Von Clean Architecture zu evolutionärer Architektur

Falk Sippach

JAVAFORUM stuttgart

Stuttgart, 31. Juli 2024

embarc

0

Abstract

Clean Architecture als Unterstützung für evolutionäre Architektur

Softwaresysteme werden immer komplexer und die Anforderungen an sie wandeln sich ständig. Evolutionäre Softwarearchitektur versucht, sich kontinuierlich und effizient an sich verändernde Rahmenbedingungen, Anforderungen, Technologien und Umgebungen anzupassen. Die Ideen einer Clean Architecture stellen wichtige Grundlagen für evolutionäre Architektur dar. Denn durch die klare Trennung von Verantwortlichkeiten, die Unabhängigkeit von Bibliotheken/Frameworks, die erhöhte Testbarkeit und Flexibilität bietet Clean Architecture eine robuste Struktur, um die Weiterentwicklung einfacher zu ermöglichen.

In diesem Vortrag diskutieren wir, wie diese beiden Ansätze den Weg für eine nachhaltige Softwareentwicklung ebnen, insbesondere wie sie Risiken bei Änderungen reduzieren, die Portabilität fördern und die Zusammenarbeit im Entwicklerteam verbessern.

1

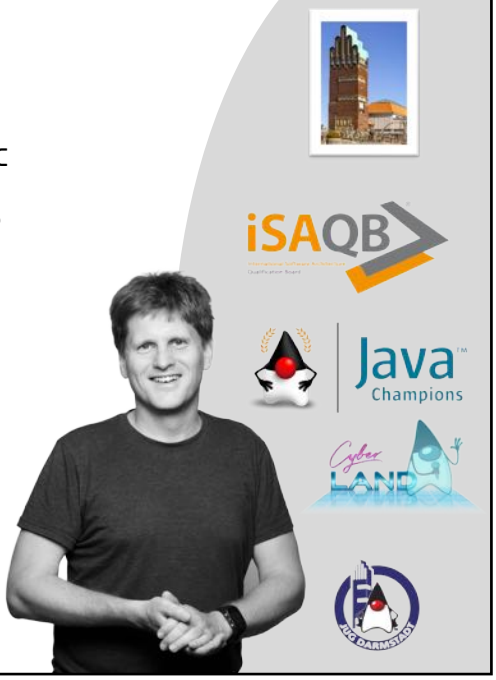
Falk Sippach

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

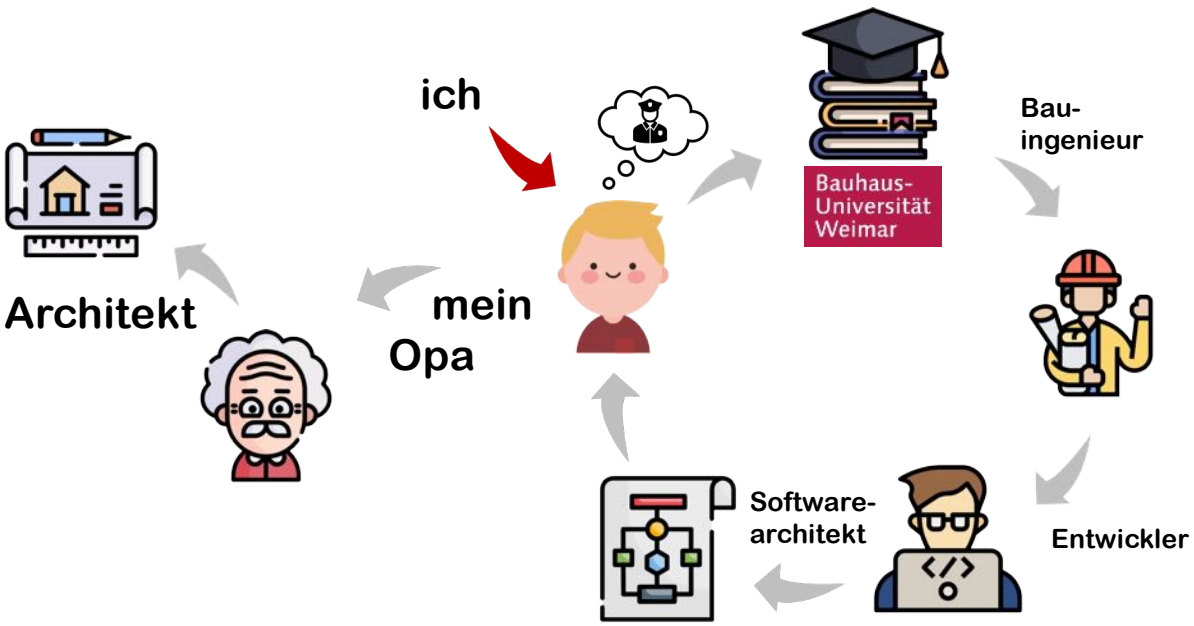
Schwerpunkte

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

embarc



2



3

Agenda

Was euch erwartet

01 Von Hexagonal zu Clean

02 Herausforderungen

03 Evolutionärer Ansatz

04 Clean & Evolutionär

05 Fazit + Ausblick

4

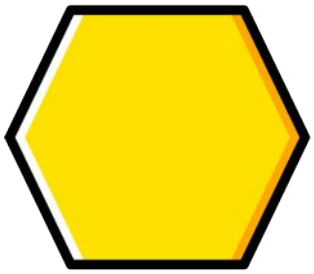
01.

Hexagonal zu Clean

Über welche (modernen) Architekturansätze reden wir nochmal?

5

Schalen: Von Ports and Adapters zu Clean Architecture



2005

2008

2012

6

Von Ports and Adapters zu Clean Architecture

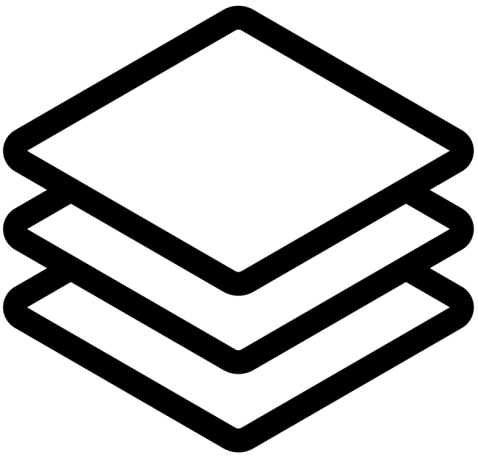
Alistair Cockburn

Jeffrey Palermo

Robert C. Martin

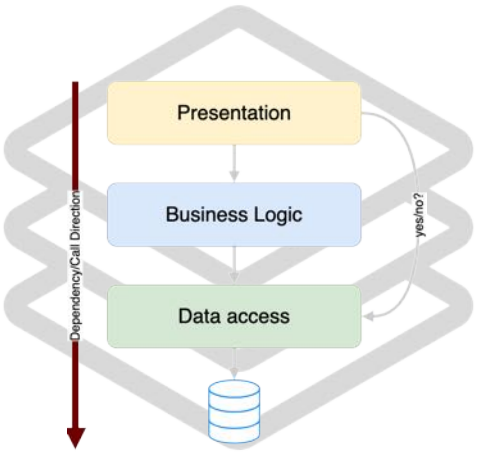
7

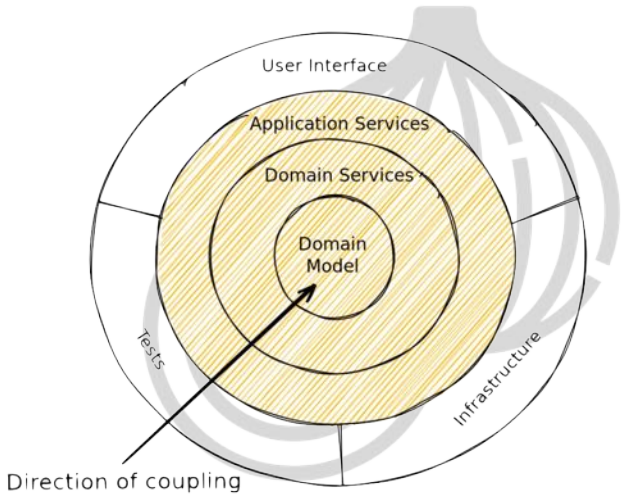
Layer vs. Rings



8

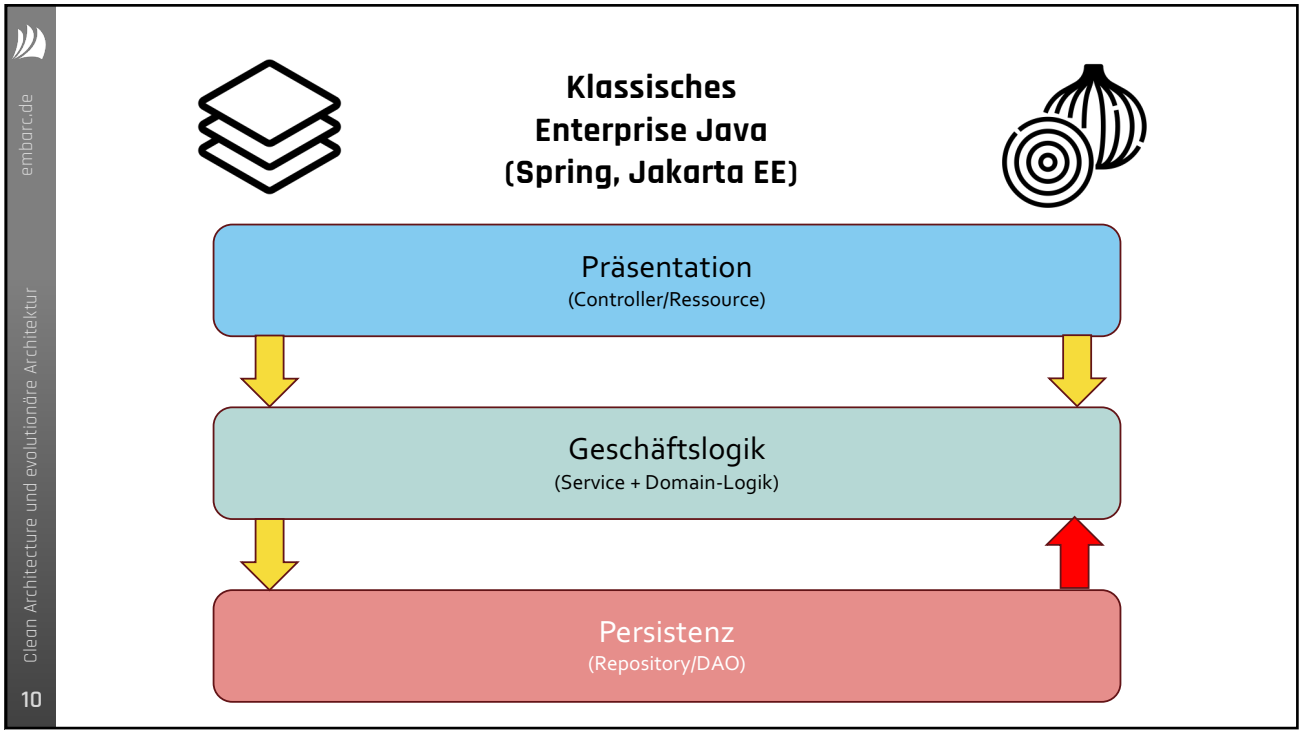
Layer vs. Rings



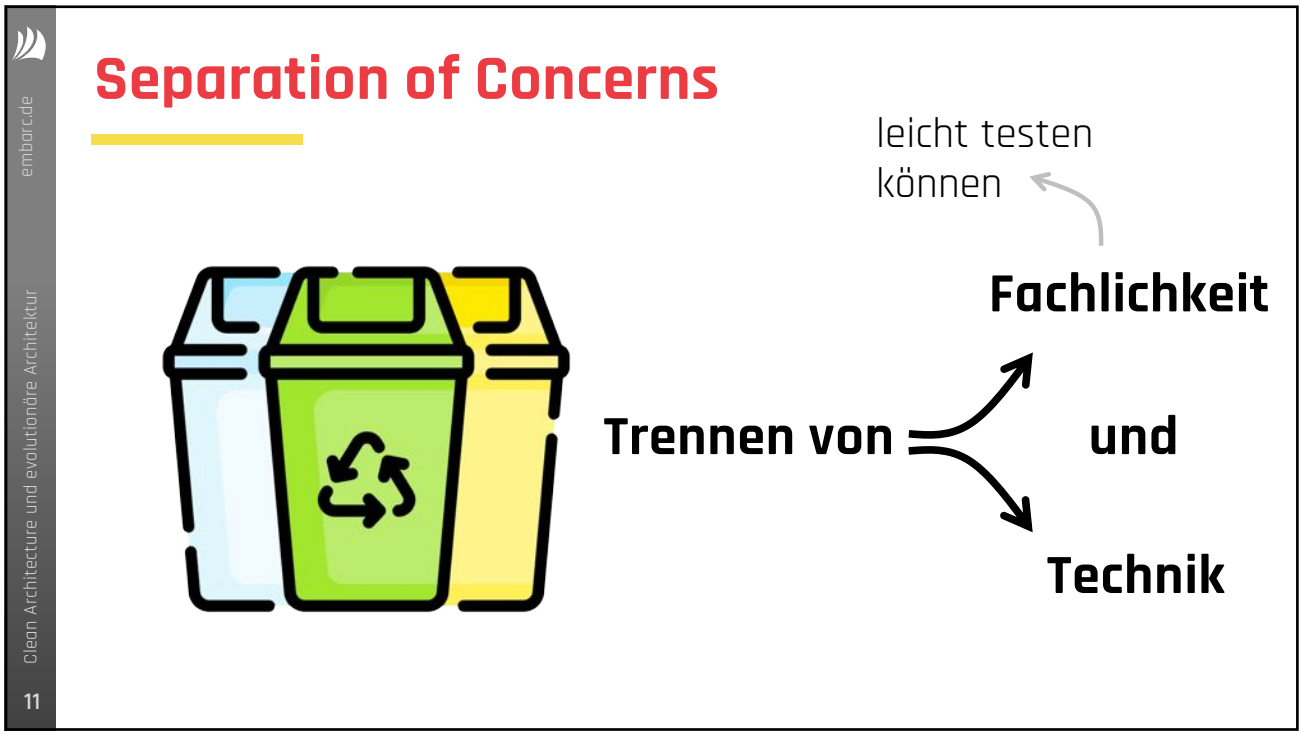


https://dev.to/barrymcauley/onion-architecture-3fgl

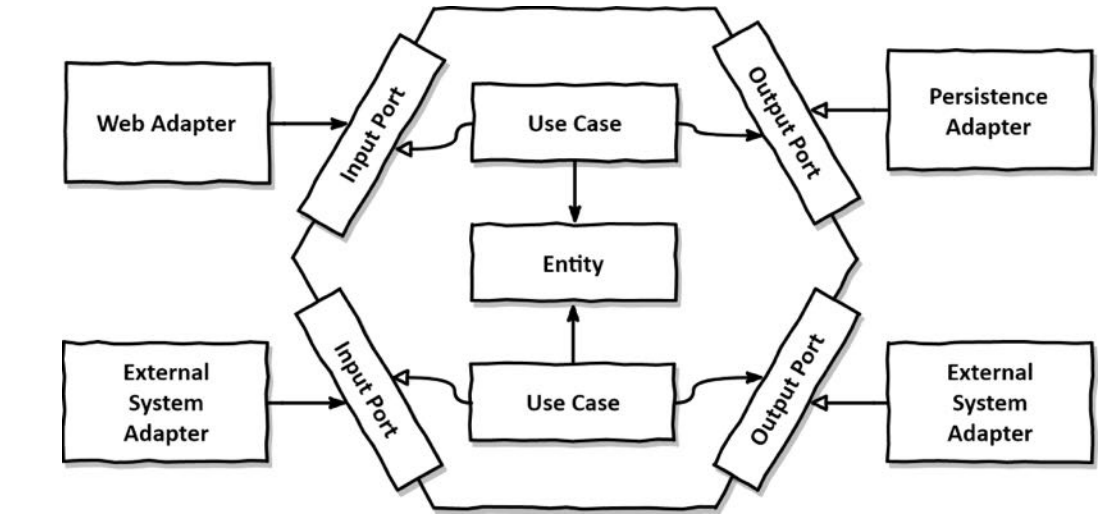
9


embarc.de
Clean Architecture und evolutionäre Architektur
10
Klassisches
Enterprise Java
(Spring, Jakarta EE)
Präsentation
(Controller/Ressource)
Geschäftslogik
(Service + Domain-Logik)
Persistenz
(Repository/DAO)

10

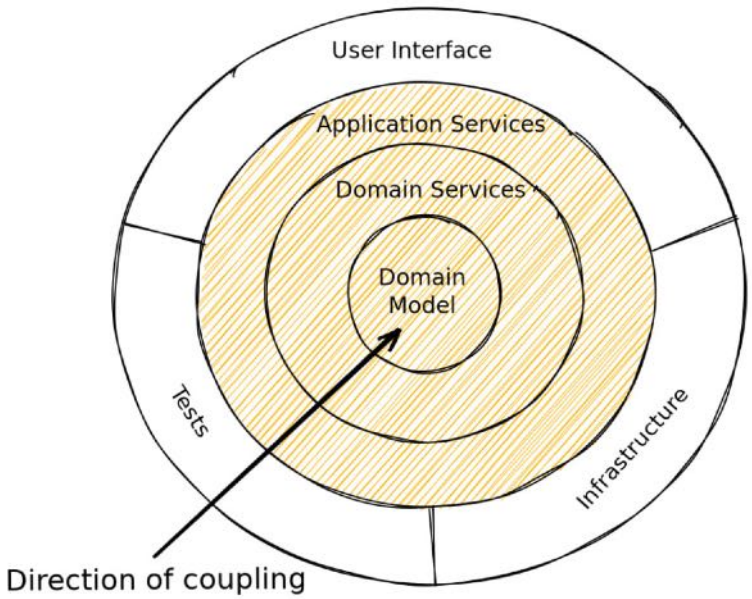

embarc.de
Clean Architecture und evolutionäre Architektur
11
Separation of Concerns
leicht testen
können
Fachlichkeit
Trennen von
und
Technik

11



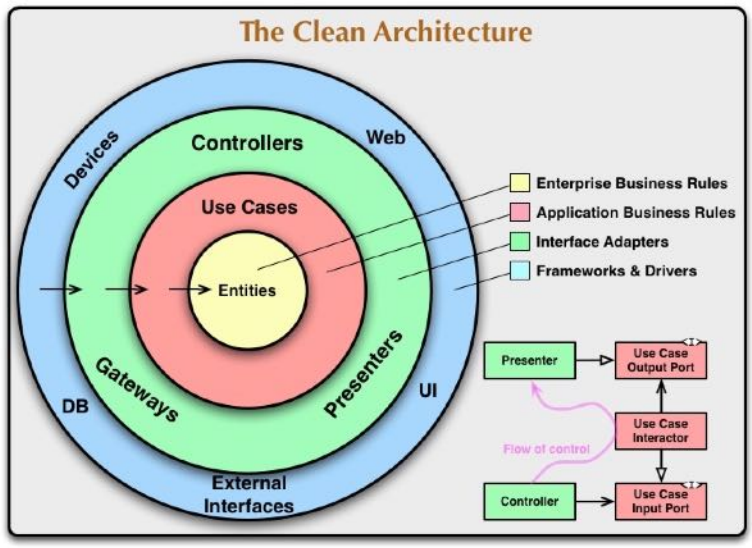
https://www.dineshonjava.com/a-practical-example-of-hexagonal-architecture-in-java/

14



https://dev.to/barrymcauley/onion-architecture-3fgl

15



https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

16

Vorteile

Isolation Fachlichkeit

Leichter ändern

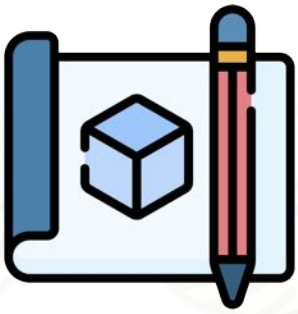
Besser testbar

Einfacher entwickeln

17

Herausforderung

Höhere Komplexität



Mehr Indirektionen (Performance, ...)

18

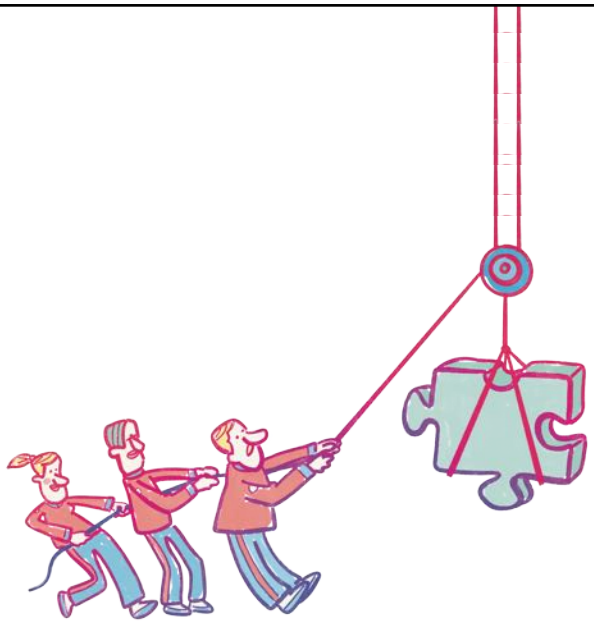
Annahme für diesen Vortrag

19

02.

Herausforderungen

Was macht Softwareentwicklung eigentlich komplex?



20

21

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“

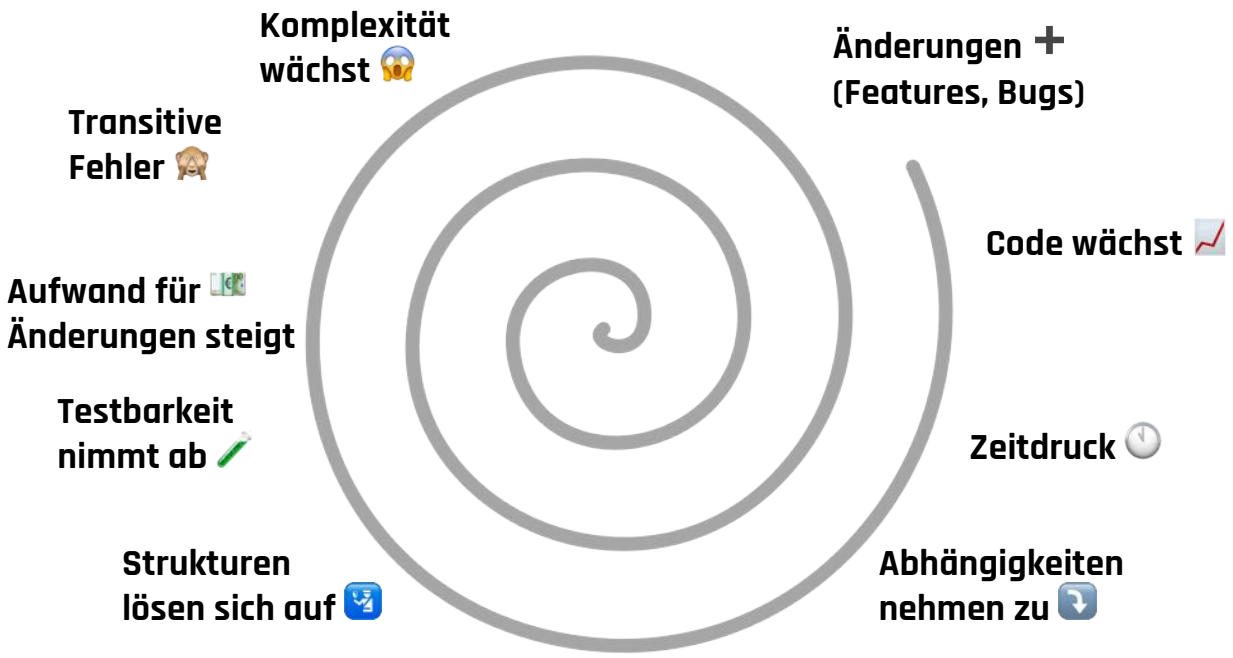
„Feature X wäre echt toll, der Konkurrent bietet das schon an“

„Entwickler:innen brauchen so lang“

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

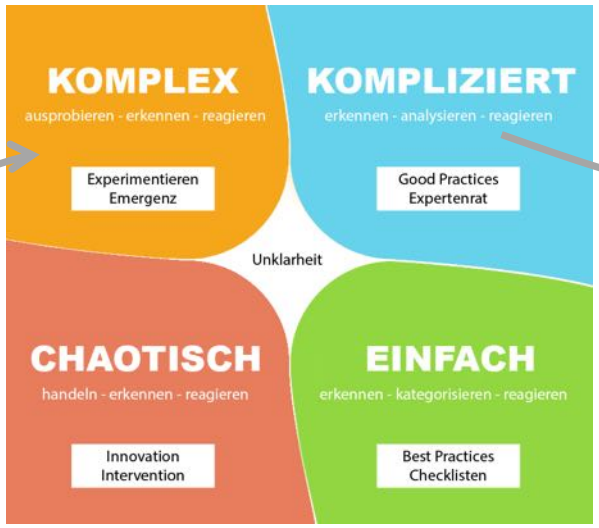
22

23



Cynefin-Framework

Softwareprojekte sind **heutzutage** eher **komplex**!

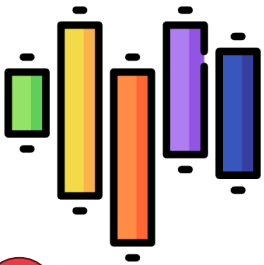


Aber wir versuchen nach **diesen Spielregeln** zu arbeiten!

Herausforderungen

- "einzige **Konstante** ist die **Änderung**"
- Software muss **anpassungs-** und **zukunftsfähig** sein
- **Architekturentscheidungen** sind grundlegend und **nicht** mehr so **leicht änderbar**

Ansätze



1

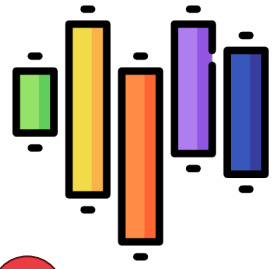
Modularisierung/ Vertikalisierung

Basis: Iterativ, inkrementelle Arbeitsweise

2

Kontinuierliche Durchführung von Experimenten

(1) und neue Herausforderungen



1

Modularisierung/
Vertikalisierung

= Microservices?

Komplexität: Verteilte Systeme sind komplex im Betrieb
Konsistenz: Konsistenz über Services hinweg ist schwierig
Evolution von APIs: Versionierung und rückwärtskompatible Änderungen
Latenz: Services sind nicht verfügbar, Latenz erhöht, …
Betrieb: Debugging und Monitoring ist komplex

Lösungsansatz führt zu neuen technischen Herausforderungen!

(2) Lasst uns experimentieren

Evolutionäre Architektur
und **Fitness Functions**
als ganzheitlicher Ansatz

2

Kontinuierliche Durchführung von Experimenten

03.

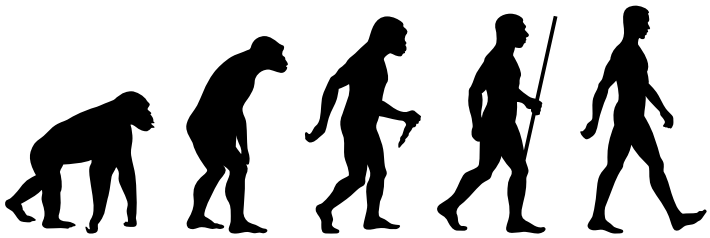
Evolutionärer Ansatz

Wie geht die Natur mit Herausforderungen um?

30

Die Natur hat Lösungen …

… für komplexe Probleme …



… durch kleinteilige, stetige Anpassung

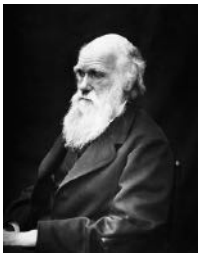
31

31

Charles Darwin: Natürliche Auslese

Diese Nachkommen sind nach dem Zufallsprinzip alle mit etwas unterschiedlichen Merkmalen ausgestattet. Einige Individuen sind dadurch besser an ihre Umwelt angepasst als andere, sie überleben und können sich vermehren.

So setzen sich ganz automatisch die vorteilhaften Merkmale durch. Darwin nennt dies "**Survival of the Fittest**". Damit meint er "**der am besten Angepasste überlebt**" und nicht etwa "der Stärkste".



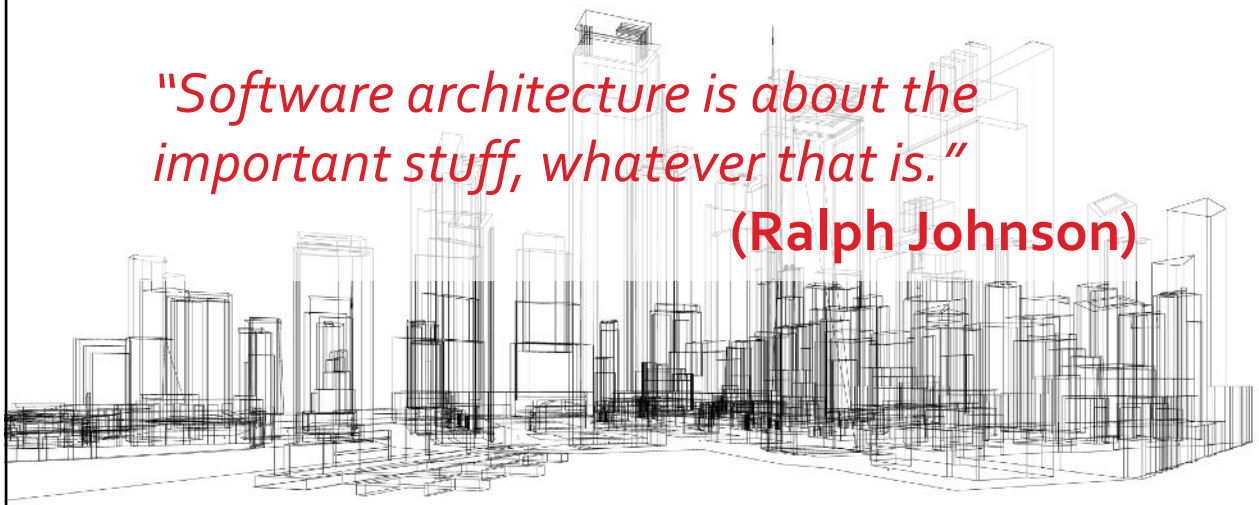
https://www.planet-wissen.de/natur/forschung/evolutionsforschung/pwiesurvivalofthefittestdiehauptthesenderevolutionstheorie100.html

Evolutionäre Algorithmen

- **Evolutionäre Algorithmen** sind Optimierungsverfahren, deren Funktionsweise von der Evolution natürlicher Lebewesen inspiriert ist.
- In Anlehnung an die Natur werden Lösungskandidaten für ein bestimmtes Problem künstlich evolviert.
- Evolutionäre Algorithmen finden meist **nicht die beste Lösung** für ein Problem, aber bei Erfolg eine **hinreichend gute** - in der Praxis bereits wünschenswert.

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

… und in der Architektur?

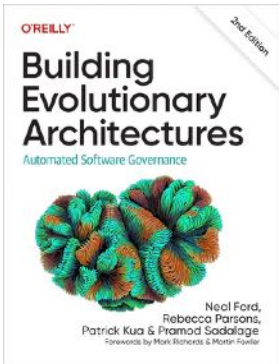


"Software architecture is about the important stuff, whatever that is."

(Ralph Johnson)

34

Übertragung auf Softwarearchitektur



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

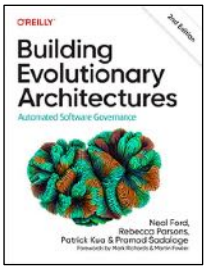
N. Ford, R. Parsons, P. Kua, P. Sadalage:
Building Evolutionary Architectures: Automated Software Governance
O'Reilly 2022

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

37

37

> Eine **Fitness Function** misst objektiv, wie gut eine Lösung die an sie gesetzten Ziele erreicht.
>
> - Building Evolutionary Architectures – Ford, Parsons, Kua, Sadalage



39

Gutes Feedback …

- wird **möglichst nah am realen Einsatz** gesammelt, z.B. nah an Nutzer*innen oder am physischen Endgerät
- erfolgt **regelmäßig** und **zeitnah**, idealerweise bei jedem Release
- ist **aussagekräftig**, entweder durch das (Nicht-)Erreichen eines Zielwerts oder durch Trends
- ist für alle relevanten Zielgruppen **zugreifbar**
- ist **von zentralen Zielen abgeleitet**

40

Einteilung / Kategorisierung

Art der Ausführung
- angestoßen
- kontinuierlich

Qualitätsmerkmal
- Wartbarkeit
- Zuverlässigkeit
- Effizienz
- ...

Art der Ergebniskontrolle
- o/1
- Wert
- Tendenz

Breite der Ausführung
- atomar
- holistisch

Ort und Zeitpunkt der Ausführung
- CI/CD
- Testumgebung
- Produktion

Gültigkeit der Tests
- Temporär
- Permanent

Kategorisierung = Dimensionen des "Lösungsraums"

41

Beispiele für Fitness Functions

atomar

Code-Metriken

Monitoring & Logging

kontinuierlich

angestoßen

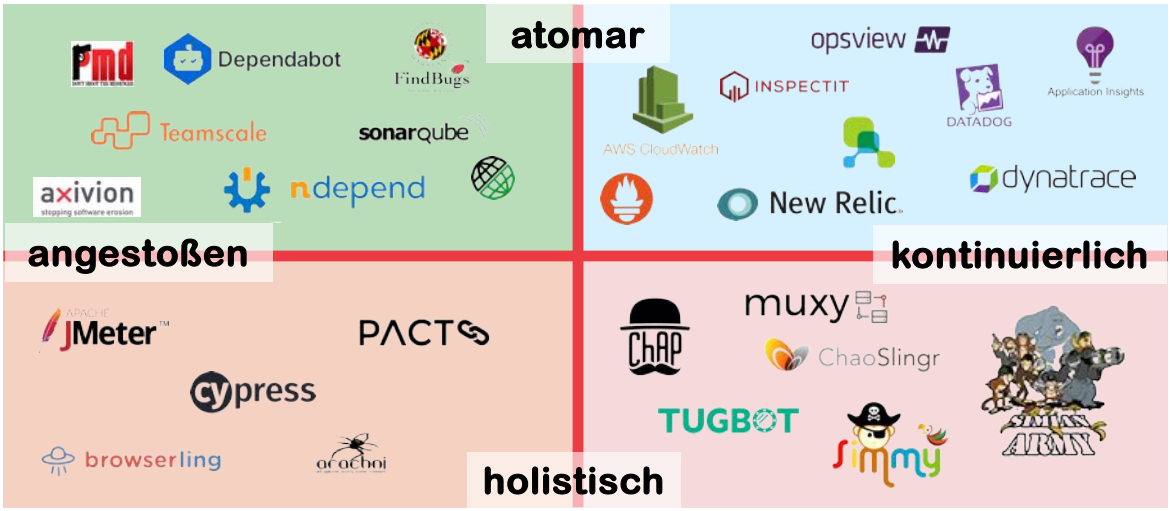
Software-Reviews

Chaos Engineering + Messung der Zuverlässigkeit

holistisch

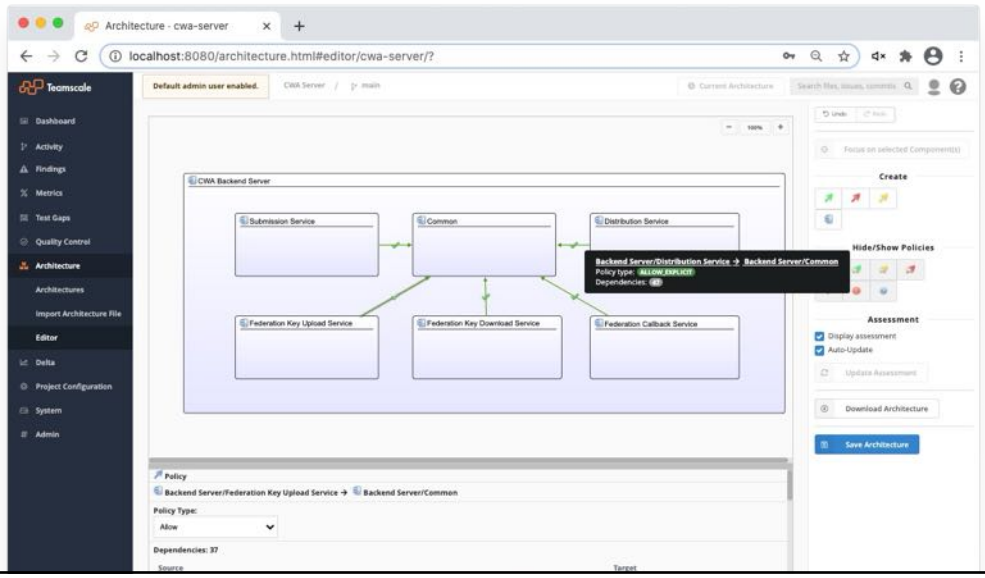
42

Fitness Function Quadrant - Tooling



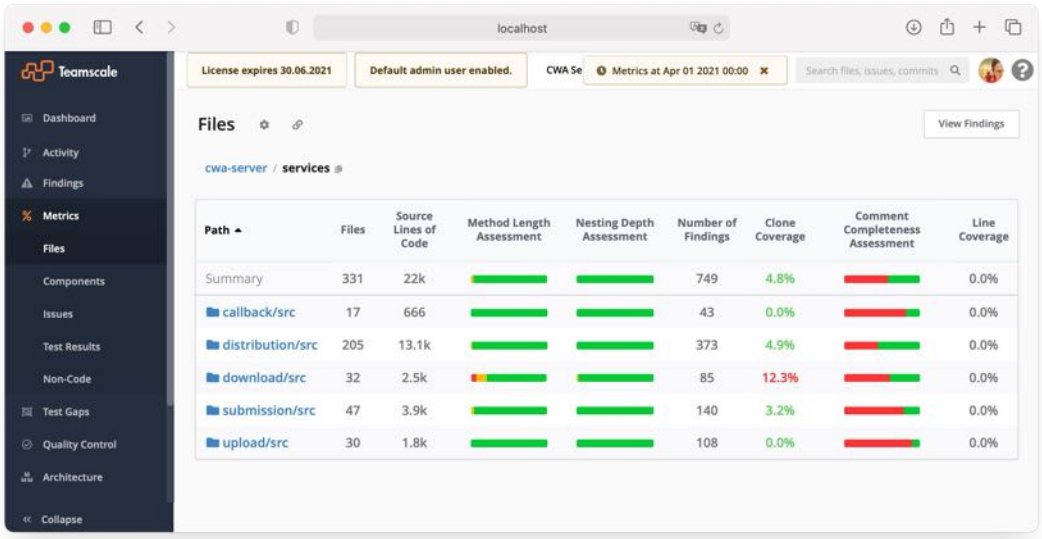
43

Architekturregeln



44

Metriken



45

Beispiele für Fitness Functions

Fragestellung – Wartbarkeit (Testbarkeit)
Ist unsere Testabdeckung ausreichend?

Zielgruppe:
Entwickler

Umsetzungsidee:
Regelmäßige Überprüfung Test-Coverage

Zeitpunkt:
Während jedem CI-Build
Während Nightly Integration-Test Build

Fitness Function(s):

- Testabdeckung (Unit) > 0.9
- Testabdeckung (Integration) > 0.6

46

Beispiele für Fitness Functions

Fragestellung – Zuverlässigkeit (Verfügbarkeit)

Funktioniert unsere Service auch bei höherer Latenz zuverlässig?

Zielgruppe:

Entwickler / Betrieb

Umsetzungsidee:

Regelmäßige Tests mit erhöhter Latenz

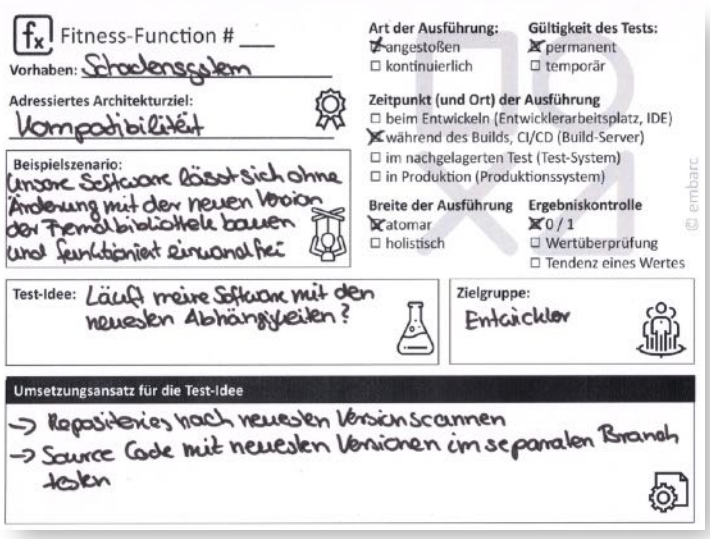
Zeitpunkt:

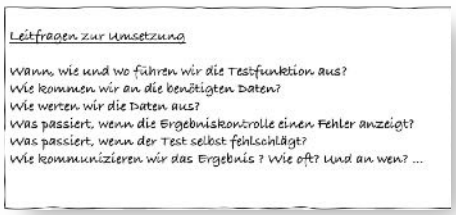
Während Nightly Integration-Test Build

Fitness Function(s):

- Integration-Test Fehler = 0% (Netzwerklatenz für 3rd Party Service = 5s)

Ein Steckbrief für Fitness-Functions

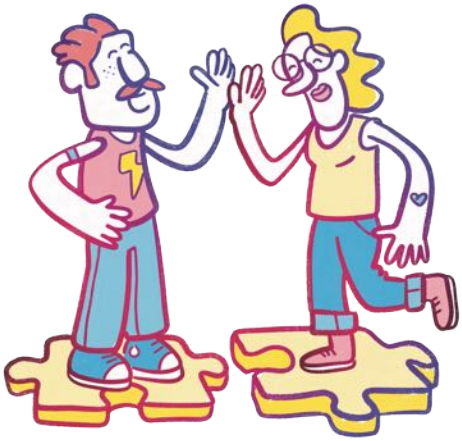




04.

Clean & Evolutionär

Wie hilft Hexagonal/Onion/Clean Architecture bei evolutionärer Architektur?



49

Erinnerung: Vorteile (Hexagonal/Onion/Clean)

Isolation Fachlichkeit

Leichter ändern

Besser testbar

Einfacher entwickeln

50

3 Ziele der hexagonalen Architektur

Steuerbar durch User, Anwendungen und automatisierte Tests (gleichermaßen)

Isoliertes Entwickeln und Testen (von DB, Infrastruktur und 3rd-Party-Systemen)

Keine Abhängigkeiten von Geschäftslogik zu Infrastruktur

https://alistair.cockburn.us/hexagonal-architecture/

51

3 Ziele der hexagonalen Architektur

Steuerbar durch User, Anwendungen und automatisierte Tests (gleichermaßen)

- klare Trennung Domain- und technische Logik (**Separation of Concerns**)
- **Treiben** der **Business-Logik** durch UI, Rest-APIs, Tests, …
- **Business-Logik treibt** Datenzugriffe, Mail-Versand, … (Dependency Inversion)

52

3 Ziele der hexagonalen Architektur

Isoliertes Entwickeln und Testen (von DB, Infrastruktur und 3rd-Party-Systemen)

- **Austauschbarkeit first** führt zu sauberen Strukturen
- **Fitness Functions leicht** an Schnittstellen (z. B. Ports) **andockbar**
- leichter **fachliche Prozesse überwachen** und steuern/testen

3 Ziele der hexagonalen Architektur

Keine Abhängigkeiten von Geschäftslogik zu Infrastruktur

- klar definierte Abhängigkeitsrichtung, Domain-Logik hängt nicht von Technik ab (**Dependency Inversion**)
- saubere Modulgrenzen (Schnittstellen)
- **leichte Austauschbarkeit** von technischer Logik (Fachlogik ist länger stabil, Technologien ändern sich)

Entwurfsprinzpien

Single Responsibility Principle

Open Closed Principle

Liskov Substitution Principle

Interface Segregation Principle

Dependency Inversion Principle

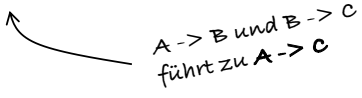
Information Hiding Principle

High Cohesion & Low Coupling

Abhängigkeitsprinzipien

Abhängigkeiten sollten immer **in Richtung des stabileren Elements** (Business-Logik) zeigen.

Abhängigkeiten sind **transitiv** (nicht alles soll von allem abhängig).



Vermeide **Abhängigkeitszyklen**.

05.

Fazit und Ausblick

Zusammenfassung, Canvas und weitere Hinweise

57

Ein Spannungsfeld …

Evolutionäre Architektur versucht den **Widerspruch aufzulösen**, dass

a) **Architekturentscheidungen schwer zu ändern** sind
b) Software, die lange leben soll, **sich Veränderungen** im Umfeld **anpassen muss**

Fragen

Sind wir mit unseren Entscheidungen noch **auf dem richtigen Weg**?
Erfordern Veränderungen das Revidieren der Architektur?

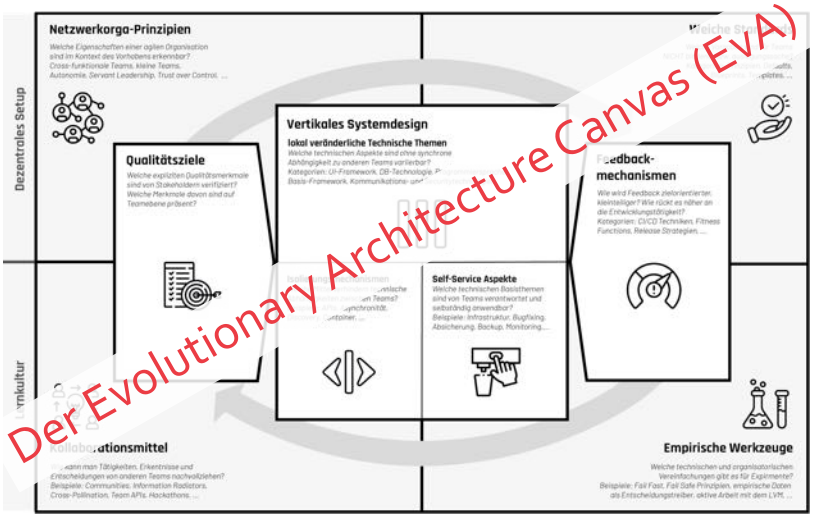
58

Ist evolutionär die Lösung?

> Eine **Evolutionäre Architektur** ermöglicht Produktentwicklung, die durch inkrementelle, emergente Praktiken und regelmäßiges, qualitatives Feedback in komplexen Umfeldern dynamisch agiert.

Geschwindigkeit trotz Komplexität

Wo steht Euer Projekt/Produkt?



Der Evolutionary Architecture Canvas (EvA)

Selbstreflektion zum Ausfüllen.

Sprecht uns auch gern **an!**

https://www.embarc.de/themen/evolutionaere-architektur/

Weitere Infos

https://www.embarc.de/microservices-evolutionaer-oop24/

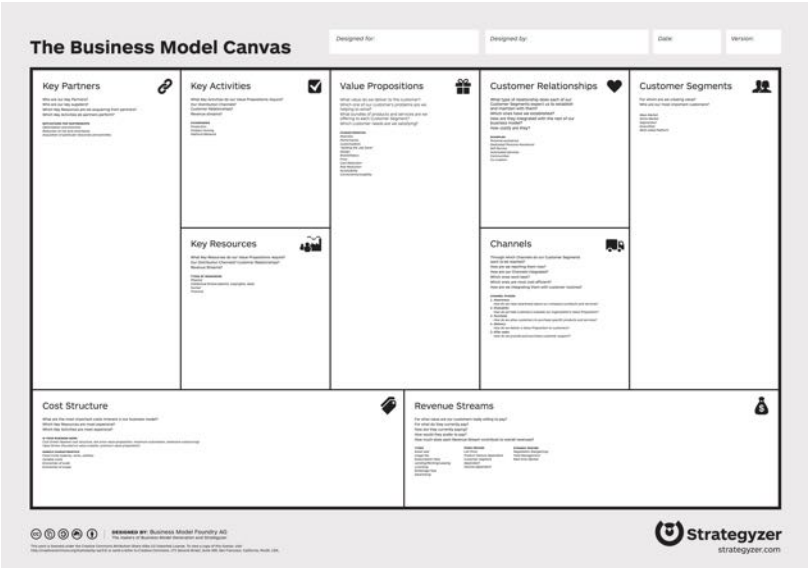
61

Was leistet ein Canvas?

NICHT einfach nur Doku...

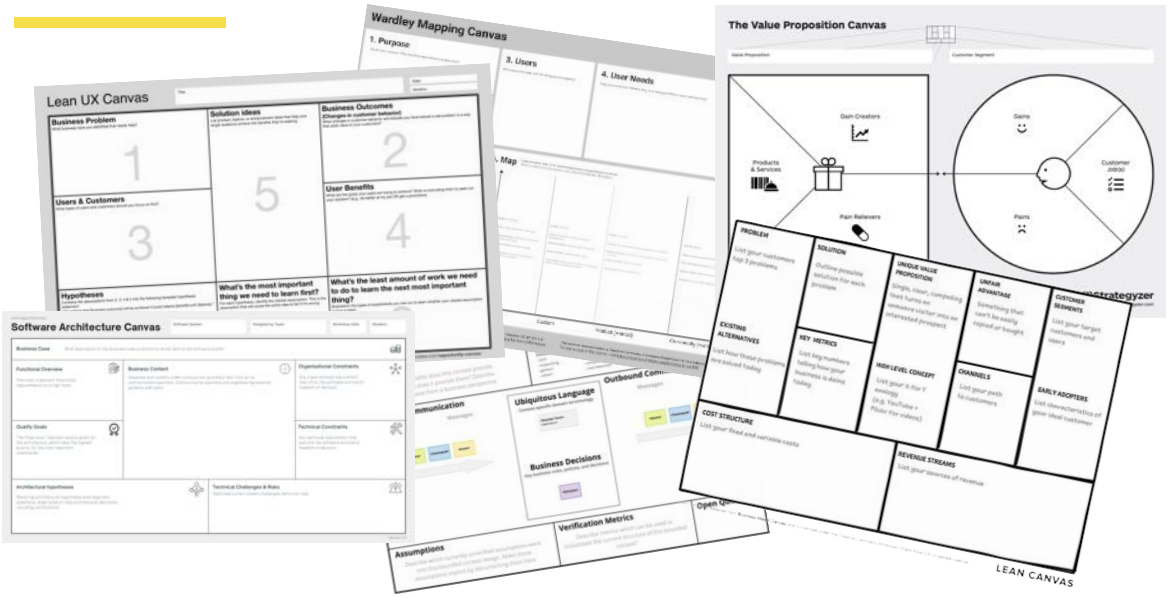
Arbeitswerkzeug, das:

- **Verständnis** erzeugt
- **Diskussion** fördert
- **Kreativität** fordert
- **Analyse** ermöglicht



62

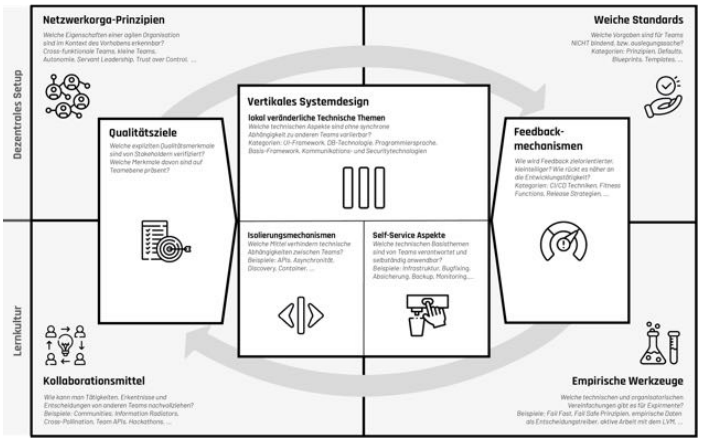
Ein beliebtes Instrument…



64

Der Evolutionary Architecture Canvas (EvA)

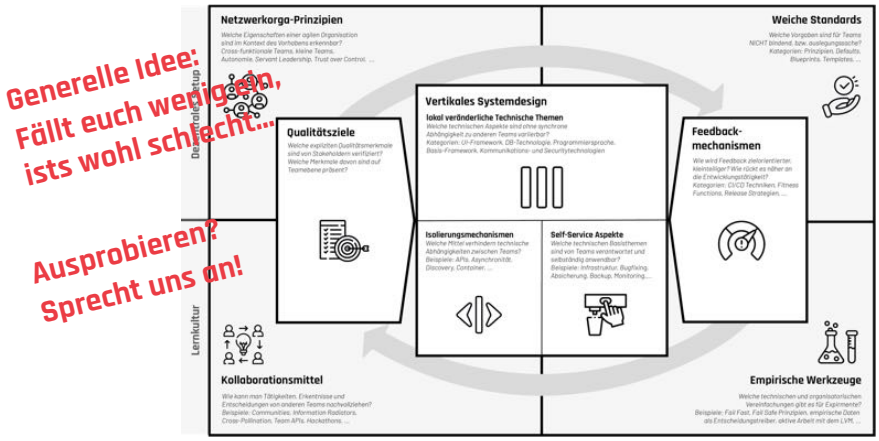
- Die **Vorteile** von z. B. Microservices sind **nur mit guter Orga & Methodik** einlösbar
- Der **Canvas zeigt** die wichtigsten **Aspekte** auf und **macht sie evaluierbar**



65

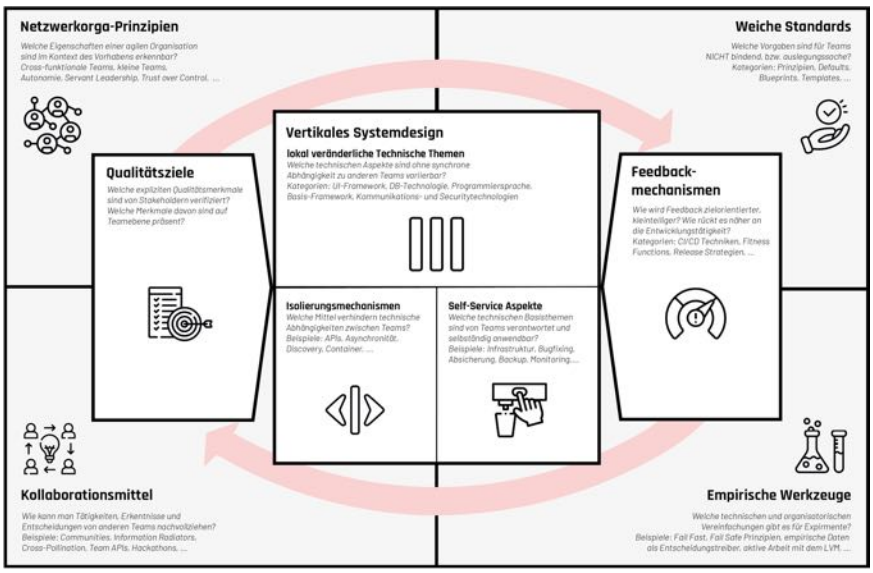
Der Evolutionary Architecture Canvas (EvA)

- Die **Vorteile von Microservices** sind **nur mit guter Orga & Methodik** einlösbar
- Der **Canvas zeigt** die wichtigsten **Aspekte** auf und **macht sie evaluierbar**



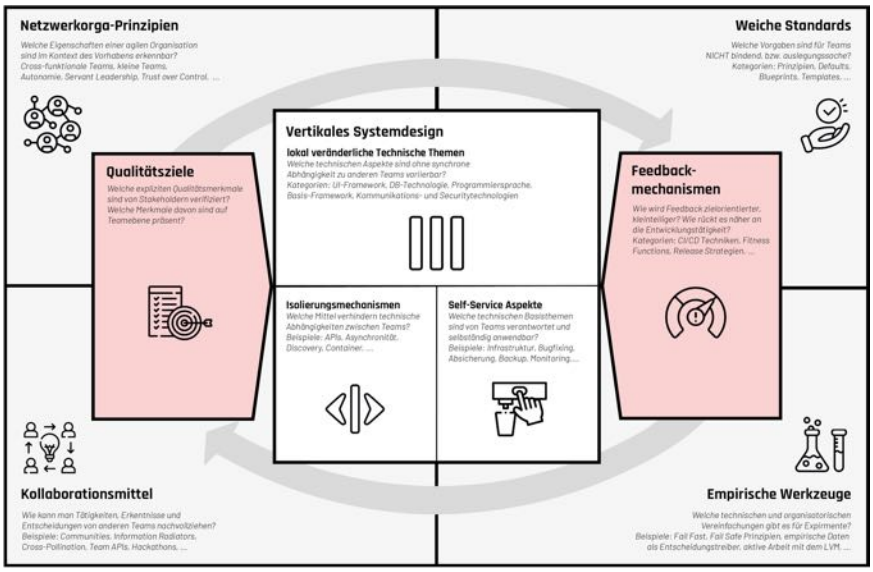
66

1 inkrementelle Praktiken



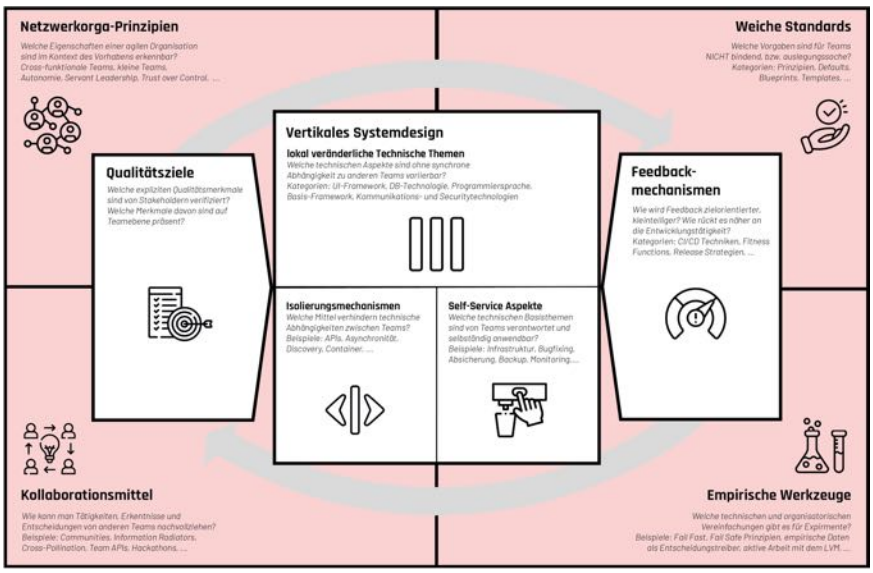
67

2 regelmäßiges, qualitatives Feedback



68

3 emergente Praktiken

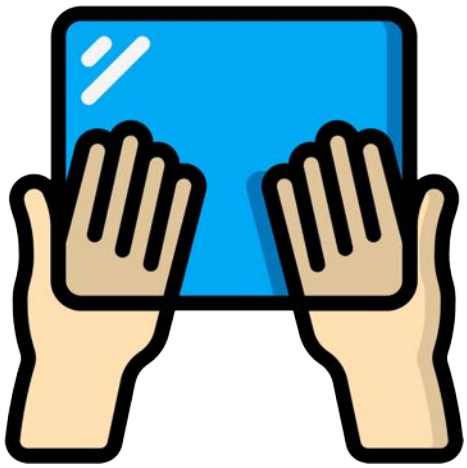


69

It's all about transparency …

Architektur gemeinsam erarbeiten

Architektur protokollieren/ festhalten

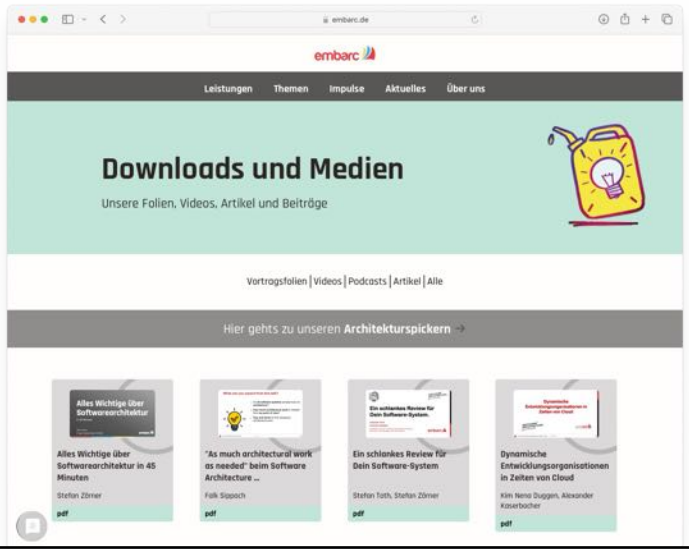


Architektur: Soll-Ist-Abgleich

Wissen verteilen

70

Folien als PDF zum Download

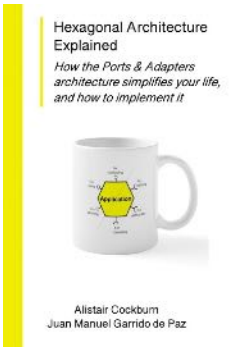


➔ embarc.de/download/

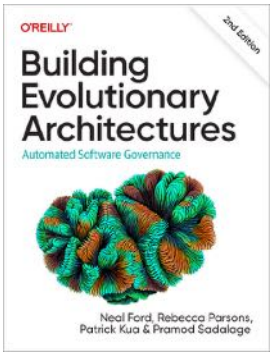
71

Vertiefen …

https://reflectoring.io/book/



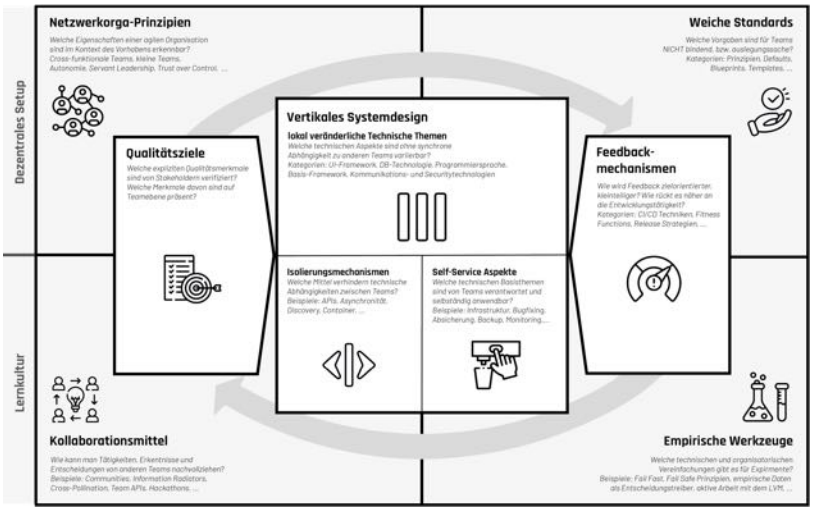
https://store7710079.company.site/Hexagonal-Architecture-Explained-p655931616



https://www.thoughtworks.com/en-es/insights/books/building-evolutionaryarchitectures-second-edition

72

Evolutionary Architecture Canvas (EvA)



Selbstreflektion zum Ausfüllen.

Sprecht uns auch gern **an!**

https://www.embarc.de/themen/evolutionaere-architektur/

73

Architektur-Spicker

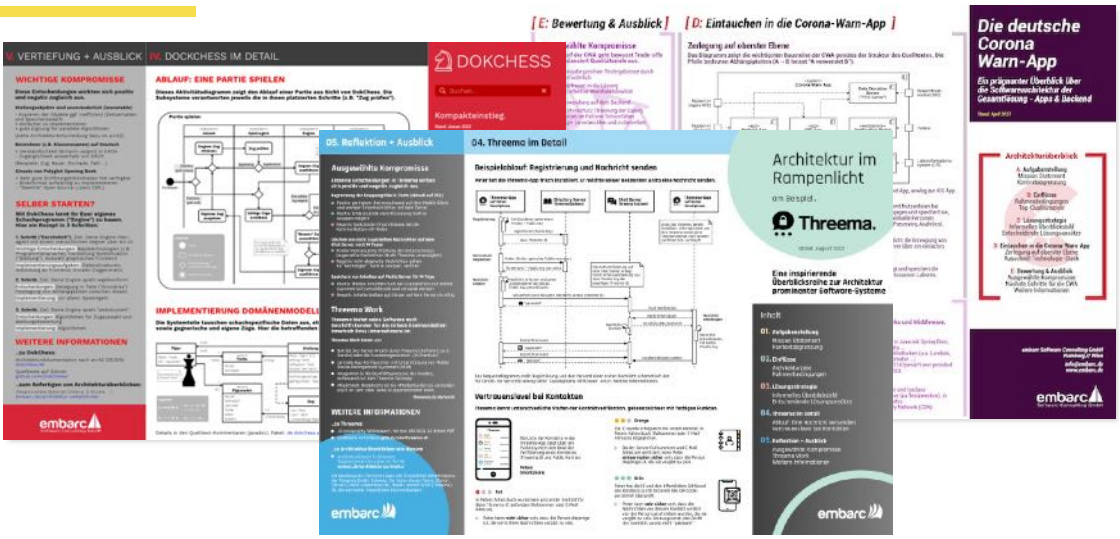
„Mit unseren Architektur-Spickern beleuchten wir die konzeptionelle Seite der Softwareentwicklung."

Architektur-Spicker #1
Der Architekturüberblick

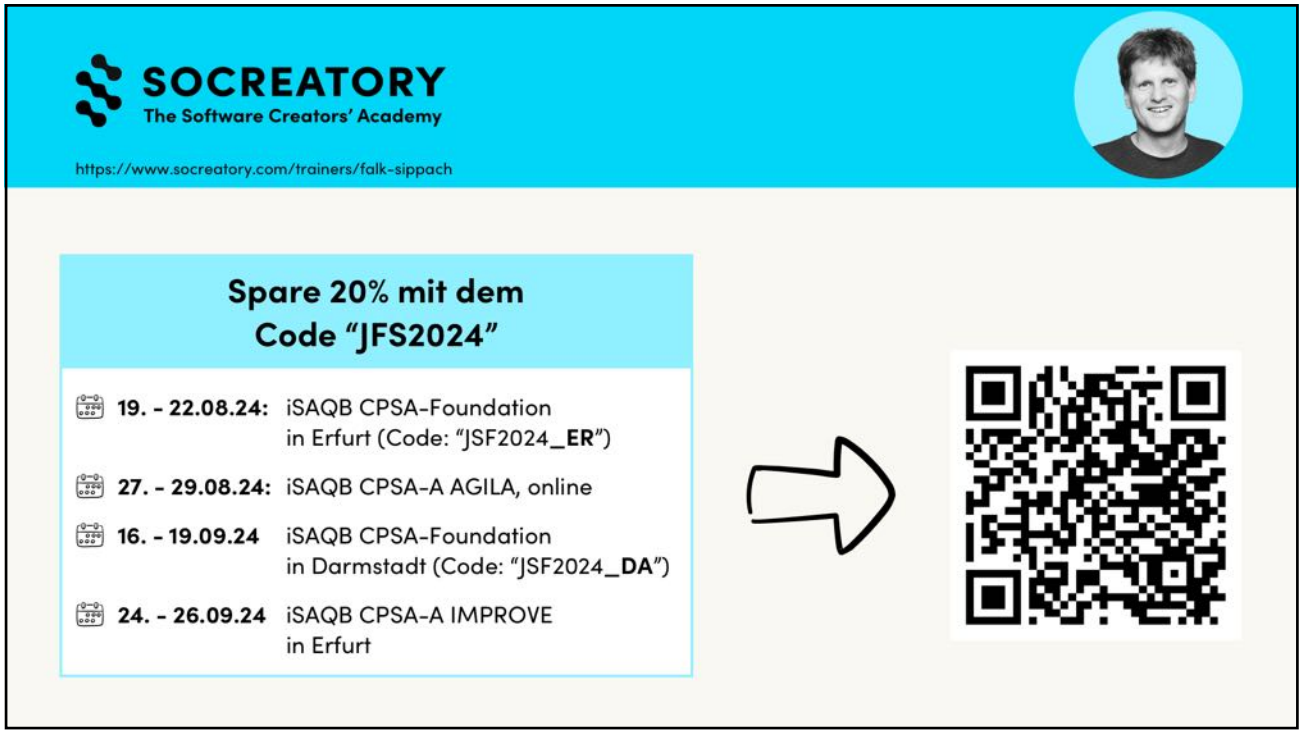
PDF, 4 Seiten
Kostenloser Download.

➔ embarc.de/architektur-spicker/

Architekturüberblicke als Vorlage



➔ embarc.de/architektur-ueberblicke/



76

Feedback & Fragen?

Wir freuen uns auf Fragen, Diskussionen, Anregungen!

78

Vielen Dank.

Ich freue mich auf Eure Fragen!

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

embarc

79