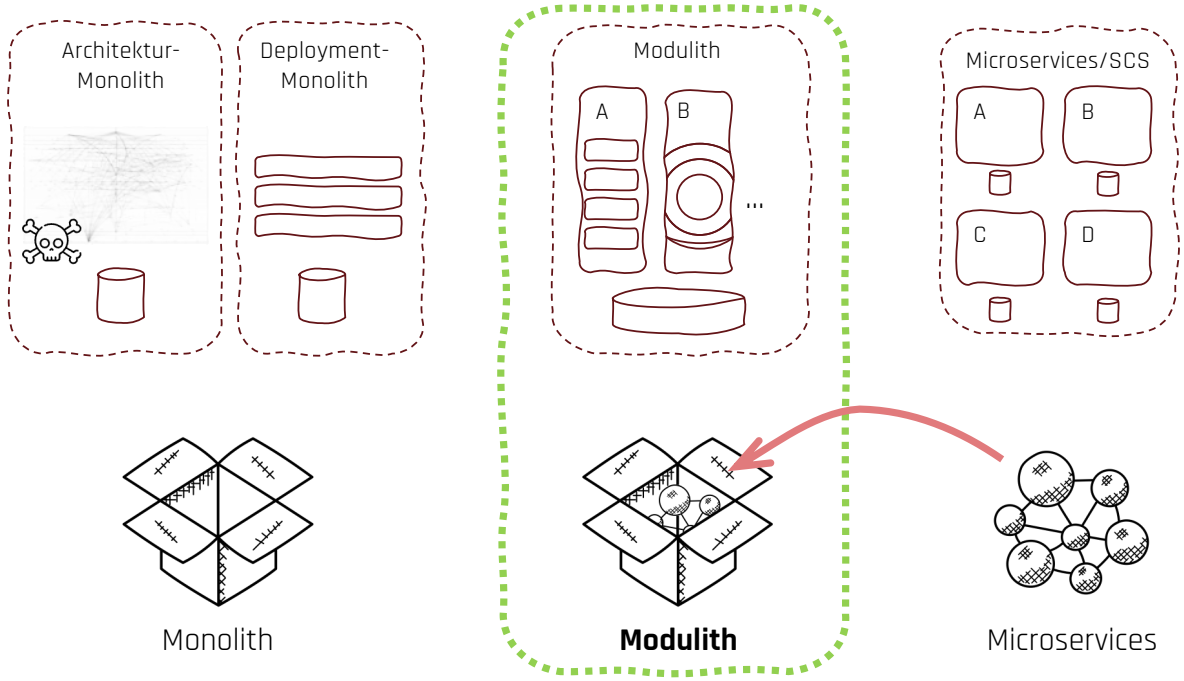
87

Modulith = Modularer Monolith

>> Modulith architecture is a style of software design that **emphasizes modularity within a monolithic application.** It aims to combine the simplicity and straightforward deployment model of a monolithic architecture with the modularity and maintainability typically associated with microservices.

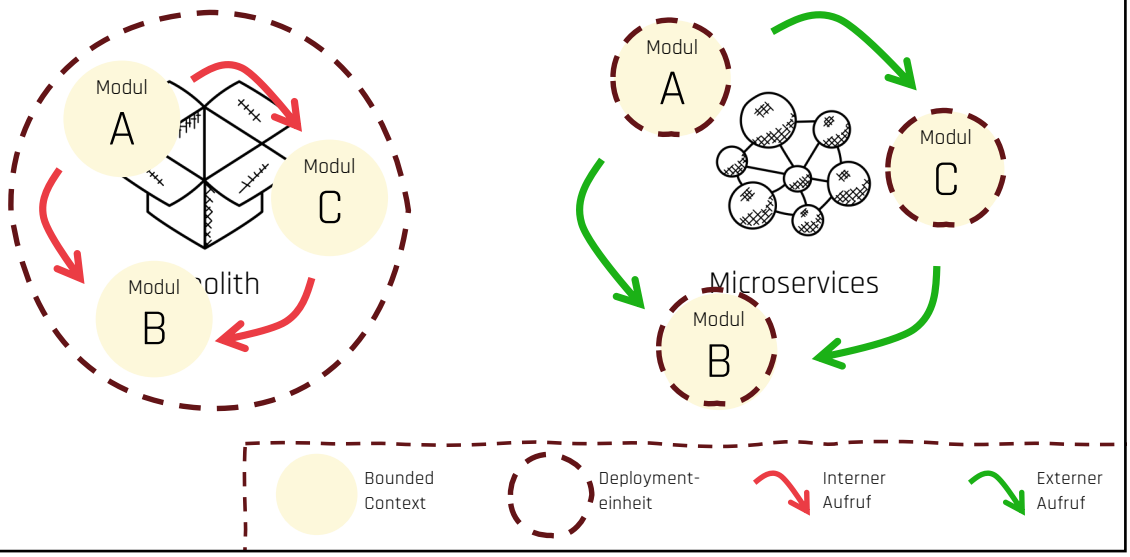
(https://dzone.com/articles/architecture-style-modulith-vs-microservices) <<

88



89

Modulith vs. Microservices



90

Migrationsaspekte

- **Fachlicher Schnitt**
- **Technischer Umbau: Anwendung**
- **Technischer Umbau: Auslieferung / Betrieb**
- **Transition: Migrationsstrategien**

94

Migrationsaspekte

Fachlicher Schnitt

Technischer Umbau: Anwendung

Technischer Umbau: Auslieferung / Betrieb

Transition: Migrationsstrategien

Vergleich zum Microservices Umbau	Neutral	Geringer Aufwand

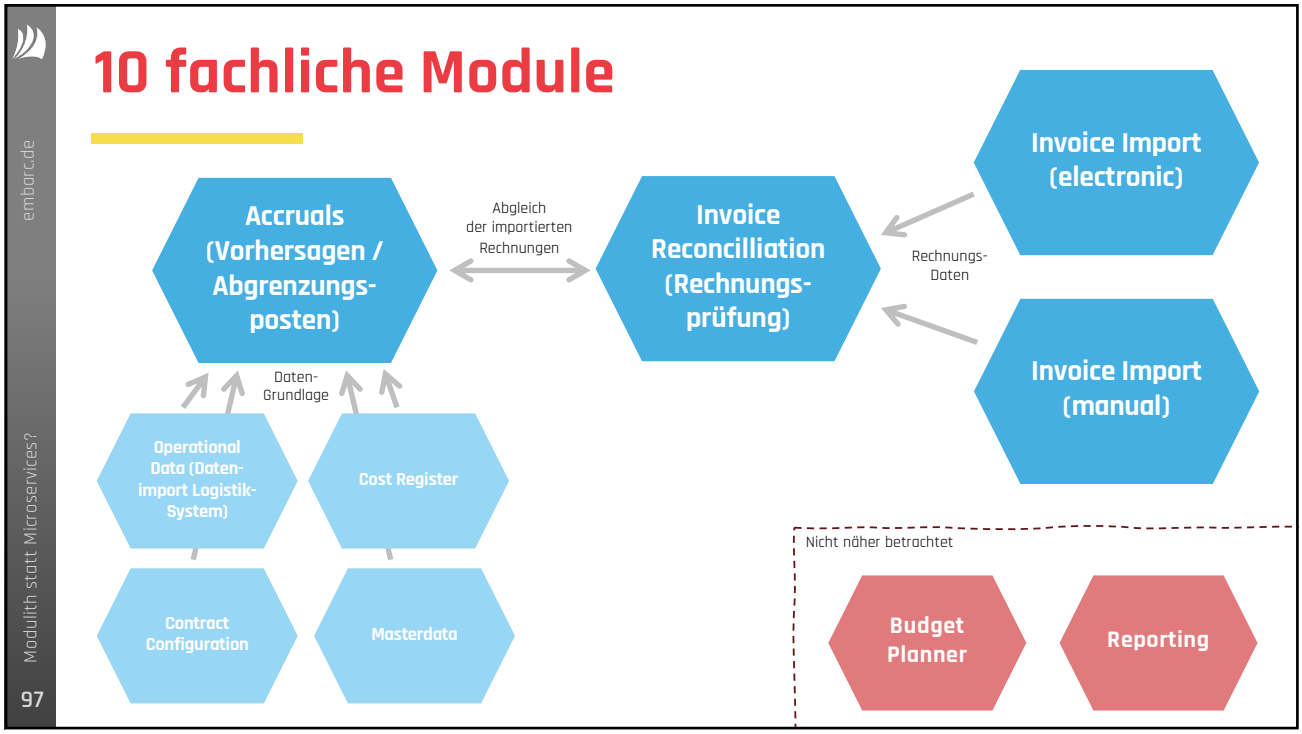
95

Zerlegungidee

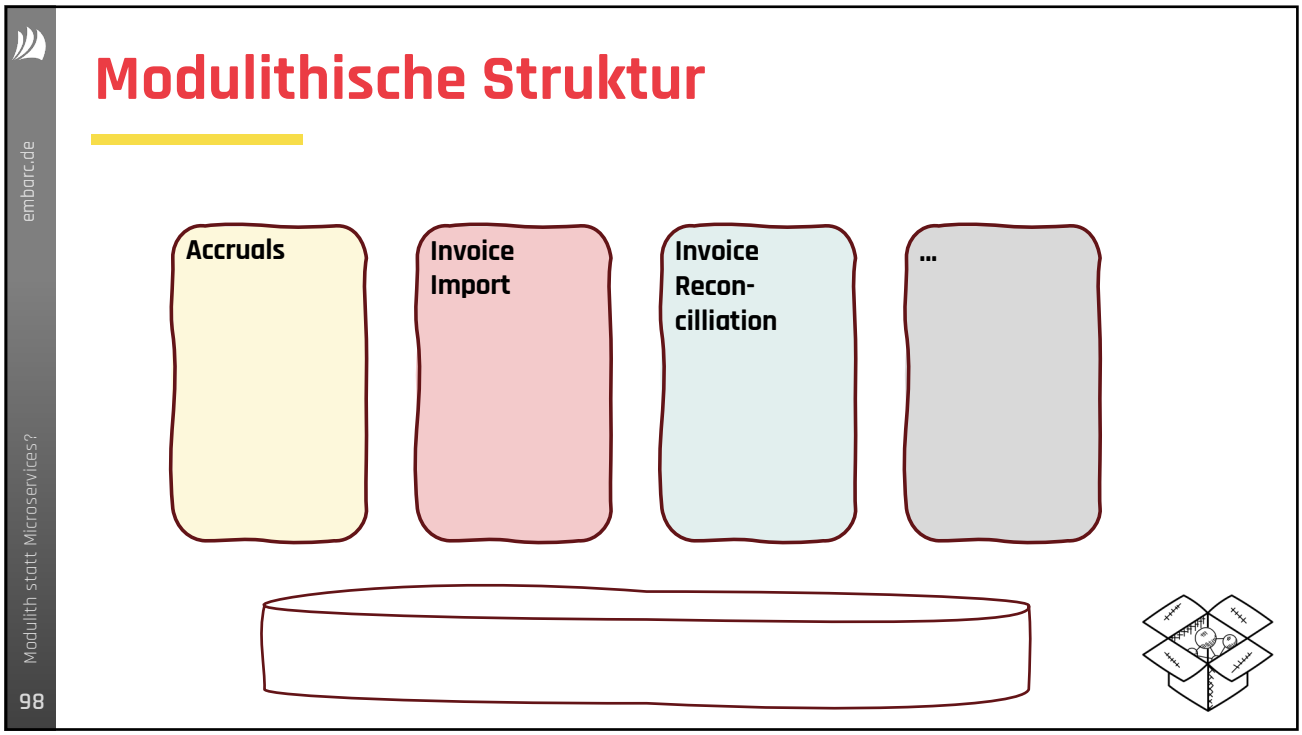
Analog Microservices

- Operational Data (Logistik-Sys. Datenimport)
- Cost Register
- Accruals (Vorhersagen / Abgrenzungs-posten)
- Contract Configuration
- Masterdata
- Invoice Reconcilliation (Rechnungs-prüfung)
- Invoice Import (manual)
- Invoice Import (electronic)
- Budget Planner
- Reporting

96

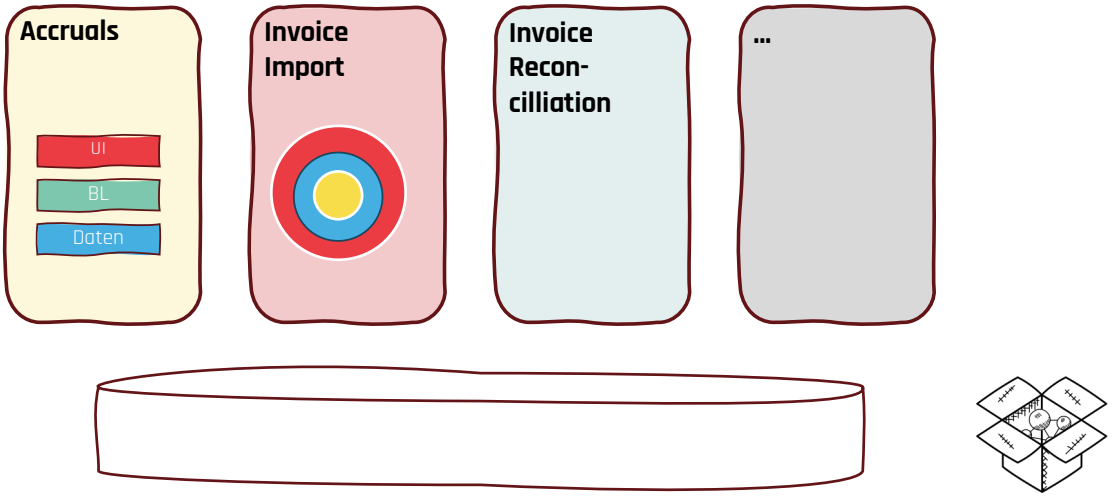

10 fachliche Module
embarc.de
Modulith statt Microservices?
97
Accruals (Vorhersagen / Abgrenzungs-posten)
Abgleich der importierten Rechnungen
Invoice Reconcilliation (Rechnungs-prüfung)
Invoice Import (electronic)
Rechnungs-Daten
Invoice Import (manual)
Daten-Grundlage
Operational Data (Daten-import Logistik-System)
Cost Register
Contract Configuration
Masterdata
Nicht näher betrachtet
Budget Planner
Reporting

97


Modulithische Struktur
embarc.de
Modulith statt Microservices?
98
Accruals
Invoice Import
Invoice Recon-cilliation
...

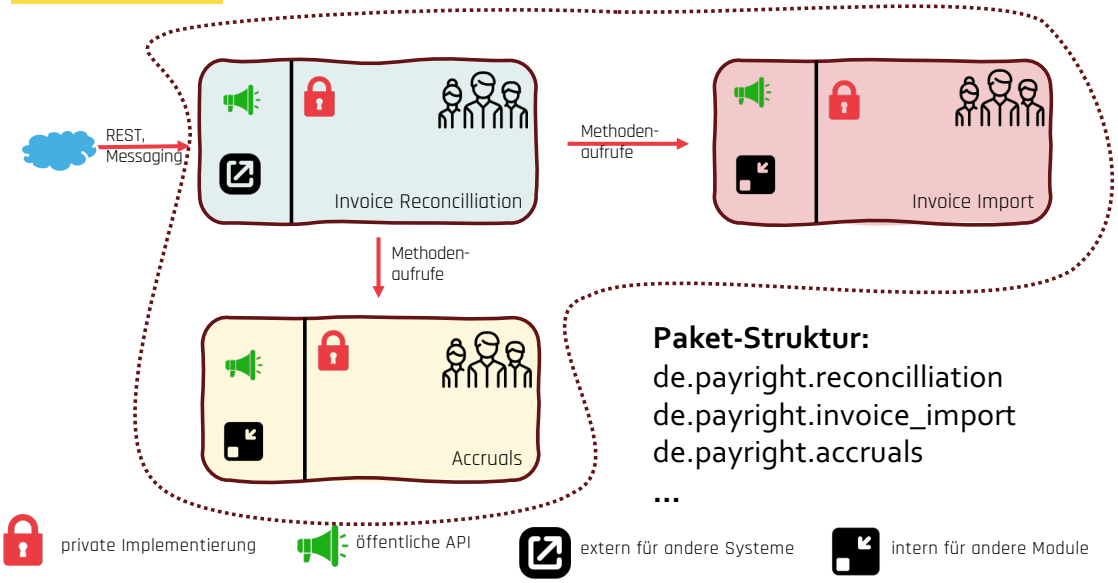
98

Modulithische Struktur



99

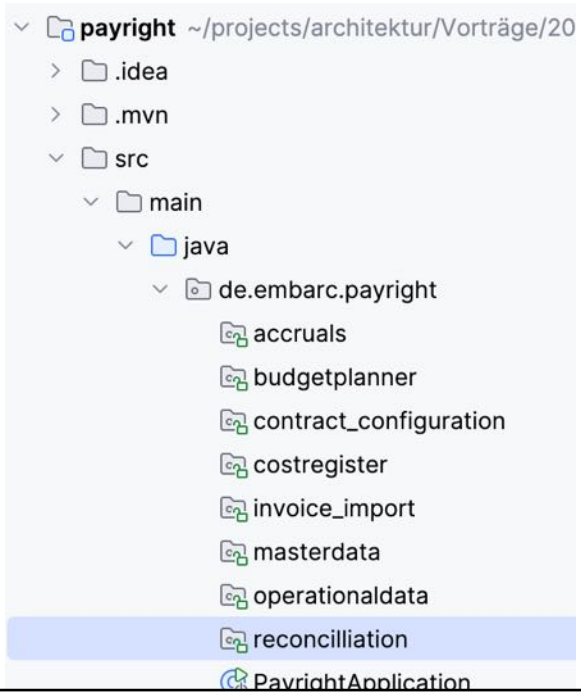
Entkopplung, Information Hiding



100

Paket-Struktur

de.payright.reconcilliation
de.payright.invoice_import
de.payright.accruals
...



101

Migrationsaspekte

Fachlicher Schnitt	**Technischer Umbau: Anwendung**
Technischer Umbau: Auslieferung / Betrieb	Transition: Migrationsstrategien

Vergleich zum Microservices Umbau: Neutral | Geringer Aufwand

102

Technische Aspekte: Anwendung

API-Gateway

Containerisierung

Service Discovery

Datenbankstrategie

UI-Strategie

Technische Aspekte: Anwendung

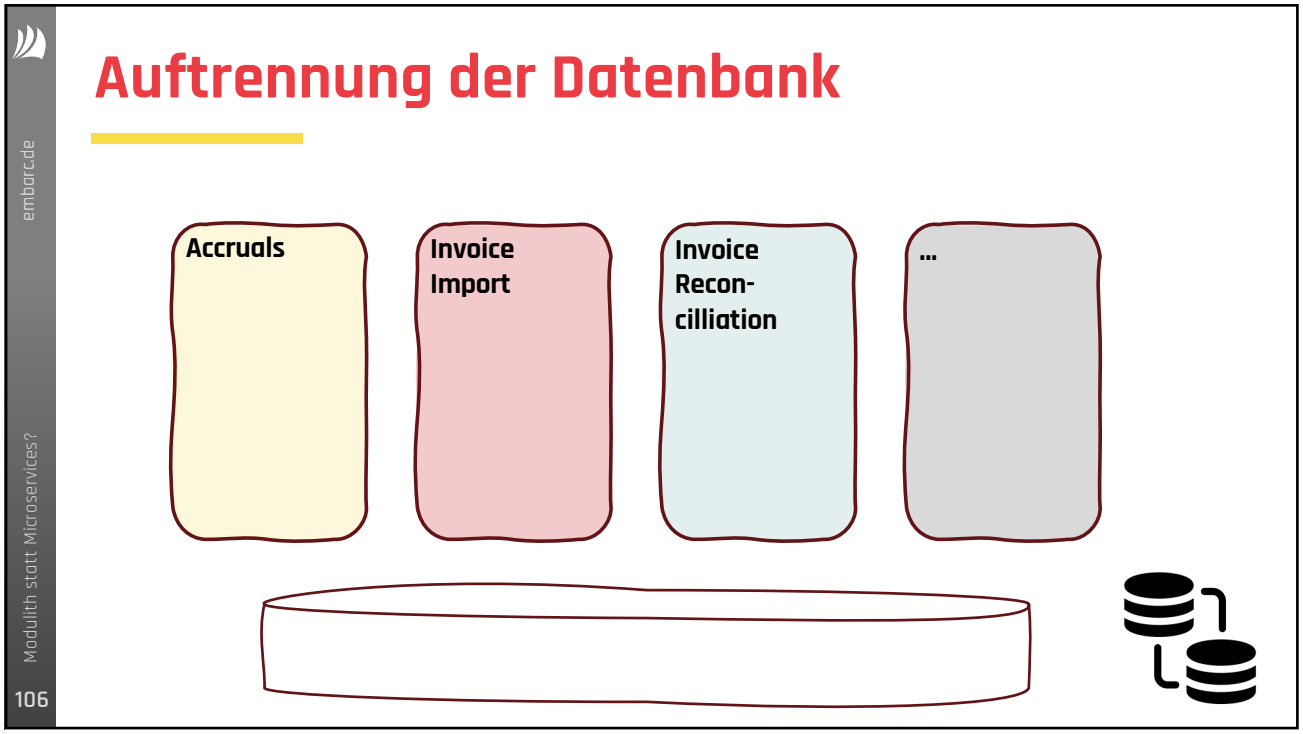
API-Gateway

Containerisierung

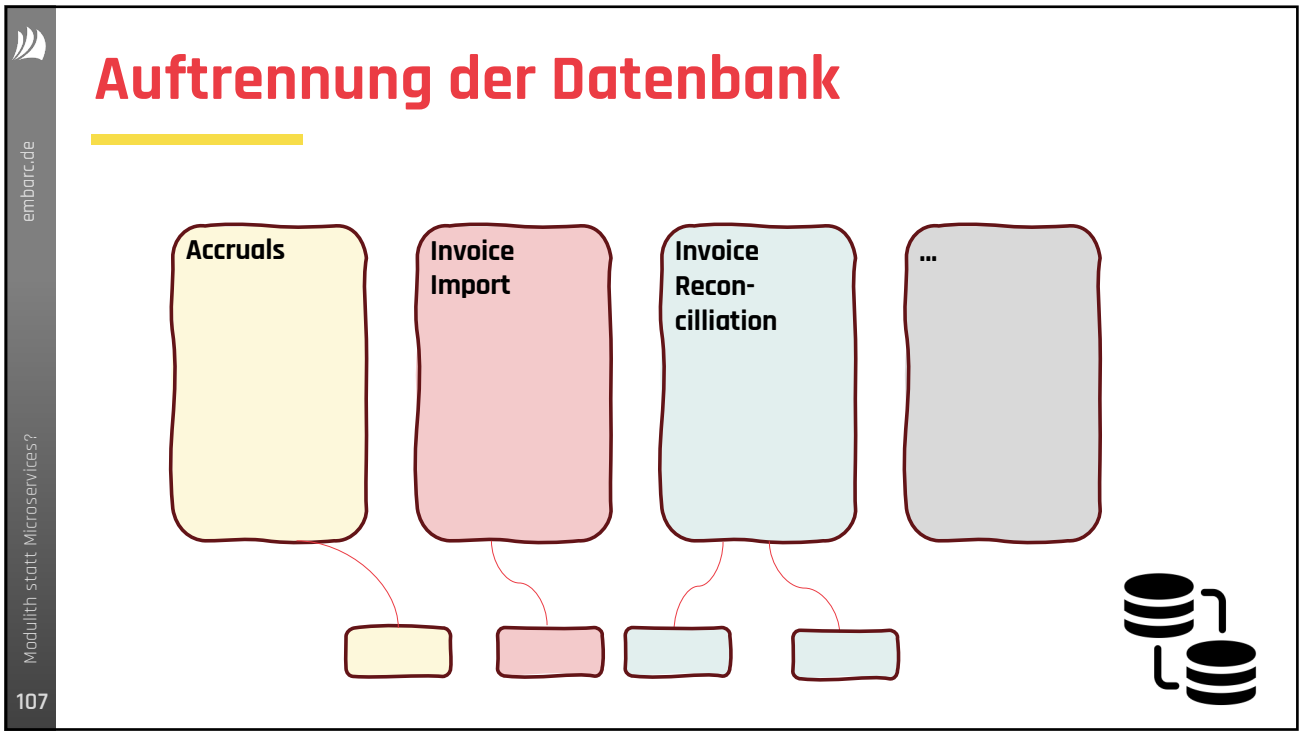
Service Discovery

Datenbankstrategie

UI-Strategie


embarc.de
Modulith statt Microservices?
106
Auftrennung der Datenbank
Accruals
Invoice Import
Invoice Recon-cilliation
...

106


embarc.de
Modulith statt Microservices?
107
Auftrennung der Datenbank
Accruals
Invoice Import
Invoice Recon-cilliation
...

107

Typische Qualitätsziele der Anwendung

Sicherstellen von Resilienz

Konsistenz

Security

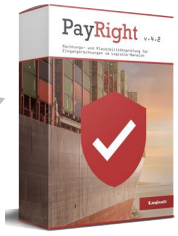
Performance

Interoperabilität

Skalierbarkeit

Konsistenz

Diese Aussagen sind nicht allgemeingültig, sie beziehen sich auf das Fallbeispiel!



- starke Konsistenz durch ACID-Transaktionen in einer Datenbank
- keine Synchronisierung der Daten notwendig
- keine Workarounds (SAGA-Pattern, fachliche Kompensationen, ...)

Konsistenz

Migrationsaspekte

Fachlicher Schnitt	**Technischer Umbau: Anwendung**
Technischer Umbau: Auslieferung / Betrieb	**Transition: Migrationsstrategien**

Vergleich zum Microservices Umbau	Neutral	Geringer Aufwand

111

Technische Aspekte: Auslieferung & Betrieb

- CI/CD-Pipelines
- Observability
- Disaster Recovery
- Test-Strategie

112

embarc

embarc.de

Modulith statt Microservices?

Technische Aspekte: Auslieferung & Betrieb

weniger Aufwand
besserer Überblick

CI/CD-Pipelines

zentrales Logging ausreichend
einfachere Fehlersuche

Observability

Disaster Recovery

Bordmittel (Pakete, Sichtbarkeiten, ...), kein CDCT
Module isoliert testbar
einfache Integrationstests

Test-Strategie

113

113

embarc.de

Modulith statt Microservices?

Migrationsaspekte

Fachlicher Schnitt

Technischer Umbau: Anwendung

Technischer Umbau: Auslieferung / Betrieb

Transition: Migrationsstrategien

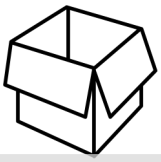
Vergleich zum Microservices Umbau	Neutral	Geringer Aufwand

117

117

„Große“ Refactorings

Big Bang



White Box Ablöse

Commercial of the Shelf

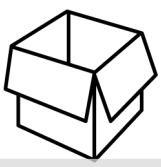
Change by Split

Strangler Approach

118

White Box Ablöse

- bei bereits vorhandenen Strukturen (fachliche Module) können einzelne Module neu implementiert und ausgetauscht werden
- aber alte Modulstruktur (häufig eher technisch strukturiert) bleibt erhalten



White Box Ablöse

119

Big Bang vs. Strangler Approach

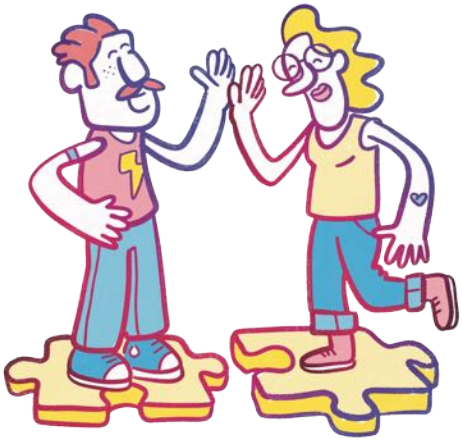
- Big Bang hat viele Nachteile
- Strangler Approach ist Kompromiss zur Risikominderung

Big Bang

Strangler Approach

120

04. Werkzeuge Modulith



121

QuellCode != Dokumentation

Sichtbar- machen von ...?

Architektur-konzepten

Architektur-entscheidungen

fachliche & technische Modulstrukturen

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

Validierung der Architekturregeln

Explizite Architektur-konzepte im Code

Transparenz durch Dokumentation

Architekturvalidierung

- Überprüfung der Erreichbarkeit angestrebter Qualitätsziele
  - Funktionalität → Unit-, Integrations- & Systemtests
  - Sicherheit → Penetrationstests, Vulnerability-Scans
  - Performance → Lasttests
- Art der Validierung
  - automatisiert vs. manuell
  - direkt vs. indirekt
  - 0/1, Value, Trend

Fitness-funktionen

Architekturvalidierung

- Wartbarkeit
  - Prüfung von Code-Strukturen
  - manuelle Reviews vs. automatisierte Tests
  - Coding-Guidelines vs. Architekturkonzepte und- Constraints

sonarqube QAssistant CQSE ArchUnit

  - Indirekte Validierung der Erreichbarkeit des Qualitätsmerkmals
    - Angemessenheit vs. Einhaltung von Strukturen

ArchUnit

- Testing-Framework für Architekturrichtlinien in Java/.NET

```
@ArchTest
public static final ArchRule services_should_not_access_controllers =
        noClasses().that().resideInAPackage("..service..")
        .should().dependOnClassesThat().resideInAPackage("..controller..");
```

- Durchsetzen von Hexagonal Architecture, Clean Architecture, Layered Architecture, …
- Vermeidung von unerlaubten Abhängigkeiten
- Sicherstellen von Namenskonventionen

ArchUnit

QAssistant

5 **Verifizierung** mit Cypher-Queries

4

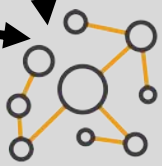
Regeln und **Konzepte** ergänzen (Cypher-Queries)

3

Strukturen im Software-Projekt (Quellcode, Git-Historie, …)

1

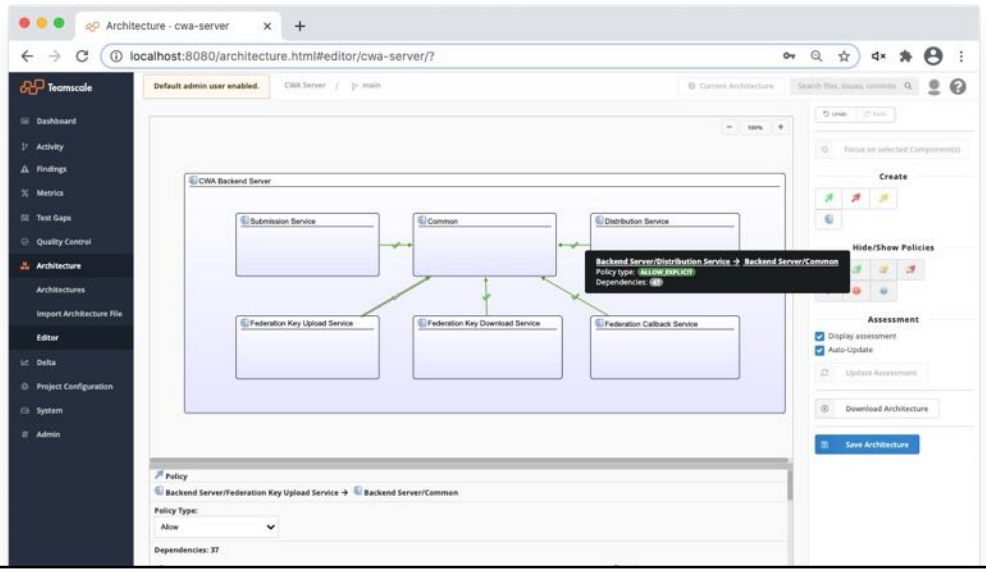
2 Struktur als Graph



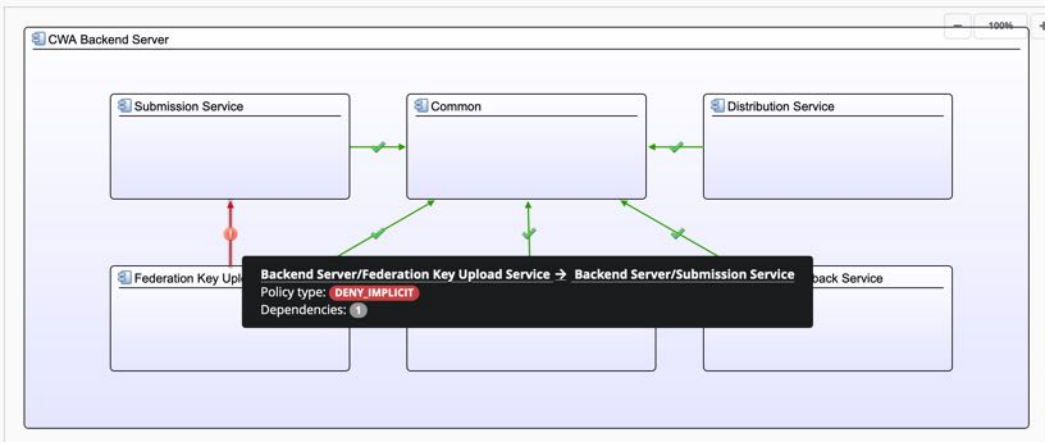
neo4j Graphdatenbank

Maven Build-Management

Abhängigkeiten in Teamscale



So sähe ein Verstoß aus …



Hinweis: Die Abhängigkeit zwischen Upload und Submission Service hatten wir von Hand im Quelltext eingebaut zu Demonstrationszwecken.

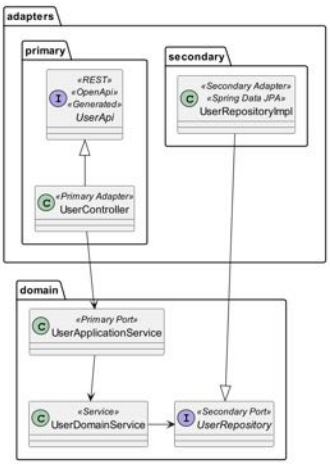
Validierung der Architekturregeln

Explizite Architektur-konzepte im Code

Transparenz durch Dokumentation

130

xMolecules, ContextMapper




```
public interface UserRepository {
```

```
@Repository
public class UserRepositoryImpl
extends SimpleJpaRepository<User, Long>
implements UserRepository {
```

```
@SecondaryPort
public interface UserRepository {
```

```
@SecondaryAdapter
@Repository
public class UserRepositoryImpl
extends SimpleJpaRepository<User, Long>
implements UserRepository {
```

131

jMolecules - Beispiele

Architekturstil/-muster	Konzepte/Annotationen
CQRS	@Command, @CommandDispatcher, @CommandHandler, @QueryModel
Layered	@DomainLayer, @ApplicationLayer, @InfrastructureLayer, @InterfaceLayer
Onion Classic	@DomainModelRing, @DomainServiceRing, @ApplicationServiceRing, @InfrastructureRing
Onion Simplified	@DomainRing, @ApplicationRing, @InfrastructureRing
Hexagonal	@Application, @PrimaryAdapter, @SecondaryAdapter, @PrimaryPort, @SecondaryPort

132

Validierung der Architekturregeln

Explizite Architektur-konzepte im Code

Transparenz durch Dokumentation

133

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 konventioneller Werkzeuge demotiviert und **bremst beim Dokumentieren** sogar aus.

134

Validierung der Architekturregeln

Explizite Architekturkonzepte im Code

Transparenz durch Dokumentation

135

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)

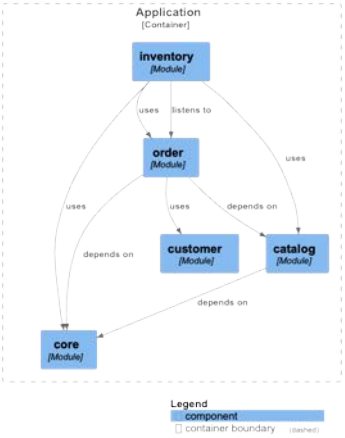
Spring Modulith

- 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

Spring Modulith

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



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

Spring Modulith

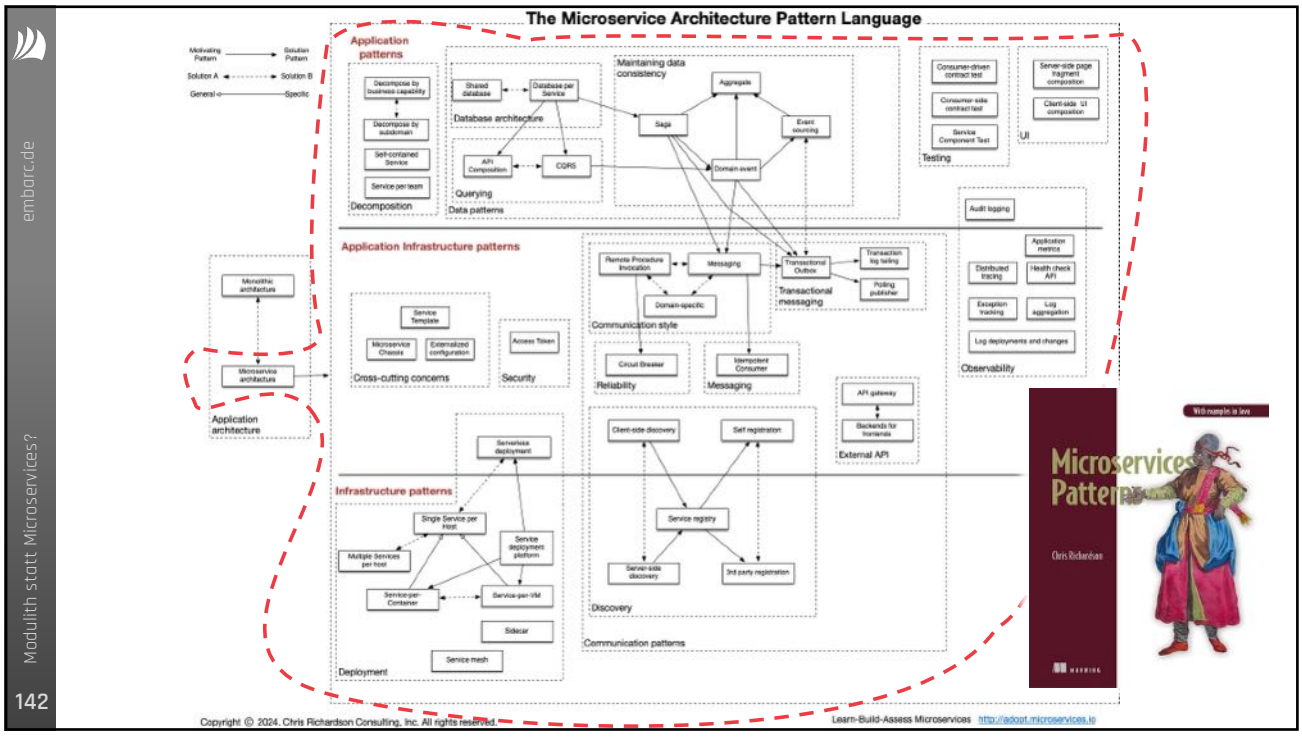
- **@ApplicationModuleTest** – Hochfahren von einem Modul und den Abhängigkeiten
- **@ApplicationModuleListener** – asynchrone, persistierte Events

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

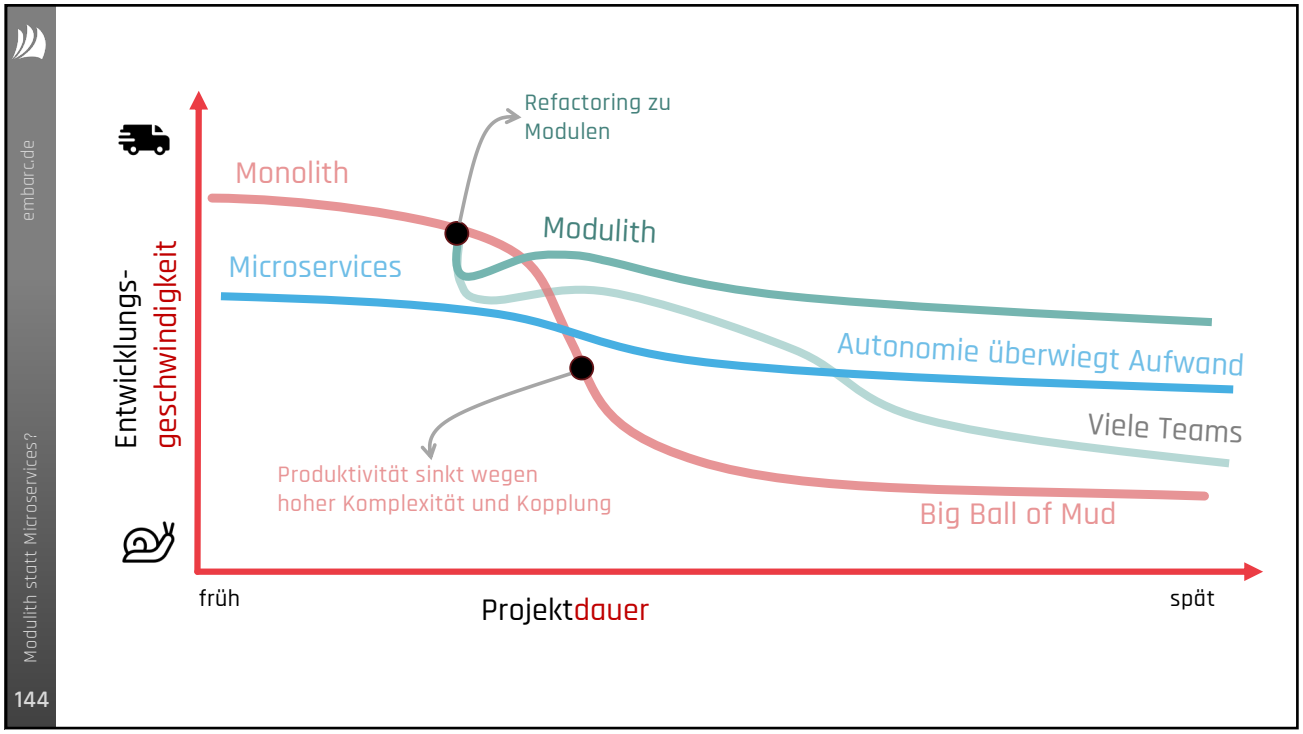
05.

Fazit + Ausblick

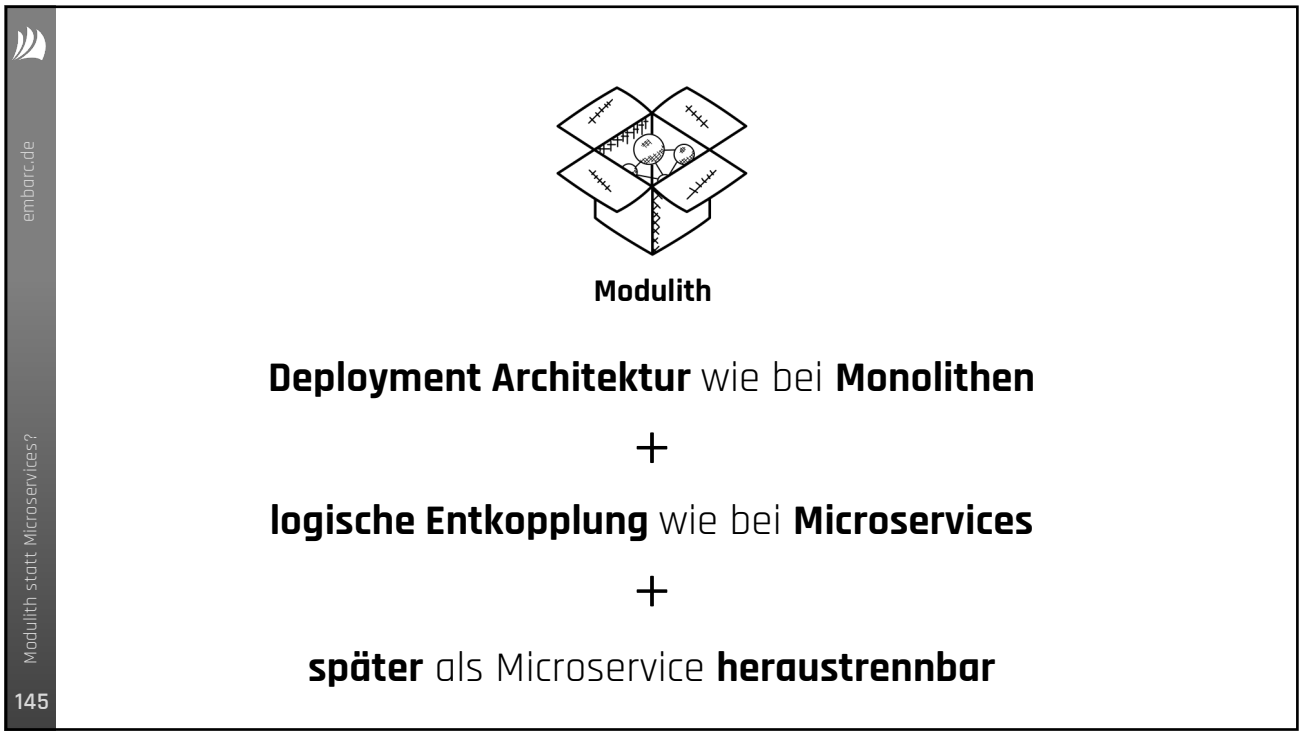
140



142


embarc.de
Modulith statt Microservices?
144
Entwicklungs-geschwindigkeit
Refactoring zu Modulen
Monolith
Modulith
Microservices
Autonomie überwiegt Aufwand
Viele Teams
Produktivität sinkt wegen hoher Komplexität und Kopplung
Big Ball of Mud
früh
Projektdauer
spät

144


embarc.de
Modulith statt Microservices?
145
Modulith
Deployment Architektur wie bei Monolithen
+
logische Entkopplung wie bei Microservices
+
später als Microservice heraustrennbar

145

Gegenüberstellung

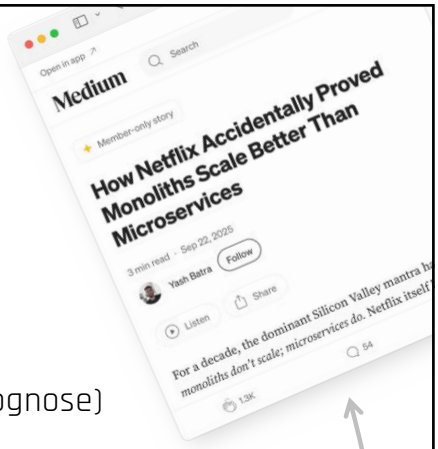
Monolith	Modulith	Microservices
ggf. schon ausreichend strukturiert (Layer, Ringe)	guter Kompromiss bei Modernisierung eines Big Ball of Mud	hohe Skalierungs-anforderungen wie Netflix
einfache Entwicklung und Betrieb	Modularisierung aufwändiger sicherzustellen	Time2Market
für kleine/unerfahrene Teams		
eine Code-Basis		

bevorzugt → bei Bedarf →

Zusammenfassung

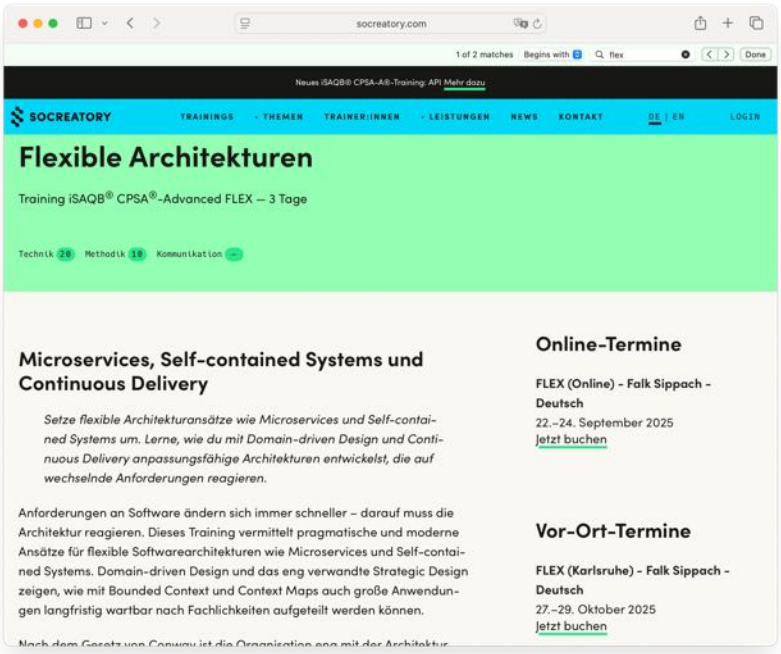
- Modulith: **weniger Komplexität, weniger Technologien**
- mehr auf die **Fachlichkeit konzentrieren**, Kapselung der Fachlichkeit in **isolierte Module**
- **übergreifende Refactorings** und einfaches Testen zw. Modulen
- **kein verteiltes System**, keine unnötige Remote-Kommunikation

Fazit



- Der Microservices **Hype ist vorbei**
- "Es **kommt** (wie immer) **drauf an**."
- Kelsey Hightower: "**Monoliths** are the **future**." (Prognose)
- Sam Newman: "**Microservices** should **never be** the **default choice**"
- **Monolith kann günstiger** (Amazon) und **schneller** (Stackoverflow) sein

149



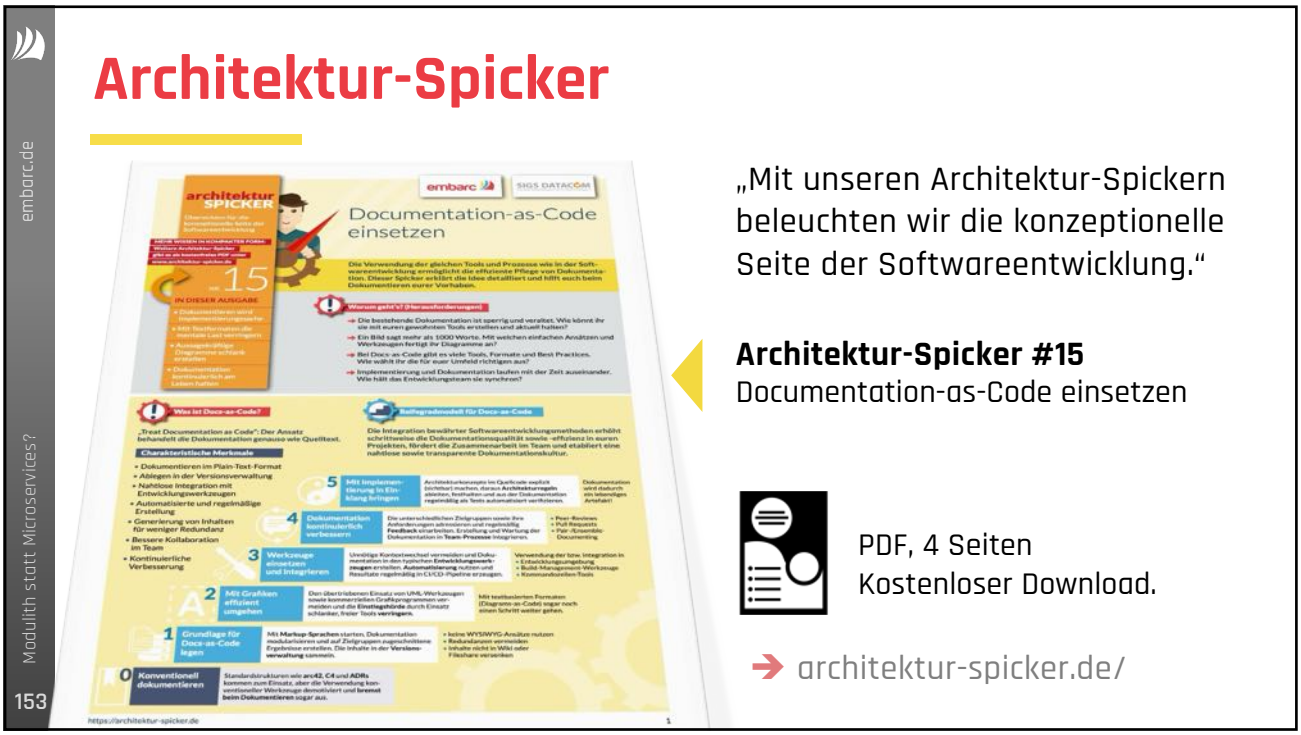
iSAQB ACCREDITED Training

➔ socreatory.com/de/trainings/flex

151

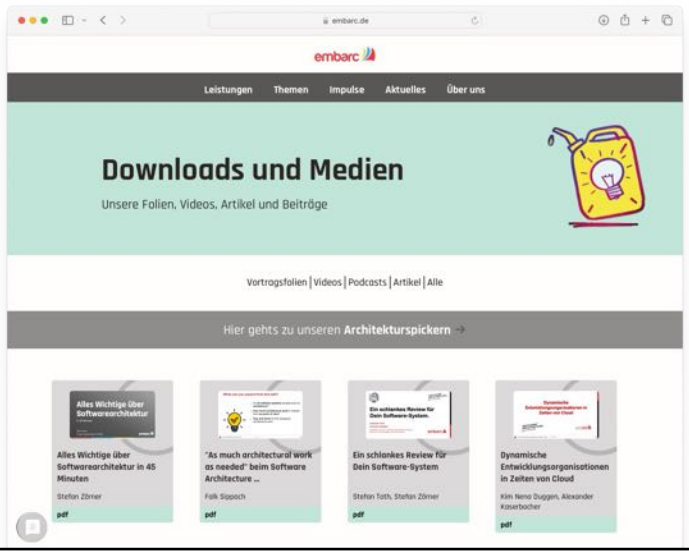

Architektur-
Punsch
Unser Line-Up
embarc.de/architektur-punsch-2025/
Kim Duggen
Oliver Zeigermann
Arnold Franke
Sven Waltmann
Cosima Laube
Alexander Kaserbacher
Felix Kammerlander
Falk Sippach
Alex Thurow
Stefan Zörner
Stefan Toth
mehr Infos...
Workshops, neue Perspektiven & Plaudern in lockerer
Punsch-Atmosphäre am 08. Dezember 2025!

152


embarc.de
Modulith statt Microservices?
153
Architektur-Spicker
architektur SPICKER
Documentation-as-Code einsetzen
„Mit unseren Architektur-Spickern beleuchten wir die konzeptionelle Seite der Softwareentwicklung."
Architektur-Spicker #15
Documentation-as-Code einsetzen
PDF, 4 Seiten
Kostenloser Download.
architektur-spicker.de/

153

Folien als PDF zum Download



➔ embarc.de/download/

154

Feedback & Fragen?

Wir freuen uns auf Fragen, Diskussionen, Anregungen!

155

Falk Sippach

Softwarearchitekt, Berater,
Trainer

falk.sippach@embarc.de

@sippsack

linkedin.com/in/falk-sippach

www.embarc.de

embarc

156