

Ausführbare Dokumentation -

Architekturegeln mit Docs-as-Code festhalten und testen

Falk Sippach

betterCode (ArchDoc)

12.05.2025

The embarc logo, identical to the one at the top of the slide.

0

Abstract

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

Es ist kein Mythos, dass eine gute Architektdokumentation 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 Architektdokumentation. 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 Architekturekonzepte 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 Architektdokumentation wird dadurch lebendig und Teil Eurer kontinuierlichen Qualitätsmaßnahmen.



Falk Sippach

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



Schwerpunkte

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



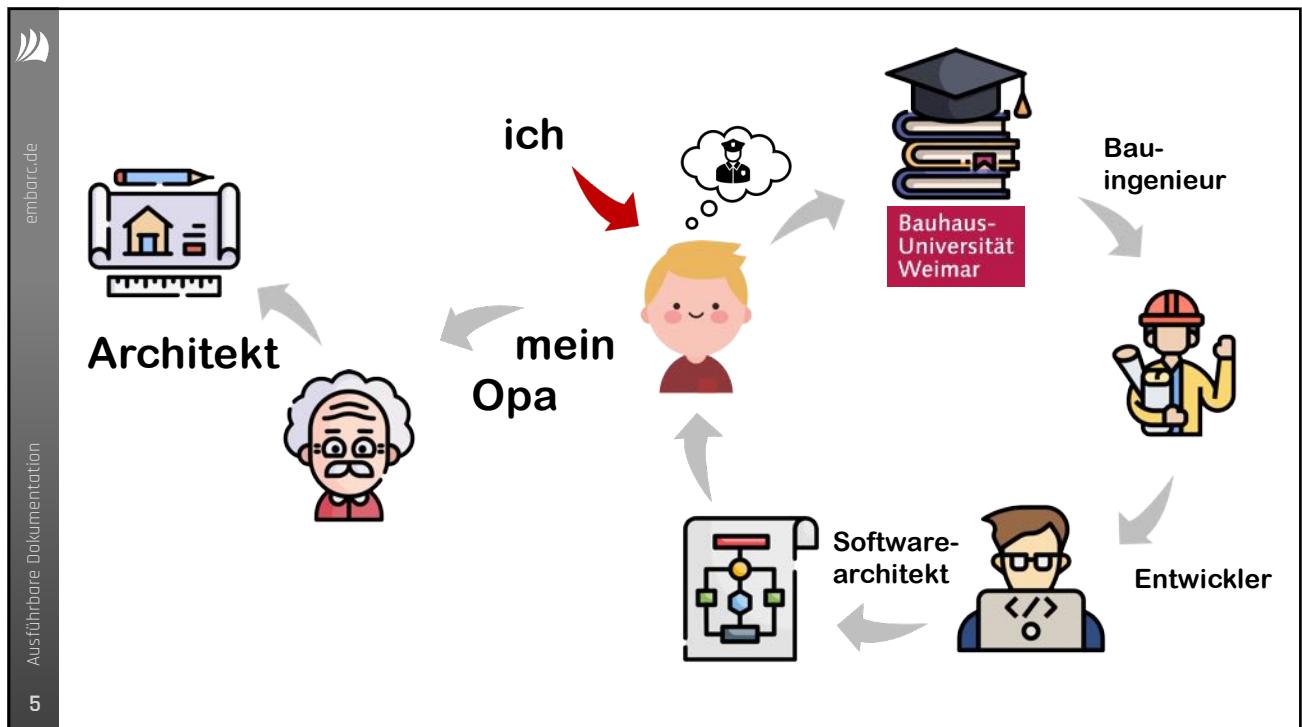
Wir müssen über ein ernstes Thema reden:

Dokumentation





4



5

D0.

Agenda

Was euch erwartet

Docs-as-Code

Werkzeuge

Warum dokumentieren?

01

02

03

04

Lebendig + ausführbar

05

Fazit + Ausblick

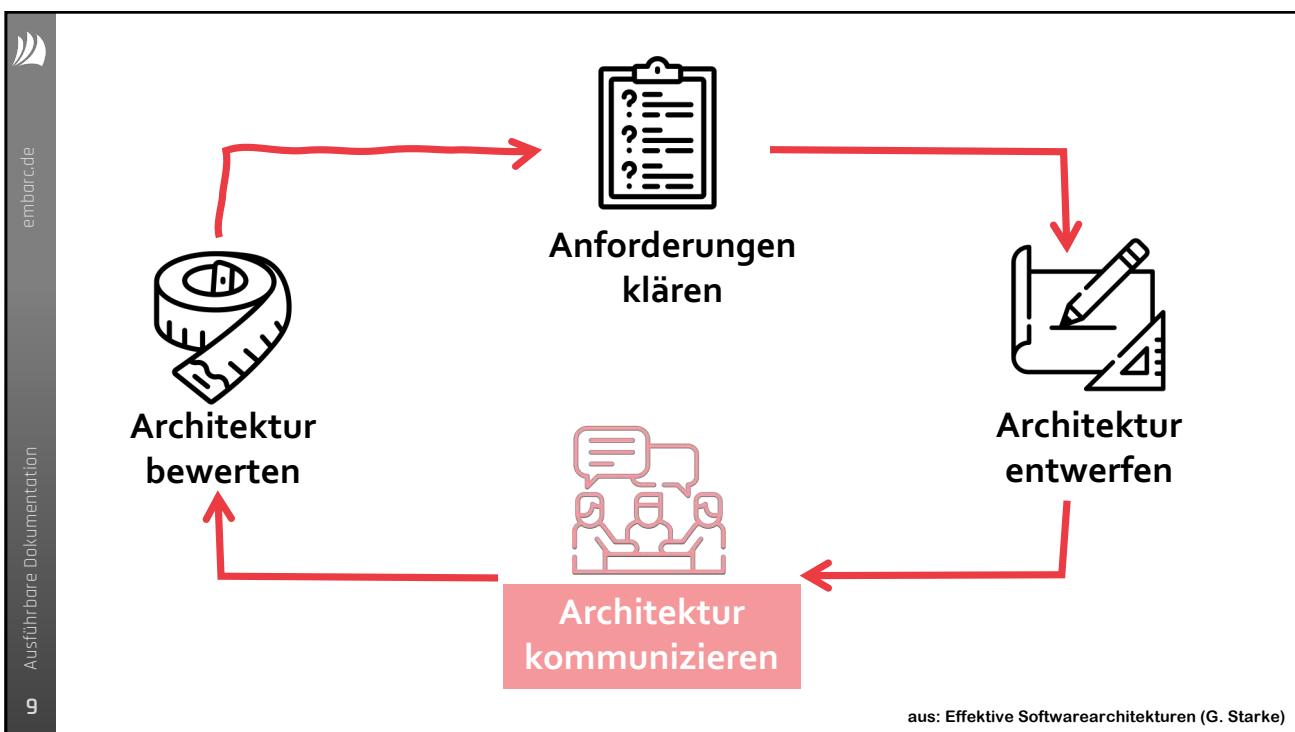
7

D1.

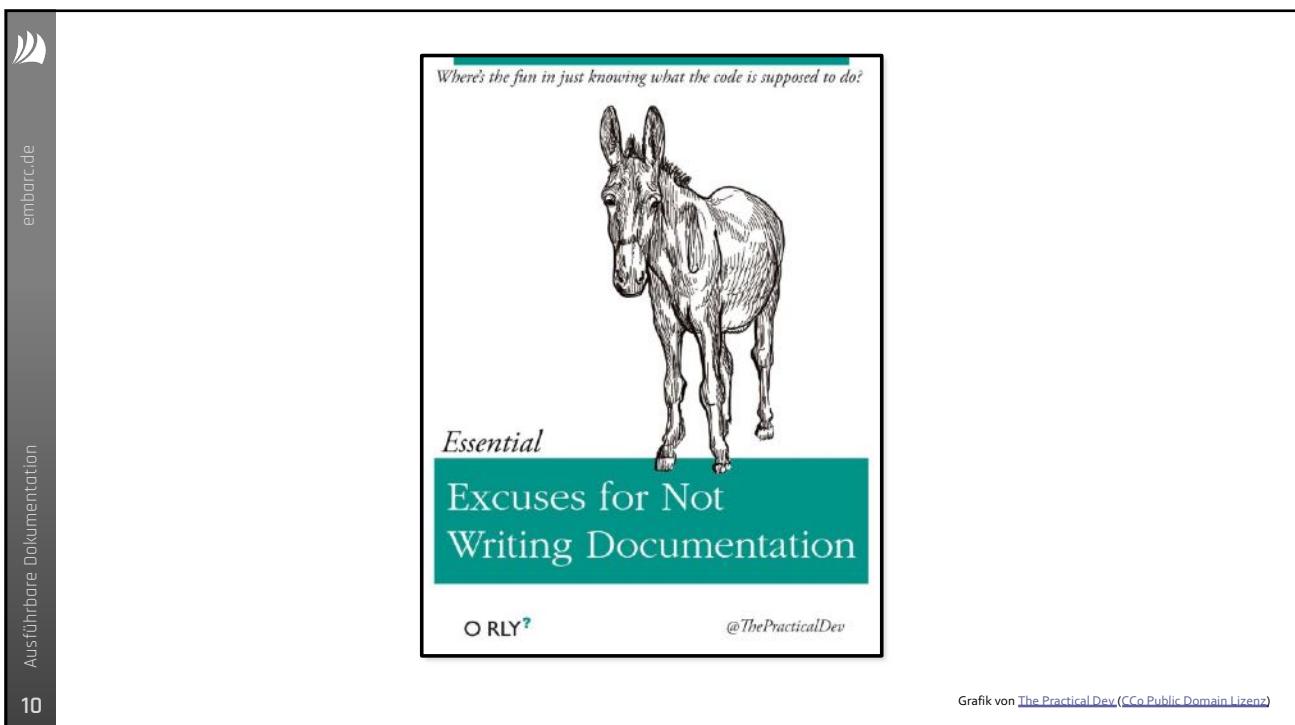
Warum dokumentieren?

8

4



9



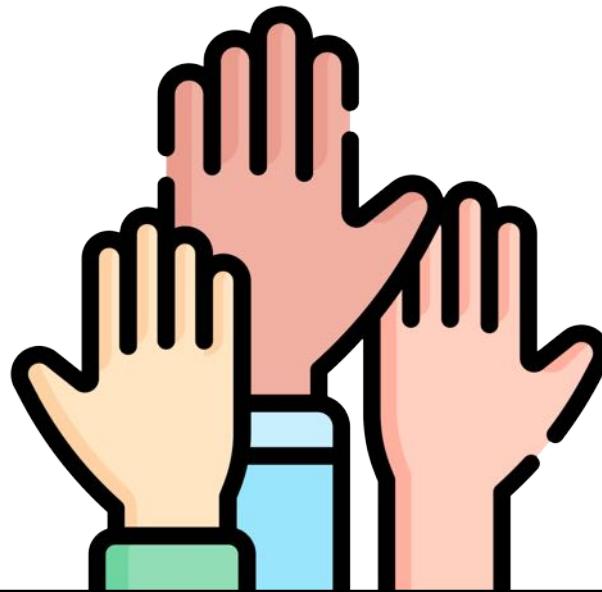
10

Eure Meinung ist gefragt

embarc.de

Ausführbare Dokumentation

11



Bitte schreibt in den Chat:

Welche **Ausreden** und
Gründe kennt ihr,
warum **nicht**
dokumentiert wird

11

Gute Gründe für Dokumentation

embarc.de

Ausführbare Dokumentation

12



Entwurfs-
unterstützung



Kommunikation



Bewertbarkeit



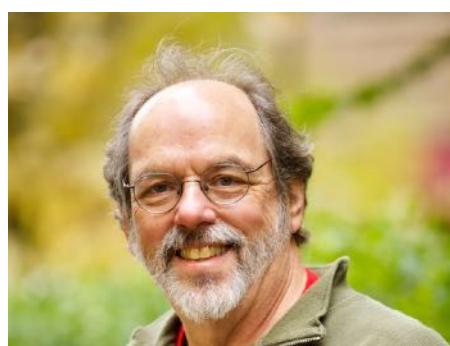
Frage nach
dem Warum



Neue Mitarbeiter



Was ist mit Wikis?



Wiki = Where Information Kills Itselfs



embarc.de

Ausführbare Dokumentation

15

I PREFER CODE



OVER CONFLUENCE

02.

Docs-as-Code

16

Maturity Model

Docs-as-Code



Grundlage für Docs-as-Code legen



Konventionell dokumentieren

Mit Grafiken effizient umgehen



Den übertriebenen Einsatz von UML-Werkzeugen sowie kommerziellen Grafikprogrammen vermeiden und die **Einstiegschürde** durch Einsatz schlanker, freier Tools **verringern**.



Dokumentation kontinuierlich verbessern



Mit Implementierung in Einklang bringen

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 **Entwicklungs-werkzeugen** erstellen. **Automatisierung** nutzen und Resultate regelmäßig in CI/CD-Pipeline erzeugen.

Mit Markup-Sprachen starten, Dokumentation modularisieren und auf Zielgruppen zugeschnittene Ergebnisse erstellen. Die Inhalte in der **Versionsverwaltung** sammeln.



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

Architekturkonzepte im Quellcode explizit (**sichtbar**) machen, daraus **Architekturenregeln** ableiten, festhalten und aus der Dokumentation regelmäßig als Tests automatisiert verifizieren.

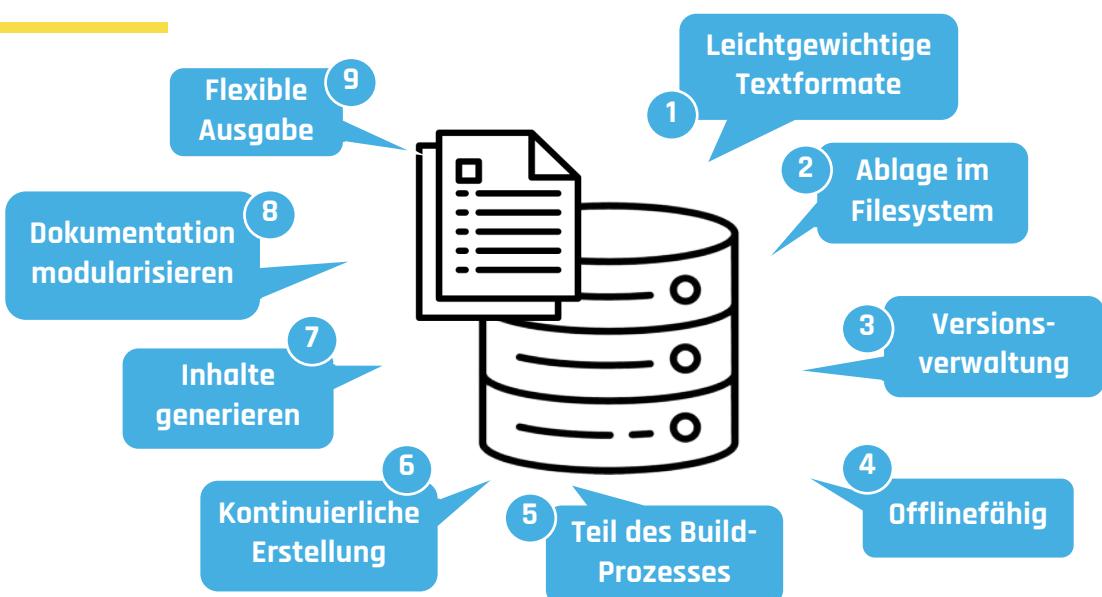
(Unser) täglich Entwickler-Brot

- Plain-Text
- Entwicklungsumgebung
- Kommandozeilenwerkzeuge
- **Versionsverwaltung**



Foto von geralt: <https://pixabay.com/de/unternehmer-start-start-up-karriere-606076/> (CC0 Public Domain Lizenz)

Documentation-as-Code



03.

Lebendige & ausführbare Dokumentation

22

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)



embarc.de

Ausführbare Dokumentation

23

23

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 Annotations oder OpenAPI-Spezifikationen generiert wird
- Systemdiagramme, die sich automatisch aus Code- oder Konfigurationsanalysen ergeben



Continuous Documentation





Lebendige Architekturdokumentation

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





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

Ausführbare Dokumentation

Auf die **Zielgruppen** ausgerichtete Artefakte erzeugen



Architekturenregeln und -tests integrieren



Synchrone Auslieferung ermöglicht die stets aktuelle Bereitstellung



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



Feedback durch die Leser ermöglichen



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?





Validierung der Architekturregeln



Explizite Architekturkonzepte im Code



Transparenz durch Dokumentation

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



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



Validierung mit QAssistant

- 1. Integration in Build-Prozess**
- 2. Erstellung von Regeln: Abbildung von Architekturkonzepten**

```

MATCH
  (dto:Java:type)
WHERE
  dto.name ends with "DTO"
RETURN
  dto as DTO
  
```

Validierung mit JQAssistant

3. Erstellung von Regeln: Formulierung von Constraints

```

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

Validierung mit JQAssistant

Lernen
durch
Fehler

4. Verletzungen behandeln → Schmerzen (er)leben

```

[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.AssignedDTO
[ERROR]   DTO=org.jqassistant.demo.architecture.hexagonal.shared.adapters.primary.rest.exception.ErrorDTO
[ERROR] -----
```

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 {
```

Docs-As-Code mit jQAssistant

- 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
imolecules.ddd.IIllegalDependenciesBetweenBoundedContexts	Checks that dependencies between <code>BoundedContext</code> nodes exist only where allowed.	MAJOR	FAILURE
maven.DependencyMustBeManaged	All used Maven dependencies must be managed in a <code>dependencyManagement2</code> section.	MAJOR	FAILURE
rest:DTOMustBeLocatedInModelPackage	All DTO classes must be located in packages named "model".	MAJOR	FAILURE
imolecules.ddd.MutableEntityId	Checks that the <code>id</code> field of an entity (<code>Identifiable</code>) is only manipulated via the constructor.	MAJOR	SUCCESS

11.2.3. rest:DTOMustBeLocatedInModelPackage

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`

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





Validierung der Architekturregeln

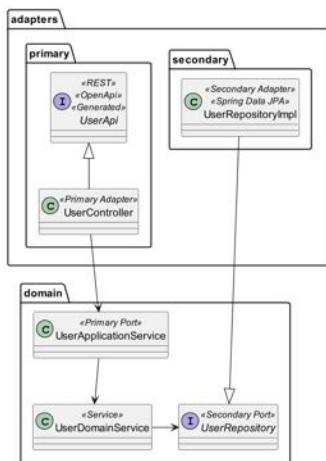


Explizite Architekturkonzepte im Code

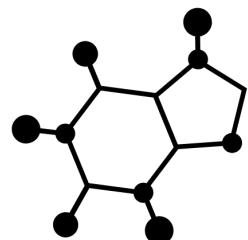


Transparenz durch Dokumentation

xMolecules, ContextMapper



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

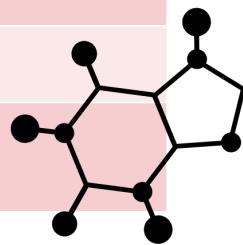


```
    @SecondaryPort  
    public interface UserRepository {  
        @SecondaryAdapter  
        @Repository  
        public class UserRepositoryImpl  
            extends SimpleJpaRepository<User, Long>  
            implements UserRepository {
```



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



```

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

```
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
}
}
```

49

49

Diagramme generieren
Code-Generatoren nutzen
Architekturregeln ableiten



**CONTEXT
MAPPER**

embarc.de

Spring Modulith

Ausführbare Dokumentation

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

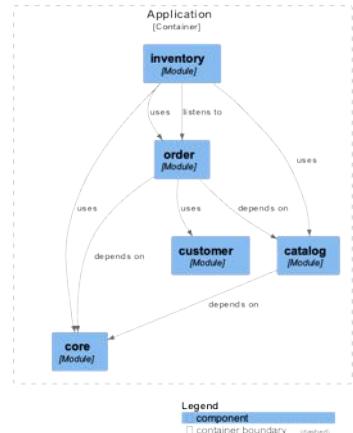


<https://spring.io/projects/spring-modulith>

56

Spring Modulith

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



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

57

Spring Modulith

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

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

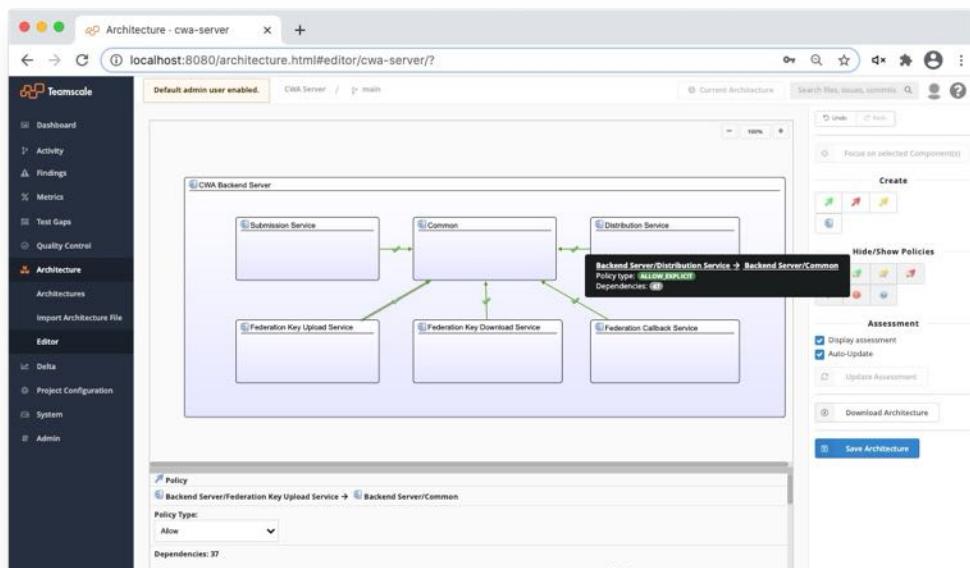
58

Abhängigkeiten in Teamscale

embarc.de

Ausführbare Dokumentation

59



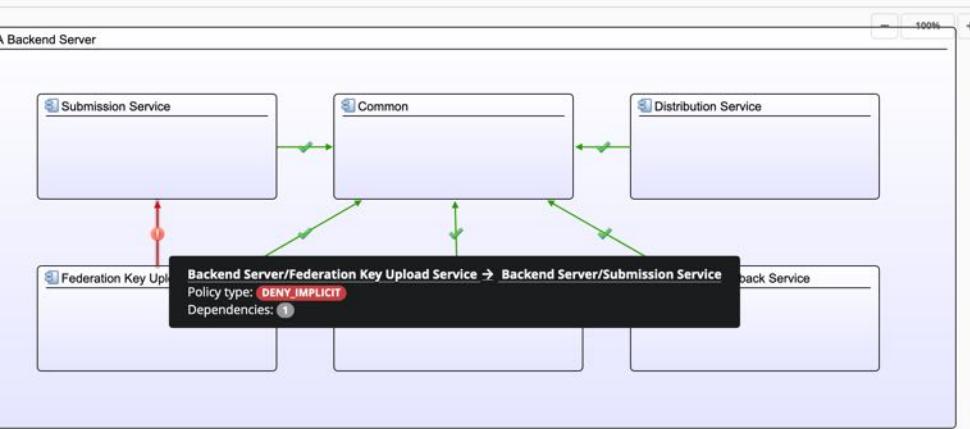
59

So sähe ein Verstoß aus ...

embarc.de

Ausführbare Dokumentation

60



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

60

05.

Fazit & Ausblick

Völlig losgelöst von der Erde
Schwebt das Raumschiff
schwerelos ...

Vorteile

- Reduziert das Risiko veralteter Dokumentation
- Fördert die Kommunikation im Team
- Unterstützt architektonische Governance
- Macht implizite Regeln explizit und überprüfbar



Architektur-Spicker

embarc.de

Ausführbare Dokumentation

63



„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/

63

Folien als PDF zum Download

embarc.de

Ausführbare Dokumentation

64

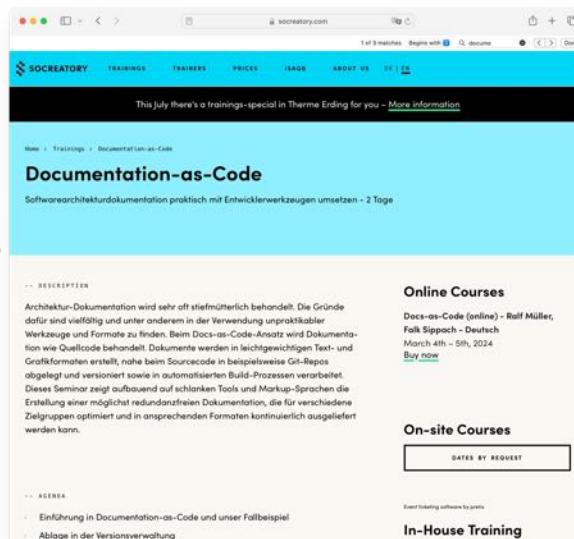


→ embarc.de/download/

Training zu Docs-as-Code

Softwarearchitekturdokumentation praktisch mit Entwicklerwerkzeugen umsetzen

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

The screenshot shows a web browser displaying a training page. The header reads "SOCREATORY TRAININGS TRAINERS PRICES IMAGE ABOUT US". Below it, a banner says "This July there's a trainings-special in Therme Erding for you - More information". The main title is "Documentation-as-Code". A subdescription states: "Softwarearchitekturdokumentation praktisch mit Entwicklerwerkzeugen umsetzen - 2 Tage". The page includes sections for "DESCRIPTION", "Online Courses", "On-site Courses", and "In-House Training". A red arrow points from the text "Wir kommen auch zu Euch in die Firma!" to the "In-House Training" section.

Weiterführende Blog-Posts

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




The screenshot shows a blog post with the title "Eine Einführung in docToolchain". The text discusses the introduction to docToolchain, mentioning its use in the construction industry and how it can be used to generate documentation from source code. It includes a photo of a modern building under construction.



The screenshot shows a blog post with the title "Was ist eigentlich Docs-as-Code?". The text explains what Docs-as-Code is, how it differs from traditional documentation, and its benefits. It includes a photo of a modern office interior with large windows.

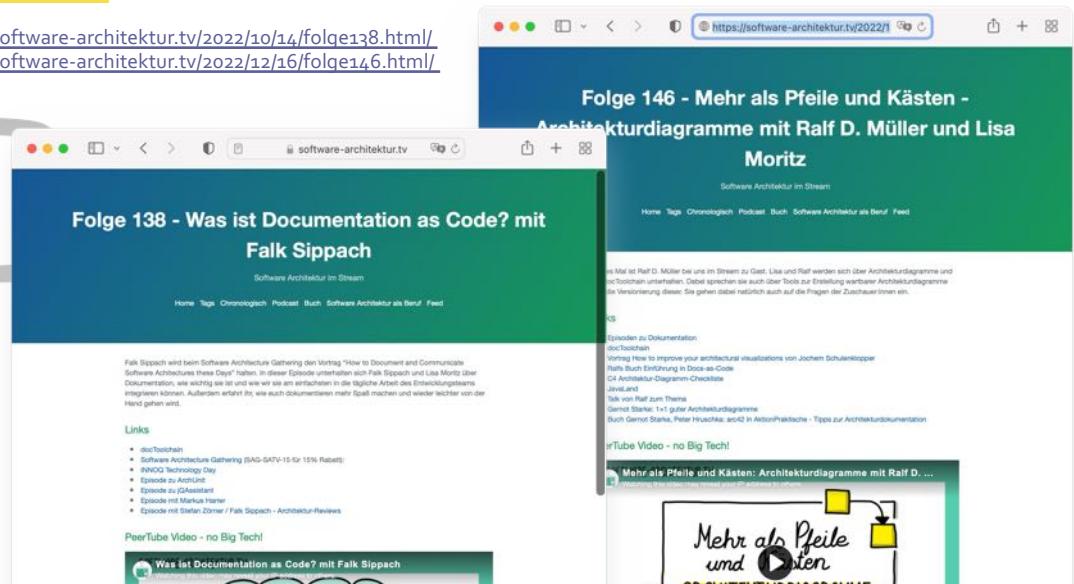
Weiterführende Videos/Podcasts

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

embarc.de

Ausführbare Dokumentation

67



67

Feedback & Fragen?

Wir freuen uns auf Fragen,
Diskussionen, Anregungen!



68

Vielen Dank.

Ich freue mich auf Eure Fragen!

-  falk.sippach@embarc.de
-  linkedin.com/in/falk-sippach
-  [@sippsack](https://twitter.com/sippsack)
-  [@sippsack@ijug.social](https://sippsack.ijug.social)



embarc 

Falk Sippach

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



iSAQB
International Software Quality Board
Qualification Board

 **Java**™
Champions

 **Cyber LAND**



Schwerpunkte

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

embarc 