Ausführbare Dokumentation -

Architekturregeln mit Docs-as-Code festhalten und testen

Falk Sippach

10.03.2026

0

Abstract

Ausführbare Dokumentation – Architekturregeln mit Docs-as-Code festhalten und testen

Es ist kein Mythos, dass eine gute Architekturdokumentation ein Garant für erfolgreiche Softwareprojekte ist. Die Kunst und Herausforderung zugleich ist es jedoch, den Aufwand überschaubar und mit geeigneten Konzepten die Dokumentation lebendig zu halten. Leichtgewichtige Methoden, Vorlagen und Werkzeuge aus dem Docs-as-Code-Umfeld bringen das Dokumentieren näher an den Sourcecode und unterstützen Entwicklungsteams bei der Erstellung einer hochwertigen Architekturdokumentation. Leider veraltet Dokumentation auch sehr schnell.

In diesem Vortrag betrachten wir daher, wie wir eine selbst validierende, ausführbare Dokumentation nutzen können, um die Qualitätsansprüche der Softwarearchitektur wiederkehrend zu prüfen und mit Fitness Functions abzusichern.

Dazu machen wir Architekturkonzepte im Code sichtbar, zum Beispiel mit xMolecules und ContextMapper, binden Software-Analysewerkzeuge wie jQAssistant ein und führen regelmäßig programmatische Tests beispielsweise mit ArchUnit aus - für einen kontinuierlichen Abgleich der Soll- und Ist-Struktur. Ihr könnt diese Ideen in Eure Projekte übertragen, Eure Architekturdokumentation wird dadurch lebendig und Teil Eurer kontinuierlichen Qualitätsmaßnahmen.

1

Falk Sippach

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

Schwerpunkte

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

embarc

iSAQB

Java Champions

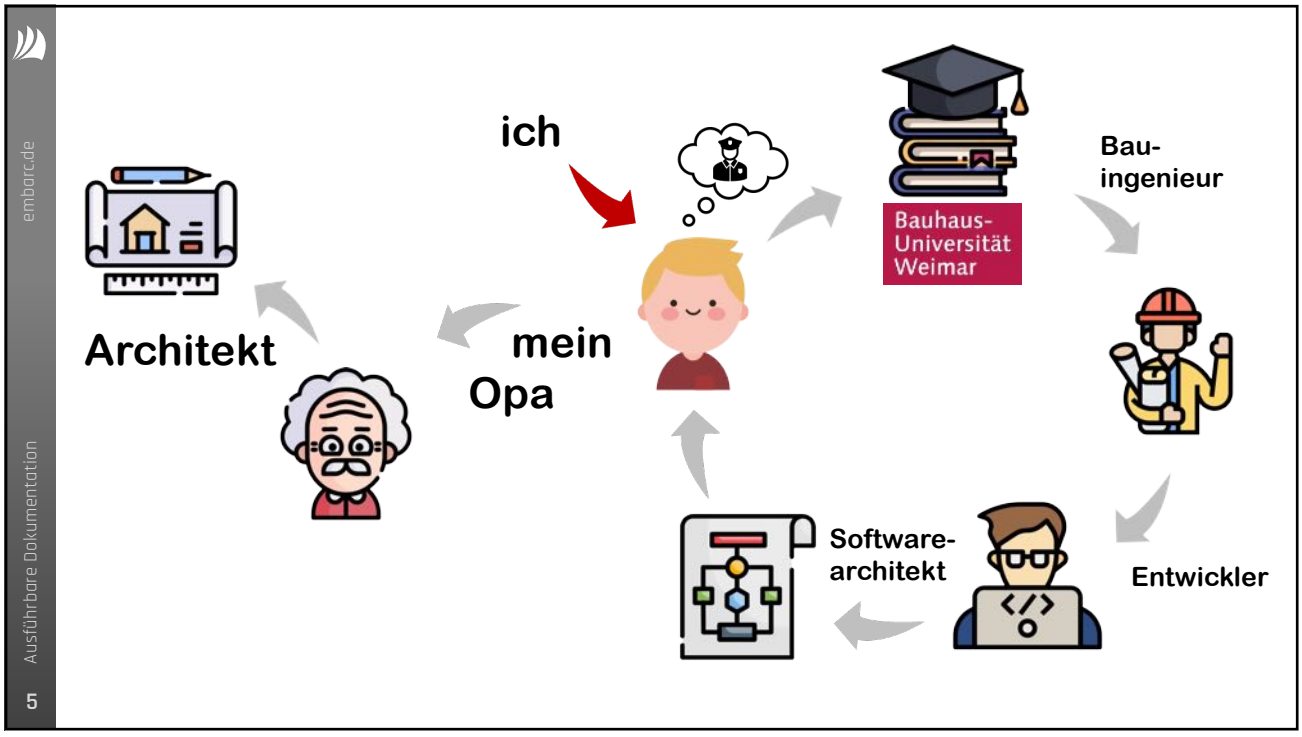
Cyber LAND

2

3

Wir konzentrieren uns
hier hauptsächlich auf
Softwarearchitektur!
Lets make
Documentation
great again!

4


embarc.de
Ausführbare Dokumentation
5
ich
Bauhaus-
Universität
Weimar
Bau-
ingenieur
Entwickler
Software-
architekt
mein
Opa
Architekt

5

Teil der Java Community

Nächster Termin am 19.03.

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

monatliche Opensource Code Camps

https://cyberland.ijug.eu/

6

00.

Agenda

Was euch erwartet

01 Warum dokumentieren?

02 Docs-as-Code

03 Lebendig + ausführbar

04 Werkzeuge

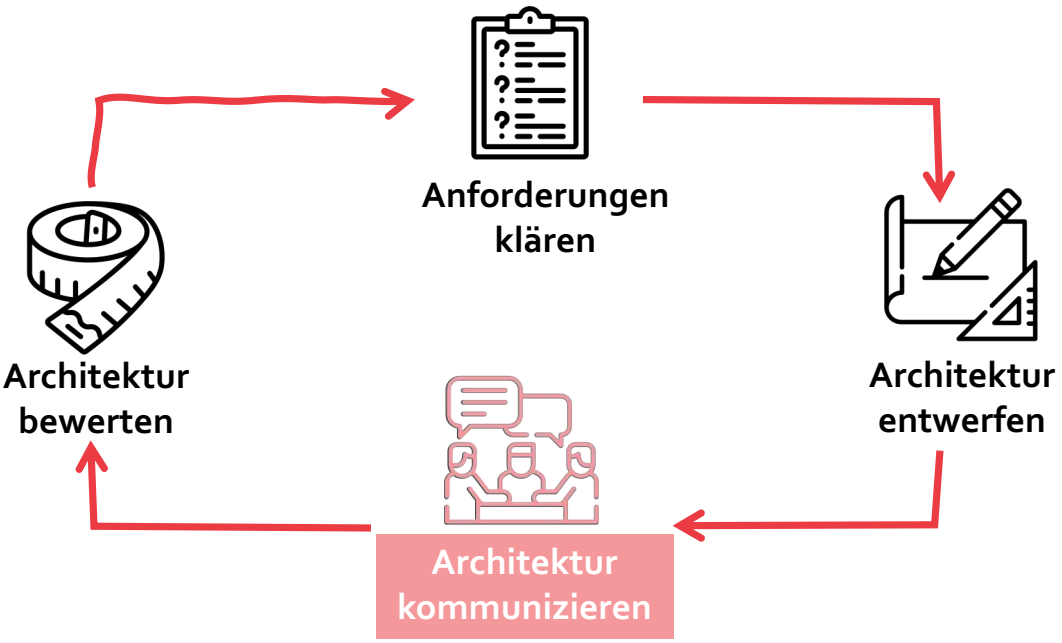
05 Fazit + Ausblick

8

01.

Warum dokumentieren?

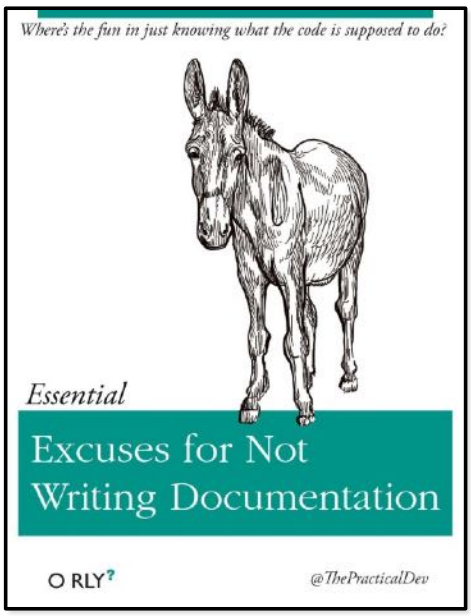
9



aus: Effektive Softwarearchitekturen (G. Starke)

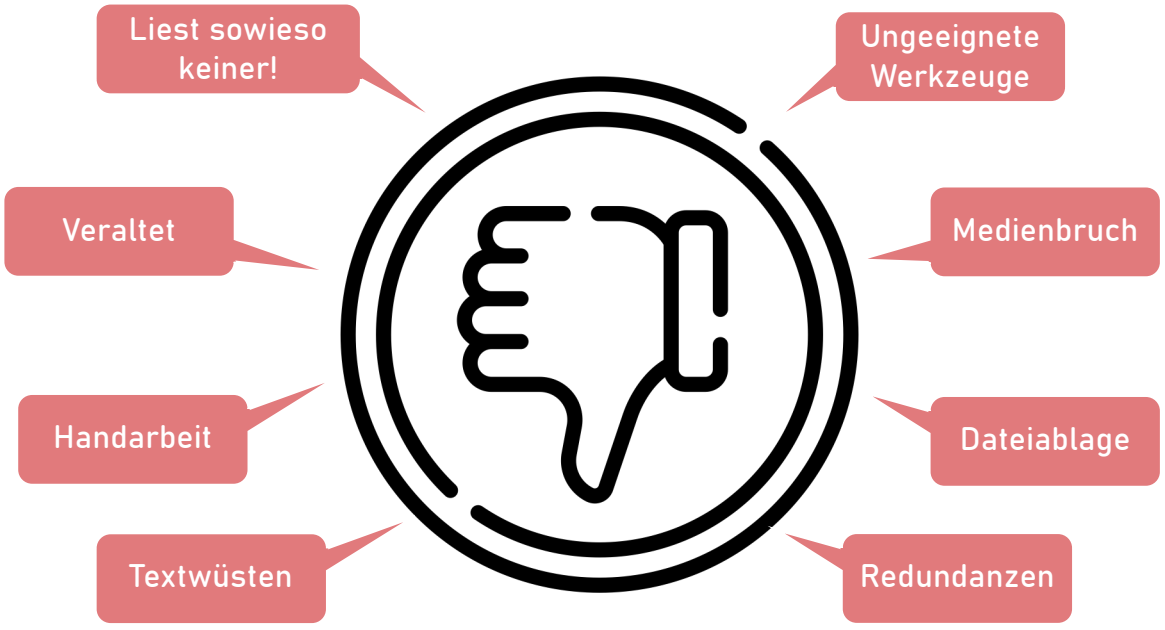
10

10



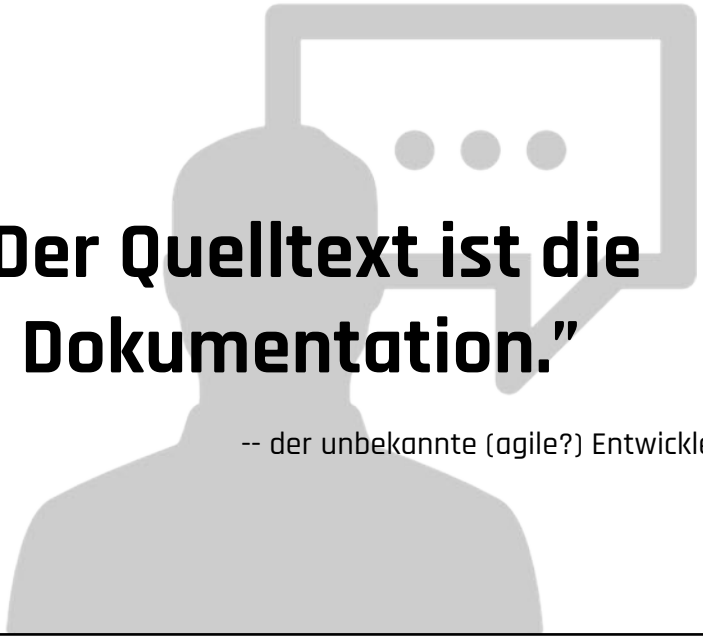
Grafik von The Practical Dev (CCo Public Domain Lizenz)

11



13

Zitat



“Der Quelltext ist die Dokumentation.”

-- der unbekannte (agile?) Entwickler

14

Replik

“Der Quelltext erzählt nicht die ganze Geschichte.”

-- Simon Brown

15

QuellCode != Dokumentation

Sichtbar-
machen
von ...?

Architektur-konzepten

Architektur-entscheidungen

Rollen von Bausteinen (DTO, ...) und Verortung

fachliche & technische Modulstrukturen

Gute Gründe für Dokumentation

Entwurfs-unterstützung

Bewertbarkeit

Frage nach dem Warum

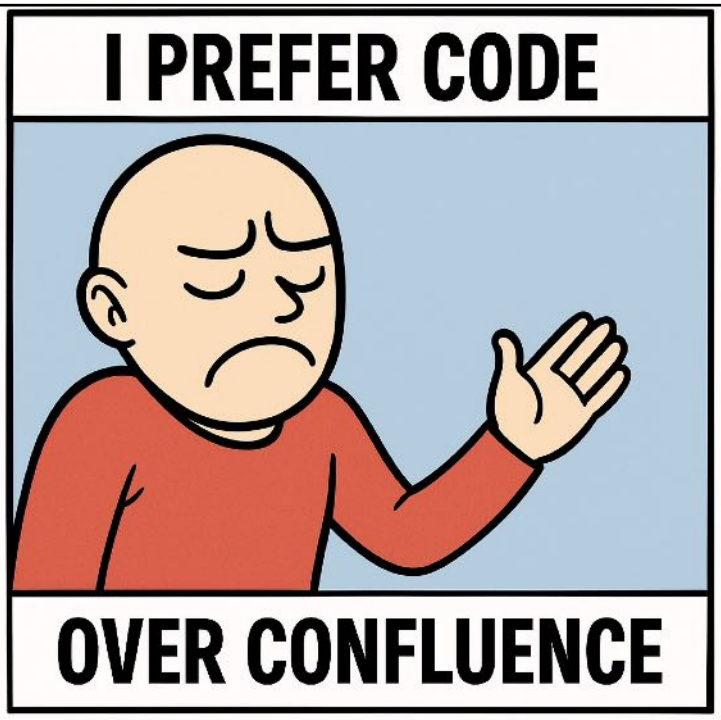
Kommunikation

Neue Mitarbeiter

Was ist mit Wikis?

Wiki = **W**here **I**nformation **K**ills **I**tselfs

19



20

02.

Docs-as-Code

21

Maturity Model

Docs-as-Code

22

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 kontinuierlich verbessern
Die unterschiedlichen Zielgruppen sowie ihre Anforderungen adressieren und regelmäßig **Feedback** einarbeiten. Erstellung und Wartung der Dokumentation in **Team-Prozesse** integrieren.

Werkzeuge einsetzen und integrieren
Unnötige Kontextwechsel vermeiden und Dokumentation in den typischen **Entwicklungswerkzeugen** erstellen. **Automatisierung** nutzen und Resultate regelmäßig in CI/CD-Pipeline erzeugen.

Mit Grafiken effizient umgehen
Den übertriebenen Einsatz von UML-Werkzeugen sowie kommerziellen Grafikprogrammen vermeiden und die **Einstiegshürde** durch Einsatz schlanker, freier Tools **verringern**.

Grundlage für Docs-as-Code legen
Mit **Markup-Sprachen** starten, Dokumentation modularisieren und auf Zielgruppen zugeschnittene Ergebnisse erstellen. Die Inhalte in der **Versionsverwaltung** sammeln.

Konventionell dokumentieren
Standardstrukturen wie **arc42, C4 und ADRs** kommen zum Einsatz, aber die Verwendung konventioneller Werkzeuge demotiviert und **bremst beim Dokumentieren** sogar aus.

23

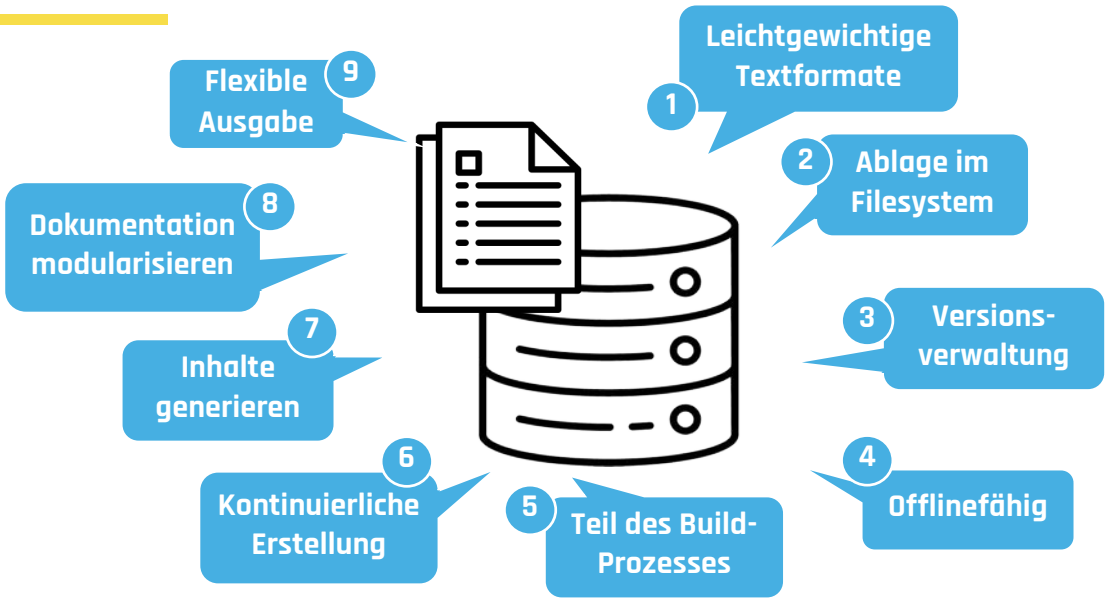
(Unser) täglich Entwickler-Brot

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

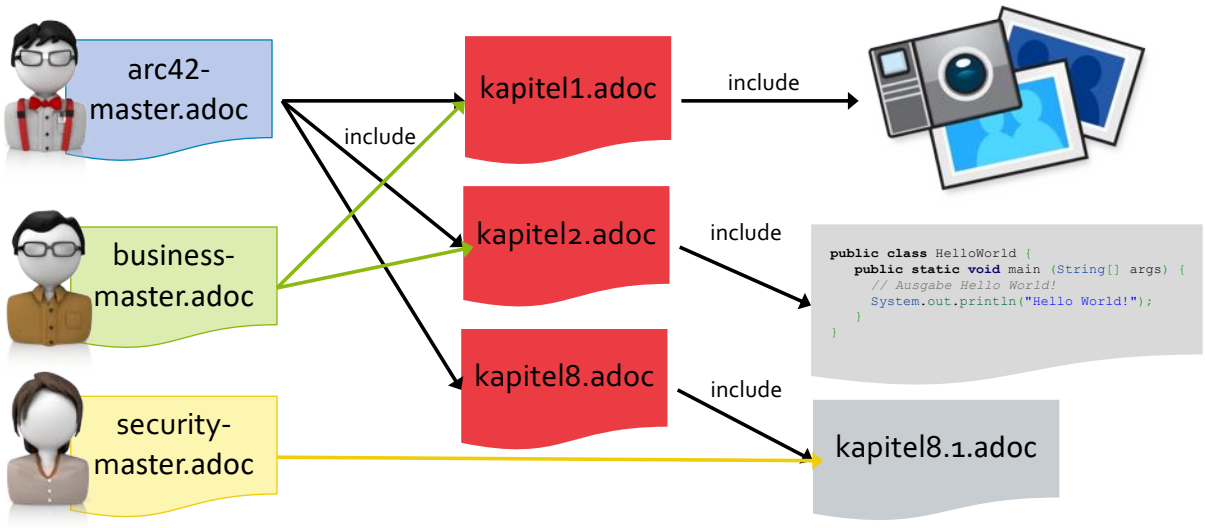
Foto von geralt: https://pixabay.com/de/unternehmer-start-start-up-karriere-696976/ (CCo Public Domain Lizenz)

24

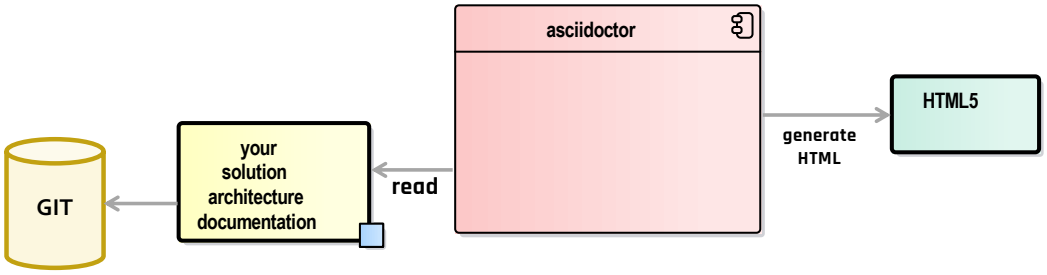
Documentation-as-Code



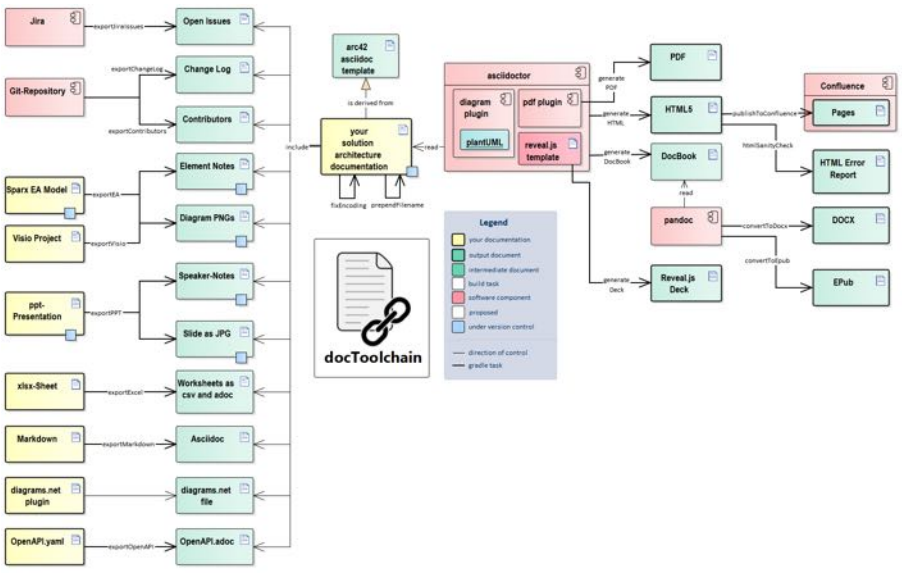
Modulare Dokumentation



Basic Docs-as-Code with AsciiDoc



Werkzeugkette: docToolchain



docToolchain @docToolchain · 26. Sep.

A full blown docu toolchain with @arc42Tipps as template installed in just one tweet:

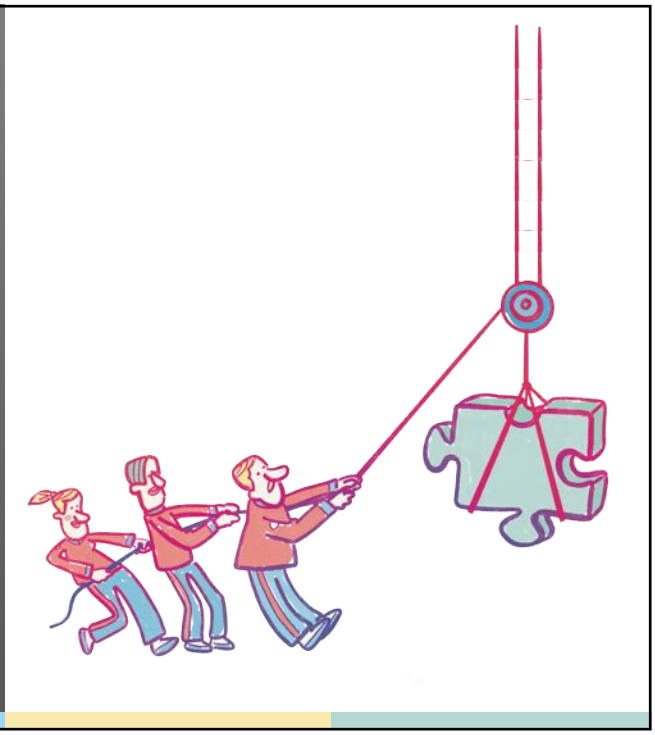
```
wget doctoolchain.github.io/dtcw
chmod +x dtcw
./dtcw downloadTemplate
./dtcw generateSite
```

=> doctoolchain.org

30

03.

Lebendige & ausführbare Dokumentation



31

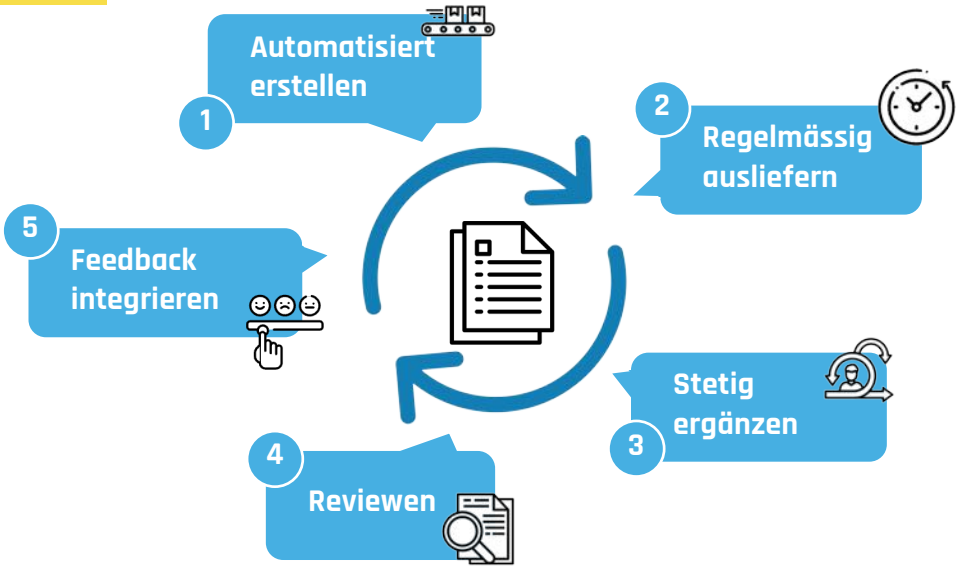
Lebendige Dokumentation

- **Automatisiert aktualisiert** (Inhalte teilweise **automatisch generiert**)
- **Teil des Entwicklungsprozesses** (Code Repo, Review über Pull Requests)
- **Maschinenlesbar** und **testbar**
- Für **Menschen verständlich** (gut strukturiert, modularisiert)

Beispiele

- Aus **BDD-Tests** (z. B. Gherkin) generierte Spezifikationen
- **Architekturregeln in Code** (z. B. „kein Zugriff von Modul A auf B"), regelmäßig getestet
- API-Dokumentation, die **aus Anotations** oder **OpenAPI-Spezifikationen generiert** wird
- **Systemdiagramme**, die sich automatisch **aus Code- oder Konfigurationsanalysen** ergeben

Continuous Documentation



34

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 kontinuierlich verbessern
Die unterschiedlichen Zielgruppen sowie ihre Anforderungen adressieren und regelmäßig **Feedback** einarbeiten. Erstellung und Wartung der Dokumentation in **Team-Prozesse** integrieren.

Werkzeuge einsetzen und integrieren
Unnötige Kontextwechsel vermeiden und Dokumentation in den typischen **Entwicklungswerkzeugen** erstellen. **Automatisierung** nutzen und Resultate regelmäßig in CI/CD-Pipeline erzeugen.

Mit Grafiken effizient umgehen
Den übertriebenen Einsatz von UML-Werkzeugen sowie kommerziellen Grafikprogrammen vermeiden und die **Einstiegshürde** durch Einsatz schlanker, freier Tools **verringern**.

Grundlage für Docs-as-Code legen
Mit **Markup-Sprachen** starten, Dokumentation modularisieren und auf Zielgruppen zugeschnittene Ergebnisse erstellen. Die Inhalte in der **Versionsverwaltung** sammeln.

Konventionell dokumentieren
Standardstrukturen wie **arc42, C4 und ADRs** kommen zum Einsatz, aber die Verwendung konventioneller Werkzeuge demotiviert und **bremst beim Dokumentieren** sogar aus.

35

Lebendige Architekturdokumentation

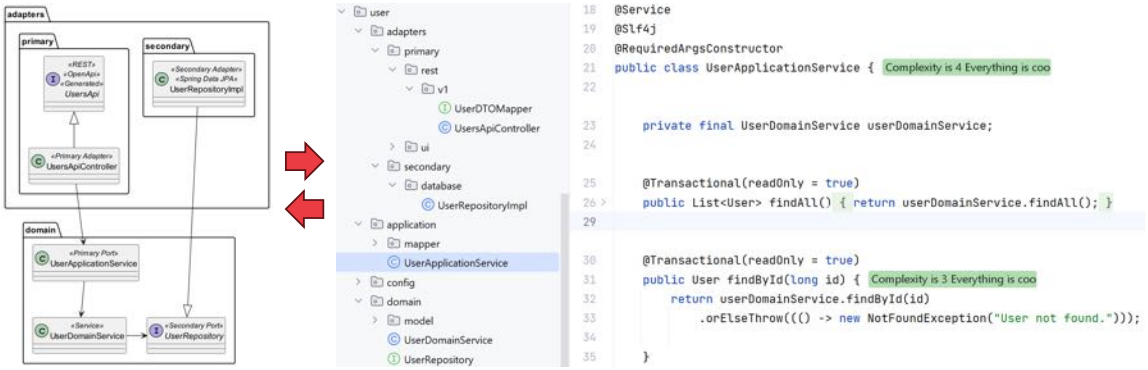
- … **prüft sich** selbst
- … passt sich an **Codeänderungen** an
- … **unterstützt Entwicklungsteams**, statt sie zu bremsen

36

Wie stellen wir sicher, dass die **Empfänger** die Dokumentation **letztendlich lesen**, **Feedback geben** und dieses in den **Überarbeitungs-** und **Aktualisierungsprozess zurückfließt**?

37

Mind The Gap



- Welche Unterschiede existieren zwischen Dokumentation (Soll) & Code (Ist)?
- Welche davon sind (un-)beabsichtigt? → Technische Schulden

38

Mind The Gap

- Beispiel: Java-Applikation à la DDD

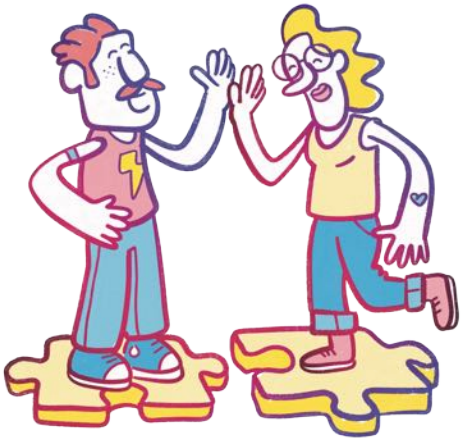
Architekturkonzept	Code
Bounded Context (BC)	Maven-Module
Technischer Layer	Java-Packages
Transaktionale Fassade	Spring-@Service-Klassen
Aggregate und Aggregate-Root	JPA @Entity-Klassen
Data Transfer Object	POJOs mit Namens-Suffix „DTO“

Projekt-spezifische Festlegung

→ Sind Architekturkonzepte im Code explizit erkennbar?
→ Was bedeuten die Konzepte?
→ Welchen Constraints unterliegen sie?

39

04.
Vorgehen & Werkzeuge



40

Validierung der Architekturregeln

Explizite Architektur-konzepte im Code

Transparenz durch Dokumentation

41

41

Architekturvalidierung

- Überprüfung der Erreichbarkeit angestrebter Qualitätsziele
  - Funktionalität → Unit-, Integrations- & Systemtests
  - Sicherheit → Penetrationstests, Vulnerability-Scans
  - Performance → Lasttests
- Art der Validierung
  - automatisiert vs. manuell
  - direkt vs. indirekt
  - 0/1, Value, Trend

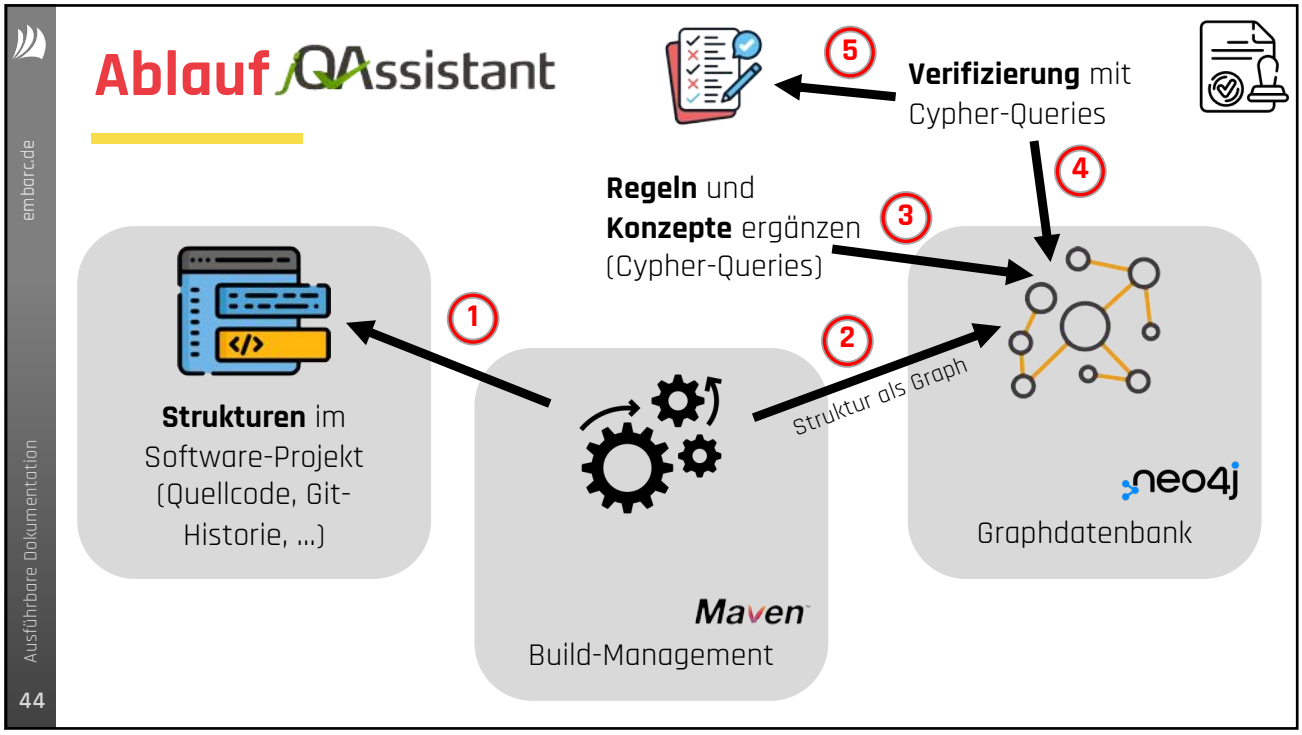
Fitness-funktionen

Architekturvalidierung

- Wartbarkeit
  - Prüfung von Code-Strukturen
  - manuelle Reviews vs. automatisierte Tests
  - Coding-Guidelines vs. Architekturkonzepte und- Constraints

sonarqube QAssistant CQSE ArchUnit

  - Indirekte Validierung der Erreichbarkeit des Qualitätsmerkmals
    - Angemessenheit vs. Einhaltung von Strukturen


Ablauf QAssistant
5
Verifizierung mit Cypher-Queries
4
Regeln und Konzepte ergänzen (Cypher-Queries)
3
1
2
Struktur als Graph
Strukturen im Software-Projekt (Quellcode, Git-Historie, ...)
Maven
Build-Management
neo4j
Graphdatenbank
embarc.de
Ausführbare Dokumentation
44

44

Validierung mit QAssistant
1. Integration in Build-Prozess 1 2
2. Erstellung von Regeln: Abbildung von Architekturkonzepten 3
MATCH
(dto:Java:Type)
WHERE
dto.name ends with "DTO"
RETURN
dto as DTO
embarc.de
Ausführbare Dokumentation
45

45

Validierung mit QAssistant

3. Erstellung von Regeln: Formulierung von Constraints 4

```
MATCH
  (package:Package)-[:CONTAINS]->(dto:Java:Type)
WHERE
  package.name <> "model"
RETURN
  dto as DTO
```


Validierung mit QAssistant

4. Verletzungen behandeln → Schmerzen (er)leben 5

```
[ERROR] --[ Constraint Violation ]---------------------------------------
[ERROR] Constraint: rest:DTO
[ERROR] Severity: MAJOR
[ERROR] Number of rows: 2
[ERROR]            All DTO classes must be located in packages named "model".
[ERROR]
[ERROR]   DTO=org.jqassistant.demo.AssigneeDTO
[ERROR]   DTO=org.jqassistant.demo.architecture.hexagonal.shared.adapters.primary.rest.exception.ErrorDTO
[ERROR] ----------------------------------------------------------------
```

Lernen durch Fehler

All DTO classes must be located in packages named "model".
DTO:org.jqassistant.demo.architecture.hexagonal.shared.adapters.primary.rest.exception.ErrorDTO

```
1  dirk…   package org.jqassistant.demo.architecture.hexagonal.shared.adapters.primary.rest.exception;
2
3          import lombok.Builder;
4          import lombok.Getter;
5
6          @Getter
7          @Builder
8          public class ErrorDTO {
9
```

Docs-As-Code mit QAssistant

embarc.de

Ausführbare Dokumentation

5. Dokumentation des Ist-Standes

```
[[section-technical-risks]]
== Risks and Technical Debts

=== Overview

include::jQAssistant:Summary[constraints="*"]

=== Details

:leveloffset: +1
include::jQAssistant:Rules[]
:leveloffset: -1
```

11. Risks and Technical Debts

11.1. Overview

Id	Description	Severity	Status
jmolecules-ddd:IllegalDependenciesBetweenBoundedContexts	Checks that dependencies between :BoundedContext nodes exist only where allowed.	MAJOR	FAILURE
maven:DependencyMustBeManaged	All used Maven dependencies must be managed in a "dependencyManagement2 section.	MAJOR	FAILURE
rest:DTOMustBeLocatedInModelPackage	All DTO classes must be located in packages named "model".	MAJOR	FAILURE
jmolecules-ddd:MutableEntityId	Checks that the id field of an entity (identifiable) is only manipulated via the constructor.	MAJOR	SUCCESS

11.2.3. rest:DTOMustBeLocatedInModelPackage

ERROR

All DTO classes must be located in packages named "model".
Status: FAILURE, Severity: MAJOR

DTO
org.jqassistant.demo.architecture.hexagonal.shared.adapters.primary.rest.exception.ErrorDTO

Docs-As-Code mit QAssistant

embarc.de

Ausführbare Dokumentation

6. Reporting

```
== Risks and Technical Debts
=== Overview
include::jQAssistant:Summary[]

=== Details

:leveloffset: +1

include::jQAssistant:Rules[]

:leveloffset: -1
```

arc42

11.2.30. bounded-context:DependencyReport

SUCCESS

Reports the dependencies between bounded contexts.
Status: SUCCESS, Severity: INFO

GraphML

Component Diagram

ArchUnit

- Testing-Framework für Architekturrichtlinien in Java/.NET

```
@ArchTest
public static final ArchRule services_should_not_access_controllers =
        noClasses().that().resideInAPackage("..service..")
        .should().dependOnClassesThat().resideInAPackage("..controller..");
```

- Durchsetzen von Hexagonal Architecture, Clean Architecture, Layered Architecture, …
- Vermeidung von unerlaubten Abhängigkeiten
- Sicherstellen von Namenskonventionen

ArchUnit

50

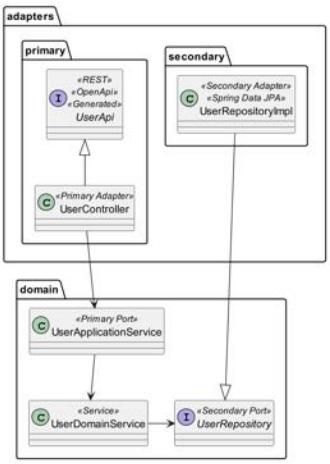
Validierung der Architekturregeln

Explizite Architektur-konzepte im Code

Transparenz durch Dokumentation

51

xMolecules, ContextMapper

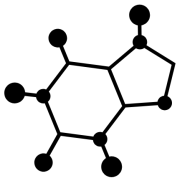



```java
public interface UserRepository {
```

```java
@Repository
public class UserRepositoryImpl
extends SimpleJpaRepository<User, Long>
implements UserRepository {
```

```java
@SecondaryPort
public interface UserRepository {
```

```java
@SecondaryAdapter
@Repository
public class UserRepositoryImpl
extends SimpleJpaRepository<User, Long>
implements UserRepository {
```

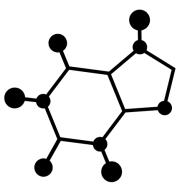


jMolecules: Architectural abstractions for Java

Ideas behind:

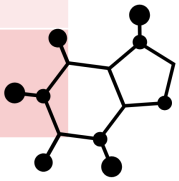
- Explicitly **express architectural concepts** for easier code reading and writing.
- **Keep** domain-specific code **free from technical dependencies**. Reduce boilerplate code.
- **Automatically generate documentation** and **validate** implementation structures and **your architecture**.

- https://github.com/xmolecules/jmolecules



jMolecules - Beispiele

Architekturstil/-muster	Konzepte/Annotationen
CQRS	@Command, @CommandDispatcher, @CommandHandler, @QueryModel
Layered	@DomainLayer, @ApplicationLayer, @InfrastructureLayer, @InterfaceLayer
Onion Classic	@DomainModelRing, @DomainServiceRing, @ApplicationServiceRing, @InfrastructureRing
Onion Simplified	@DomainRing, @ApplicationRing, @InfrastructureRing
Hexagonal	@Application, @PrimaryAdapter, @SecondaryAdapter, @PrimaryPort, @SecondaryPort



Einbindung Bibliotheken: jMolecules Plugins

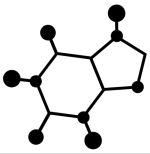
QAssistant

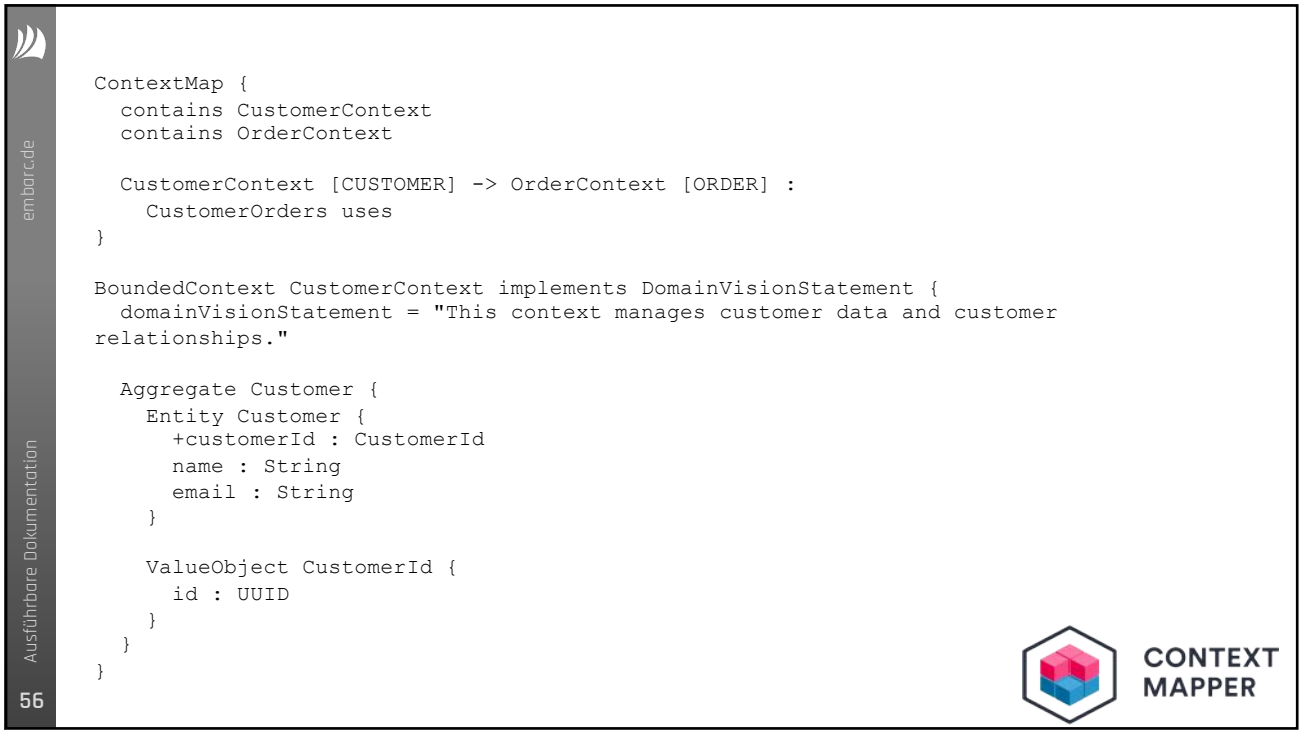
ArchUnit

Regeln für verschiedene Architekturstile, DDD Building Blocks, CQRS und Events **verifizieren**, **PlantUML Diagramm erzeugen**

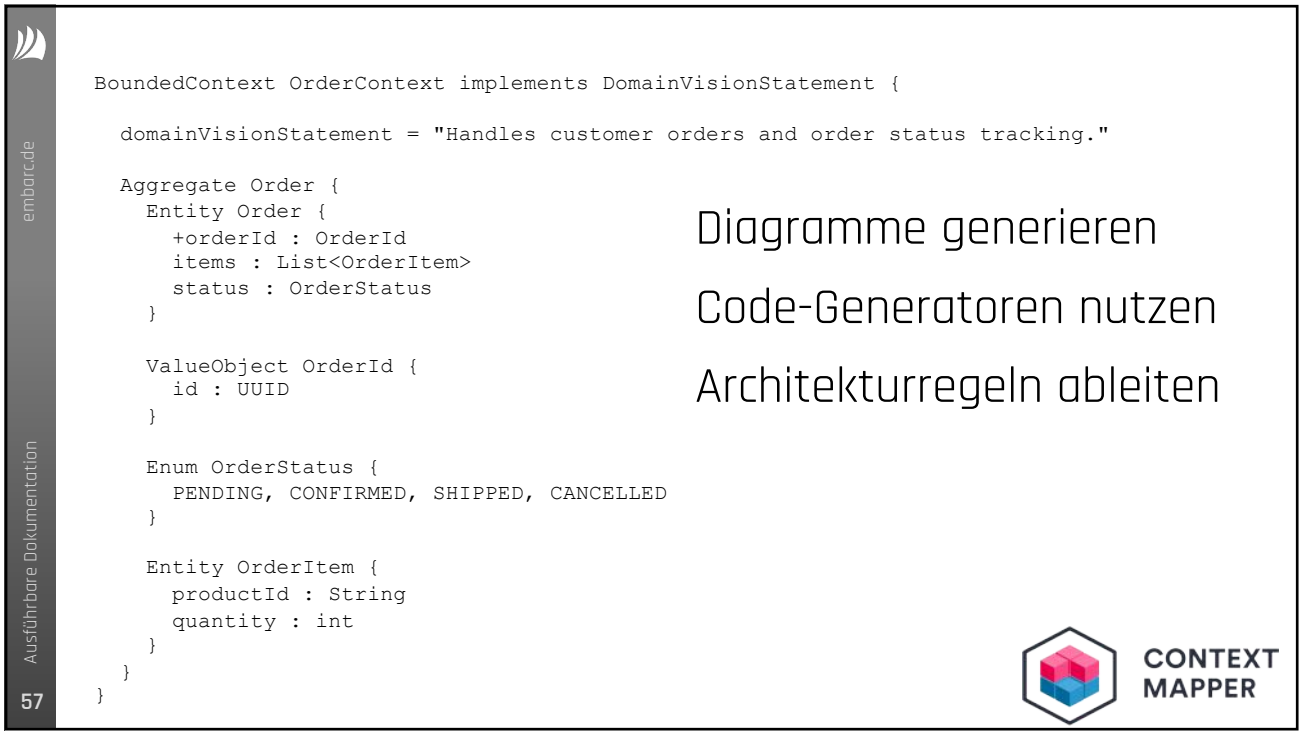
Abhängigkeiten zw. DDD Building Blocks **verifizieren**

automatische Erkennung von jMolecule Komponenten, DDD Building Blocks und Events für die **Modul-Definition** und zur **Dokumentation**




embarc.de
Ausführbare Dokumentation
56
ContextMap {
contains CustomerContext
contains OrderContext
CustomerContext [CUSTOMER] -> OrderContext [ORDER] :
CustomerOrders uses
}
BoundedContext CustomerContext implements DomainVisionStatement {
domainVisionStatement = "This context manages customer data and customer relationships."
Aggregate Customer {
Entity Customer {
+customerId : CustomerId
name : String
email : String
}
ValueObject CustomerId {
id : UUID
}
}
}
CONTEXT MAPPER

56


embarc.de
Ausführbare Dokumentation
57
BoundedContext OrderContext implements DomainVisionStatement {
domainVisionStatement = "Handles customer orders and order status tracking."
Aggregate Order {
Entity Order {
+orderId : OrderId
items : List<OrderItem>
status : OrderStatus
}
ValueObject OrderId {
id : UUID
}
Enum OrderStatus {
PENDING, CONFIRMED, SHIPPED, CANCELLED
}
Entity OrderItem {
productId : String
quantity : int
}
}
}
Diagramme generieren
Code-Generatoren nutzen
Architekturregeln ableiten
CONTEXT MAPPER

57

Alternative Werkzeugketten

58

Spring Modulith

- unterstützt Entwickler bei der Implementierung logischer Module (über Package Konventionen und Meta-Informationen)
- erzeugt intern ein Modell (über Spring Komponenten-Modell, Meta-Informationen, jMolecules, ...)

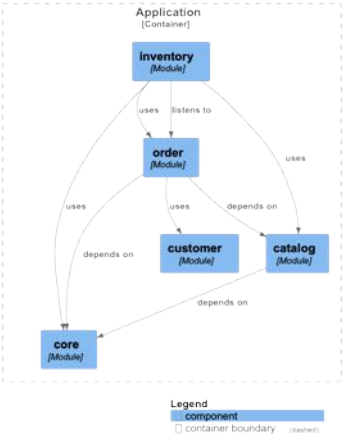
```
Example
└ src/main/java
  ├ example
  │ └ Application.java
  ├ example.inventory
  │ ├ InventoryManagement.java
  │ └ SomethingInventoryInternal.java
  ├ example.order
  │ └ OrderManagement.java
  └ example.order.internal
    └ SomethingOrderInternal.java
```

https://spring.io/projects/spring-modulith

60

Spring Modulith

- kann Verifikationen laufen lassen (ArchUnit)
- erzeugt Dokumentations-Snippets
  und -diagramme mit AsciiDoc und PlantUML



https://spring.io/projects/spring-modulith

Spring Modulith

- **@ApplicationModuleTest** – Hochfahren von einem Modul und den Abhängigkeiten
- **@ApplicationModuleListener** – asynchrone, persistierte Events

https://spring.io/projects/spring-modulith

Diagrams-as-Code 2.0

- Programmatische Erzeugung von Architekturdiagrammen (ähnlich PlantUML, Mermaid, ...)
- "multiple software architecture diagrams **from a single model**"
- Structurizr DSL, Java DSL, ...
- Integration in die Build-Prozesse

https://structurizr.com/

Structurizr

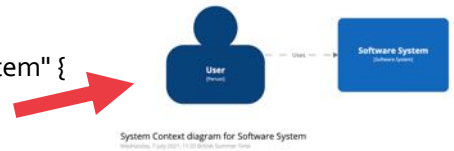
63

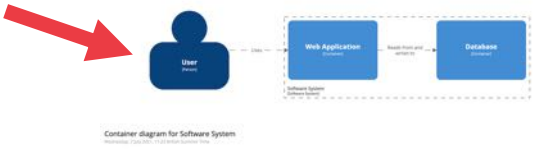
Structurizr

```
workspace {

  model {
    user = person "User"
    softwareSystem = softwareSystem "Software System" {
      webapp = container "Web Application" {
        user -> this "Uses"
      }
      container "Database" {
        webapp -> this "Reads from and writes to"
      }
    }
  }

  views { .... }
```

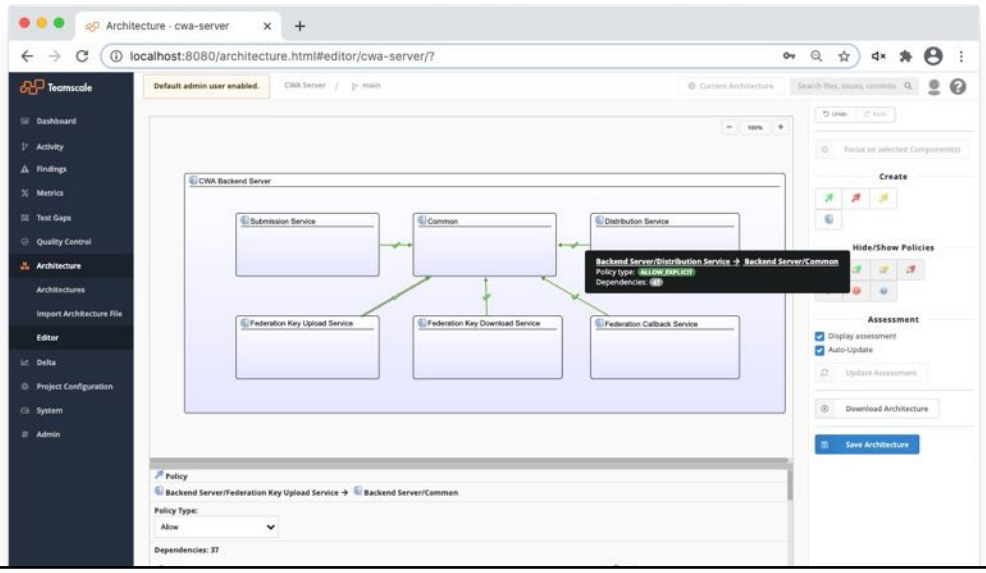




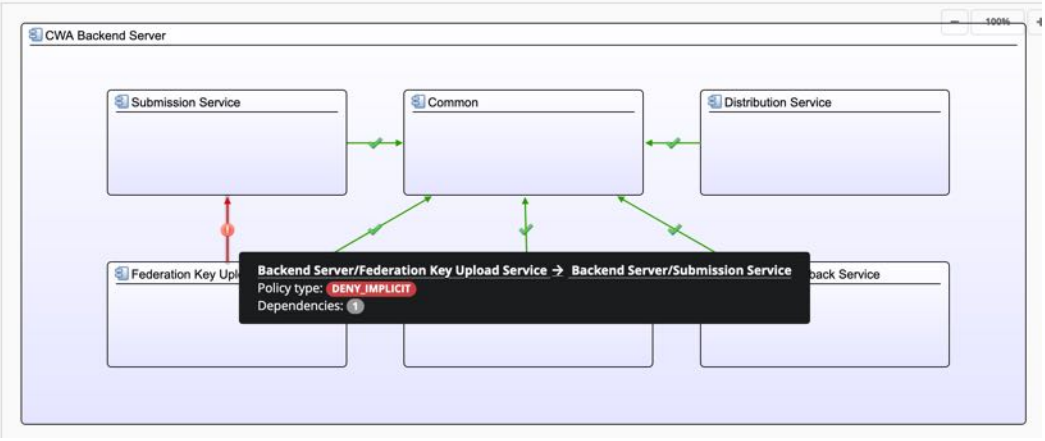
Structurizr

64

Abhängigkeiten in Teamscale



So sähe ein Verstoß aus …



Hinweis: Die Abhängigkeit zwischen Upload und Submission Service hatten wir von Hand im Quelltext eingebaut zu Demonstrationszwecken.

05.

Fazit & Ausblick

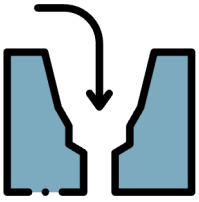
Völlig losgelöst von der Erde
Schwebt das Raumschiff
schwerelos …

67

Dokumentation –
Das notwendige Übel

~~Dokumentation –~~
~~Classic Style~~

Dokumentation –
State of the Art

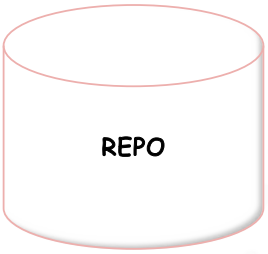


Mind the Gap

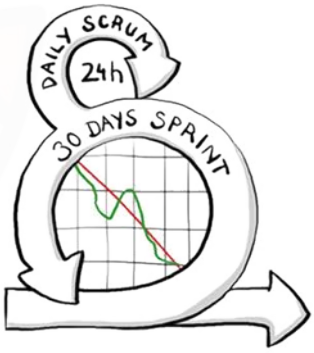
Architektur-
Validierung

Explizite Architektur-
konzepte im Code

68



Documentation as Code



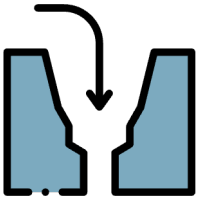
Continuous Documentation

69

Dokumentation – Das notwendige Übel

~~Dokumentation – Classic Style~~

Dokumentation – State of the Art

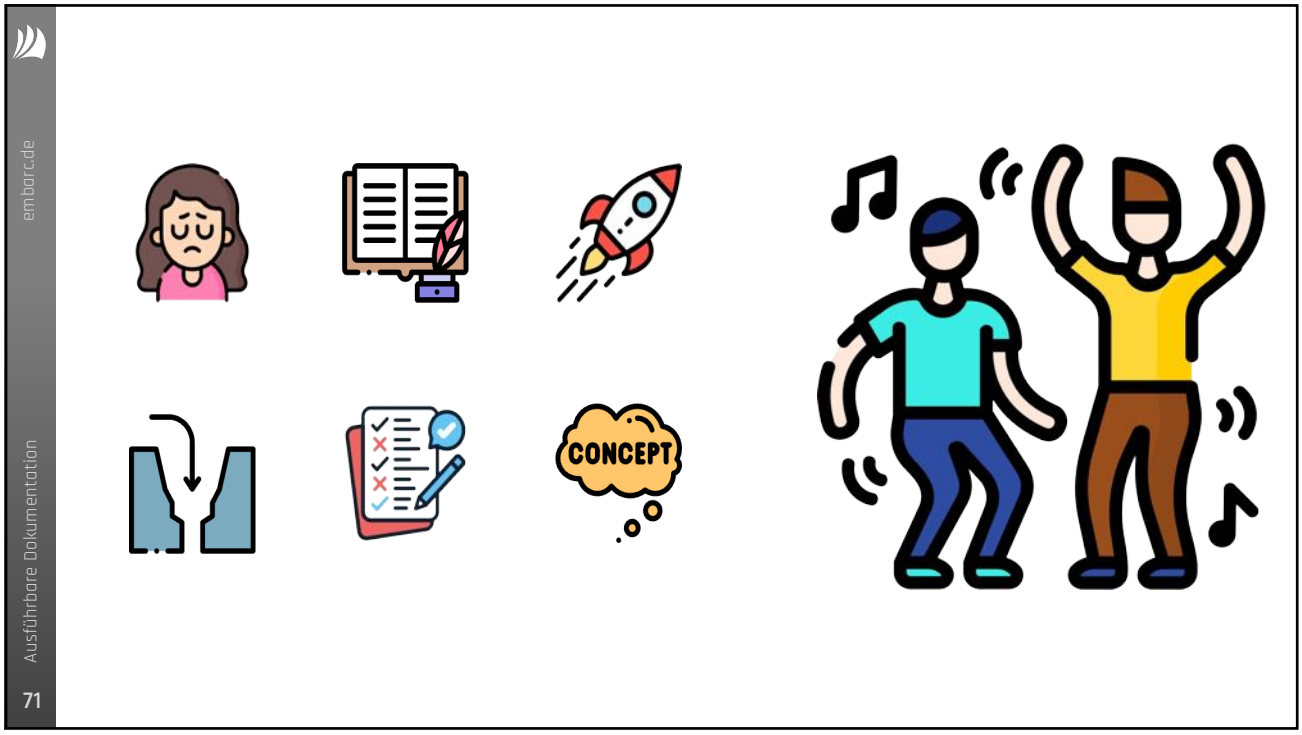


Mind the Gap

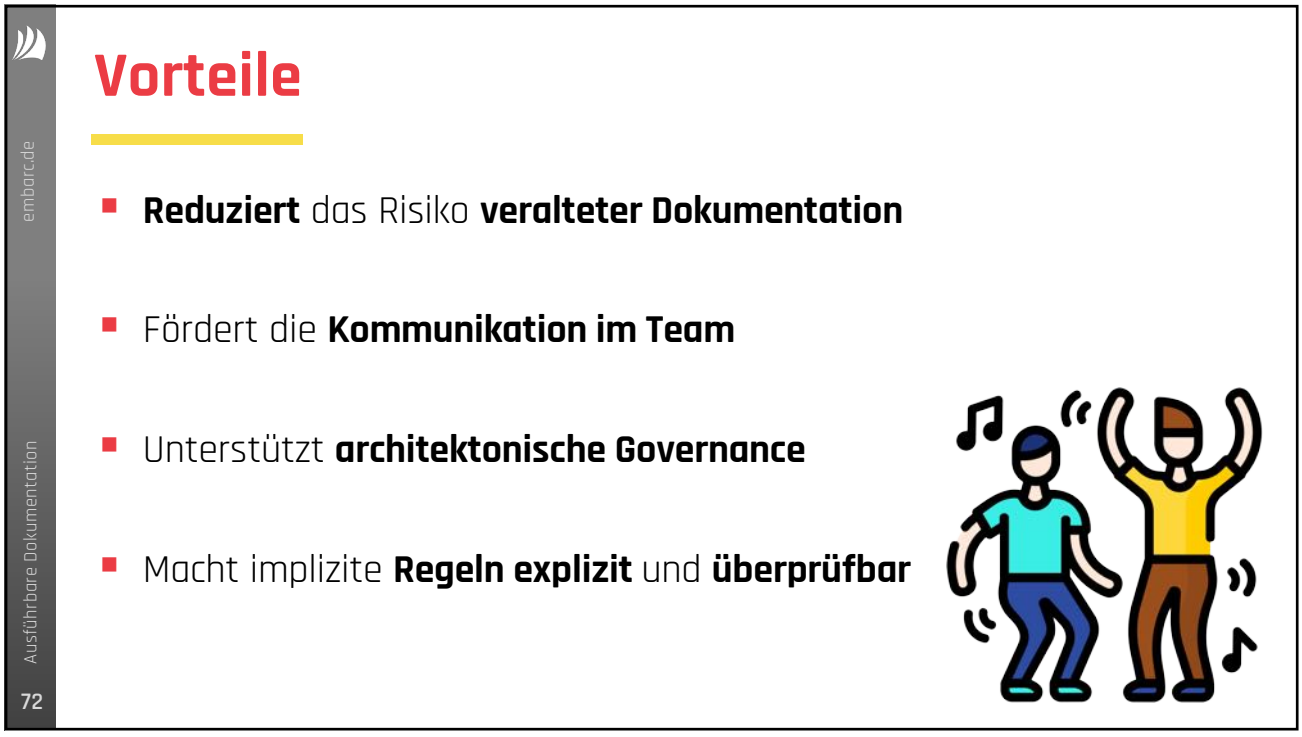
Architektur-Validierung

Explizite Architektur-konzepte im Code

70


embarc.de
Ausführbare Dokumentation
71
CONCEPT

71


embarc.de
Ausführbare Dokumentation
72
Vorteile
Reduziert das Risiko veralteter Dokumentation
Fördert die Kommunikation im Team
Unterstützt architektonische Governance
Macht implizite Regeln explizit und überprüfbar

72

Ausführbare Dokumentation

Auf die **Zielgruppen** ausgerichtete Artefakte erzeugen

Synchrone Auslieferung ermöglicht die stets aktuelle Bereitstellung

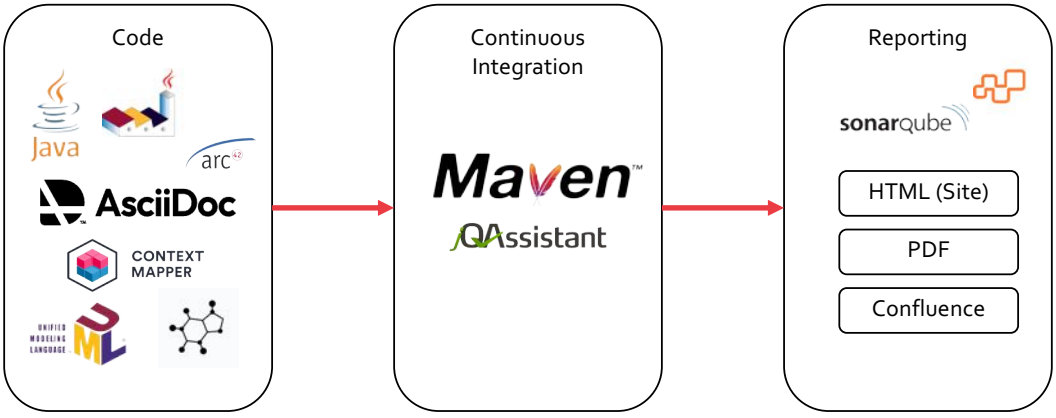
Architekturregeln und **-tests** integrieren

Einbetten von **Code-Snippets** und **API-Aufrufen**

Feedback durch die Leser **ermöglichen**

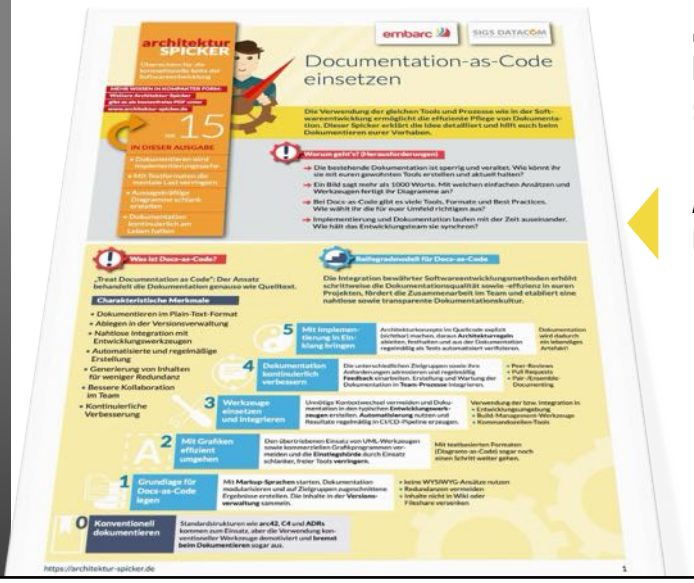
73

Lebendige Architektur-Dokumentation



74

Architektur-Spicker



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

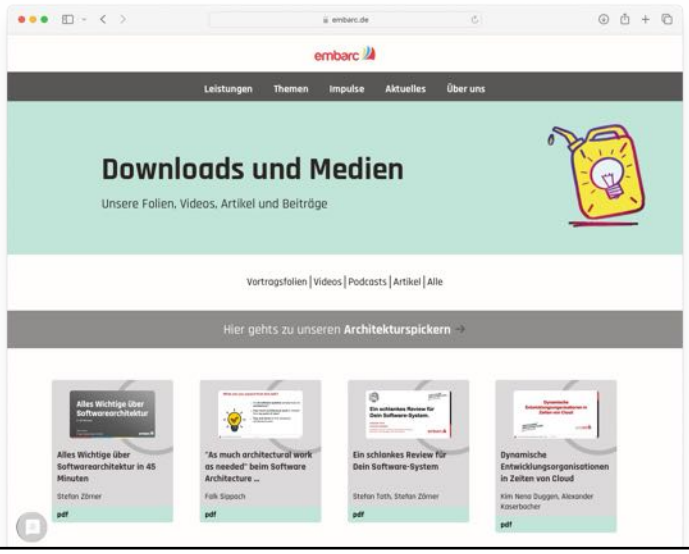
Architektur-Spicker #15
Documentation-as-Code einsetzen

PDF, 4 Seiten
Kostenloser Download.

➔ architektur-spicker.de/

78

Folien als PDF zum Download



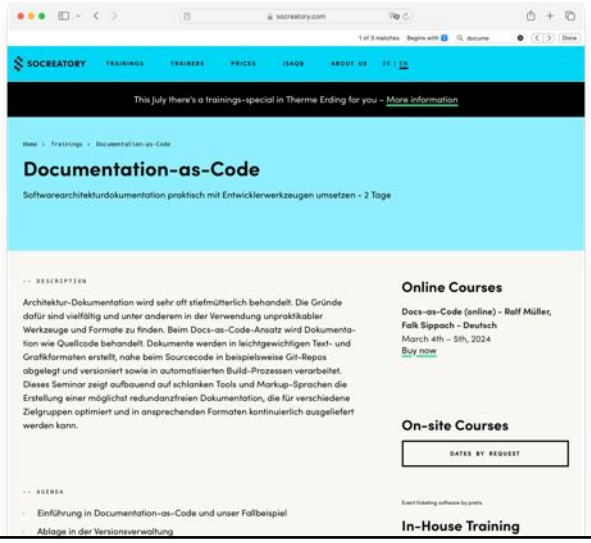
➔ embarc.de/download/

79

Training zu Docs-as-Code

Softwarearchitekturdokumentation praktisch mit Entwicklerwerkzeugen umsetzen

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

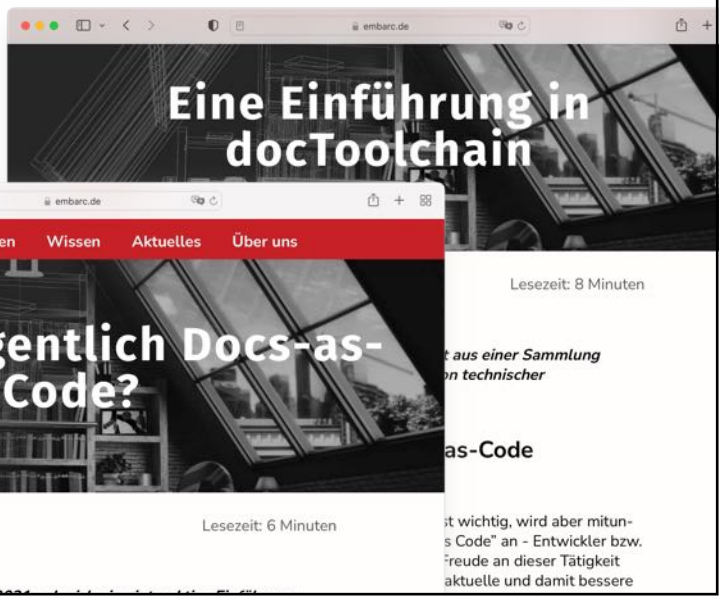


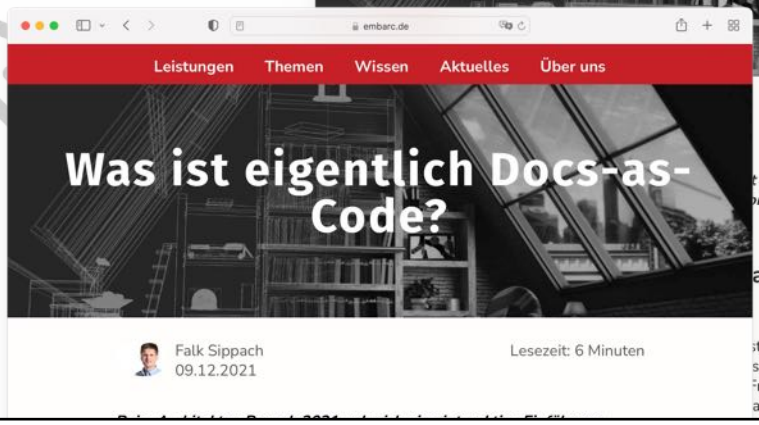
Wir kommen auch zu Euch in die Firma!

80

Weiterführende Blog-Posts

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

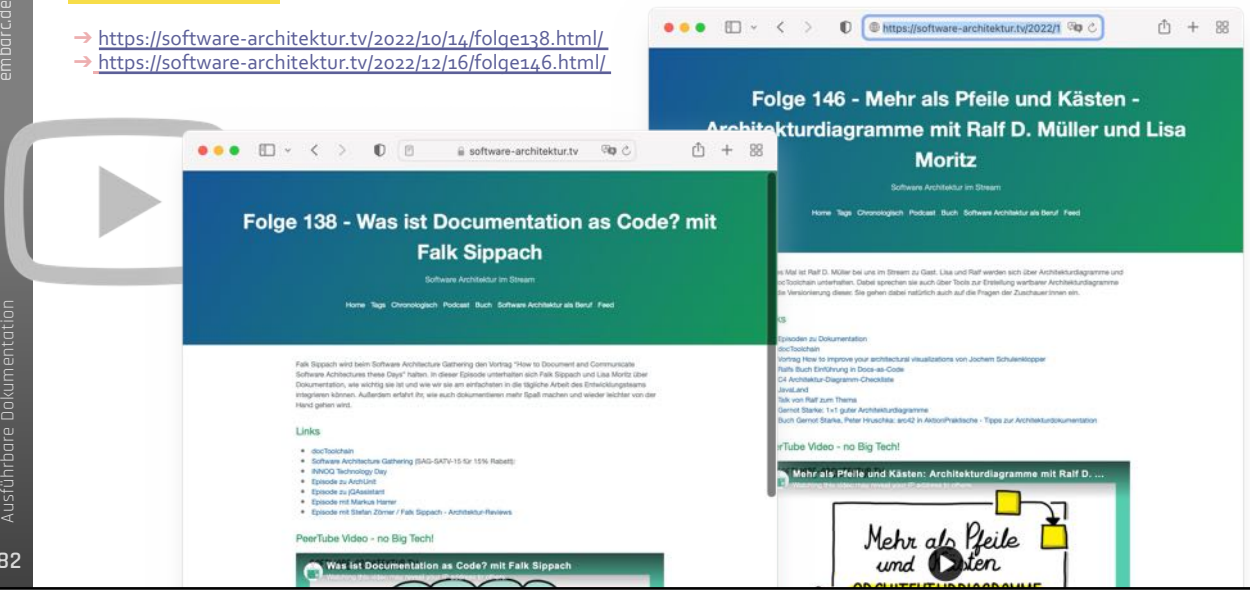




81

Weiterführende Videos/Podcasts

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



82

Feedback & Fragen?

Wir freuen uns auf Fragen, Diskussionen, Anregungen!

83

Vielen Dank.

Ich freue mich auf Eure Fragen!

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

embarc

84

Falk Sippach

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

Schwerpunkte

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

embarc

85