embarc
Software Consulting GmbH

Mit der Zeit gehen ...

Flexible Architekturen

FALK SIPPACH // EMBARC

BaselOne 2021

Dienstag, 21. Oktober 2021

BaselOne21

1

2

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

3

Mit der Zeit gehen - Flexible Architekturen

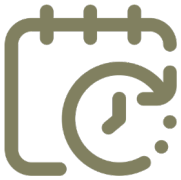
In der agilen Softwareentwicklung gibt es die Rolle des Architekten eigentlich nicht mehr. Paradoxerweise sind die Herausforderungen an den Entwurf moderner Softwaresysteme höher als früher. Um die Digitalisierung voranzutreiben, sich von den Mitbewerbern abzuheben, sich möglichst auch einen Wettbewerbsvorteil 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.

4

Agenda



1 Motivation

2 Anforderungen klären

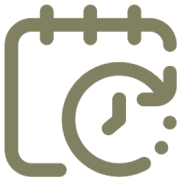
3 Flexible Lösungsansätze

4 Architektur festhalten

5 Ausblick

5

Agenda



1 **Motivation**

2 Anforderungen klären

3 Flexible Lösungsansätze

4 Architektur festhalten

5 Ausblick

1

6

7

8

Experten zu: Was ist Softwarearchitektur?

> *"… 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)

9

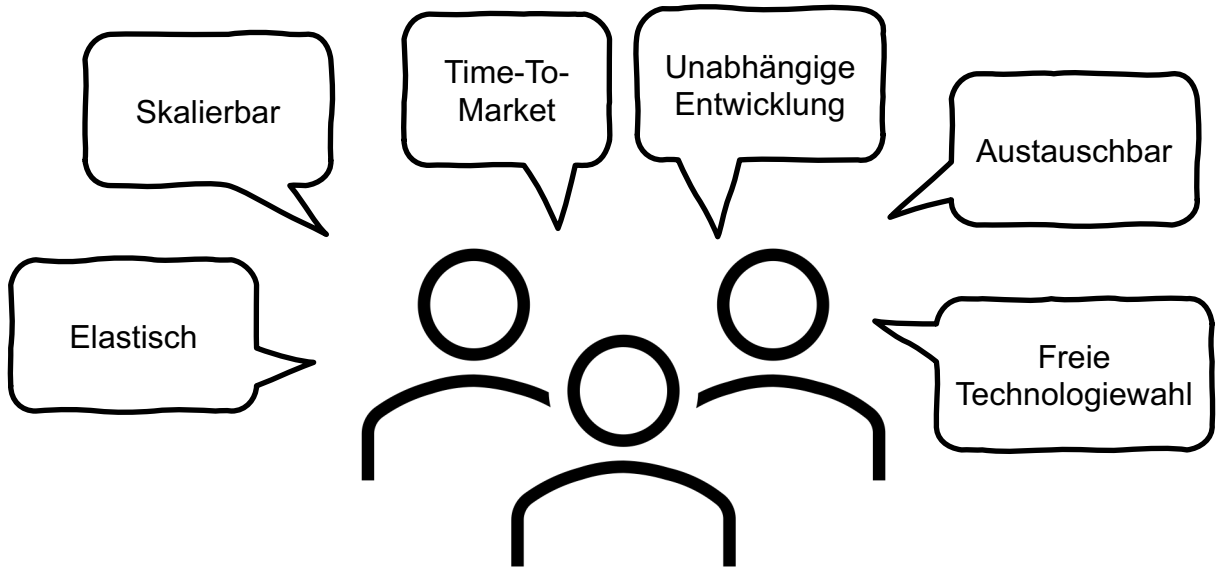
Was ist Architektur für mich?

fundamentale Strukturen, Konzepte, Entscheidungen und Lösungsansätze

… die man nicht mehr leicht los bekommt!

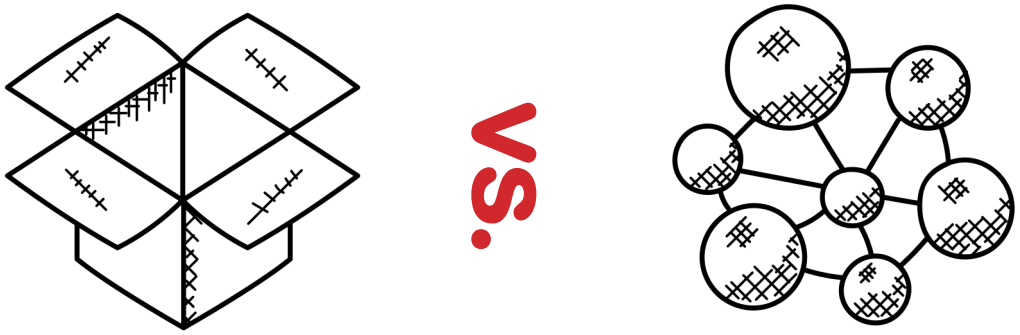
10

Was heißt nun eigentlich flexibel?



11

Der Klassiker



Laufzeitmonolith

Microservices

12

Warum den ~~Partner~~ Monolith loswerden?

> *“Ich verstehe häufig nicht, was er eigentlich will.”*
>
> ***(Wartbarkeit)***

> *“Er wird immer mehr zur Couch Potato.”*
>
> ***(Erweiterbarkeit)***

> *“Auf spontanen Parties mit vielen Leuten ist er überfordert.”*
>
> ***(Skalierbarkeit)***

> *“Ich würde gerne öfter mal was Neues ausprobieren. Das geht mit ihm nicht ...”*
>
> ***(Neue Technologien)***

> *“Er klammert mir zu sehr.”*
>
> ***(Unabhängigkeit)***

Folie aus: „Wie werde ich ihn los – in 10 Tagen? Vom Monolithen zu Microservices“ von Harm Gnoyke (IT-Tage 2016)

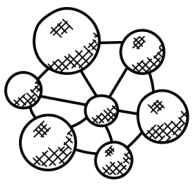
13

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

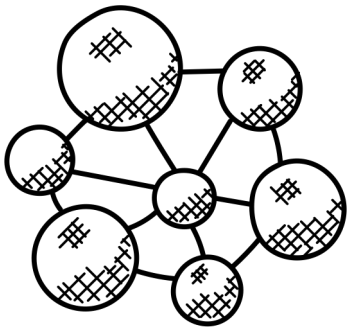
14

Warum finden wir Microservices toll?

Schnelle Anpassungen,
Teams agieren **unabhängig**

Elastisch auf
Lastanforderungen reagieren

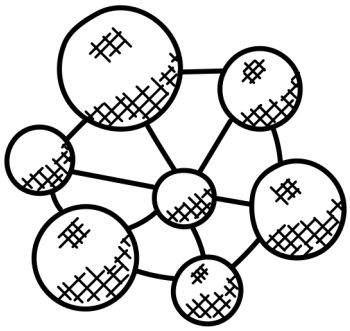
Technologische Trends
schnell aufgreifen



15

Microservices in der Praxis

CORONA
WARN-APP



https://coronawarn.app/ https://github.com/corona-warn-app

16

17

Aber: Sind die Services gut geschnitten?

“If you can’t build a well-structured monolith, what makes you think you can build a well-structured set of microservices?”

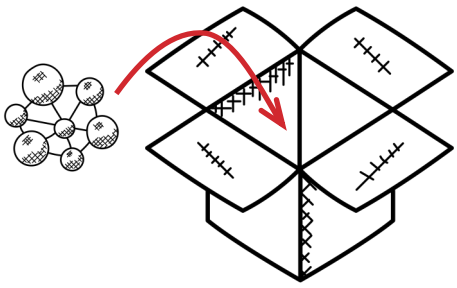
Simon Brown

18

Oder etwas drastischer ausgedrückt …

19

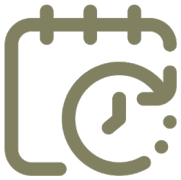
Kompromisse/Alternativen



Modulith

20

Agenda



1 Motivation

2 **Anforderungen klären**

3 Flexible Lösungsansätze

4 Architektur festhalten

5 Ausblick

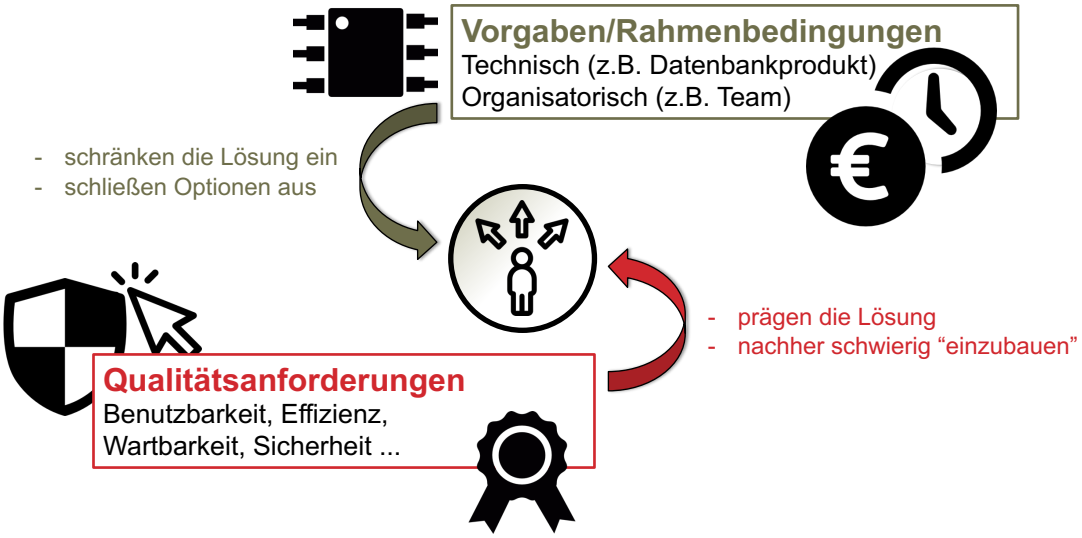
2

21

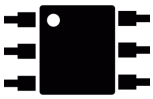
aus: Effektive Software-architekturen (G. Starke)

22

Einflüsse auf Entscheidungen



Arten von Rahmenbedingungen



Technologisch

Hardware-Vorgaben
Software-Vorgaben
Vorgaben des Systembetriebs
Programmiervorgaben

Organisatorisch

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

Zentrale Rahmenbedingungen Corona-Warn-App

Technisch

- Betrieb in der **Cloud**
- **Native mobile** Clients
- Einsatz des **Exposure Notification Framework**

Organisatorisch

- **Große Medienaufmerksamkeit**, gewisse **Skepsis** am Mehrwert innerhalb der Bevölkerung
- Konsortium aus **zwei Auftragnehmern** (SAP und Deutsche Telekom)
- **Enger Zeitrahmen**
- Hoher **politischer Druck**, viele Parteien involviert (Ministerien, Behörden, RKI)
- **Hohe Datenschutzanforderungen**

25

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.

26

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? ...

27

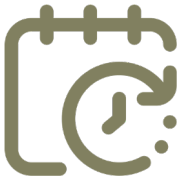
Top-Qualitätsziele Corona-Warn-App

	Ziel	Beschreibung
	Höchster Datenschutz	Der Schutz der personenbezogenen Daten hat oberste Priorität. *(Sicherheit)*
	Effektive Warnfunktionalität	Die App ist ein effektiver Baustein bei der Pandemie-Bekämpfung. *(Funktionale Eignung)*
	Attraktive Lösung für App-Nutzer	Die App ist leicht zu installieren sowie intuitiv und effizient zu bedienen. *(Benutzbarkeit)*
	Hohe Zuverlässigkeit	Die Lösung geht mit Lastspitzen wegen hoher Nutzer- oder Infektionszahlen ebenso souverän um, wie mit böswilligen Angriffen. *(Zuverlässigkeit)*
	Gute Änderbarkeit	Die Software lässt sich leicht anpassen, wenn z. B. Nutzer/-innen oder neue Forschungsergebnisse es erfordern. *(Wartbarkeit/Erweiterbarkeit)*

Die Reihenfolge gibt Orientierung bezüglich der Wichtigkeit.

28

Agenda



1 Motivation

2 Anforderungen klären

3 **Flexible Lösungsansätze**

4 Architektur festhalten

5 Ausblick

3

29

Beschreibung

In der Softwareentwicklung gibt es die Rolle des Architekten eigentlich nicht mehr. Pa… Herausforderungen an den Entwurf moderner Softwaresysteme höher als f… anzutreiben, sich von den Mitbewerbern abzuheben, sich … zu erarbeiten und die sowohl technische als auch … zu den altbekannten ganz neue

- Server…
- Relationale Da…
- Lang- oder Kurzlebigkeit

Je nach eingesetzten Architekturstilen u… Herausforderungen auf euch zu. Wir diskutieren … und vor allem auch robuste Softwarearchitektur entwerfen, pflegen können. Und das auch, wenn von der grünen Wiese nach e… zu sehen ist.

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

30

High Level Lösungsansätze

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

31

Einordnung Lösungsansätze

Konservativ?

Laufzeitmonolith
Klassischer Betrieb
On Premises
Relationale DBs
Langlebigkeit

Modern?

Microservices
DevOps
Serverless/Cloud
NoSQL/In-Memory
Kurzlebigkeit

32

Stärken und Herausforderungen

Konservativ?

- 👎 "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

Modern?

33

Typische Rahmenbedingungen

Konservativ?

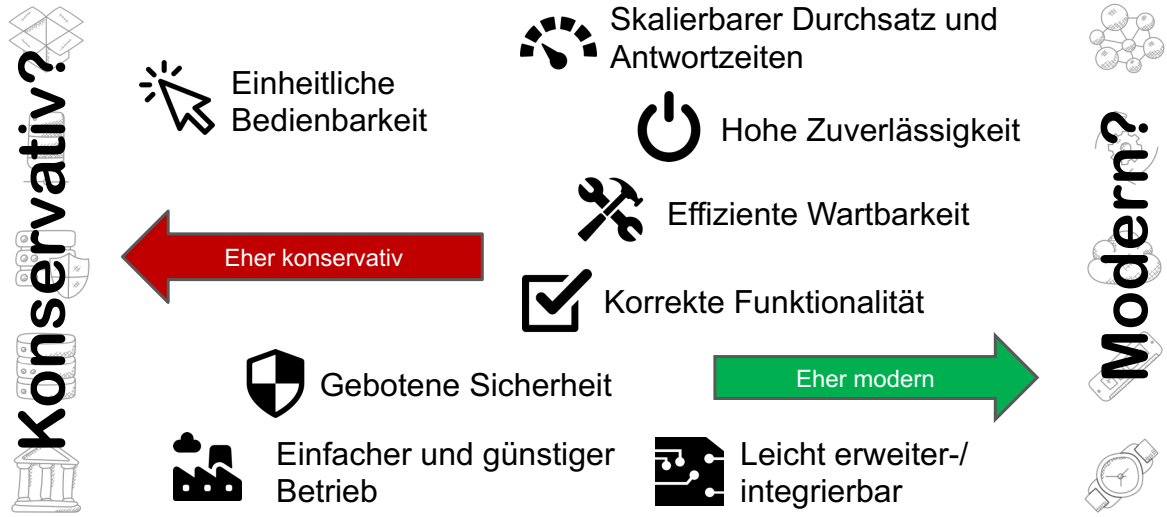
- Klassische Organisationsstrukturen
- Lange Produktlaufzeiten und Kundenbindungen
- Datenschutz-/sicherheit
- Unflexibler Inhouse-Betrieb
- Keine agilen Prozesse
- Langlaufende Lizenzverträge

- Cloudausrichtung/-guidelines
- (Public) Cloud first Ansatz
- Agile Prozesse etabliert
- Offene Mitarbeiter
- Ausreichende Entwicklungskapazitäten
- Fortgeschrittene Automatisierung und Infrastruktur als Code

Modern?

34

Typische Qualitätsziele

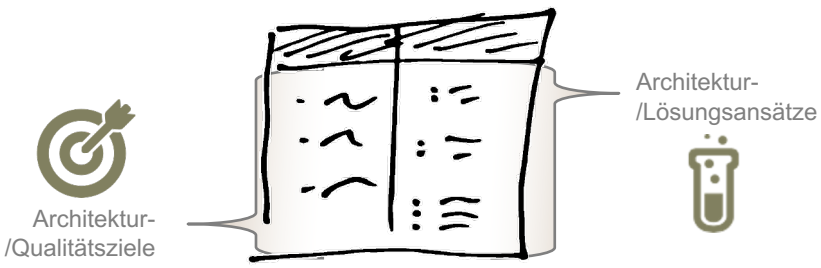


35

Lösungsstrategie

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

Form: Tabelle ([Ziele | Lösungsansätze]).



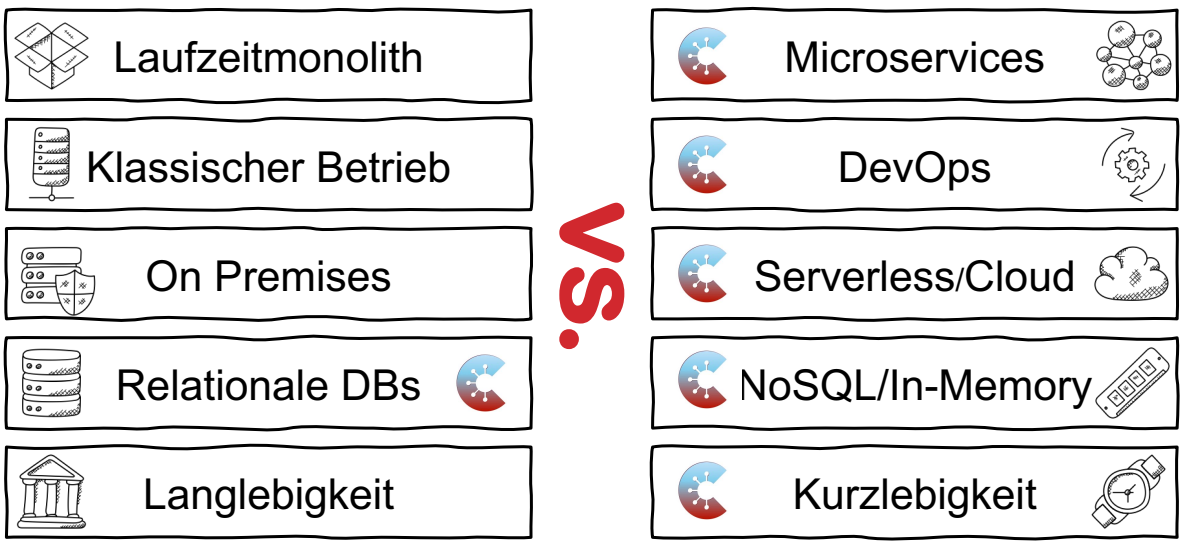
36

Lösungsstrategie Corona-Warn-App

Ziel	Passende Lösungsansätze (Auswahl)
Höchster Datenschutz	• Speicherung der **Daten lokal** • **Verschlüsselung** aller Bewegungsdaten • Senden der Daten nur **nach Aufforderung** • **Transparente** Entwicklung (Open Source)
Effektive Warnfunktionalität	• Verwendung **Exposure Notification Framework** • **Digitale Abläufe** bevorzugt • Optionales, manuelles **Kontakt-Tagebuch**
Attraktive Lösung für App-Nutzer	• **Native Clients** (L&F) • **Übersichtliche** Gestaltung und **simple** Bedienung
Hohe Zuverlässigkeit	• **Microservices,** Docker, Kubernetes, **Public Cloud** • hohe **Testabdeckung** und **automatisierte Builds** • Bereitstellung von zu lesenden Daten über **CDN**
Gute Änderbarkeit	• Hoher **Modularisierungsgrad** • **Open Source** Projekt, gute **Dokumentation** • Verwendung von **Standard** & **Open Source** Libraries • Konsortium von **mehreren Auftragnehmern** • **Code-Qualität** (SonarQube, SwiftLint, Checkstyle, …)

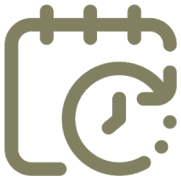
37

Lösungsansätze bei der Corona Warn App



38

Agenda



1 Motivation

2 Anforderungen klären

3 Flexible Lösungsansätze

4 **Architektur festhalten**

5 Ausblick

4

44

Dokumentation des Entwurfs

Architekturüberblick

- Mission Statement
- Kontextabgrenzung

Einflüsse

- Vorgaben ✓
- Architekturziele ✓

Lösungsansätze ✓

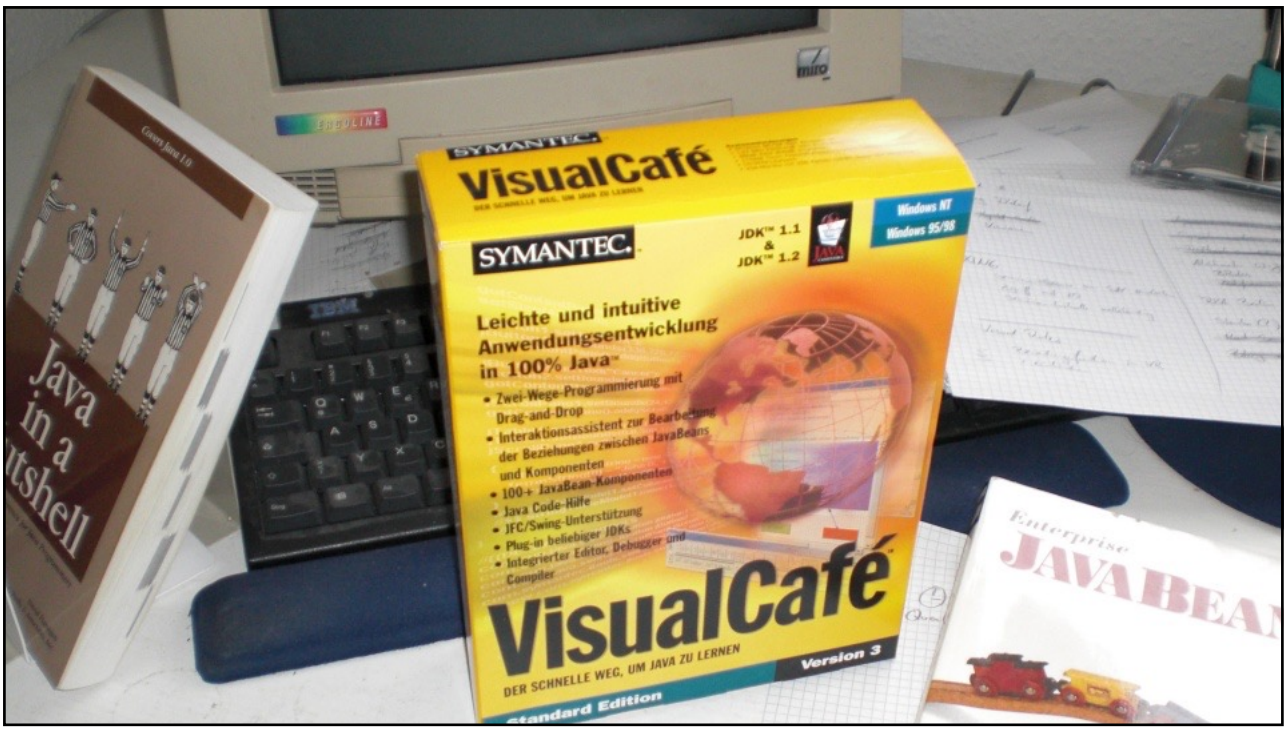
- Architekturstil
- Technologie-Stack
- Konzepte
- Prinzipien
- Zerlegung
- ....

45

Was ist ein Architekturüberblick?

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

46



47

Mission Statement

CORONA WARN-APP

> Die **Corona-Warn-App** ist eine App, die hilft, **Infektionsketten** des SARS-CoV-2 (COVID-19-Auslöser) in Deutschland **nachzuverfolgen** und zu **unterbrechen**. Die App basiert auf Technologien mit einem **dezentralisierten Ansatz** und informiert Personen, wenn sie mit einer infizierten Person in Kontakt standen. **Transparenz** ist von entscheidender **Bedeutung**, um die Bevölkerung zu schützen und die **Akzeptanz zu erhöhen**.

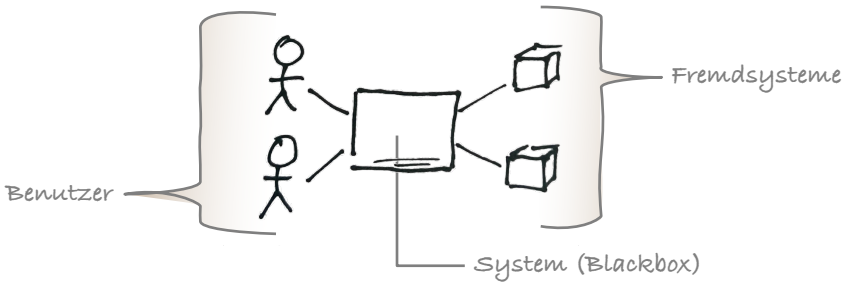
Quelle: https://www.coronawarn.app/de/

48

Kontextabgrenzung

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

Form: Graphik, ergänzt um kurze Beschreibungen.

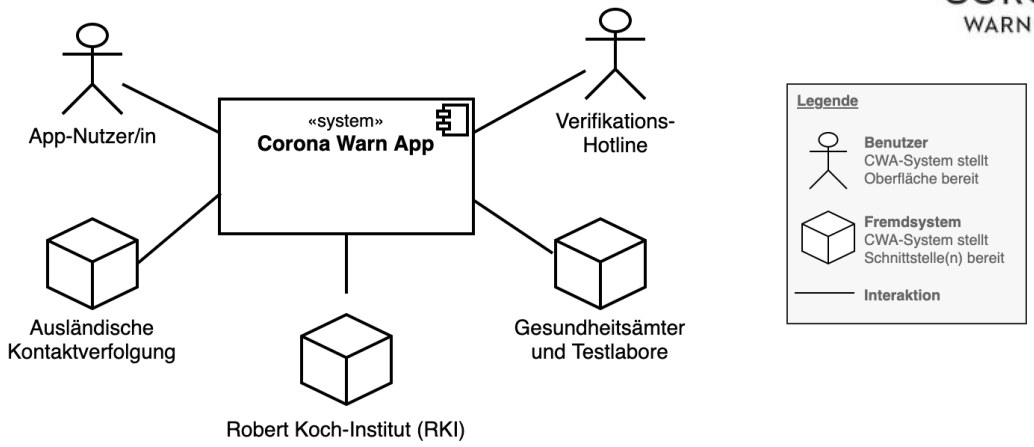


49

Kontextabgrenzung

Dieser fachliche Systemkontext zeigt das Corona-Warn-App-System im Zusammenspiel mit den wichtigsten Benutzern und Fremdsystemen.

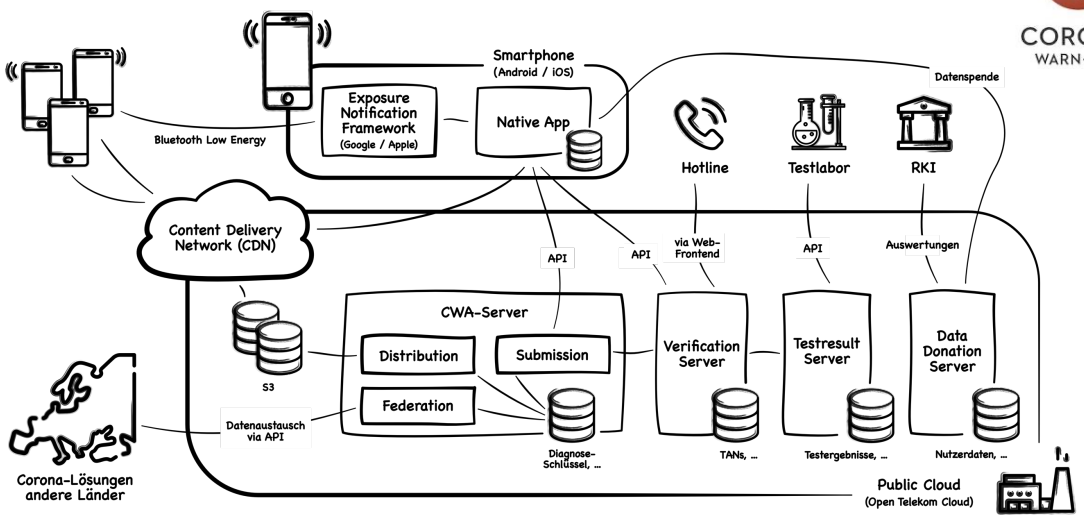
CORONA WARN-APP



50

Informelles Überblicksbild

CORONA WARN-APP



Quelle der Abbildung: S. Zörner, F. Sippach: „So gehen Architektur-Reviews! Entlang der Corona-Warn-App", OOP 2021

52

„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

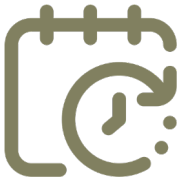
12. Glossar

arc42

➔https://arc42.de/

53

Agenda



1 Motivation

2 Anforderungen klären

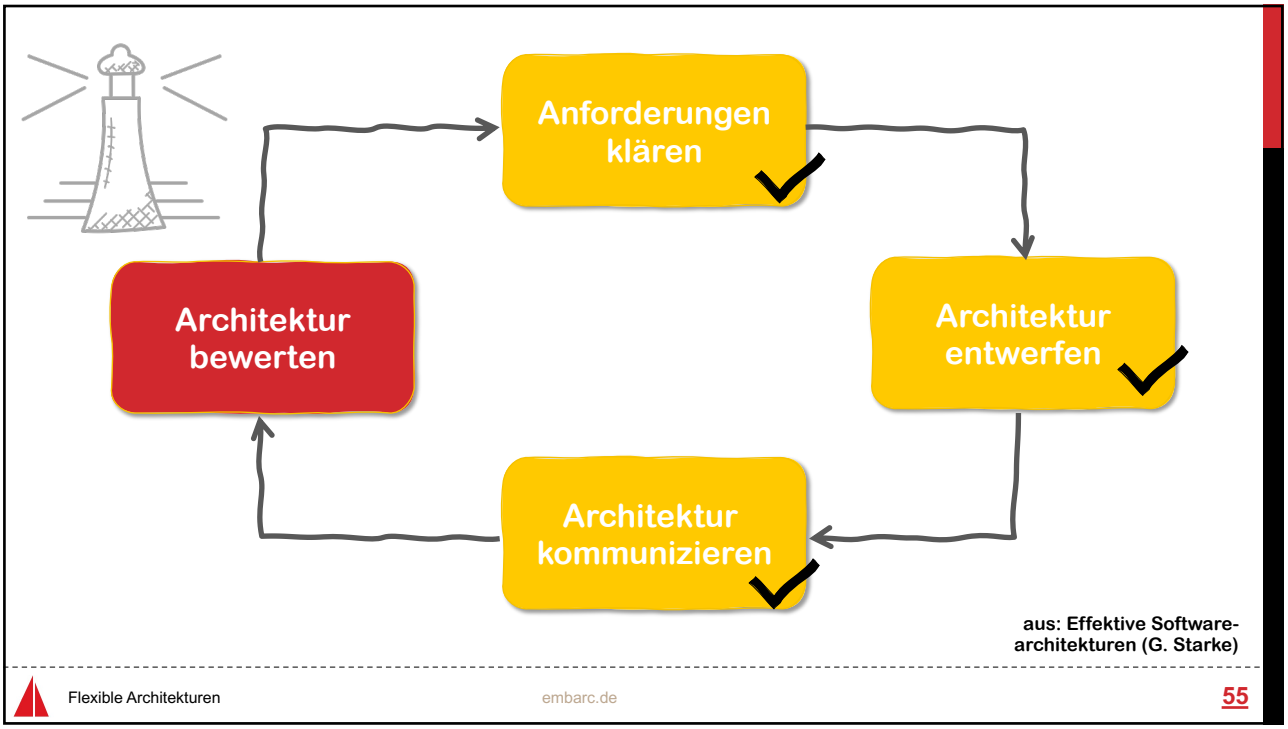
3 Flexible Lösungsansätze

4 Architektur festhalten

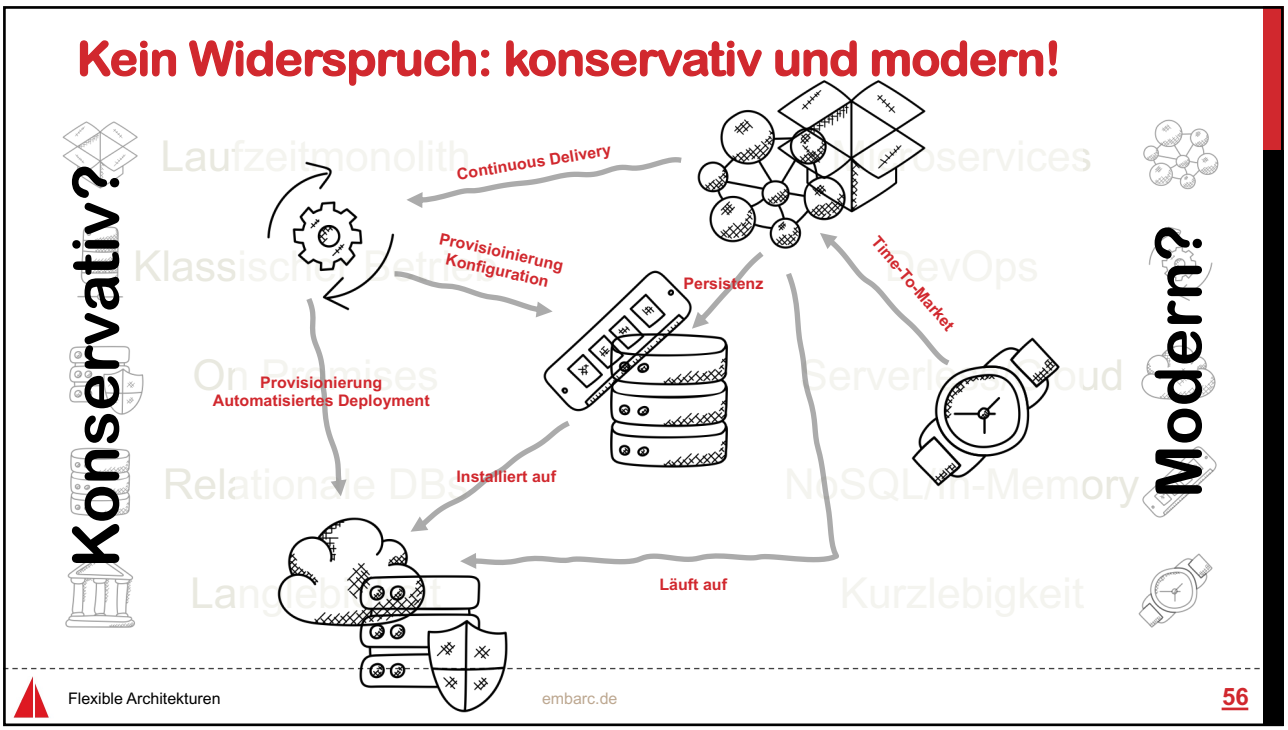
5 **Ausblick**

5

54


Anforderungen klären
Architektur entwerfen
Architektur kommunizieren
Architektur bewerten
aus: Effektive Software-architekturen (G. Starke)
Flexible Architekturen
embarc.de
55

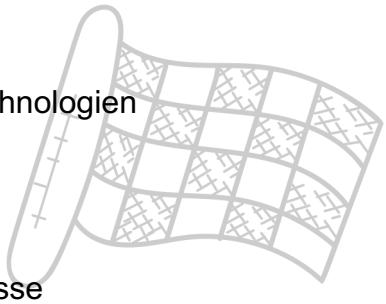
55


Kein Widerspruch: konservativ und modern!
Konservativ?
Modern?
Continuous Delivery
Provisioinierung Konfiguration
Persistenz
Time-To-Market
Provisionierung Automatisiertes Deployment
Installiert auf
Läuft auf
Laufzeitmonolith
Microservices
Klassisches Betriebsmodell
DevOps
On Premises
Serverless Cloud
Relationale DB
NoSQL/In-Memory
Langlebigkeit
Kurzlebigkeit
Flexible Architekturen
embarc.de
56

56

Was bleibt/wird wichtig?

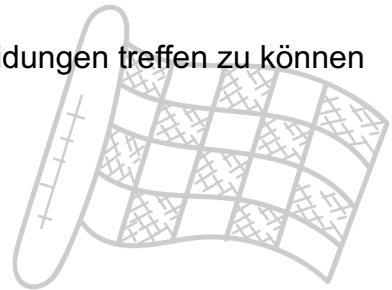
- **Schnelligkeit**
  - Zügige Entwicklung und Ausrollen für schnelle Anpassung an den Markt
  - Fehler akzeptieren, daraus lernen
- **Flexibilität**
  - Einfacher Austausch von Bausteinen und Technologien
  - Redundanzen in Kauf nehmen
- **Skalierbarkeit**
  - Hohe Anpassbarkeit an wechselnde Bedürfnisse



57

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



58

Schlusswort(e)

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

59

Vielen Dank.

Ich freue mich auf Eure Fragen!

Falk Sippach

fs@embarc.de

@sippsack

➔ xing.to/fsi

60


Basler Versicherungen
bluesky
CONSIDER IT DONE
KARAKUN
sympany versicherungen
magnolia
JAVA USER GROUP CH
ELCA
We make it work.
Gradle
OPTRAVIS
Transfer Pricing Solutions
#BaselOne21
baselone.ch

61