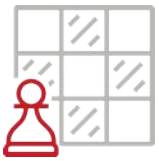
Microservices & Makro-Architektur

Drei zentrale Entwurfsfragen bei vertikalen Anwendungsarchitekturen



STEFAN ZÖRNER // EMBARC

OOP-Konferenz, München

Mittwoch, 05.02.2020

1

Stefan Zörner

- Softwareentwickler + -architekt bei embarc in Hamburg
- Vorher oose, IBM, Mummert + Partner, Bayer AG, …

Schwerpunkte:

- Softwarearchitektur (Entwurf, Bewertung, Dokumentation)
- Java Technologien

Stefan.Zoerner@embarc.de

@StefanZoerner

➔ xing.to/szr

3

Agenda

1 Heilsversprechen reloaded

2 Themen für die Makro-Architektur?

3 Thema I

4 Thema II

5 Thema III

6 Weitere Informationen und Fazit

4

Agenda

1 **Heilsversprechen reloaded**

2 Themen für die Makro-Architektur?

3 Thema I

4 Thema II

5 Thema III

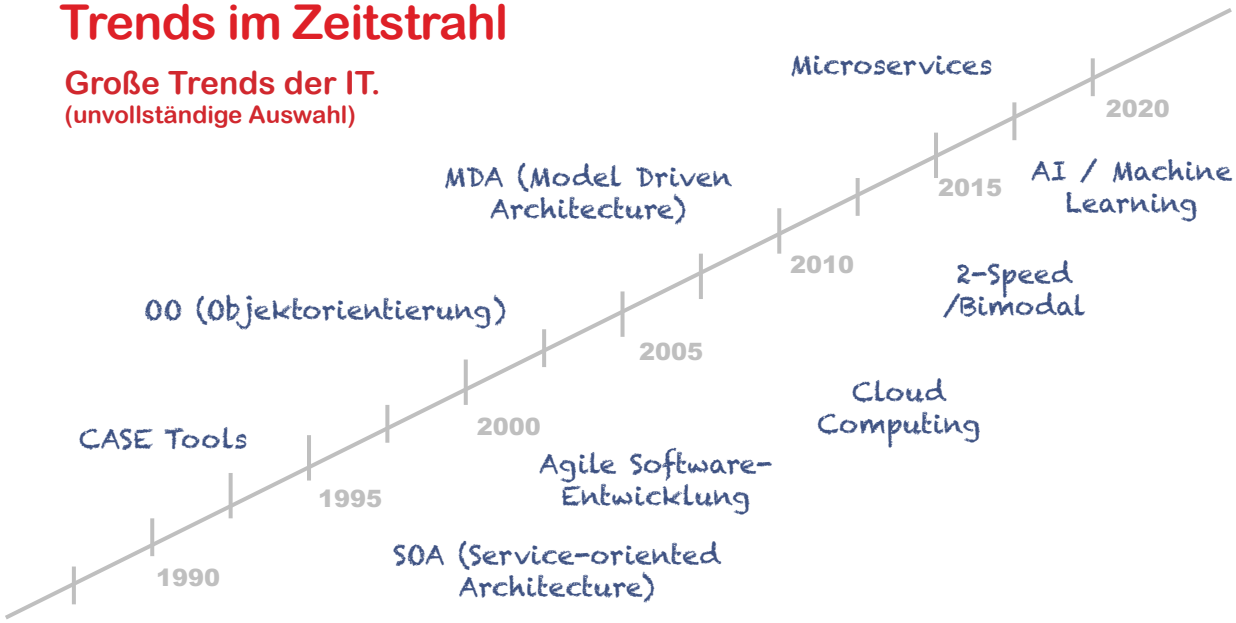
6 Weitere Informationen und Fazit

1

5

Trends im Zeitstrahl

Große Trends der IT.
(unvollständige Auswahl)



6

Zwei Heilsversprechen

Diese Trends versprechen zwei Formen des Glücks ...

1

Kosteneffizienz

Kosten **sparen** in Entwicklung und Betrieb, z.B. durch ...

- effizientere Werkzeuge und Methoden
- Wiederverwendung
- Einsparungen in Personal, Lizenzen, ...

2

Flexibilität

Schnell auf Veränderungen **reagieren** können, z.B. auf ...

- neue fachliche Anforderungen
- technologische Trends
- Schwankungen in der Last

7

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

8

Das Versprechen einhalten …

Wie versuchen Microservices leicht auf Änderungen zu reagieren?

Neue fachliche Anforderungen schnell umsetzen

- Kleinteiligkeit („Micro“), als Gegenmodell zum Monolith
- Lose Kopplung der Services, leichte Austauschbarkeit

Technologische Trends schnell aufgreifen

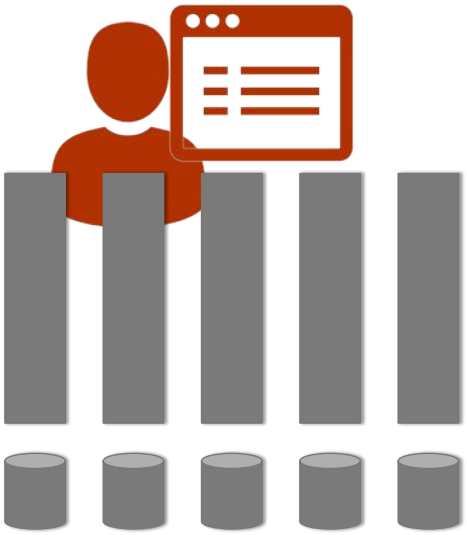
- Hoher Freiheitsgrad bei Technologieauswahl
- Services mit unterschiedlichem Technologie-Stack möglich

Schwankungen in der Last auffangen

- Einzelne Services einzeln skalierbar
- Services schnell start- und wegwerfbar

9

Ein Spannungsfeld



***Eine* Anwendung**
So sollte es sich für den Benutzer auch anfühlen.

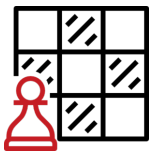
Technologische Vielfalt in den "Vertikalen"

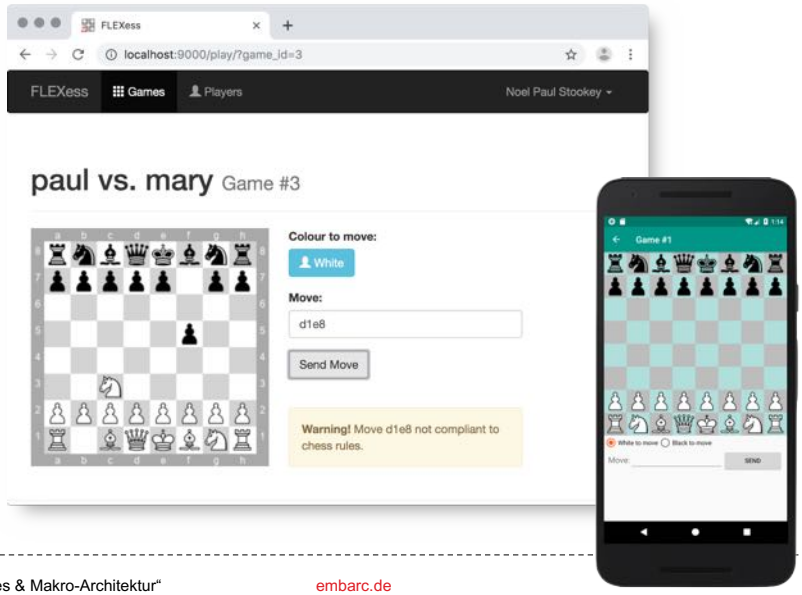
- Paradigmen
- Programmiersprachen
- Bibliotheken und Frameworks
- ...

z.B. in der Persistenz (polyglott)

10

FLEXess – Ein Beispiel

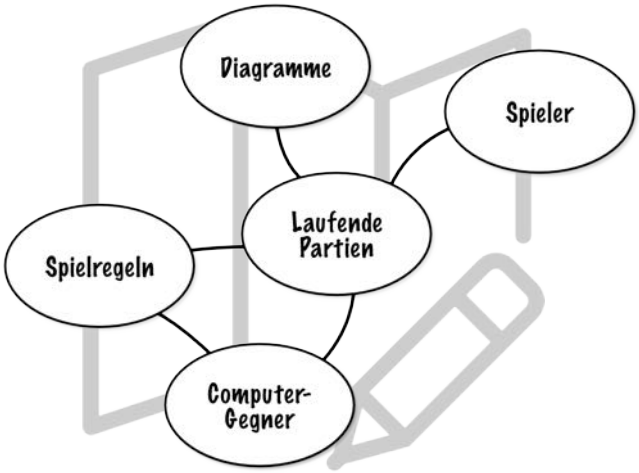




11

FLEXess – Ein Beispiel

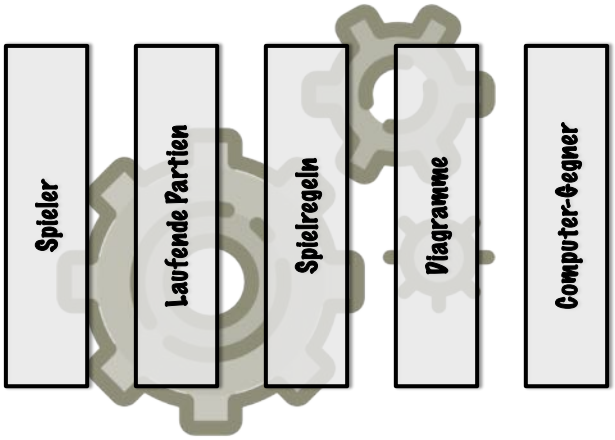
Erste Context Map für die Domäne einer Online-Schachplattform.



12

FLEXess – Ein Beispiel

Daraus Kandidaten ableiten für Vertikalen (Services) ...



13

Agenda

1 Heilsversprechen reloaded

2 **Themen für die Makro-Architektur?**

3 Thema I

4 Thema II

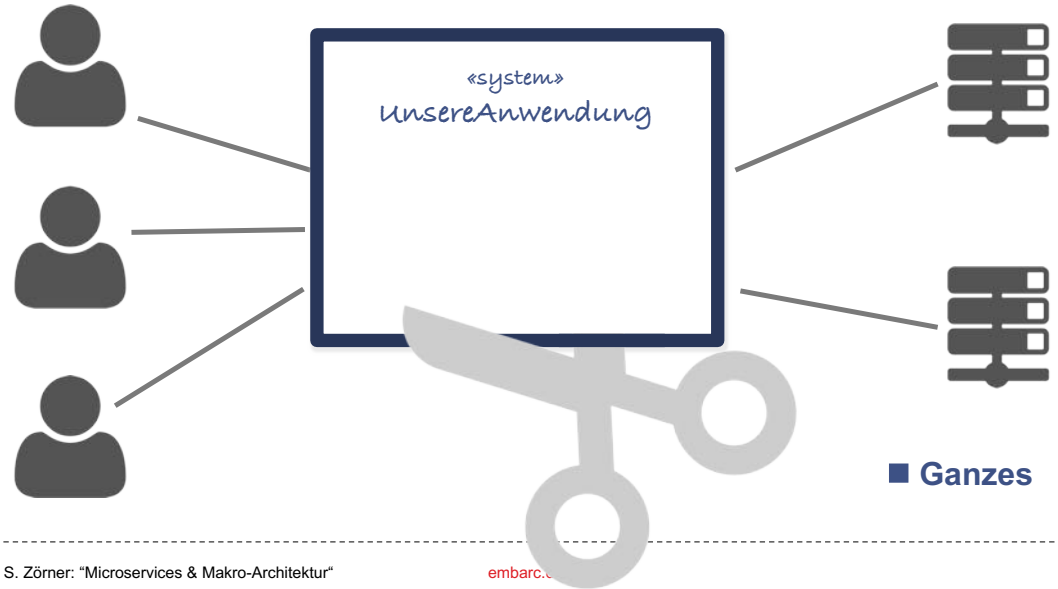
5 Thema III

6 Weitere Informationen und Fazit

2

14

Zu allererst: Den Kontext festlegen



15

Einsatzgebiete Makro vs. Mikro

Worüber reden wir?

- Eine **Anwendung,** die in **Module, Services, Komponenten** ... zerfällt?
  (Self-contained Systems, Microservices …)

Farb-Code:
- **Ganzes**
- **Bestandteile**

- Eine **Produktfamilie** oder -linie, die in **Varianten** oder Ausprägungen zerfällt?
- Eine **Systemlandschaft**, die in **Anwendungen** zerfällt?
- ...

Makro- vs. Mikroarchitektur

Entscheidungen ...

Welche Aspekte sind für alle Bestandteile gleich?

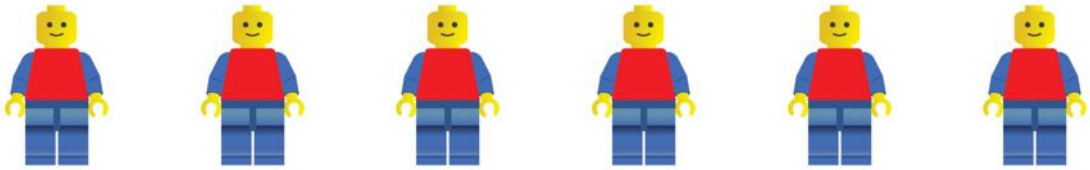
(Makro, Standardisierung)

Wo ist Spielraum?

(Mikro, Individualisierung)

Vorteile von Standardisierung

- Entwickler wechseln leicht zwischen Teams und Projekten
- Konzentration auf Applikationsspezifika (z.B. Fachlichkeit) leichter möglich
- Wiederverwendung von technischen Lösungen
- Vermeidung von Fehlern in kritischen Bereichen durch erprobte Konzepte



18

Vorteile von Individualisierung

- Einsatz optimaler Lösungen für spezifische Probleme möglich
- Neue Trends lassen sich schneller aufgreifen
- Fehlentscheidungen haben geringere Relevanz
- Geringere Abhängigkeit von einzelnen Lieferanten ...

19

Übergreifende Themen (Kandidaten)

20

Themenkomplex: Benutzer und Fremdsysteme

21

Themenkomplex: „Unter der Haube“

22

Themenkomplex: Entwicklung + Weiterentwicklung

23

Themenkomplex: Zielumgebung und Betrieb

24

Und jetzt?

25

Und jetzt?

Entscheidungen ...

Zu bestimmten Themen gibt es Best Practices, Industrie-Standards, …

Zu einigen Themen gibt es meiner Erfahrung nach regelmäßig Diskussionen „ob und wenn ja wie man das zentral macht ...“

Drei dieser Themen habe ich jetzt mal rausgepickt, um genau das zu diskutieren ...




Agenda

1 Heilsversprechen reloaded

2 Themen für die Makro-Architektur?

3 **Thema I: Die UI-Frage**

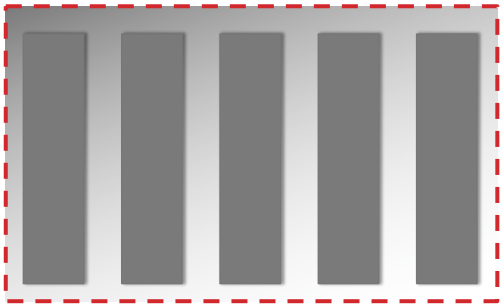
4 Thema II

5 Thema III

6 Weitere Informationen und Fazit

3

Die UI-Frage



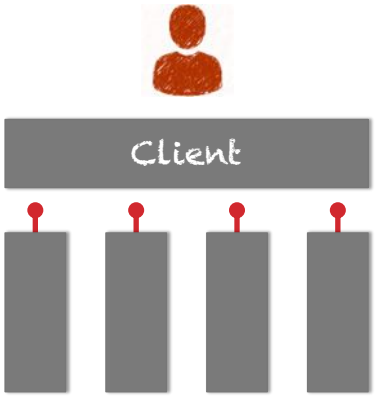
Anforderung:
Trotz mehrerer Teile präsentiert sich die Anwendung dem Benutzer “aus einem Guss“.

Frage:
Wie realisieren wir mit mehreren Teilen *ein* UI?

28

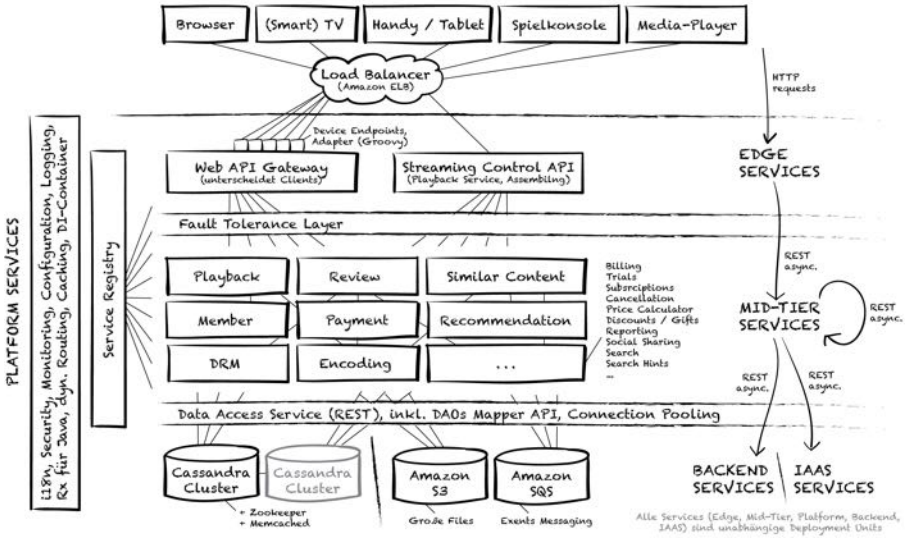
Antworten: (Extreme) Option A

Microservices (allgemeiner: Vertikalen) haben keinen UI-Anteil, sondern nur eine (z.B. REST)-Schnittstelle. Ein gemeinsamer Client greift auf alle Services zu.



29

Beispiel: Netflix

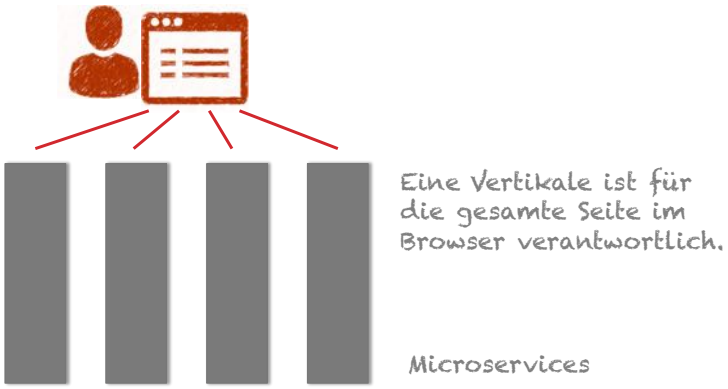


Quelle: Stefan Toth, Stefan Zörner "Gut das ist? Umgekehrte Architekturbewertung eines Internetgiganten"

30

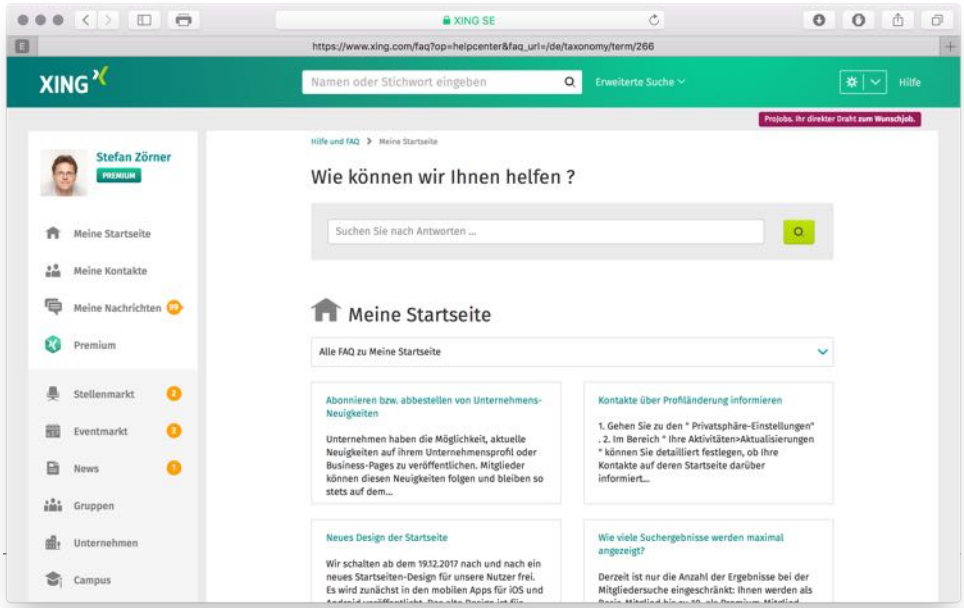
Antworten: (Extreme) Option B

Vertikalen (z.B. Microservices) bringen jeweils die komplette UI für „ihr Thema" mit. Die Integration erfolgt z.B. über Links im Browser.



31

Beispiel: Xing



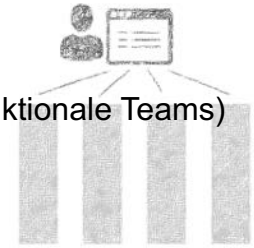
Besondere Stärken der Ansätze

Gemeinsames UI für UI-lose Vertikalen

- Inhalte und Funktionen aus verschiedenen Themen lassen sich nahtlos integrieren – keine Brüche
- Einheitliches User Interface ist leicht erreichbar
- Spezialisiertes Team für eine optimale UX denkbar

Separate UIs für die Vertikalen

- Teams vollumfänglich für Thema verantwortlich (Cross-funktionale Teams)
- Technologische Freiheiten im UI-Bereich
- Unabhängiges Arbeiten der Teams leichter möglich



Kompromisse ...

Begriffe nach ISO 25010

Benutzbarkeit (Usability)
Ist die Software intuitiv zu bedienen, leicht zu erlernen, attraktiv?

Portabilität (Portability)
Ist die Software leicht auf andere Zielumgebungen (z.B. anderes OS) übertragbar?

Funktionale Eignung (Functional Suitability)
Sind die berechneten Ergebnisse genau genug / exakt, ist die Funktionalität angemessen? ...

Effizienz (Performance)
Antwortet die Software schnell, hat sie einen hohen Durchsatz, einen geringen Ressourcenverbrauch? ...

Kompatibilität (Compatibility)
Ist die Software konform zu Standards, arbeitet sie gut mit anderen zusammen?

Zuverlässigkeit (Reliability)
Ist das System verfügbar, tolerant gegenüber Fehlern, nach Abstürzen schnell wieder hergestellt? ...

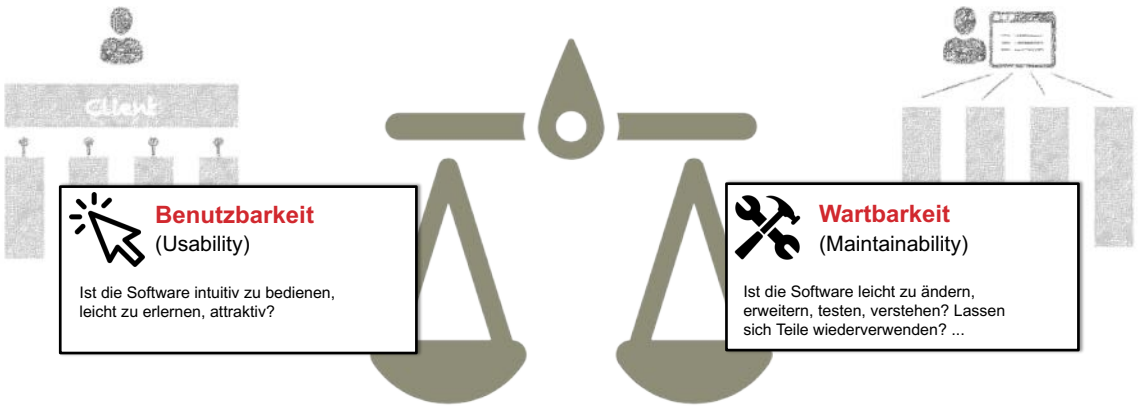
Sicherheit (Security)
Ist das System sicher vor Angriffen? Sind Daten und Funktion vor unberechtigtem Zugriff geschützt? ...

Wartbarkeit (Maintainability)
Ist die Software leicht zu ändern, erweitern, testen, verstehen? Lassen sich Teile wiederverwenden? ...

34

Eine Frage der Balance

Mit dem Beantworten der UI-Frage balancieren Sie vor allem Benutzbarkeit und Wartbarkeit aus. Die beiden Antworten entsprechen Extrem-Ausschlägen.

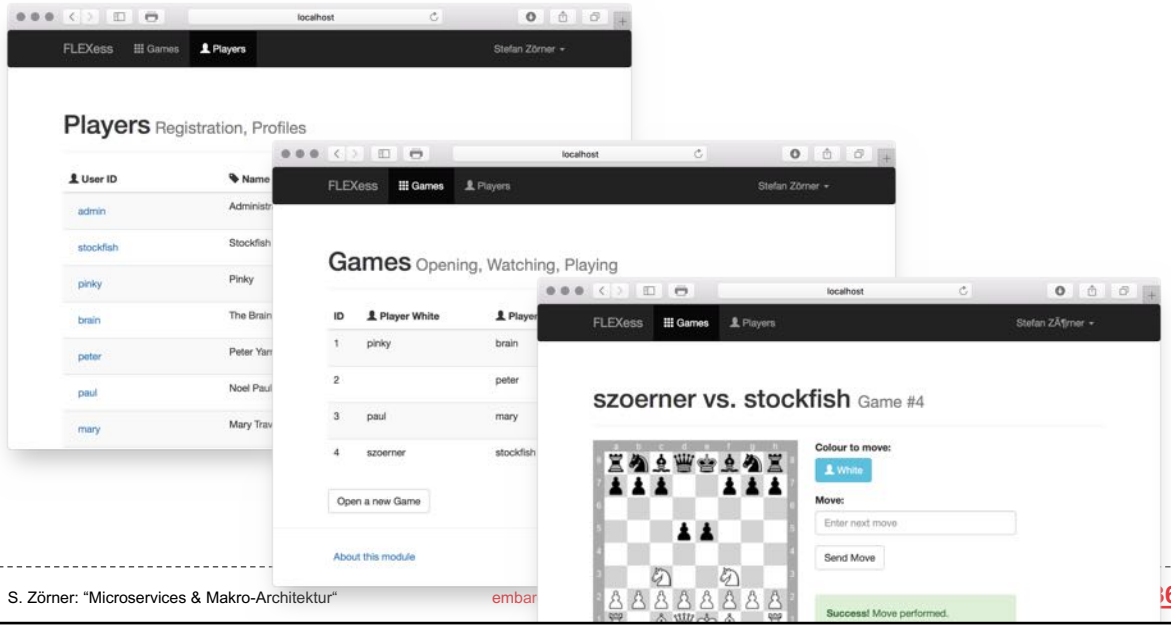


Benutzbarkeit (Usability)
Ist die Software intuitiv zu bedienen, leicht zu erlernen, attraktiv?

Wartbarkeit (Maintainability)
Ist die Software leicht zu ändern, erweitern, testen, verstehen? Lassen sich Teile wiederverwenden? ...

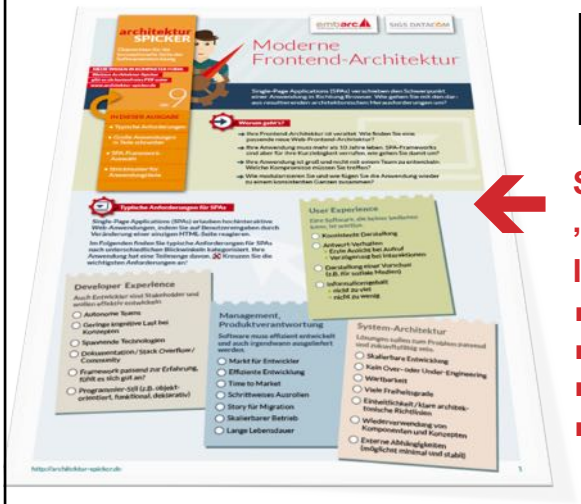
35

Realisierung in “FLEXess”



36

Architektur-Spicker zu Frontend-Architektur



Unsere Architektur-Spicker beleuchten die konzeptionelle Seite der Softwareentwicklung.

Spicker Nr. 9:
„Moderne Frontend-Architektur“

In dieser Ausgabe:

- Typische Anforderungen für SPAs
- Große Anwendungen in Teile schneiden
- SPA-Framework-Auswahl
- Strickmuster für Anwendungsteile

PDF, 4 Seiten. Kostenloser Download. ➔ https://www.architektur-spicker.de

37

Agenda

1 Heilsversprechen reloaded

2 Themen für die Makro-Architektur?

3 Thema I: Die UI-Frage

4 **Thema II: Kommunikation** 

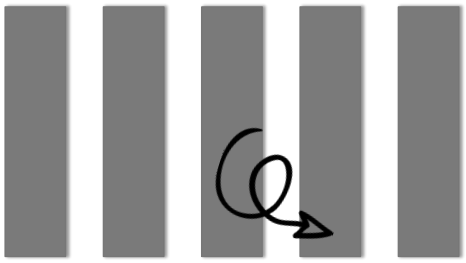
5 Thema III

6 Weitere Informationen und Fazit

4

38

Die Kommunikationsfrage

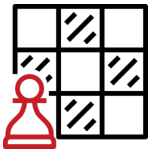


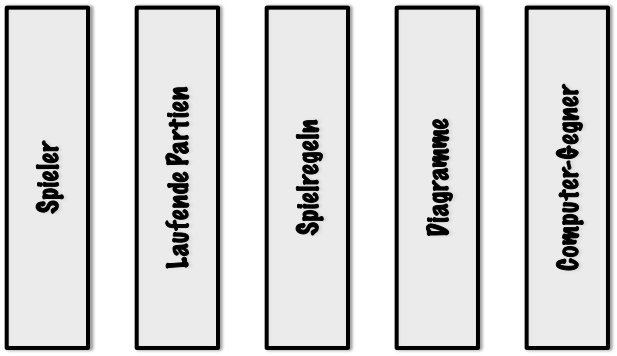
Anforderung:
Ein Service benötigt Funktionalität und/oder Daten eines anderen Services, um seine Aufgabe zu erledigen.

Frage:
Dürfen Services miteinander reden, und wenn ja wie?
(Und wenn nein – wie realisieren wir dann obige Anforderung?)

39

Beispiel in FLEXess

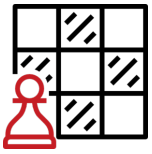


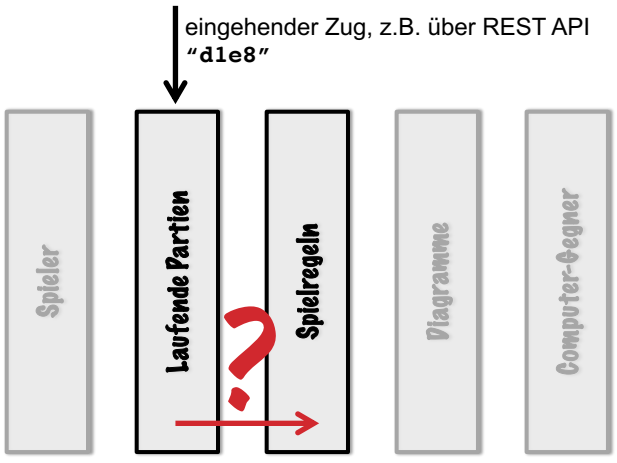


Nutzung der Spielregeln aus den laufenden Partien, um eingehende Züge zu prüfen.

40

Beispiel in FLEXess

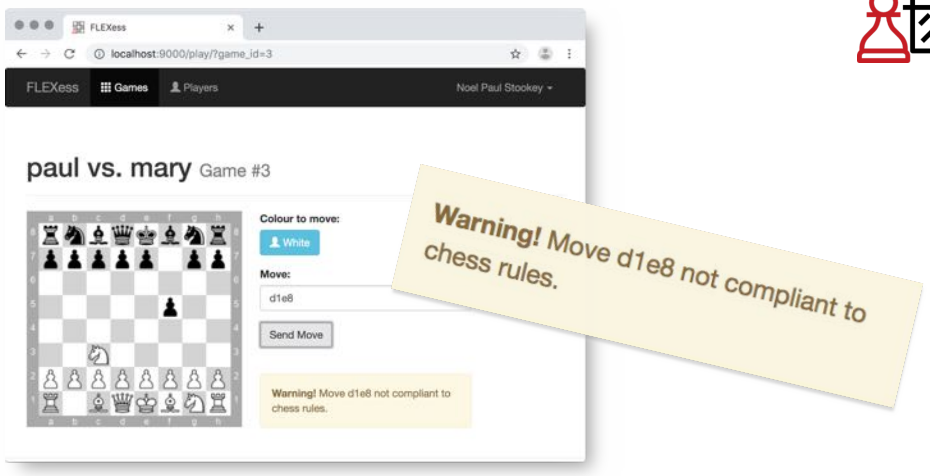




Nutzung der Spielregeln aus den laufenden Partien, um eingehende Züge zu prüfen.

41

Beispiel in FLEXess



Die Frage im Detail (1)

direkt vs. indirekt

Direkt
Der Client „kennt" den Service und spricht ihn direkt an.

Indirekt
Der Client kommuniziert über eine Middleware, die ihn vom Service entkoppelt. Client und Server „kennen" sich nicht.

Die Frage im Detail (2)

synchron vs. asynchron

Synchron
Der Client wartet auf die Antwort des Service und blockiert.

Asynchron
Der Client wartet nicht auf die Antwort. Er erhält diese falls nötig später, ggf. über einen anderen Kanal.

44

Eine Option: direkt + synchron

Top-Herausforderung aus „direkt"
Der Client muss den (ggf. redundanten) Service auffinden.

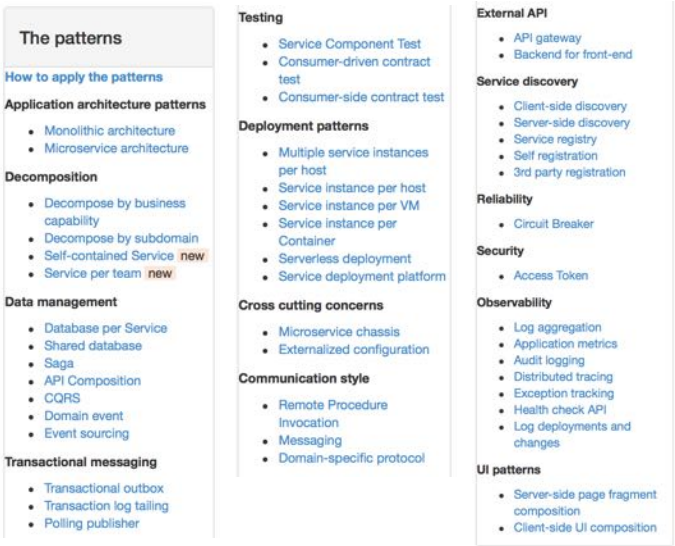
(Typische Lösung: Service Registry)

Top-Herausforderung aus „synchron"
Wenn der Service nicht verfügbar ist oder fehlerhaft bzw. langsam antwortet, zieht das weitere Systemteile runter.

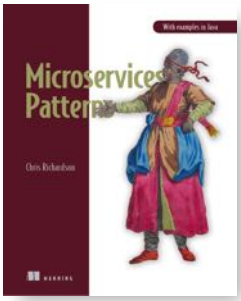
(Typische Lösung: Circuit Breaker, Bulk Heads ...)

45

Microservices Patterns



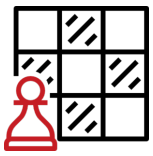
Chris Richardson
"Microservice Patterns"
Manning, Oktober 2018

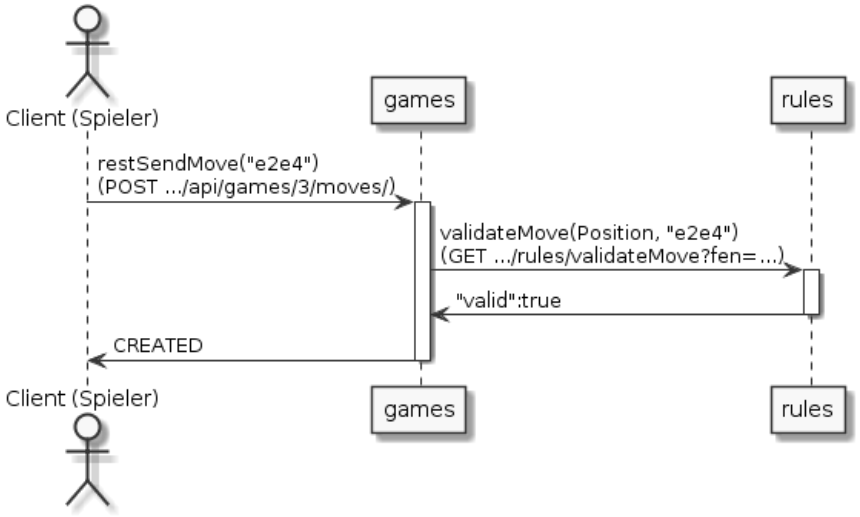


➔ https://microservices.io/patterns/

46

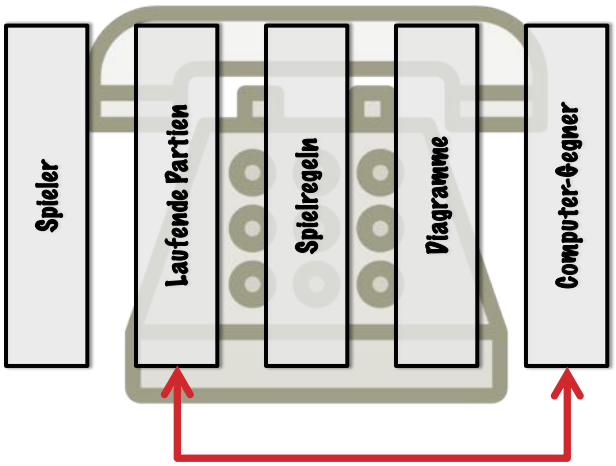
Implementierung FLEXess





47

Weitere Kommunikationsaspekte



z.B. Anbindung von Computer-Gegners als Integrationsaufgabe.

48

Agenda

1 Heilsversprechen reloaded

2 Themen für die Makro-Architektur?

3 Thema I: Die UI-Frage

4 Thema II: Kommunikation

5 Thema III: Security ----------

6 Weitere Informationen und Fazit

5

49

Die Security-Frage

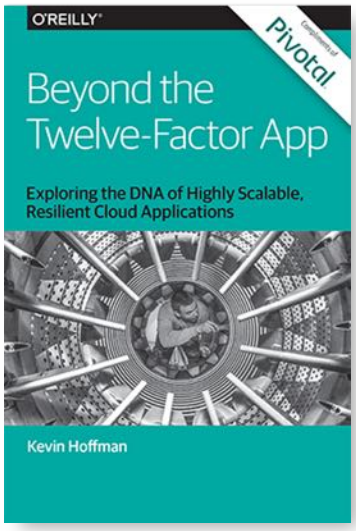
Anforderung:
„Security sollte niemals ein nachträglicher Einfall bei Deiner Anwendungsentwicklung sein.“

(aus „Beyond the Twelve Factor App“)

Frage:
Welche Themen rund um Security adressieren wir in der Makro-Architektur? (und wie?)

50

“Beyond the Twelve-Factor App”



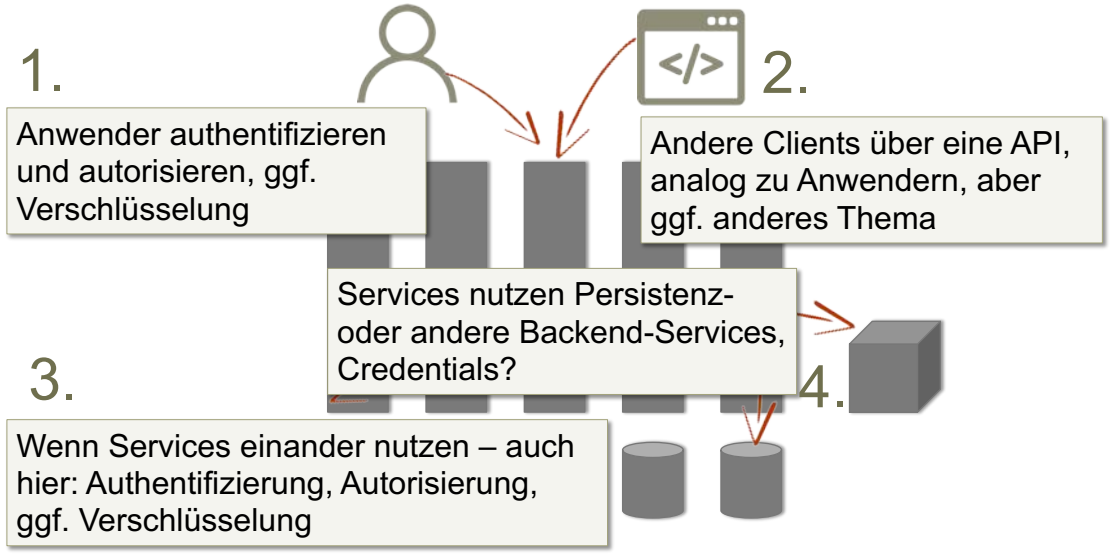
Kevin Hoffman
Beyond the Twelve-Factor App
Exploring the DNA of Highly Scalable, Resilient Cloud Applications

O’Reilly Media 2016
ISBN 978-1-491-94401-1

72 Seiten (PDF)

51

Security unter der Lupe



Eine häufige Antwort für (1)

Die Anwender der Services authentifizieren und autorisieren ...

Authentifizierung zentral (Makro-Architektur)

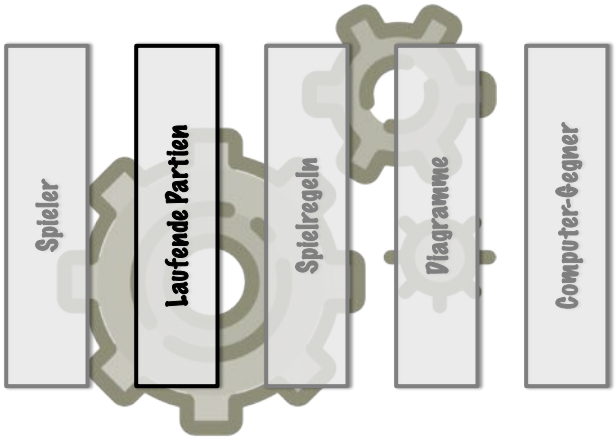
- Gemeinsamer Service, Single Sign-on
- Einzelne Services erhalten Token z.B. via JWT (JSON Web Token)

Autorisierung obliegt den einzelnen Services

- Service erhält Identität des Aufrufers und ggf. weitere Eigenschaften (z.B. globale Rollen)
- Die Überprüfung der Berechtigung verantwortet der Service

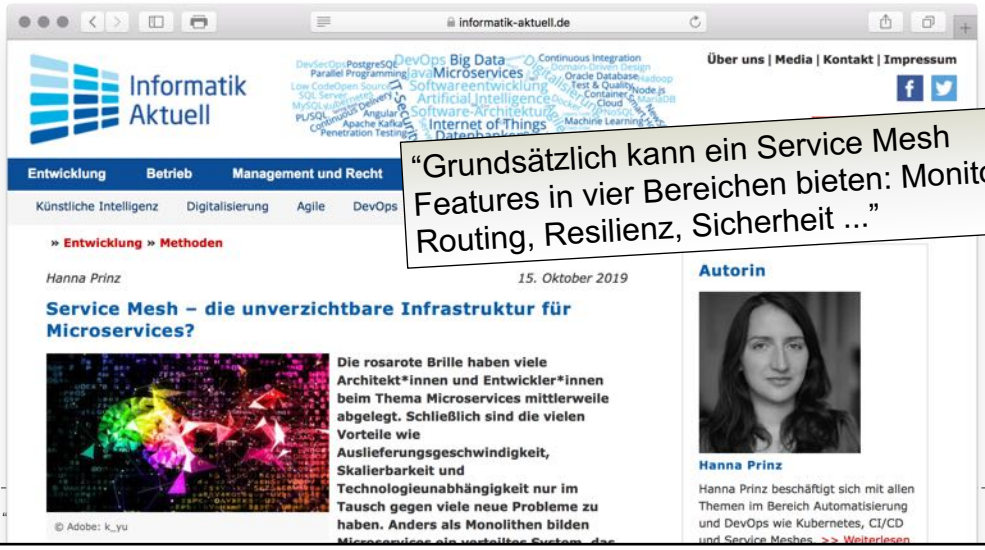
FLEXess – Ein Beispiel

Der „Laufende Partien“-Service entscheidet, ob ein Spieler einen Zug machen darf.



Service Mesh. Ein Lösungsansatz?

➔ informatik-aktuell.de/entwicklung/methoden/service-mesh-fuer-microservices-unverzichtbar.html



“Grundsätzlich kann ein Service Mesh Features in vier Bereichen bieten: Monitoring, Routing, Resilienz, Sicherheit ...”

Agenda

6

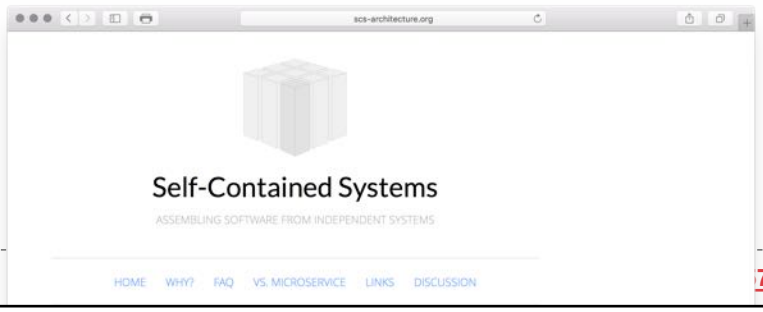
1 Heilsversprechen reloaded
2 Themen für die Makro-Architektur?
3 Thema I: Die UI-Frage
4 Thema II: Kommunikation
5 Thema III: Security
6 **Weitere Informationen und Fazit**

56

Self-contained Systems (SCS)

*"The Self-contained System (SCS) approach is an architecture that focuses on a separation of the functionality into **many independent systems**, making the complete logical system a collaboration of many smaller software systems. This avoids the problem of large monoliths that grow constantly and eventually become unmaintainable."*

➔ http://scs-architecture.org



57

SCS – Charakteristika

Der Architekturstil Self-contained Systems ist klarer definiert als Microservices bei Fowler et. al

- Jedes SCS ist eine autonome Web-Applikation.
- Für jedes SCS ist ein Team verantwortlich
- Wo immer möglich erfolgt die Kommunikation zwischen verschiedenen SCS asynchron.
- Ein SCS kann eine Service API besitzen (zusätzlich zum UI)
- Jedes SCS beinhaltet Datenhaltung und Geschäftslogik
- Ein SCS stellt seine Funktionalität über sein eigenes UI bereit.
- Eine SCS teilt keine Geschäftslogik mit einer anderen SCS.
- Geteilte Infrastruktur sollte wo immer möglich vermieden werden.

→ http://scs-architecture.org

58

SCS – Charakteristika

Der Architekturstil Self-contained Systems ist klarer definiert als Microservices bei Fowler et. al

- **Jedes SCS ist eine autonome Web-Applikation.**
- Für jedes SCS ist ein Team verantwortlich
- Wo immer möglich erfolgt die Kommunikation zwischen verschiedenen SCS asynchron.
- Ein SCS kann eine Service API besitzen (zusätzlich zum UI)
- Jedes SCS beinhaltet Datenhaltung und Geschäftslogik
- Ein SCS stellt seine Funktionalität über sein eigenes UI bereit.
- Eine SCS teilt keine Geschäftslogik mit einer anderen SCS.
- Geteilte Infrastruktur sollte wo immer möglich vermieden werden.

→ http://scs-architecture.org

59

SCS – Charakteristika

Der Architekturstil Self-contained Systems ist klarer definiert als Microservices bei Fowler et. al

- Jedes SCS ist eine autonome Web-Applikation.
- Für jedes SCS ist ein Team verantwortlich
- **Wo immer möglich erfolgt die Kommunikation zwischen verschiedenen SCS asynchron.**
- Ein SCS kann eine Service API besitzen (zusätzlich zum UI)
- Jedes SCS beinhaltet Datenhaltung und Geschäftslogik
- Ein SCS stellt seine Funktionalität über sein eigenes UI bereit.
- Eine SCS teilt keine Geschäftslogik mit einer anderen SCS.
- Geteilte Infrastruktur sollte wo immer möglich vermieden werden.

→ http://scs-architecture.org

60

SCS – Charakteristika

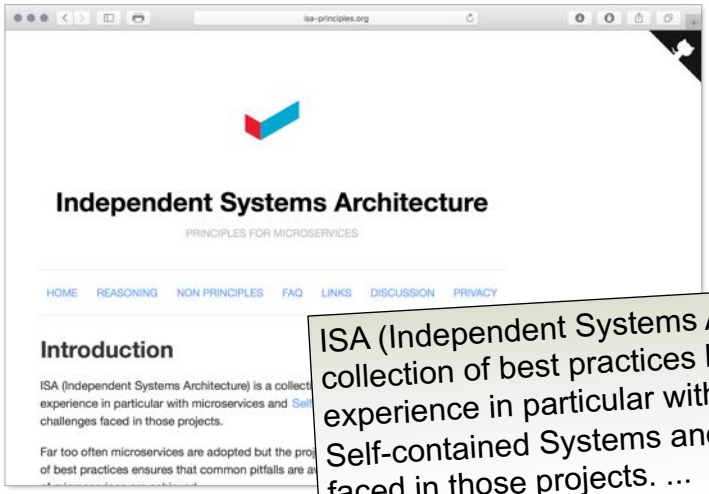
Der Architekturstil Self-contained Systems ist klarer definiert als Microservices bei Fowler et. al

- Jedes SCS ist eine autonome Web-Applikation.
- Für jedes SCS ist ein Team verantwortlich
- Wo immer möglich erfolgt die Kommunikation zwischen verschiedenen SCS asynchron.
- Ein SCS kann eine Service API besitzen (zusätzlich zum UI)
- Jedes SCS beinhaltet Datenhaltung und Geschäftslogik
- Ein SCS stellt seine Funktionalität über sein eigenes UI bereit.
- Eine SCS teilt keine Geschäftslogik mit einer anderen SCS.
- **Geteilte Infrastruktur sollte wo immer möglich vermieden werden.**

→ http://scs-architecture.org

61

ISA Principles



ISA (Independent Systems Architecture) is a collection of best practices based on experience in particular with microservices and Self-contained Systems and the challenges faced in those projects. ...

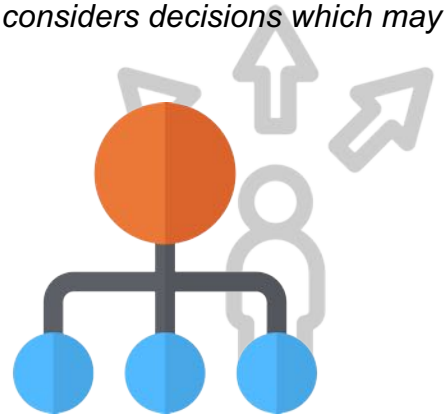
➔ http://isa-principles.org

62

ISA Principle 3 (/9)

" … 3. The system must have two clearly separated levels of architectural decisions: ***The Macro Architecture*** *comprises decisions that cover all modules. All further principles are part of the Macro Architecture.*
The ***Micro Architecture*** *considers decisions which may be taken individually for each module. …"*

➔ http://isa-principles.org



63

tl;dr (too long; didn't read)

*Ein gutes Verständnis von **Makro- und Mikroarchitektur** ist zentral für den Einsatz moderner Architekturstile wie Microservices und Self-contained Systems.*

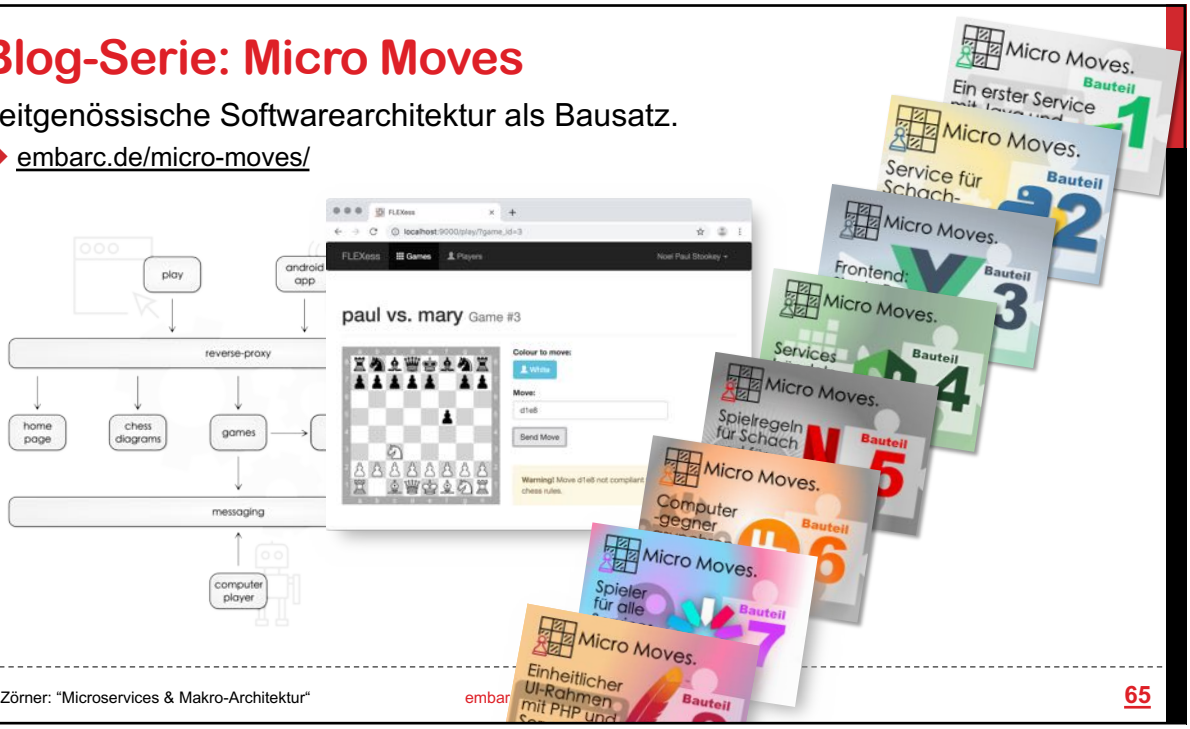
***UI-, Kommunikations-,** und **Security**-Themen sind häufige Brennpunkte für eine frühe initiale Bearbeitung in der Makro-Architektur.*

*Entscheidungen dort bergen oftmals **Kompromisse**, die entlang der **Qualitätsziele** ausbalanciert gehören. Dazu muss man diese kennen.*

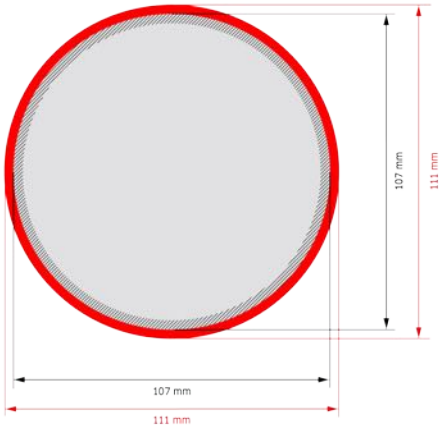
Blog-Serie: Micro Moves

Zeitgenössische Softwarearchitektur als Bausatz.

➔ embarc.de/micro-moves/



Makro-Architektur auf einem Bierdeckel?



66

Architekturüberblick auf einer DVD-Hülle

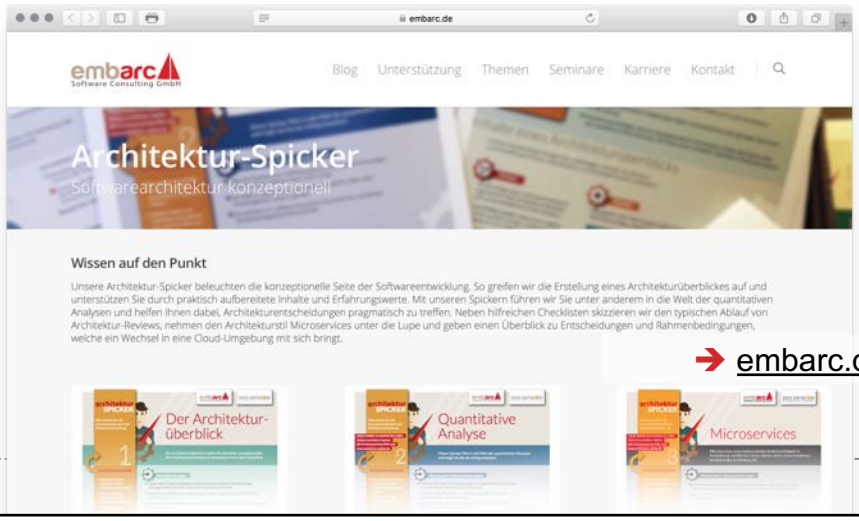
67

Zum Mitnehmen bei uns am Stand.

68

Spicken erlaubt!

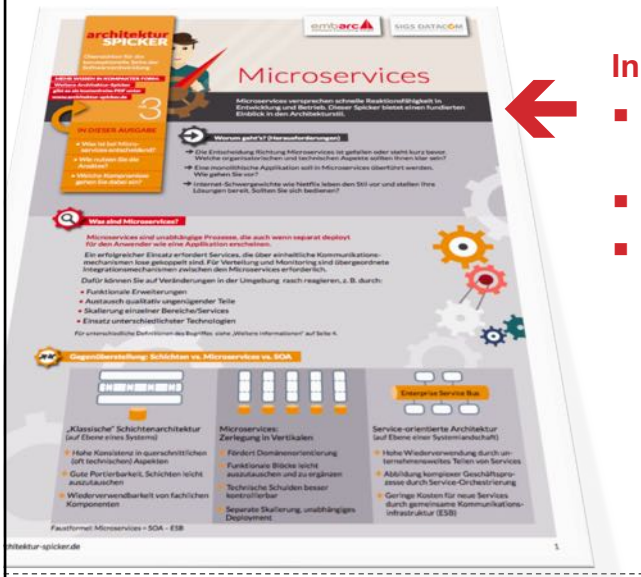
Unsere Architektur-Spicker beleuchten die konzeptionelle Seite der Softwareentwicklung.



➔ embarc.de/architektur-spicker/

69

Spicker #3: „Microservices“



In dieser Ausgabe:

- Was ist bei Microservices entscheidend?
- Wie nutzen Sie die Ansätze?
- Welche Kompromisse gehen Sie dabei ein?

PDF, 4 Seiten
Kostenloser Download.

➔ embarc.de/architektur-spicker/

70

Vielen Dank.

Ich freue mich auf Eure Fragen!

sz@embarc.de

@StefanZoerner

➔ xing.to/szr

DOWNLOAD FOLIEN:
https://www.embarc.de/blog/

71