KI braucht Architektur

Warum Docs-as-Code jetzt unverzichtbar wird

Falk Sippach

16.09.2026
Software Architecture Meetup Nürnberg

0

Abstract

KI braucht Architektur: Warum Docs-as-Code jetzt unverzichtbar wird

Architekturdokumentation ist in vielen Projekten wichtig, aber oft schwer auffindbar, veraltet oder zu weit vom Entwicklungsalltag entfernt. Gleichzeitig halten KI-Werkzeuge und LLMs Einzug in Softwareentwicklung, Reviews und Wissensarbeit. Und sie brauchen dafür verlässlichen Kontext. Genau hier entsteht ein Problem: Wenn Architekturwissen nur implizit in Köpfen, Tickets, Chats oder verstreuten Wikis existiert, kann KI es nicht sinnvoll nutzen. Im schlimmsten Fall erzeugt sie plausible, aber falsche Antworten über Architekturentscheidungen, Qualitätsziele, Schnittstellen oder Systemgrenzen.

Docs-as-Code bietet eine pragmatische Grundlage: Architekturwissen liegt nahe am Code, ist versioniert, review- und automatisierbar. ADRs, arc42-Dokumentation, C4-Diagramme, OpenAPI-Spezifikationen und Architekturregeln können so nicht nur von Menschen gelesen, sondern auch als Kontext für LLMs genutzt werden. Der Vortrag zeigt, wie KI bei Architekturdokumentation konkret helfen kann: beim Formulieren und Reviewen von ADRs, beim Zusammenfassen technischer Diskussionen, beim Erkennen veralteter Dokumentation und beim Ableiten prüfbarer Architekturregeln. Denn KI ersetzt Architekturdokumentation nicht – sie macht gute, strukturierte und überprüfbare Dokumentation wichtiger denn je.

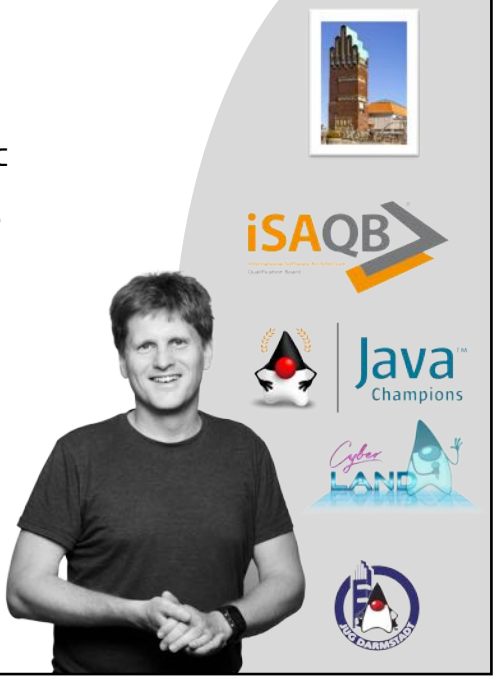
1

Falk Sippach

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

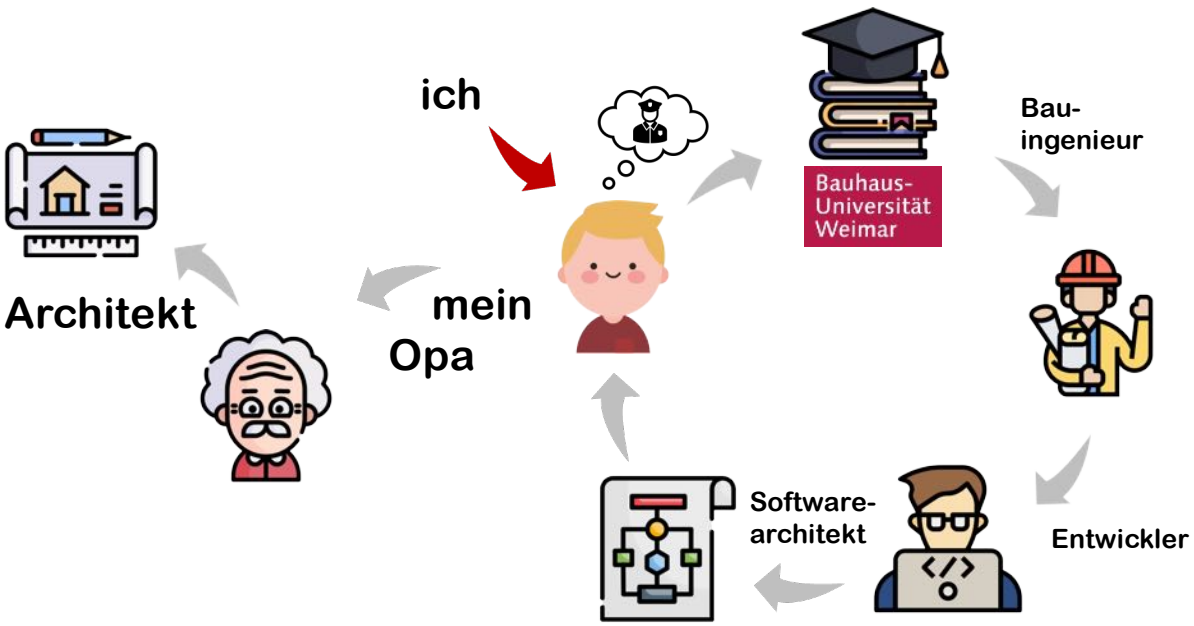
Schwerpunkte

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



embarc

2



3

Teil der (Java) Community

Nächster Termin am 17.09.

https://www.jug-da.de/

monatliche Opensource Code Camps

https://cyberland.ijug.eu/

4

5

6

Aber **schreiben wir** in Zukunft **überhaupt noch** Architekturdokumentation?

7

8

Hands-up:

Welcher KI-Typ bist du?

9


SKEPTICAL DEVELOPERS BUILD BETTER SOLUTIONS
Better Software Together
SAME PEOPLE BRIGHTER IDEAS
Der Skeptiker
Die Neugierige
Der Power-User
Good Code Better Humans
Curiosity builds better software
More Prompts Better Products
Prompt Engineering
AI for Developers
Building with LLMs
Clean Code
Design Patterns
Refactoring
AI Assistant
Generating...

10


SKEPTICAL DEVELOPERS BUILD BETTER SOLUTIONS
Better Software Together
SAME PEOPLE BRIGHTER IDEAS
AI Assistant
Plan
Build
Test
Deploy
Improve
Szenario 1: Wird uns die KI bald komplett ersetzen?

11

... oder (Szenario 2) räumen wir in 20 Jahren den heute durch KI erzeugten Code auf?

12

Was machen die Junioren von heute dann?

13

Was machen die Junioren von heute dann?

14

Was ist Softwarearchitektur?

Softwarearchitektur :=

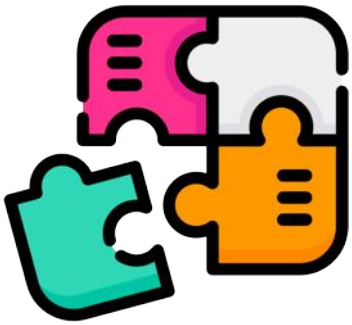
Σ wichtige Entscheidungen

wichtig =
- fundamental (betrifft viele)
- im weiteren Verlauf nur schwer zu ändern
- entscheidend für den Erfolg Eures Softwaresystems

15

Problem- und Lösungsraum

Anforderungen/
Architekturziele



Lösungsansätze/
Entscheidungen

16

Zutaten

Systemkontext

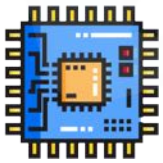
Metapher
"Produktkarton"

Entscheidungen

Architektur-
Stil/Muster/...

Vorgaben

Qualitätsziele



Technologien

Informeller Überblick

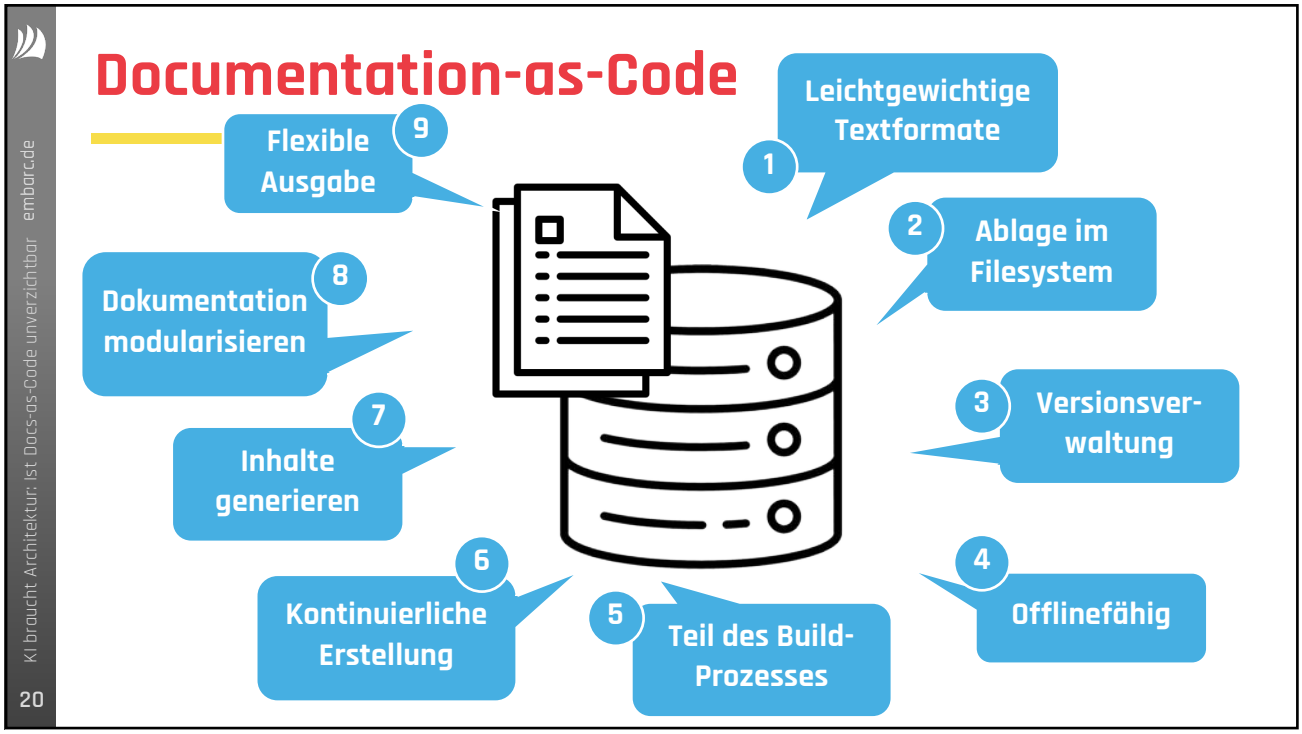
17

https://www.isaqb.org/

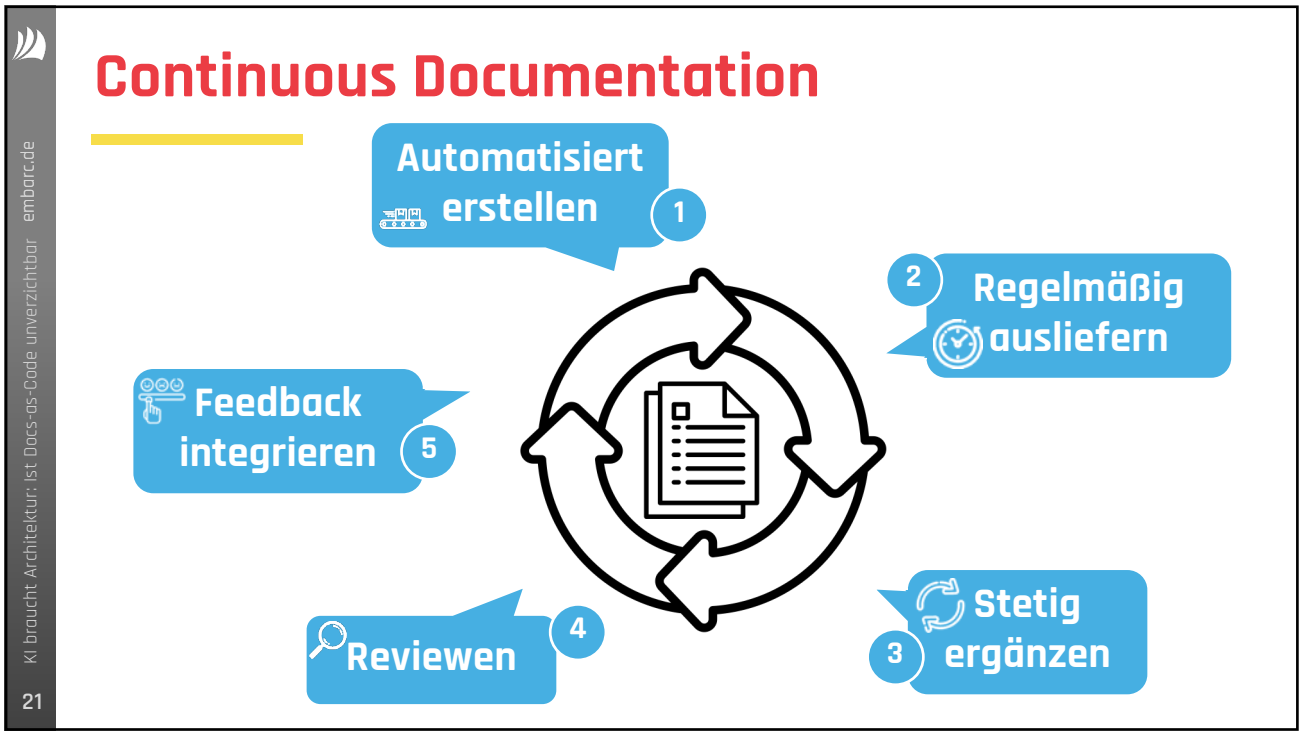
(Unser) täglich Entwickler-Brot

- Plain-Text
- Entwicklungsumgebung
- Kommandozeilen-werkzeuge
- **Versionsverwaltung**

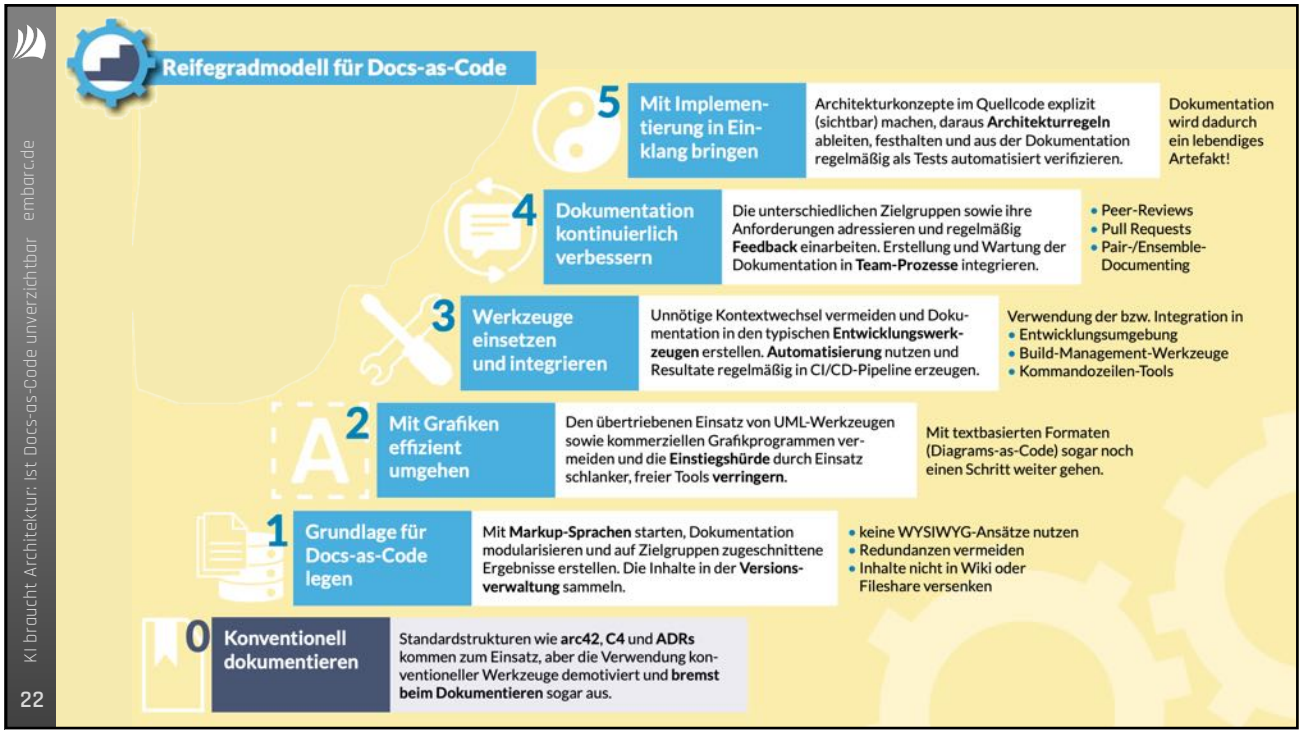
Foto von geralt: https://pixabay.com/de/unternehmer-start-start-up-karriere-696976/ (CCo Public Domain Lizenz)


Documentation-as-Code
1 Leichtgewichtige Textformate
2 Ablage im Filesystem
3 Versionsver-waltung
4 Offlinefähig
5 Teil des Build-Prozesses
6 Kontinuierliche Erstellung
7 Inhalte generieren
8 Dokumentation modularisieren
9 Flexible Ausgabe
KI braucht Architektur: Ist Docs-as-Code unverzichtbar embarc.de
20

20


Continuous Documentation
1 Automatisiert erstellen
2 Regelmäßig ausliefern
3 Stetig ergänzen
4 Reviewen
5 Feedback integrieren
KI braucht Architektur: Ist Docs-as-Code unverzichtbar embarc.de
21

21


KI braucht Architektur: Ist Docs-as-Code unverzichtbar embarc.de
22
Reifegradmodell für Docs-as-Code
5
Mit Implementierung in Einklang bringen
Architekturkonzepte im Quellcode explizit (sichtbar) machen, daraus Architekturregeln ableiten, festhalten und aus der Dokumentation regelmäßig als Tests automatisiert verifizieren.
Dokumentation wird dadurch ein lebendiges Artefakt!
4
Dokumentation kontinuierlich verbessern
Die unterschiedlichen Zielgruppen sowie ihre Anforderungen adressieren und regelmäßig Feedback einarbeiten. Erstellung und Wartung der Dokumentation in Team-Prozesse integrieren.
• Peer-Reviews
• Pull Requests
• Pair-/Ensemble-Documenting
3
Werkzeuge einsetzen und integrieren
Unnötige Kontextwechsel vermeiden und Dokumentation in den typischen Entwicklungswerkzeugen erstellen. Automatisierung nutzen und Resultate regelmäßig in CI/CD-Pipeline erzeugen.
Verwendung der bzw. Integration in
• Entwicklungsumgebung
• Build-Management-Werkzeuge
• Kommandozeilen-Tools
2
Mit Grafiken effizient umgehen
Den übertriebenen Einsatz von UML-Werkzeugen sowie kommerziellen Grafikprogrammen vermeiden und die Einstiegshürde durch Einsatz schlanker, freier Tools verringern.
Mit textbasierten Formaten (Diagrams-as-Code) sogar noch einen Schritt weiter gehen.
1
Grundlage für Docs-as-Code legen
Mit Markup-Sprachen starten, Dokumentation modularisieren und auf Zielgruppen zugeschnittene Ergebnisse erstellen. Die Inhalte in der Versionsverwaltung sammeln.
• keine WYSIWYG-Ansätze nutzen
• Redundanzen vermeiden
• Inhalte nicht in Wiki oder Fileshare versenken
0
Konventionell dokumentieren
Standardstrukturen wie arc42, C4 und ADRs kommen zum Einsatz, aber die Verwendung konventioneller Werkzeuge demotiviert und bremst beim Dokumentieren sogar aus.

22


KI braucht Architektur: Ist Docs-as-Code unverzichtbar embarc.de
23
Fünf Thesen.
Ein roter Faden.
Was Architekturwissen leisten muss,
damit KI sinnvoll mitarbeiten kann.

23

KI braucht mehr als Code

Code zeigt, was gebaut wurde. Architekturwissen erklärt, warum.

24

Code zeigt das Was – nicht das Warum

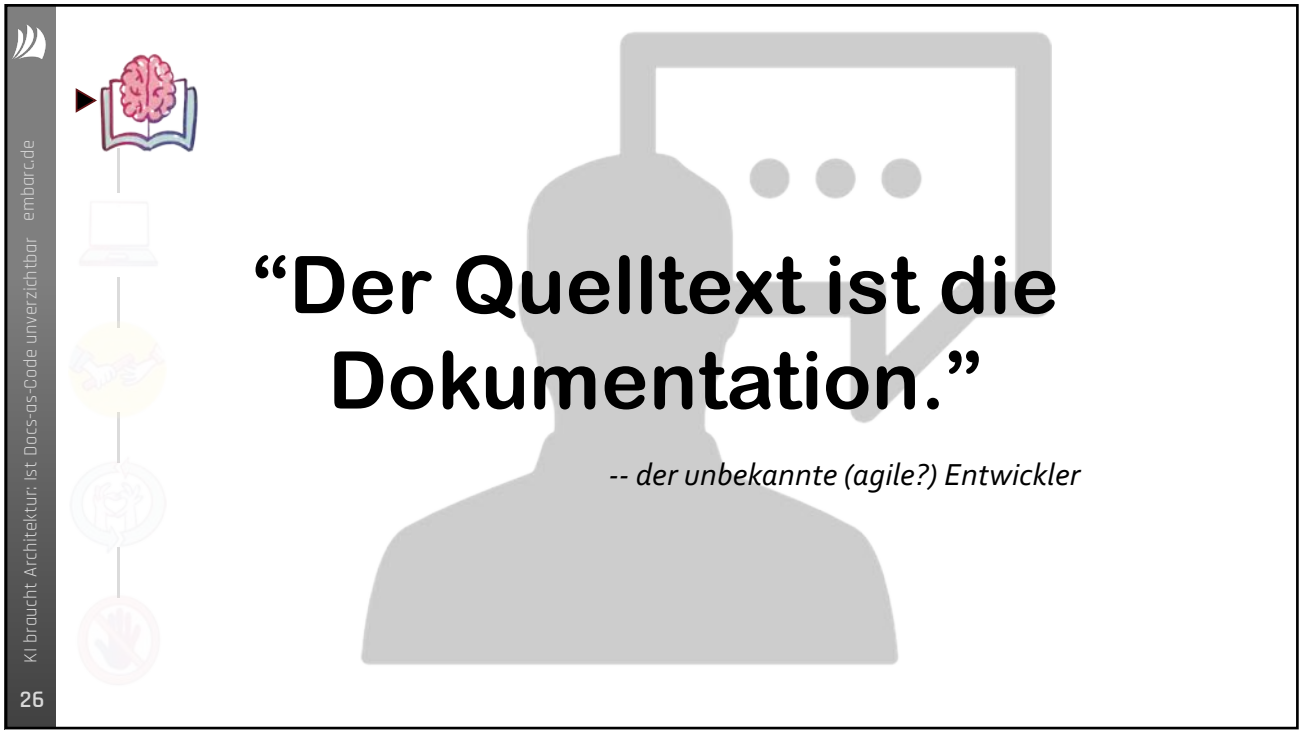
Strukturen
Module · Abhängigkeiten

Verhalten
Abläufe · Schnittstellen

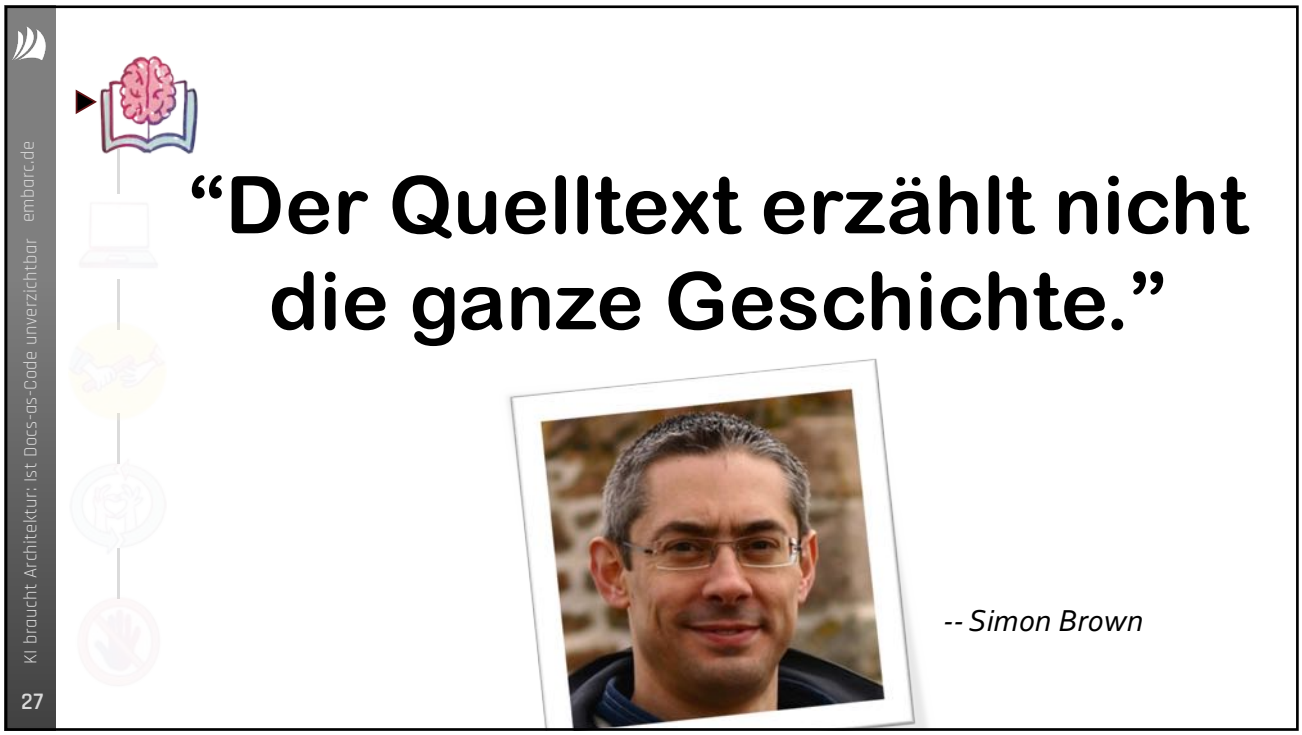
Gründe
Ziele · Zwänge · Alternativen

Code dokumentiert Ergebnisse – nicht die Gründe dahinter.

25


KI braucht Architektur: Ist Docs-as-Code unverzichtbar embarc.de
“Der Quelltext ist die Dokumentation.”
-- der unbekannte (agile?) Entwickler
26

26


KI braucht Architektur: Ist Docs-as-Code unverzichtbar embarc.de
“Der Quelltext erzählt nicht die ganze Geschichte.”
-- Simon Brown
27

27

Architekturwissen ist verteilt

Artefakte
ADRs · Tickets · Wikis

Gespräche
Reviews · Chats · Meetings

Köpfe
Erfahrung · implizite Regeln

Was nicht auffindbar und verständlich ist, existiert für die KI nicht.

28

Fehlender Kontext macht KI überzeugend falsch

Plausible Antwort
klingt fachlich richtig

Verdeckte Vorgabe
Datenschutz · Latenz · Teams

Falsche Richtung
lokal logisch · global falsch

Plausibel ist nicht automatisch passend.

29

DeepWiki: Architektur aus dem Code?

Repository analysieren
Code · Struktur · README

Wiki generieren
Texte · Mermaid · Quellen

System befragen
Chat mit Codebezug

Starker Ist-Blick – das Warum bleibt Dokumentationsaufgabe.

https://www.embarc.de/deepwiki-architekturdokumentation/

30

Lesetipp: DeepWiki im Browser

https://www.embarc.de/deepwiki-architekturdokumentation/

31

Docs-as-Code wird zum KI-Kontext

Versioniert und strukturiert wird Dokumentation zur Arbeitsgrundlage für Menschen und KI.

32

Vom Dokument zum KI-Kontext

Textbasiert
Markdown · AsciiDoc · DSLs

Versioniert
gemeinsam mit dem Code

Adressierbar
Dateien · IDs · Links

Docs-as-Code macht Architekturwissen für Mensch und KI nutzbar.

33

Struktur schlägt Textmenge

Konventionen
Orte · Namen · Templates

Semantik
Status · Priorität · Owner

Beziehungen
Code · ADR · Diagramm

Die KI braucht Orientierung – nicht einfach mehr Text.

34

Der richtige Kontext zur richtigen Aufgabe

Aufgabe klären
Review · Änderung · Entscheidung

Kontext wählen
Ziele · ADRs · Modelle

Antwort belegen
Quellen · Annahmen · Lücken

Nicht das ganze Repository – der passende Ausschnitt.

35

KI ist Sparringspartner, nicht Entscheider

KI formuliert, strukturiert und hinterfragt. Menschen entscheiden und verantworten.

36

KI liefert Optionen – Menschen entscheiden

KI schlägt vor
Entwürfe · Alternativen · Fragen

Gemeinsam prüfen
Annahmen · Folgen · Konflikte

Mensch entscheidet
Priorität · Risiko · Verantwortung

Die Verantwortung lässt sich nicht prompten.

37

Qualitätsziele im Dialog

Kontext erfragen
System · Aufgaben · Stakeholder

Risiken erkunden
Brainstorming · Szenarien

Top-Ziele schärfen
max. 5 · iterativ · gemeinsam

Der Agent strukturiert den Dialog – Menschen priorisieren und entscheiden.

https://www.informatik-aktuell.de/entwicklung/methoden/ein-erster-ai-agent-fuer-die-softwarearchitektur-qualitaetsziele-im-dialog.html

38

Qualitätsmerkmale (Begriffe nach ISO 25010:2023)

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

Effizienz
(Performance Efficiency)
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? ...

Benutzbarkeit
(Interaction Capability)
Ist die Software intuitiv zu bedienen, wiedererkennbar, leicht zu erlernen, attraktiv? ...

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? ...

Übertragbarkeit
(Flexibility)
Ist die Software leicht auf andere Situationen oder Zielumgebungen (z.B. anderes OS) übertragbar? ...

Betriebssicherheit
(Safety)
Sind Personen, Tiere, Sachen und Umwelt vor Schäden durch die Software geschützt? ...

39

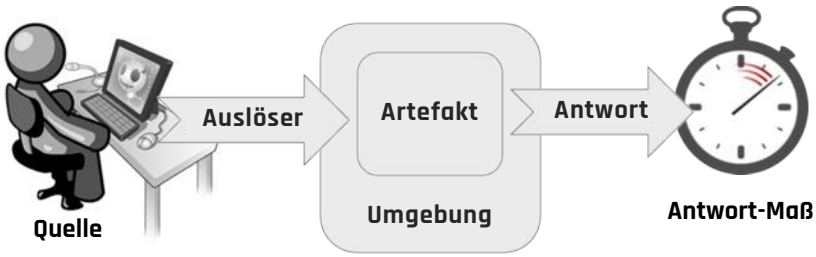
Qualitätsziele KI-generiert

Priorität	Qualitätsziel	ISO/IEC 25010:2023 Qualitätsmerkmal	Kurzbeschreibung
1	**Revisionssichere Nachvollziehbarkeit**	**Security** – insbesondere *Integrity*, *Non-repudiation* und *Accountability*	Jeder Spesenvorgang muss vom eingescannten Beleg über Regelentscheidungen, Rückfragen und Genehmigungen bis zur Auszahlung bzw. Kundenrechnung vollständig nachvollziehbar sein. Änderungen und Entscheidungen müssen manipulationsgeschützt, einer Person bzw. Regelversion zuordenbar und für Prüfungen schnell auffindbar bleiben.
2	**Korrekte Spesenverarbeitung**	**Functional suitability** – insbesondere *Functional correctness*	Steueranteile, geldwerte Vorteile, nicht erstattbare Beträge, Währungen sowie projekt- und landesspezifische Regeln müssen fachlich korrekt angewendet werden. Fehlerhafte Erstattungen oder falsche Kundenrechnungen verursachen unmittelbar finanzielle, steuerliche und vertrauensbezogene Schäden.
3	**Anpassbare Regelverarbeitung**	**Flexibility** – insbesondere	Fachliche Administratoren müssen steuerliche Vorgaben, Länderbesonderheiten, Organisationsstrukturen und Genehmigungsgrenzen kontinuierlich anpassen können. Das System soll außerdem schrittweise auf weitere Länder und hinzugekaufte

40

Qualitätsszenarien

Typische Bestandteile eines Qualitätsszenarios:



> *Eine angemeldete Schach-Polizistin ruft über das Internet den Bericht zu verdächtigen Spielern auf und erhält das Ergebnis innerhalb von 5 Sekunden.*

41

Qualitätsszenarien KI-generiert

Qualitätsziel	Qualitätsszenario
Revisionssichere Nachvollziehbarkeit	Ein Revisor prüft eine Kundenrechnung für ein Beratungsprojekt. BigSpender zeigt innerhalb von **5 Sekunden** alle zugehörigen Spesenvorgänge, lesbaren Belege, angewendeten Regelversionen, Bearbeitungsschritte und Übergaben an SAP vollständig und chronologisch an; bestätigte Historieneinträge sind nachträglich nicht veränderbar.
Korrekte Spesenverarbeitung	Ein eingescanntes Hotel- und Bewirtungsbelegpaket in Fremdwährung wird einem DACH-Beratungsprojekt zugeordnet. BigSpender wendet die zum Vorgangsdatum gültigen Länder-, Steuer- und Erstattungsregeln an, weist erstattungsfähige, nicht erstattungsfähige und weiterberechenbare Beträge getrennt aus und erzeugt in einer vorab definierten Regeltestsuite **keine fachlich falschen Buchungsdaten**.
Anpassbare Regelverarbeitung	Der fachliche Admin muss eine geänderte steuerliche Erstattungsregel für Deutschland einführen. Er hinterlegt, prüft und aktiviert die neue, versionierte Regel ohne Software-Deployment innerhalb von **einem Arbeitstag**; bereits abgeschlossene Vorgänge bleiben mit ihrer ursprünglich angewendeten Regelversion nachvollziehbar.
Ausfallsichere	Während ein Genehmiger eine Rückfrage speichert, fällt die Intranet-Web-Anwendung oder ihre Verbindung zur Datenbank aus. Nach Wiederherstellung innerhalb von **24 Stunden** ist der zuletzt bestätigte Workflow-Stand vollständig vorhanden; es entstehen weder doppelte Genehmigungen noch

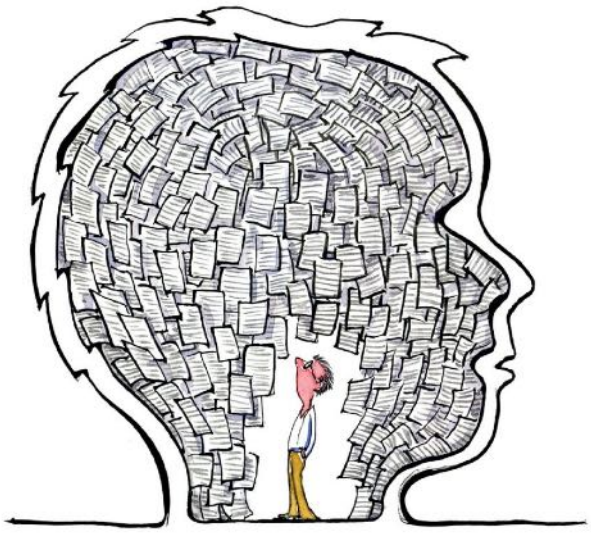
42

Lesetipp: Qualitätsziele im Dialog

https://www.informatik-aktuell.de/entwicklung/methoden/ein-erster-ai-agent-fuer-die-softwarearchitektur-qualitaetsziele-im-dialog.html

43

Ein Bild sagt mehr als 1000 Worte …



44

Sichten 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

Dr. Peter Hruschka
http://www.peterhruschka.eu/

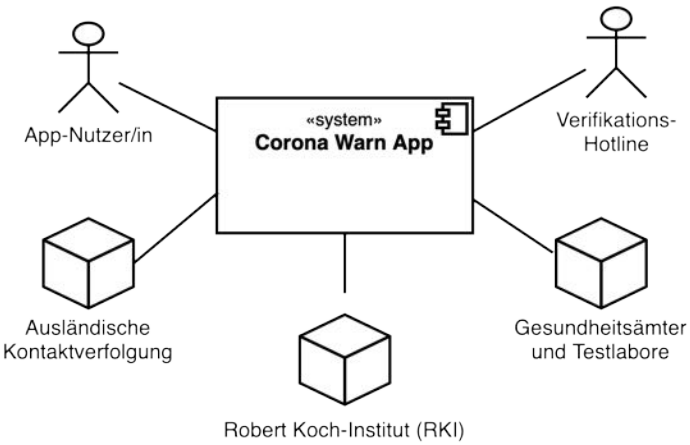
Dr. Gernot Starke
http://gernotstarke.de/

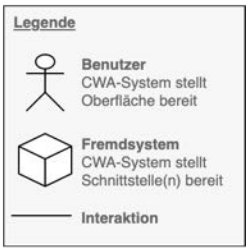
➔https://arc42.de/

45

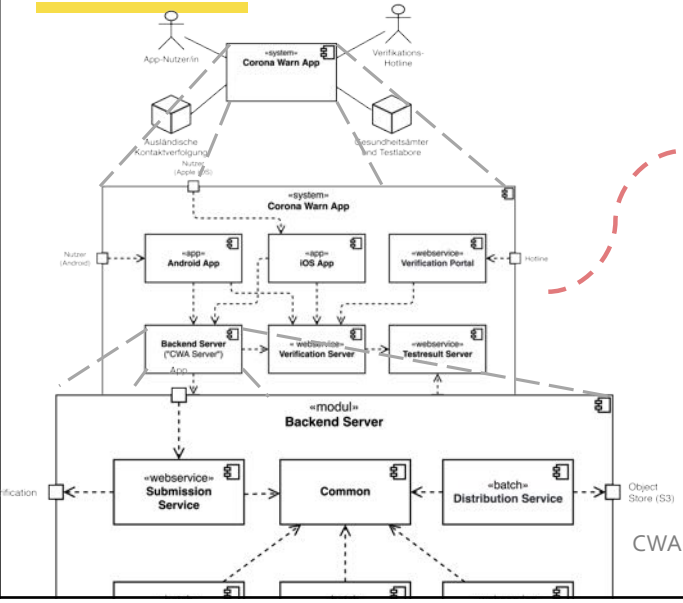
Kontextabgrenzung

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





Tatsächliche Zerlegung im Quelltext



„Bausteinsicht, Ebene 1"

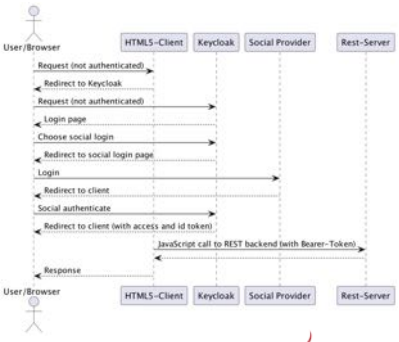
GitHub-Repositories

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

- cwa-app-android
- cwa-app-ios
- cwa-server („Backend")
- cwa-testresult-server
- cwa-verification-server
- cwa-verification-portal

CWA-Server (Backend), Bausteinsicht, Ebene 2

Sequenzdiagramm mit PlantUML

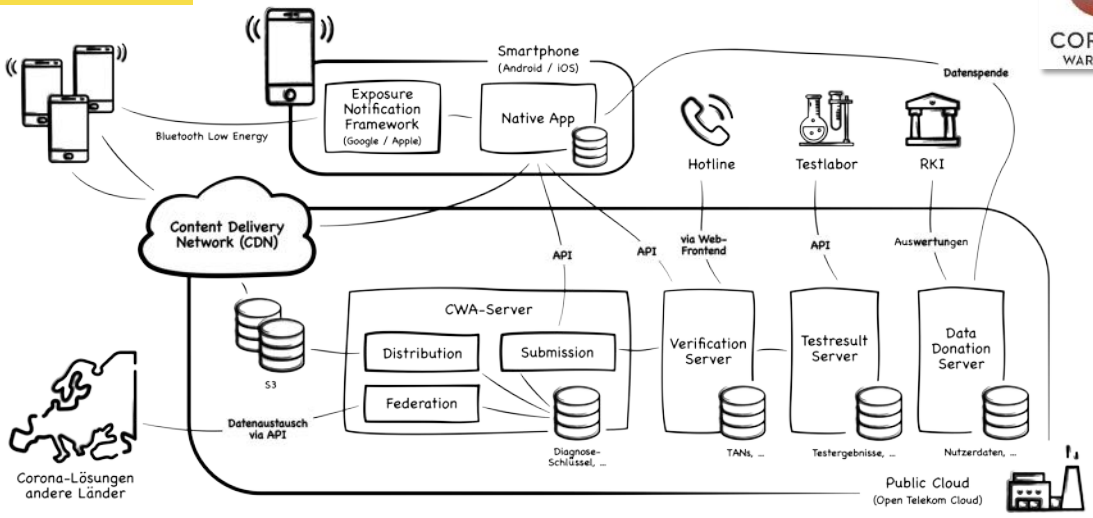


```
@startuml
actor browser as "User/Browser"
participant html5 as "HTML5-Client"
participant Keycloak
participant socialprovider as "Social Provider"
participant server as "Rest-Server"

browser -> html5 : Request (not authenticated)
browser <-- html5 : Redirect to Keycloak
browser -> Keycloak : Request (not authenticated)
browser <-- Keycloak : Login page
browser -> Keycloak : Choose social login
browser <-- Keycloak : Redirect to social login page
browser -> socialprovider : Login
browser <-- socialprovider : Redirect to client
browser -> Keycloak : Social authenticate
browser <-- Keycloak : Redirect to client (with access and id token)
html5 -> server : JavaScript call to REST backend (with Bearer-Token)
html5 <-- server
browser <-- html5 : Response
@enduml
```


48

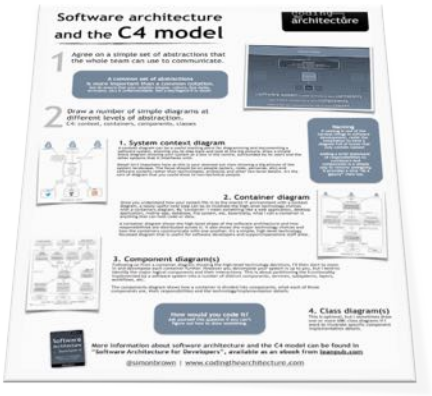
Informelles Überblicksbild



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

49

Alternative: C4 Model (Simon Brown)



- Agree on a simple set of abstractions that the whole team can use to communicate.
- Draw a number of simple diagrams at different levels of abstraction.
- C4: context, containers, components, classes (code)

➔ http://static.codingthearchitecture.com/c4.pdf

Evolution der Diagrammarten

Diagramm ist nicht gleich Diagramm …

Binärformat
JPG, PNG, …

Vektorgrafik
Visio, draw.io, …

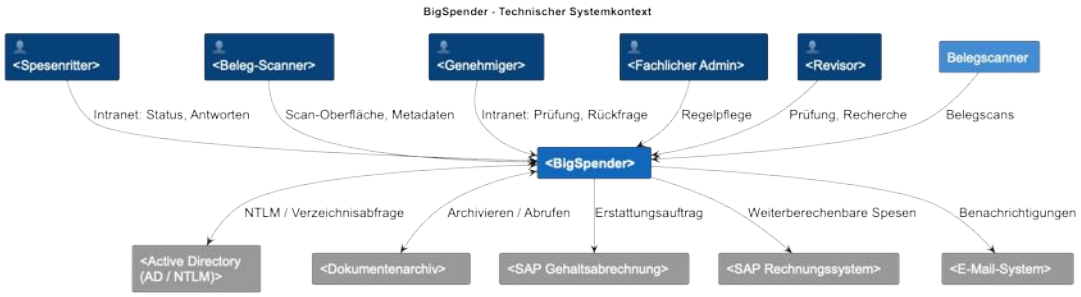
Diagrams-as-Code
PlantUML, Kroki, …

Vorschläge für Diagramme

Diagramm	Sicht / Zweck	Empfohlenes Werkzeug	Begründung
BigSpender auf einer Seite	Informelles Überblicksbild für die Folie: Beleg erfassen → Regeln prüfen → genehmigen → auszahlen bzw. weiterberechnen → revisionssicher prüfen.	**draw.io**	Für ein Präsentationsbild sind bewusst gestaltete Symbole, Farbflächen, reduzierte Beschriftungen und eine klare Prozessführung wichtiger als textbasierte Reproduzierbarkeit.
Technischer Systemkontext	Zeigt Benutzer, Belegscanner, BigSpender und die externen Systeme Active Directory, Dokumentenarchiv, SAP und E-Mail samt Kommunikationswegen.	**PlantUML**	C4-PlantUML eignet sich sehr gut für strukturierte Kontexte und technische Beziehungen. Das Diagramm bleibt bei Änderungen konsistent, versionierbar und kann als SVG für Folien exportiert werden.
Containerdiagramm	Zerlegt BigSpender in Intranet-Web-Anwendung, Scan-Client, modularen Kern, Hintergrundverarbeitung, Datenbank und Integrationsadapter.	**PlantUML**	Die C4-Notation macht Verantwortlichkeiten, Technologien und Schnittstellen auf einer Architekturfolie präzise sichtbar, ohne bereits Implementierungsdetails vorwegzunehmen.
Fachliche	Visualisiert den modularen Monolithen: Vorgangsverwaltung, Belegverwaltung, Regelverarbeitung	**Mermaid**	Ein einfaches Komponenten- bzw. Flussdiagramm ist schlank, direkt im Markdown pflegbar und für Lernmaterial

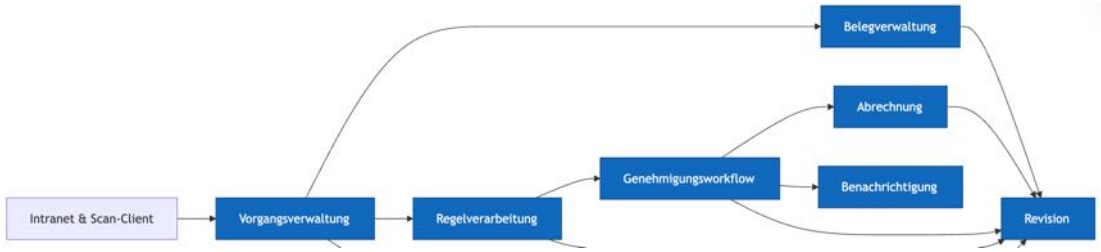
52

Systemkontext (PlantUML) generiert



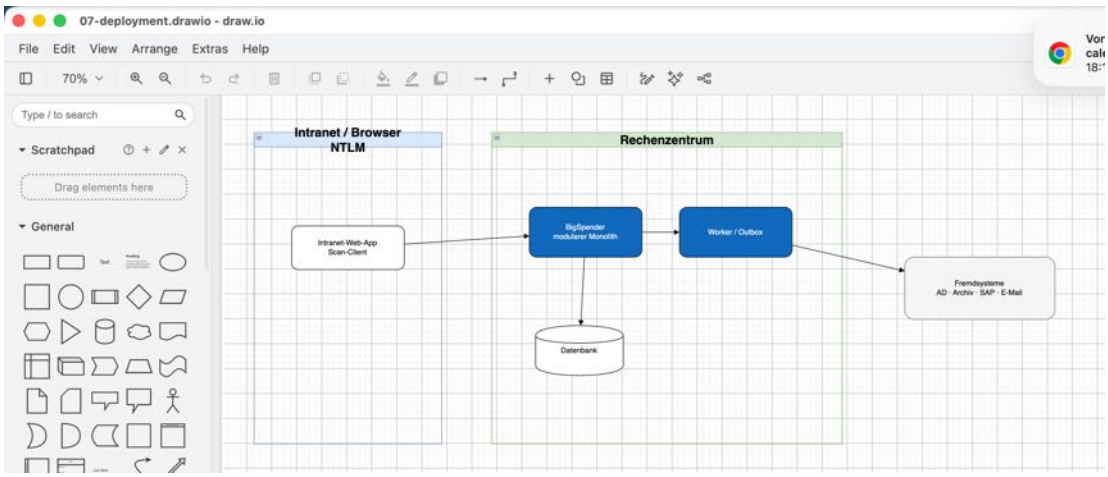
53

Bausteinsicht (Mermaid) generiert



54

Deploymentsicht (draw.io) generiert



55

Checkliste für Architekturdiagramme

Gibt es eine **Überschrift** und eine **Legende**?

Diagrammzweck ersichtlich?

Ist jedes **Element benannt**, die **Bedeutung ersichtlich** und die **Funktion verständlich**?

Autor, Erstellungs- und Änderungs**datum** vorhanden?

Sind die verwendeten **Abkürzungen ersichtlich**?

Bedeutung aller Farben und Formen ersichtlich?

Per **URL verlinkbar** statt Copy-by-Value?

Ist die **Quelle bekannt** und ist es leicht **editierbar**?

Bedeutung **unterschiedlicher Größe** und aller **Rahmenstile verständlich**?

Ist der **Diagrammtyp** klar erkennbar und wurde der **passende Typ** gewählt?

Sind **alle Verbindungen** beschriftet und die **Bedeutung der Linien klar**?

Wurde ein **passendes Werkzeug** verwendet?

Ist die Bedeutung der Pfeilspitzen und Linienstile (fett, gestrichelt, ...) ersichtlich?

KI braucht Architektur: Ist Docs-as-Code unverzichtbar embarc.de

56

56

Der größte KI-Hebel: Review und Pflege

Der größte Nutzen entsteht beim Erkennen von Lücken, Widersprüchen und veralteten Inhalten.

KI braucht Architektur: Ist Docs-as-Code unverzichtbar embarc.de

57

57

Der größte Hebel liegt im Review

Lücken finden

fehlende Inhalte · offene Fragen

Widersprüche erkennen

Text · Diagramm · ADR · Code

Veraltetes markieren

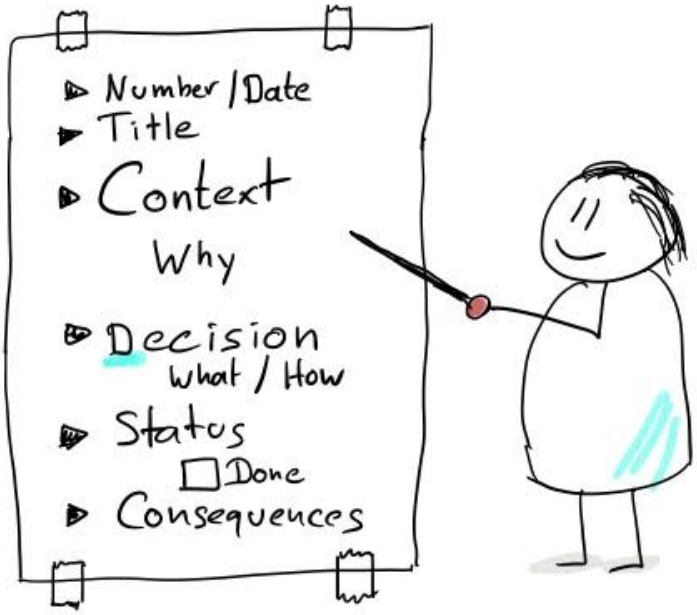
Änderung · Betroffenheit · Hinweis

Review skaliert besser als Neuerstellung.

58

Alternative ADR

- Architecture Decision Records
- weit verbreitet
- initial von Michael Nygard 2011 beschrieben



https://www.redhat.com/ja/blog/architecture-decision-records

59

ADRs

ADR-001: Modularer Monolith mit Integrationsadaptern	
Status	Vorgeschlagen
Kontext / Treiber	DACH-Pilot in 6 Monaten · 950 PT · konsistenter End-to-End-Workflow · hohe Revisionsanforderungen · viele, teils ausfallende Fremdsysteme
Entscheidung	Modularer Monolith: fachlich getrennte Module für Vorgang, Beleg, Regeln, Workflow, Abrechnung und Revision
Integrationsprinzip	Adapter kapseln AD/NTLM, Archiv, SAP und E-Mail · persistente, asynchrone und idempotente Übergaben
Alternative: Microservices	+ unabhängig deploy- und skalierbar · – hoher Initialaufwand für Betrieb, Messaging, Observability und verteilte Konsistenz
Alternative: Unstrukturierter Monolith	+ kurzfristig schnell · – wachsende Kopplung, schwer änderbar und schlecht auditierbar
Konsequenz	Schneller, risikoarmer Pilot mit konsistenten Transaktionen; spätere Extraktion einzelner Module bleibt möglich

60

Pflege wird Teil des Entwicklungsflusses

Änderung
Code · Infrastruktur · Docs

KI & Checks
Diff · Regeln · betroffene Docs

Pull Request
gemeinsam prüfen · nachführen

Dokumentation altert nicht still – Änderungen werden sichtbar.

61

KI braucht überprüfbare Guardrails

Quellen, Reviews und automatisierte Checks machen KI-Beiträge belastbar.

62

Aus Prinzipien werden prüfbare Regeln

Ziel formulieren
Was wollen wir schützen?

Regel ableiten
konkret · eindeutig · versioniert

Automatisiert prüfen
Test · Linter · Policy-as-Code

Was wichtig ist, sollte überprüfbar werden.

63

Mehrstufige Guardrails statt blindem Vertrauen

Harte Checks
Tests · Regeln · Schemas

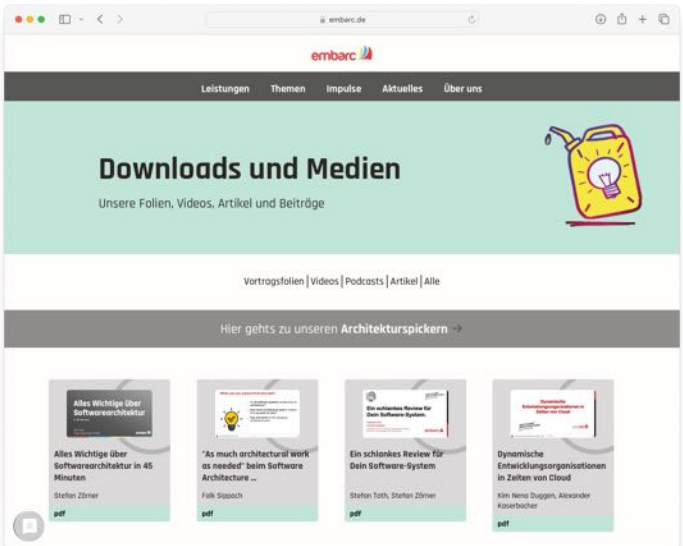
KI-Review
Semantik · Lücken · Widersprüche

Menschliche Freigabe
Risiko · Reichweite · Verantwortung

Nichtdeterministische KI braucht deterministische Sicherungen.

64

Folien als PDF zum Download



➔ embarc.de/download/

65

Architektur-Spicker

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

Architektur-Spicker #15
Documentation-as-Code einsetzen

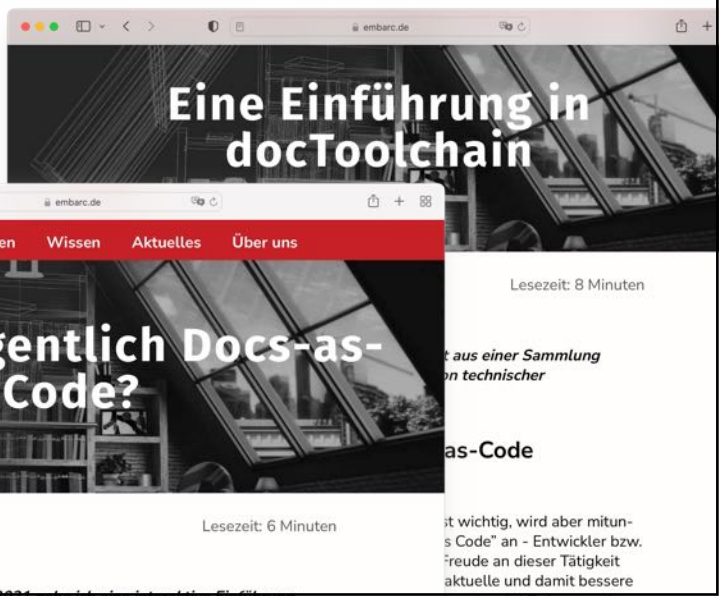
PDF, 4 Seiten
Kostenloser Download.

➔ architektur-spicker.de/

66

Weiterführende Blog-Posts

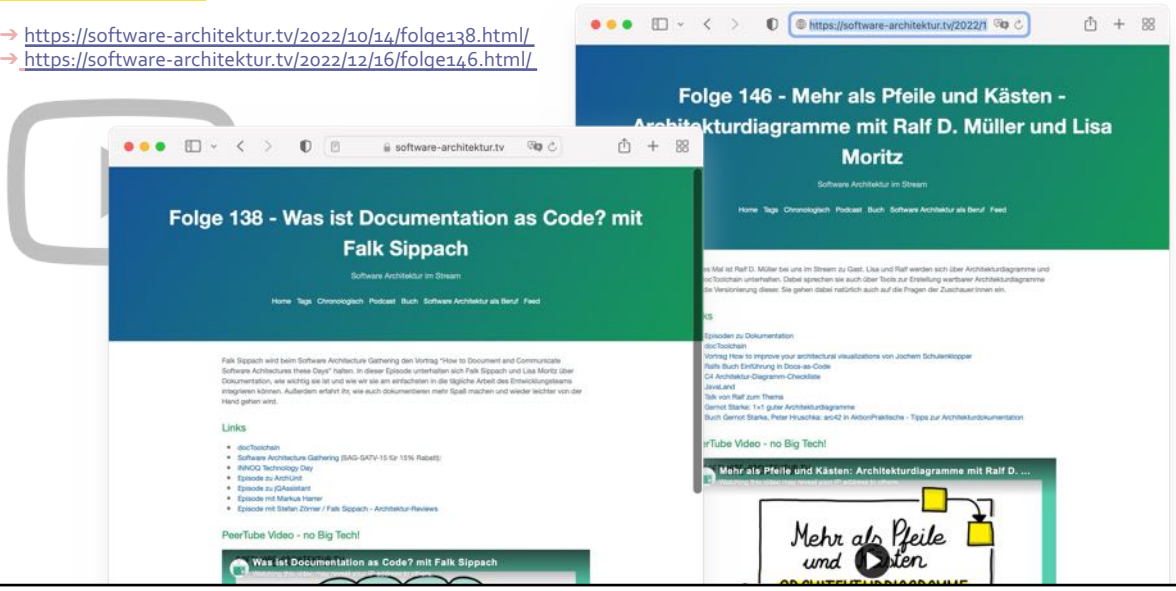
➔ https://www.embarc.de/was-ist-docs-as-code/
➔ https://www.embarc.de/doctoolchain/



67

Weiterführende Videos/Podcasts

→ https://software-architektur.tv/2022/10/14/folge138.html/
→ https://software-architektur.tv/2022/12/16/folge146.html/

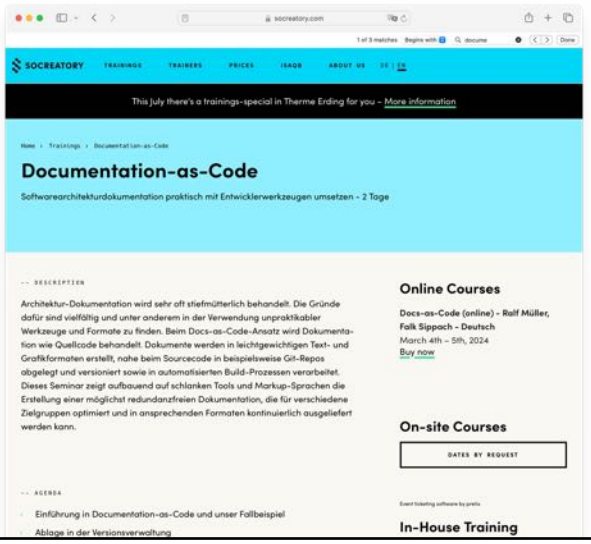


68

Training zu Docs-as-Code

Softwarearchitekturdokumentation praktisch mit Entwicklerwerkzeugen umsetzen

→ https://www.socreatory.com/en/trainings/docascode



Wir kommen auch zu Euch in die Firma!

69

70

Feedback & Fragen?

Wir freuen uns auf Fragen, Diskussionen, Anregungen!

71

Vielen Dank.

Ich freue mich auf Eure Fragen!

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

embarc