embarc
Software Consulting GmbH

Mit der Zeit gehen ...

Flexible Architekturen

FALK SIPPACH

Mainz, 09.09.2020

Falk Sippach

embarc Software Consulting GmbH

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

fs@embarc.de

@sippsack

➔ xing.to/fsi

ARCHITECTURE SPECIAL DAY

JAX & W-JAX

Die Konferenzen für Java, Architektur- und Software-Innovation

Bis Konferenz-beginn
✓ 5-Tages-Special
✓ Kollegenrabatt
✓ Bis zu 247 € sparen

Mittwoch, 09. Sep. 2020

Die Arbeit in der Softwarearchitektur hat sich insbesondere durch die agilen Methoden gewandelt und wird durch die stetig komplexer werdenden Softwareprojekte vor immer größere Herausforderungen gestellt. Der Architecture Special Day vermittelt Wissenswertes über den Aufbau moderner Softwarearchitekturen und beleuchtet die spezifischen, methodischen und technologischen Anforderungen, denen Architekturverantwortliche sich im Lichte der allgegenwärtigen Digitalisierung heute stellen müssen.

* **Warum brauchen wir Architekturarbeit überhaupt (noch)?**
* **Auf welche Prinzipien und sogar Gesetze können wir trotz der steigenden Komplexität heutiger Softwaresysteme zurückgreifen?**
* **Wie kann eine moderne und flexible Architektur konkret aussehen?**
* **Wie kann in Zeiten der wachsenden Digitalisierung die Entwicklung auch organisatorisch skalieren?**
* **Was benötigen wir noch, damit Architekturarbeit wirklich „einfacher" wird?**

MI
09. Sep 2020

10:00 - 11:00 Goldsaal A/B/C/D
SESSION
Essential architectural Thinking – why, how, what, when and how much?
Uwe Friedrichsen, *codecentric AG*

11:30 - 12:30 Goldsaal A/B/C/D
SESSION
Prinzipien der Softwarearchitektur – modern und trotzdem zeitlos
Thorsten Maier, *OIO – die Java-Experten von Trivadis*

14:00 - 15:00 Gutenbergsaal 2/3
SESSION
Mit der Zeit gehen – flexible Architekturen
Falk Sippach, *embarc*

15:30 - 16:30 Gutenbergsaal 2/3
SESSION
Matrix-Organisationen, Conway's Law und Microservices – was kann uns das sagen?
Dr. Daniel Lübke, *selbstständig*

17:00 - 18:00 Gutenbergsaal 2/3
SESSION
Panel-Diskussion: Simplification! Muss die Architekturarbeit einfacher werden?"
Sebastian Meyen, *Software & Support Media*

FILTERN NACH
SESSION-TYP
Alle Typen
THEMA
Alle Themen

Bis Konferenzbeginn:
✓ 5-Tages-Special
✓ Kollegenrabatt
✓ Bis zu 247 € sparen
JETZT ANMELDEN

LEGENDE
K Keynote
S Session
W Workshop

Mit der Zeit gehen - Flexible Architekturen

In der agilen Softwareentwicklung gibt es die Rolle des Architekten eigentlich nicht mehr. Paradoxerweise sind die Herausforderungen an den Entwurf moderner Softwaresysteme höher als früher. Um die Digitalisierung voranzutreiben, sich von den Mitbewerbern abzuheben, sich möglichst auch einen Wettbewerbsvorteil zu erarbeiten und die sowohl technische als auch organisatorische Skalierbarkeit sicherzustellen, sind zu den altbekannten ganz neue Fragestellungen hinzugekommen:

- Laufzeitmonolith oder irgendetwas Entkoppeltes?
- DevOps oder klassischer Betrieb?
- Serverless/Cloud oder On Premises?
- Relationale Datenhaltung oder NoSQL bzw. sogar In-Memory?
- Lang- oder Kurzlebigkeit?

Je nach eingesetzten Architekturstilen und -mustern kommen heutzutage ganz neue Herausforderungen auf euch zu. Wir diskutieren in diesem Vortrag, wie agile Teams eine flexible und vor allem auch robuste Softwarearchitektur entwerfen, sie festhalten, kommunizieren und pflegen können. Und das auch, wenn von der grünen Wiese nach ein paar Monaten nicht mehr viel zu sehen ist.


Softwarearchitektur


Flexible Architekturen?

Expertenmeinungen

> *"… not all design is architecture. Architecture represents the **significant design decisions** that **shape a system**, where significant is measured by cost of change."*
>
> (Grady Booch)

> *"Softwarearchitecture is about **the important stuff**, whatever that is."*
>
> (Ralph Johnson)

> *"Software architecture is the **set of design decisions** which, if made incorrectly, may cause your **project to be cancelled**."*
>
> (Eoin Woods)

Was ist Architektur für mich?

fundamentale Strukturen,
Konzepte,
Entscheidungen
und Lösungsansätze

… die man nicht mehr leicht los bekommt!

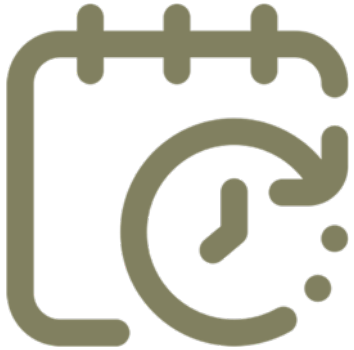
Wer oder was ist ein Software-Architekt?

> Architect is Latin for "**Can’t code anymore**".
>
> (Ted Neward)

> Das Leben von Software-Architekten besteht aus einer langen und schnellen Abfolge **suboptimaler Entwurfsentscheidungen**, die meist **im Dunkel getroffen** werden.
>
> (Philippe Kruchten)

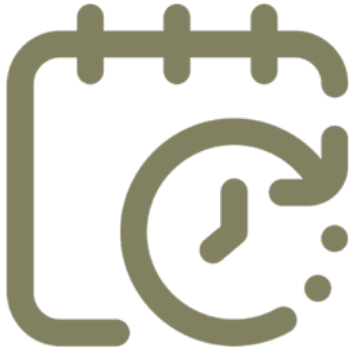
aus: Effektive Software-architekturen (G. Starke)

Agenda



1. Anforderungen klären
2. Flexible Lösungsansätze
3. Dokumentation
4. Ausblick

Agenda



1 Anforderungen klären

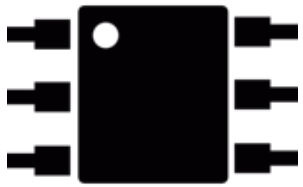
2 Flexible Lösungsansätze

3 Dokumentation

4 Ausblick

1

Einflüsse auf Entscheidungen



Vorgaben/Rahmenbedingungen
Technisch (z.B. Datenbankprodukt)
Organisatorisch (z.B. Team)

- schränken die Lösung ein
- schließen Optionen aus

Qualitätsanforderungen
Benutzbarkeit, Effizienz,
Wartbarkeit, Sicherheit ...

- prägen die Lösung
- nachher schwierig “einzubauen”

Rahmenbedingungen …

… wirken eher einschränkend auf die Optionen beim Entwurf einer Software und können im Gegensatz zu Qualitätsanforderungen oft nicht mit dem Kunden diskutiert werden. Außerdem werden solche Randbedingungen oft auf binäre Art und Weise erfüllt oder nicht, ohne dass es dazwischen Abstufungen gäbe.

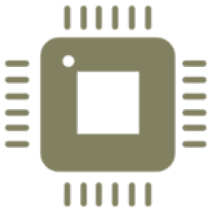
Arten von Rahmenbedingungen

Technologisch

Hardware-Vorgaben
Software-Vorgaben
Vorgaben des Systembetriebs
Programmiervorgaben

Organisatorisch

Organisation und Struktur
Ressourcen (Zeit, Budget, Personal)
Organisatorische Standards
Juristische Faktoren



Qualitätsziele

Die wichtigsten geforderten Qualitätsmerkmale für ein Softwaresystem (Synonym: **Architekturziele**).

Typischerweise werden als Qualitätsziele im Rahmen eines Architekturüberblicks die **Top-3** bis **Top-5** genannt.

Beispiel: Qualitätsziele Netflix

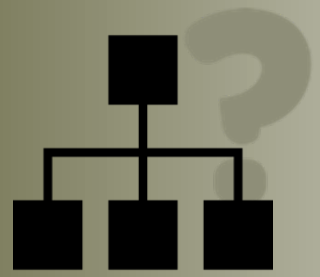
(aka „Architekturziele“ oder „Architekturtreiber“)

Ziel	Beschreibung
Verfügbarkeit	Die Lösung steht auch bei Lastspitzen uneingeschränkt zur Verfügung.
Benutzbarkeit	Auf allen wichtigen Geräten eine optimale User Experience.
Modifizierbarkeit	Es ist leicht, neue Funktionalität zu bauen und hinzuzufügen.
Attraktivität	Wir sind als Arbeitgeber für gute Entwickler attraktiv.
Betreibbarkeit	Wir haben gute Einblicke, was in unserer Umgebung läuft.

NETFLIX

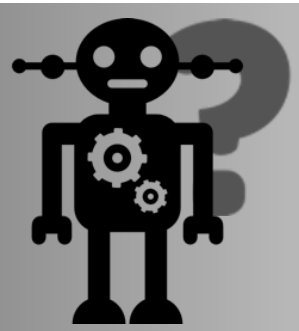
Quelle: Stefan Toth, Stefan Zörner
“Gut das ist? Umgekehrte Architekturbewertung eines Internetgiganten”

Themen für Entscheidungen



Zerlegung

Module und Abhängigkeiten
Komponentenbildung
...



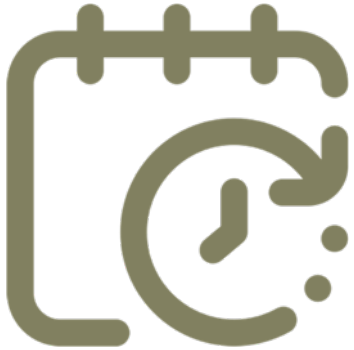
Technologie-Stack

Programmiersprache(n), Bibliotheken, Frameworks
Middleware (Kommunikation, Applikationsserver ...)
Querschnittsthemen (Persistenz, Oberfläche ...)
...

Zielumgebung

Wo läuft die Software? (Endanwender vs. RZ vs. Cloud)
Verteilungsaspekte (Redundanz, Clustering)
Virtualisierung
...

Agenda



1 Anforderungen klären

2 **Flexible Lösungsansätze**

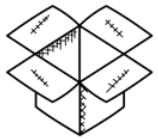
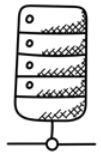
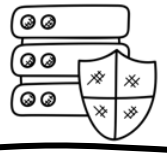
3 Dokumentation

4 Ausblick

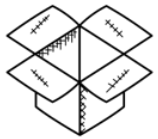
2


Beschreibung
Laufzeitmonolith oder irgendetwas Entkoppeltes?
DevOps oder klassischer Betrieb?
Serverless/Cloud oder On Premises?
Relationale Datenhaltung oder NoSQL bzw. sogar In-Memory?
Lang- oder Kurzlebigkeit?

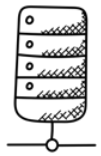
High Level Lösungsstrategien

 Laufzeitmonolith	vs.	Microservices 
 Klassischer Betrieb		DevOps 
 On Premises		Serverless/Cloud 
 Relationale DBs		NoSQL/In-Memory 
 Langlebigkeit		Kurzlebigkeit 

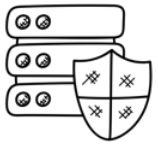
High Level Lösungsstrategien



Laufzeitmonolith



Klassischer Betrieb



On Premises

Relationale DBs

Langlebigkeit

Konservativ?

Modern?

Microservices

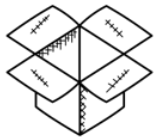
DevOps

Serverless/Cloud

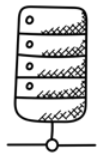
NoSQL/In-Memory

Kurzlebigkeit

Vorteile/Herausforderungen/Ziele

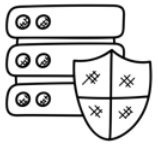


“gediegen”



Gewohnheit

zustandsbehaftet



Datensicherheit

Klassische Organisationsstrukturen

verstanden

Transaktionen

planungssicher

Belastbar/hochverfügbar

widerstandsfähig

komplex

Hohe Lernkurve

flexibel

automatisiert

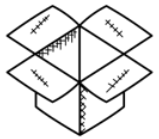
skalierbar

Neue Herangehensweisen

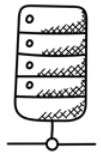
Conway’s Law beachten

Time-To-Market

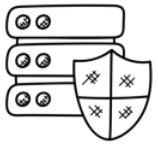
Rahmenbedingungen



Klassische Organisationsstrukturen



Lange Produktlaufzeiten und Kundenbindungen



Datenschutz-/sicherheit

Unflexibler Inhouse-Betrieb

Keine agilen Prozesse

Langlaufende Lizenzverträge

Cloudausrichtung/-guidelines

(Public) Cloud first Ansatz

Agile Prozesse etabliert

Offene Mitarbeiter

Ausreichende Entwicklungskapazitäten

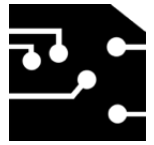
Fortgeschrittene Automatisierung und Infrastruktur als Code

Qualitätsmerkmale

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? ...

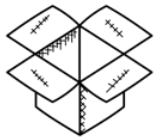
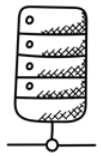
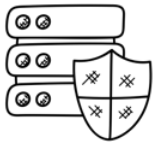
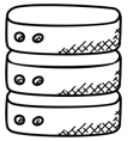
Sicherheit
(Security)

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? ...

Qualitätsziele



Einheitliche Bedienbarkeit


Skalierbarer Durchsatz und Antwortzeiten

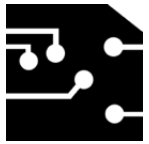

Hohe Zuverlässigkeit


Effiziente Wartbarkeit


Korrekte Funktionalität


Gebotene Sicherheit


Einfacher und günstiger Betrieb


Leicht erweiter-/integrierbar

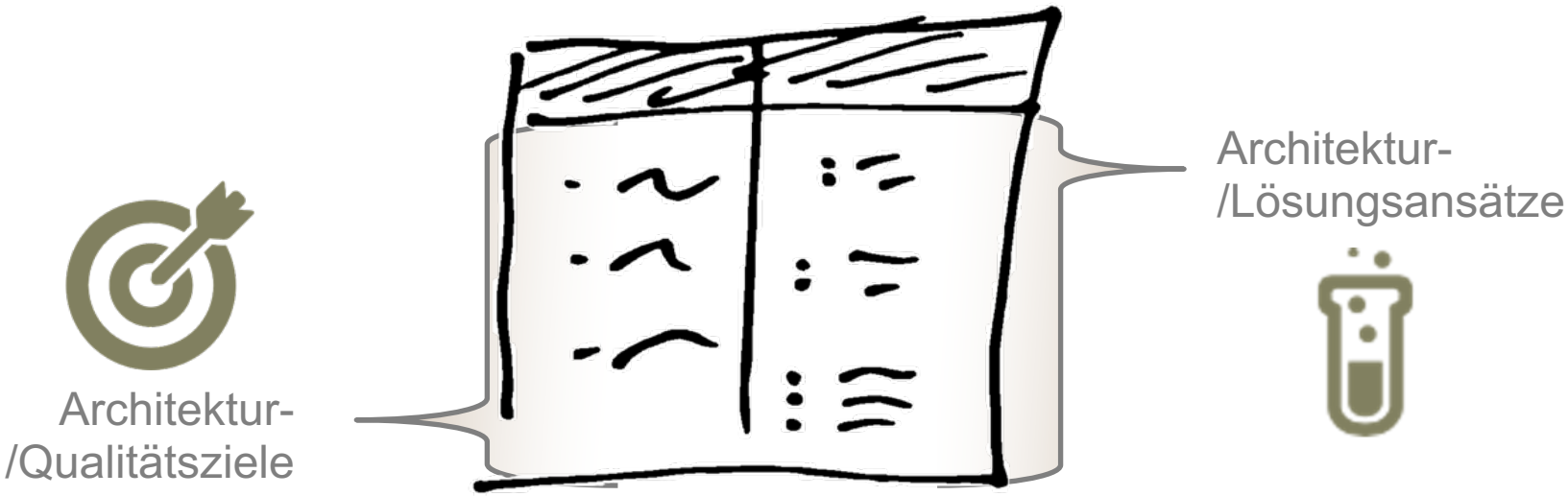




Lösungsstrategie

Stellt Qualitätsziele und zugeordnete high-level Lösungsansätze in Beziehung zueinander dar.

Form: Tabelle ([Ziele | Lösungsansätze]).



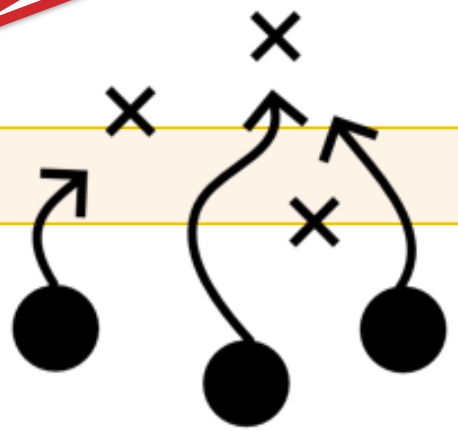
Kompakte Darstellung

Lösungsstrategie

Qualitätsziel	Architekturansatz
QZ 1	Ansatz a) Ansatz b)
QZ 2	Ansatz c)
QZ 3	Ansatz d)
QZ 4	Ansatz e) Ansatz f)
...	

Prägende Entscheidungen, Konzepte, Muster, Prinzipien ...

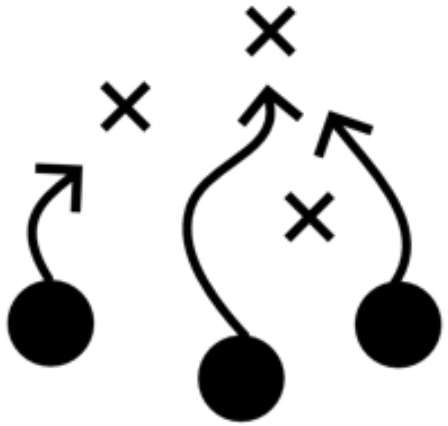
Top 3-5 Architekturziele



Typische Lösungsansätze

Wichtig ...

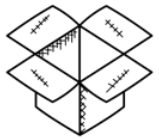
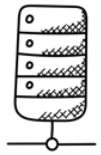
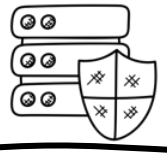
Die Lösungsstrategie schlägt die Brücke zwischen architekturrelevanten Anforderungen und Eurer Lösung.

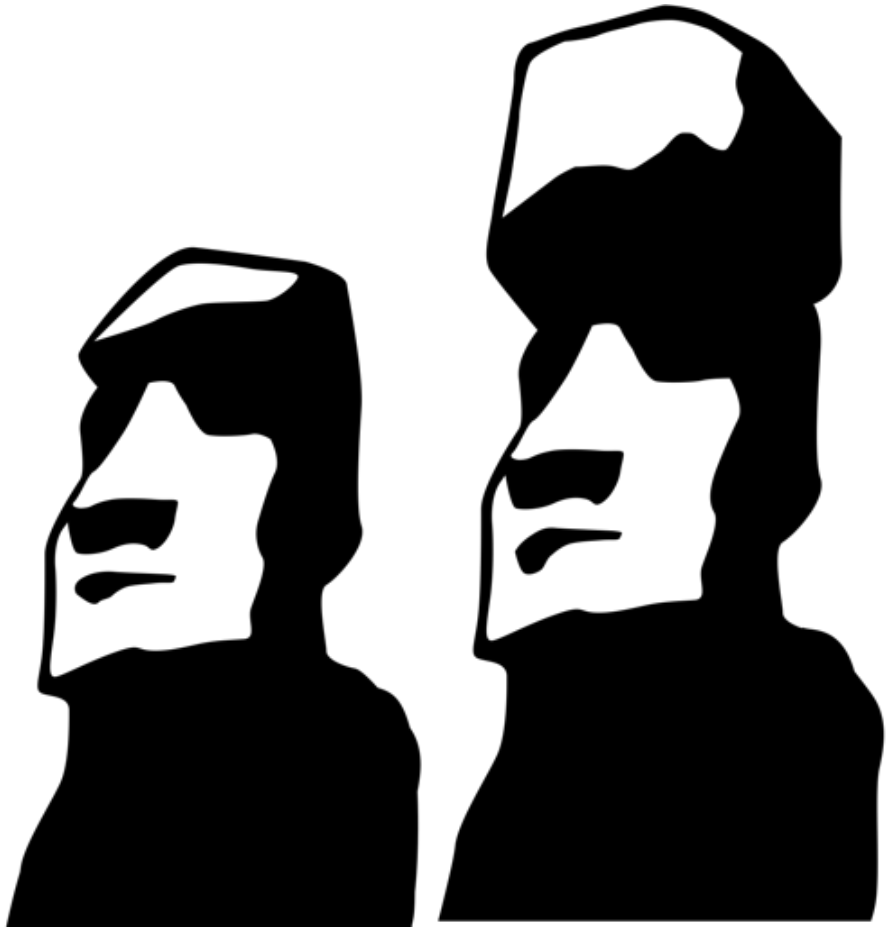


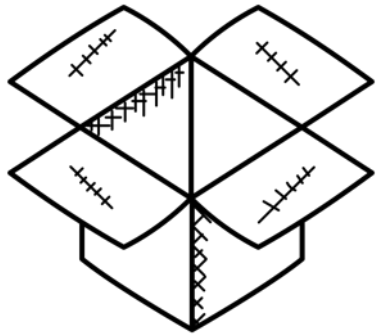
Beispiel

Ziel	Passende Lösungsansätze
Verfügbarkeit	• Betrieb in Public Cloud • Resilience Patterns
Benutzbarkeit	• Public API • Asynchrone Verarbeitung und Messaging
Modifizierbarkeit Wartbarkeit	• Microservices • Komplette Automatisierung
Attraktivität	• Einsatz modernster Technologien • Viele Lösungen sind Open Source
Sicherheit	• Oauth und OpenID • Hoher Datenschutz durch private Cloud

High Level Lösungsstrategien

 Laufzeitmonolith		Microservices 
 Klassischer Betrieb		DevOps 
 On Premises	vs.	Serverless/Cloud 
 Relationale DBs		NoSQL/In-Memory 
 Langlebigkeit		Kurzlebigkeit 





Monolith

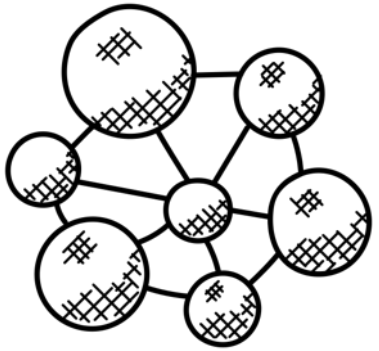
Microservices – in kurz …

“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, often an HTTP resource API.”

(James Lewis, Martin Fowler)

Charakteristische Eigenschaften

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



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

Laufzeitmonolith vs. Microservices

- ❑ **Effizienz (solange nicht zu groß)**
- ❑ **Leichte Installier- und Betreibbarkeit**
- ❑ **Strukturierung möglich (Modulith)**

- ❑ **Schwierige Wartbarkeit bei Big Ball of Mud**
- ❑ **Niedrige Strukturiertheit**
- ❑ **Unkontrollierte Abhängigkeiten**
- ❑ **Pathologische Phänomene für schlechtes Design** (hohe Kopplung, niedrige Kohäsion, zyklische Abhängigkeiten, Zerbrechlichkeit, Unbeweglichkeit, Starrheit)

- ❑ **Fördert Domänenorientierung und Fokussierung**
- ❑ **Erzwingt Modularisierung/saubere Schnittstellen**
- ❑ **Kleinere Arbeitspakete, funktionale Blöcke leicht auszutauschen und zu ergänzen**
- ❑ **Technische Schulden einfacher kontrollierbar**
- ❑ **Separate Skalierung, unabhängiges Deployment**
- ❑ **Schnelle Anpassungen im laufenden Betrieb (Time-To-Market)**
- ❑ **Autonom arbeitende Teams**
- ❑ **Elastisch auf Lastanforderungen reagieren**

- ❑ **Hohe Komplexität beim Deployment und Betrieb**
- ❑ **Netzwerk-Kommunikation**
- ❑ **Steile Lernkurve, Org.struktur anpassen**
- ❑ **Hoher Automatisierungsgrad notwendig**

Klassischer Betrieb vs. DevOps

- ❑ Jahrzehntelange Erfahrung
- ❑ Fokusierung des Betriebssystems

- ❑ Strikte Trennung zw. Betrieb und Entwicklung
- ❑ Aufwändige Fehlersuche
- ❑ Unflexibel und starr
- ❑ meist kaum oder nur einfache Automatisierungen
- ❑ Lange Beschaffungsprozesse
- ❑ Wartezeiten für die Entwicklung

- ❑ Crossfunktionale, autonome Teams
- ❑ Hoher Automatisierungsgrad
- ❑ Reproduzierbarkeit, Dokumentation und Versionierung durch Infrastructure as Code und GitOps
- ❑ Kontinuierliche Verbesserungsprozesse
- ❑ Zero-Downtime Deployments, Canary Releases
- ❑ Plattformunabhängigkeit durch Containerisierung und Kubernetes

- ❑ Hohe Teamverantwortung
- ❑ Neue Komplexitätsstufe, steile Lernkurve

On-Premises vs. Serverless/Cloud (Public)

- Daten und Software vor Ort
- Hoher Datenschutz und –sicherheit in eigener Hand
- Unterstützung verschiedener Deployment-Szenarien (Bare Metal, VMs, Private Cloud)

- Aufwändiger Betrieb (Patches, Bugfixes, Sicherstellen der Datensicherheit)
- Spezielles Personal, ggf. 24/7 Support

- Elastisch, auf Lastspitzen reagieren
- Infrastruktur up to date
- Großes Angebot an Diensten
- Bezahlung nur bei Nutzung
- Global verfügbar, regionsbasierte Lastverteilung, Verteilung geschenkt
- Zero-Downtime Deployments, Canary Releases, selbst heilende Software

- Teuer bei großer Last
- Daten außerhalb der Organisation
- Alternative Deploymentartefakte (GraalVM, Go, …)

Relationale DBs vs. NoSQL/In-Memory

- ❑ Gut verstandene Konzepte
- ❑ Ausgereifte Datenbanken und Tools
- ❑ Viele verfügbare Entwickler
- ❑ ACID Transaktionen **(aber gar nicht immer notwendig!)**
- ❑ Datenbank-Schema

- ❑ Skalierung aufwändig/unmöglich
- ❑ Allgemeines Datenmodell (Schema)
- ❑ Impedence Mismatch (Objektrelationale Unverträglichkeit)

- ❑ Hohe Verfügbarkeit
- ❑ Gute Skalierbarkeit, große Datenmengen
- ❑ Einfache Datenredundanz
- ❑ An Usecase angepasste Datenmodelle (Schemafreiheit)
- ❑ Einfacheres Programmiermodell (Aggregatorientierung)
- ❑ Polyglotte Persistenz

- ❑ Eventually Consistency (BASE)
- ❑ Technologie-Zoo, kaum Standards
- ❑ Andere Modelle - Mindchange

Langlebigkeit vs. Kurzlebigkeit

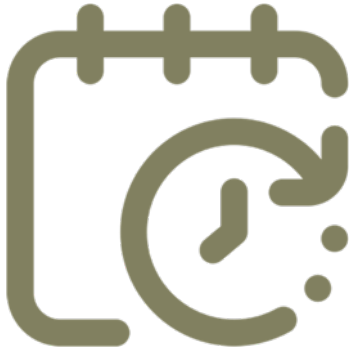
- ❑ Vorausschauende, planbare Produktentwicklung
- ❑ Zukunftssicherheit

- ❑ Langsame Entwicklungszyklen
- ❑ Schlecht bei häufigen Änderungen
- ❑ Big-Bang Releases (Stop-The-World)

- ❑ Schnell mit Ideen am Markt (A/B-Testing, Canary Releases)
- ❑ Schnelles Reagieren auf Änderungen
- ❑ Integriert sich gut mit agilen Methoden
- ❑ Automatisiertes Ausrollen durch Deployment Pipelines (Continuous Delivery)
- ❑ Automatisierte Regressions- und E2E-Tests

- ❑ Aufwändigere Qualitätskontrollen
- ❑ Ggf. Kundenunzufriedenheit in Kauf nehmen

Agenda



1 Anforderungen klären

2 Flexible Lösungsansätze

3 **Dokumentation**

4 Ausblick

3

Dokumentation des Entwurfs

Architekturüberblick

- Mission Statement
- Kontextabgrenzung

Einflüsse

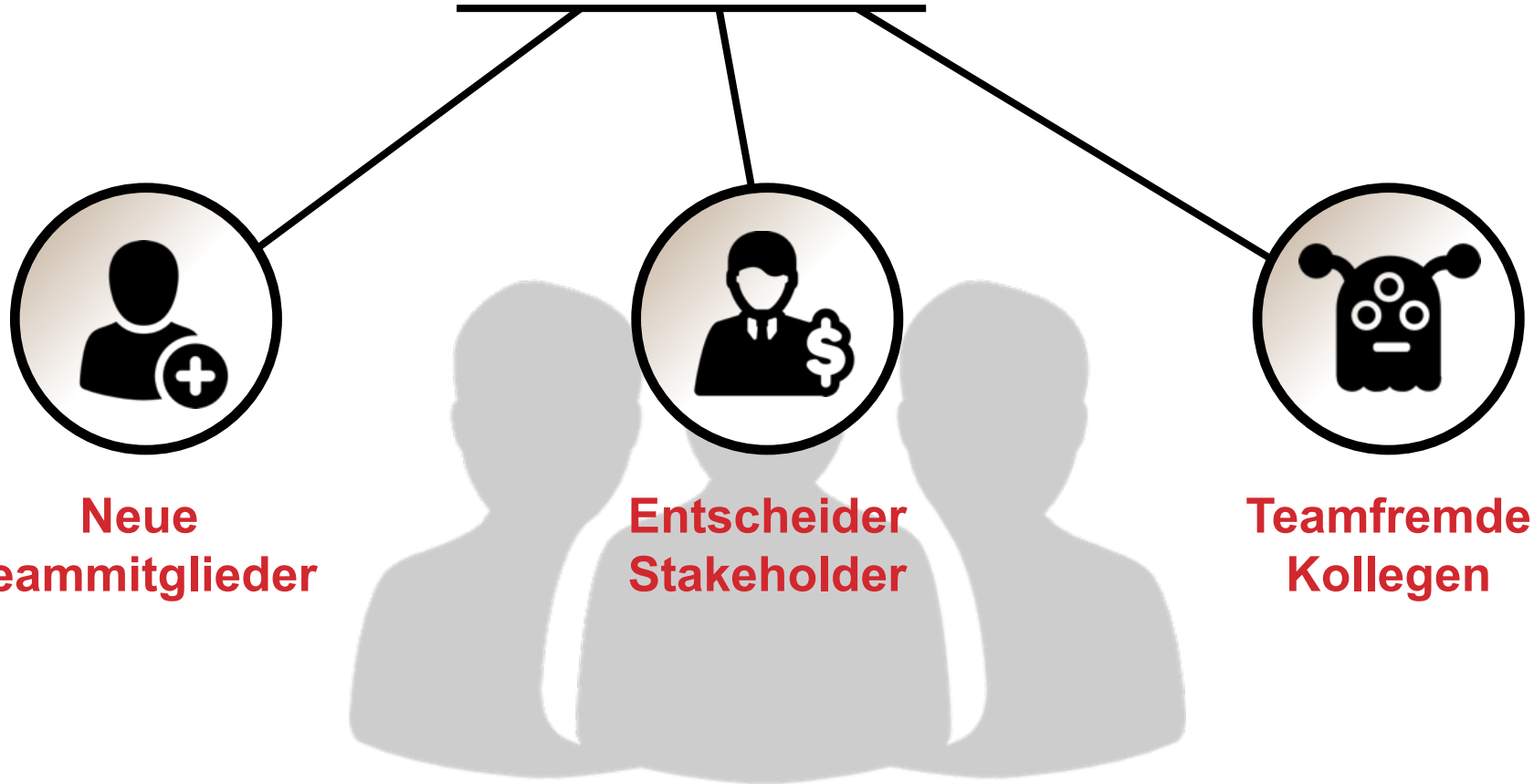
- Vorgaben
- Architekturziele

Lösungsansätze

- Architekturstil
- Technologie-Stack
- Konzepte
- Prinzipien
- Zerlegung
- ....

Was ist ein Architekturüberblick?

Ein Architekturüberblick macht die zentralen Lösungsansätze Eurer Softwarearchitektur für Außenstehende nachvollziehbar.

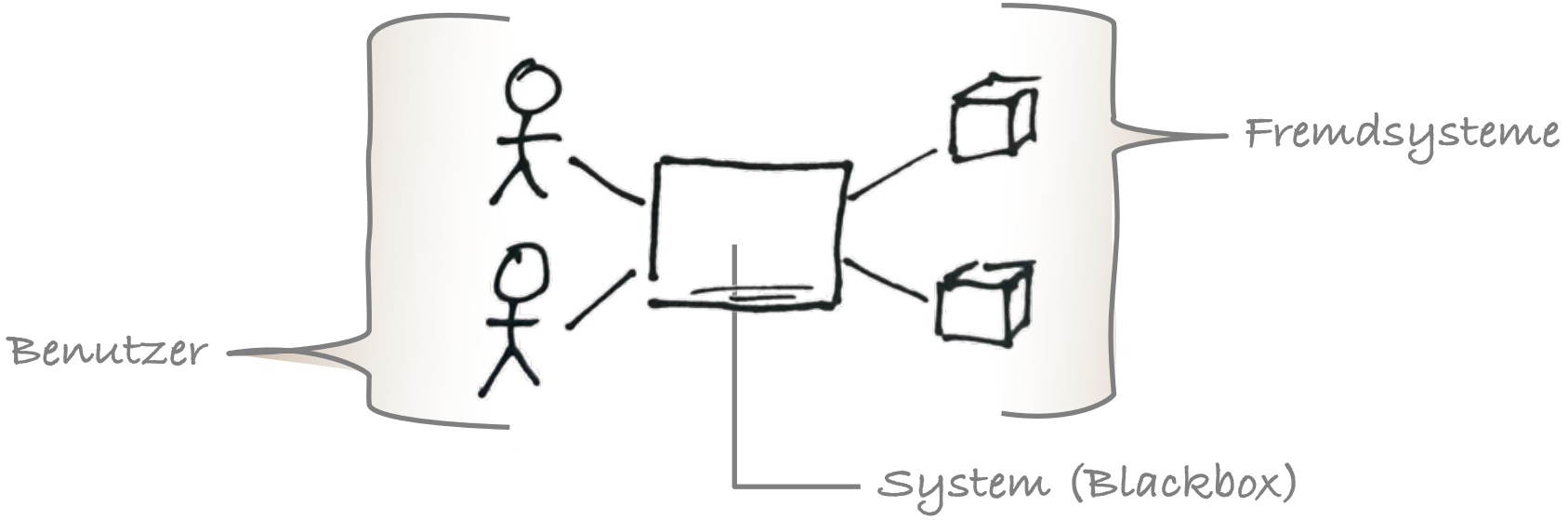



Java in a
tshell
SYMANTEC
VisualCafé
Leichte und intuitive
Anwendungsentwicklung
in 100% Java™
VisualCafé
Version 3
Standard Edition
JAVABEAN

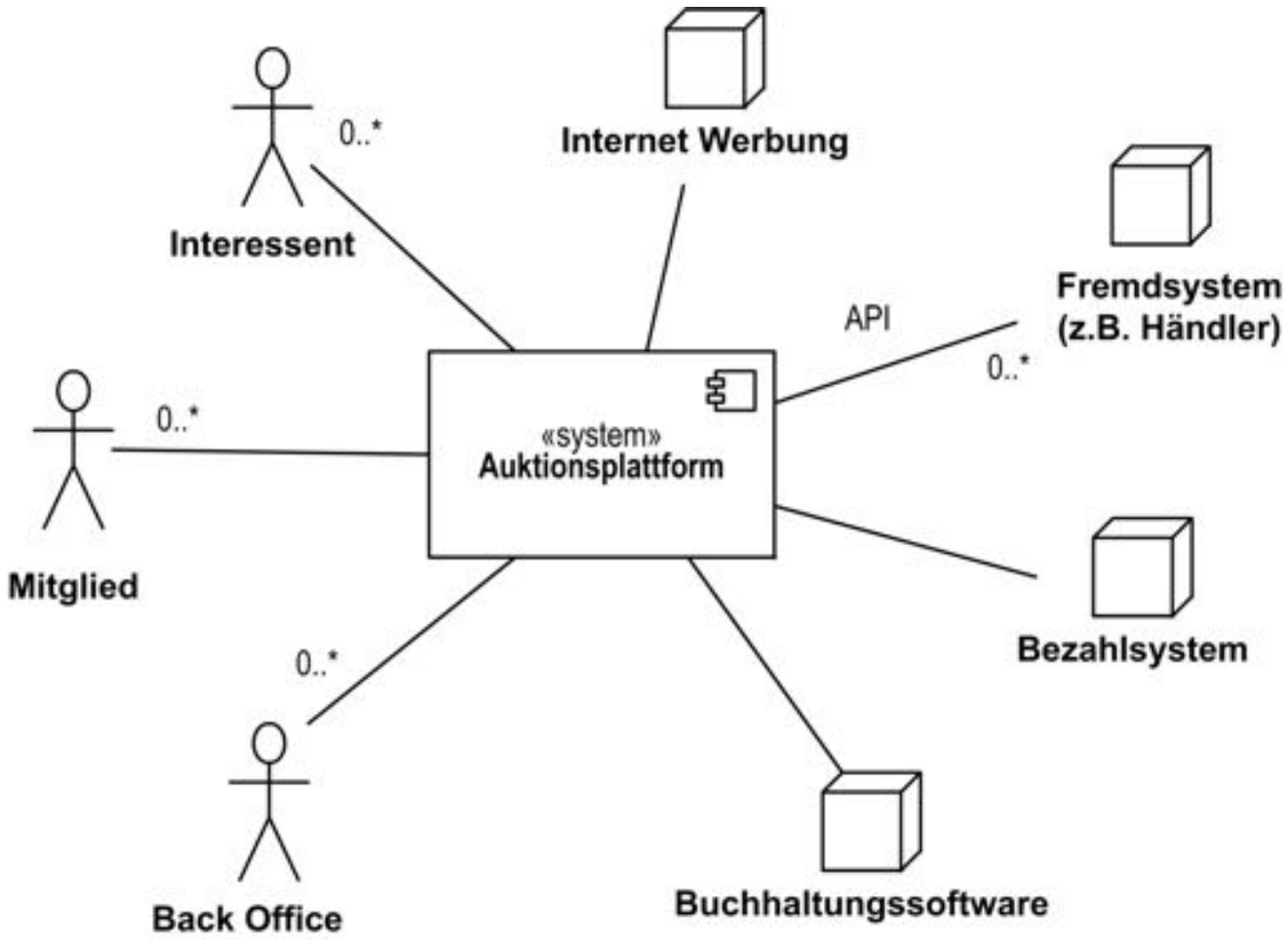
Kontextabgrenzung

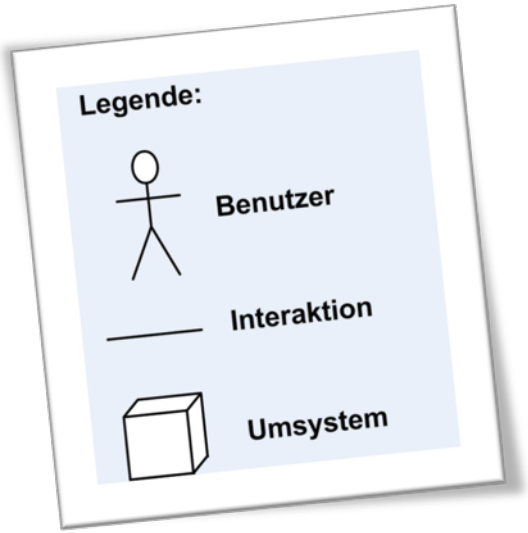
Abgrenzung des Systems und Visualisierung der Benutzer und Fremdsysteme, mit denen es interagiert.

Form: Graphik, ergänzt um kurze Beschreibungen.



Systemkontext einer Online-Platform





Was sonst noch dokumentieren?

Sichten auf das System

- **Bausteinsicht**
- **Laufzeitsicht**
- **Verteilungssicht**

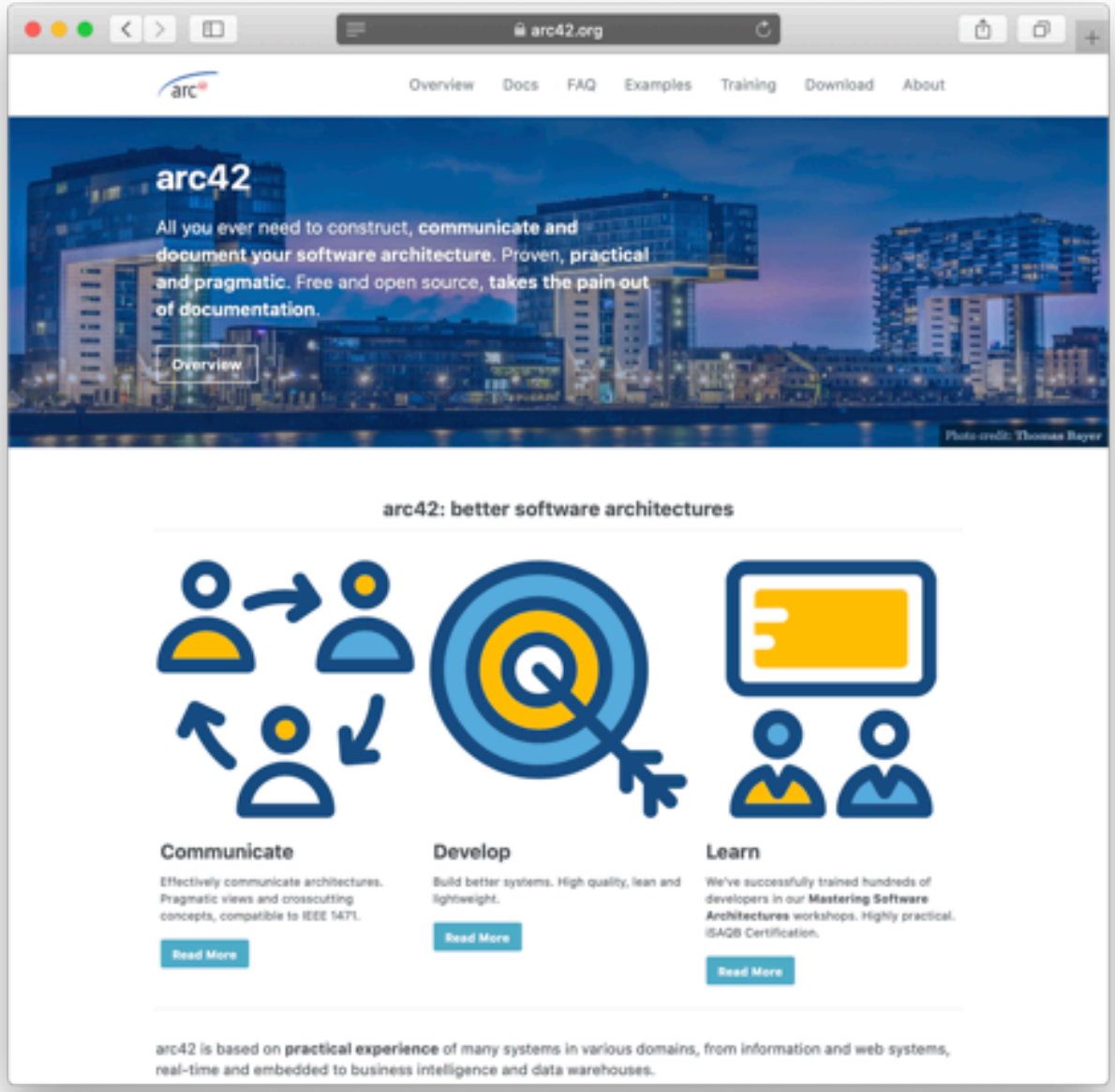
Querschnittliche Konzepte erklären

Entwurfsentscheidungen nachvollziehbar festhalten

Qualitätsszenarien als Ausgangspunkt für quantitative Bewertungsmethoden (ATAM, …)

arc42

Vorschlag für ein Template (G. Starke, P. Hruschka)



➔ http://arc42.org/

„Abheften“ in arc42

1. Einführung und Ziele
1.1 Aufgabenstellung
1.2 Qualitätsziele
1.3 Stakeholder

2. Randbedingungen
2.1 Technische Randbedingungen
2.2 Organisatorische Randbedingungen
2.3 Konventionen

3. Kontextabgrenzung
3.1 Fachlicher Kontext
3.2 Technischer- oder Verteilungskontext

4. Lösungsstrategie

5. Bausteinsicht
5.1 Ebene 1
5.2 Ebene 2
...

6. Laufzeitsicht
6.1 Laufzeitszenario 1
6.2 Laufzeitszenario 2
...

7. Verteilungssicht
7.1 Infrastruktur Ebene 1
7.2 Infrastruktur Ebene 2
...

8. Konzepte
8.1 Fachliche Strukturen und Modelle
8.2 Typische Muster und Strukturen
8.3 Persistenz
8.4 Benutzungsoberfläche
...

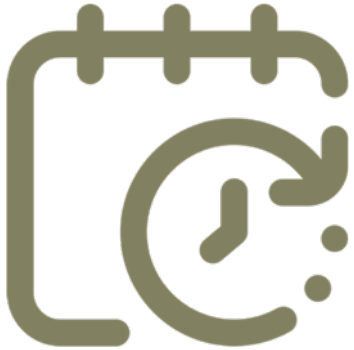
9. Entwurfsentscheidungen
9.1 Entwurfsentscheidung 1
9.2 Entwurfsentscheidung 2
...

10. Qualitätsszenarien
10.1 Qualitätsbaum
10.2 Bewertungsszenarien

11. Risiken

12. Glossar

Agenda



1 Anforderungen klären

2 Flexible Lösungsansätze

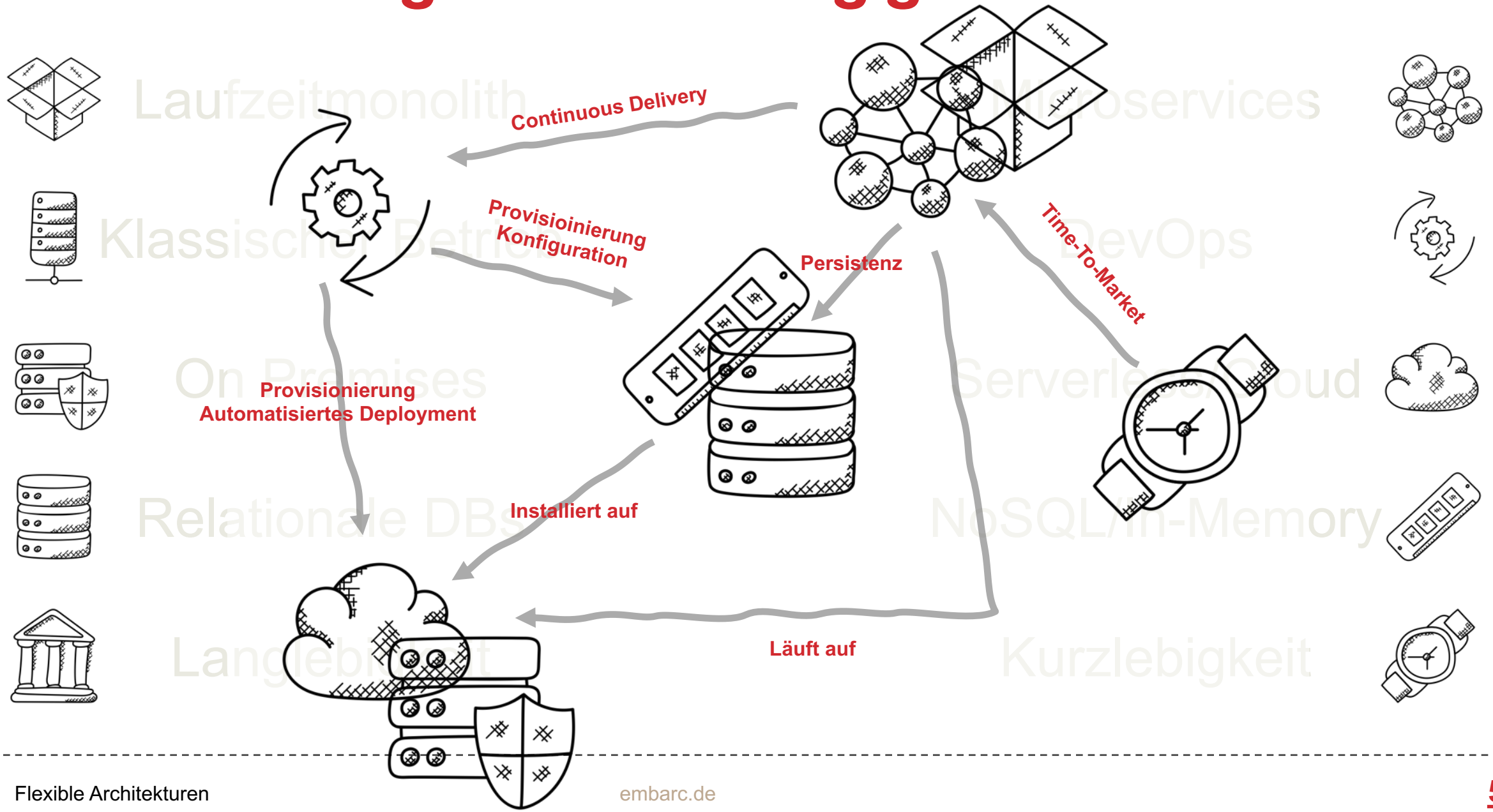
3 Dokumentation

4 **Ausblick**

4

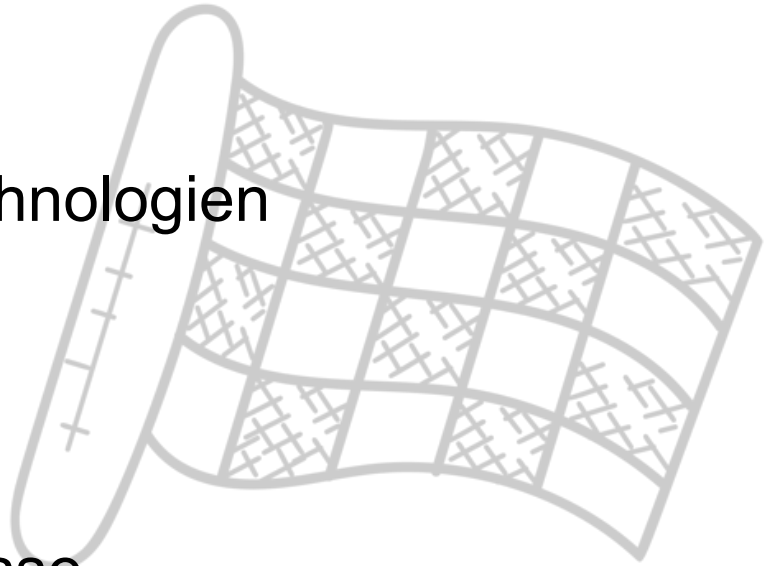
aus: Effektive Software-architekturen (G. Starke)

Beeinflussungen und Abhängigkeiten



Was bleibt/wird wichtig?

- Schnelligkeit
  - Zügige Entwicklung und Ausrollen für schnelle Anpassung an den Markt
  - Fehler akzeptieren, daraus lernen
- Flexibilität
  - Einfacher Austausch von Bausteinen und Technologien
  - Redundanzen in Kauf nehmen
- Skalierbarkeit
  - Hohe Anpassbarkeit an wechselnde Bedürfnisse



Fazit

- Es gibt nicht die eine flexible Softwarearchitektur
- Alles hat seine Vorteile, aber auch Kompromisse/Herausforderungen
- Stärken/Schwächen kennen, um gute Entscheidungen treffen zu können
  - Aber: Anforderungen ändern sich
  - Erfahrungen bauen sich auf

Schlusswort(e)

- Softwarearchitekturarbeit bleibt weiterhin anspruchsvoll
- Moderne Technologien, Architekturstile/-muster eröffnen neue Optionen, machen es aber nicht per se einfacher

Vielen Dank.

Ich freue mich auf Eure Fragen!

Falk Sippach

fs@embarc.de

@sippsack

➔ xing.to/fsi