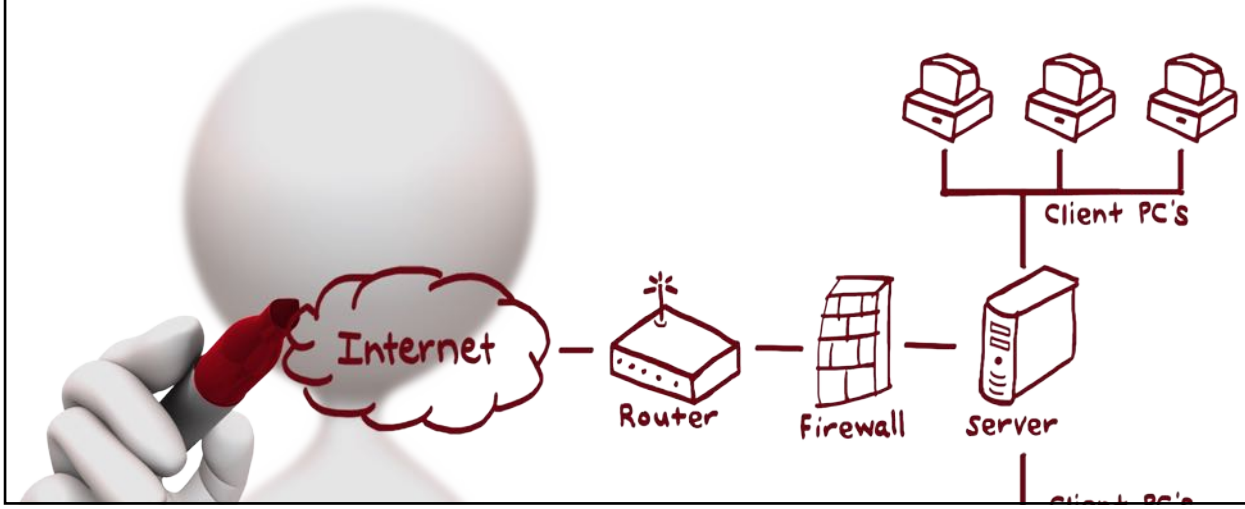
Falk Sippach
embarc

Ralf D. Müller
DB Systel GmbH

Phantastische Diagramme

…und wie Du sie selbst erstellst



1

Phantastische Diagramme

…und wie Du sie selbst erstellst

Zusammenfassung

Die wichtigste Aufgabe eines Software-Architekten besteht darin, die **Architektur zu kommunizieren**.

Neben den textuellen Inhalten gilt es auch, Grafiken zu erstellen. Getreu dem Motto "**ein Bild sagt mehr als tausend Worte**" helfen Diagramme bei einer effektiven und pragmatischen Dokumentation. Damit sie wirken, müssen sie leicht erfassbar, stets aktuell und korrekt sein..

In diesem Talk spüren wir zuerst die Schwachstellen vieler Diagramme auf und überlegen uns anschließend, wie wir sie umgehen können. Das Ergebnis ist eine **Checkliste für richtig gute, fast schon phantastische Architektur-Diagramme**.

Als Bonus werden wir aufzeigen, wie **Diagramme wartbar mit dem Docs-as-Code Ansatz umgesetzt werden**, um sie jederzeit aktuell und im Einklang mit der Architektur halten zu können.

2

Falk Sippach

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

fs@embarc.de
@sippsack
➔ xing.to/fsi

4

Ralf D. Müller

- was mit Dokumentation, Security und Architektur by der DB Systel
- Maintainer von docToolchain, Contributor arc42
- co-Autor

ralf.d.mueller@deutschebahn.com
@ralfdmueller
➔ xing.to/rdmuller

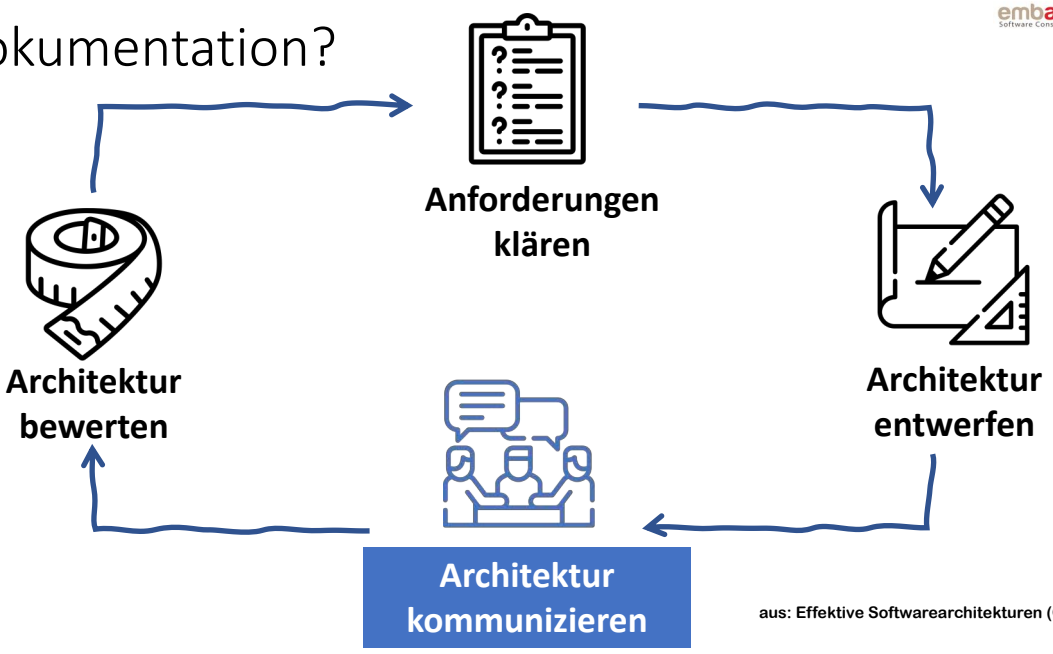
5

Mission Statement

- **Schwachstellen** vieler Diagramme aufzeigen.
- Checkliste für **richtig gute Architekturdiagramme** diskutieren.
- Für Architekturdokumentation **relevante Diagrammarten** vorstellen.
- Diagramme **wartbar mit Docs-as-Code** Ansatz umsetzen.
- **Leichtgewichtige Werkzeuge** zur Diagrammerstellung zeigen.

7

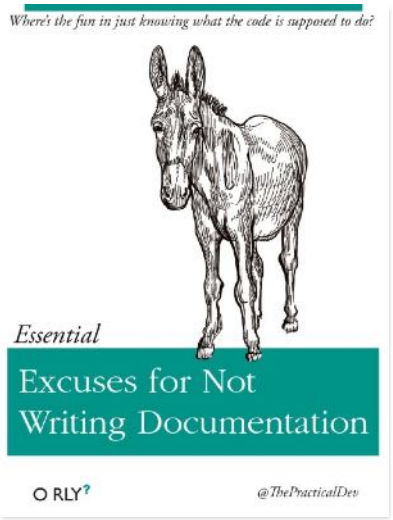
Dokumentation?



aus: Effektive Softwarearchitekturen (G. Starke)

8

Gute Gründe …



Grafik von The Practical Dev (CC0 Public Domain Lizenz)

Entwurfsunterstützung

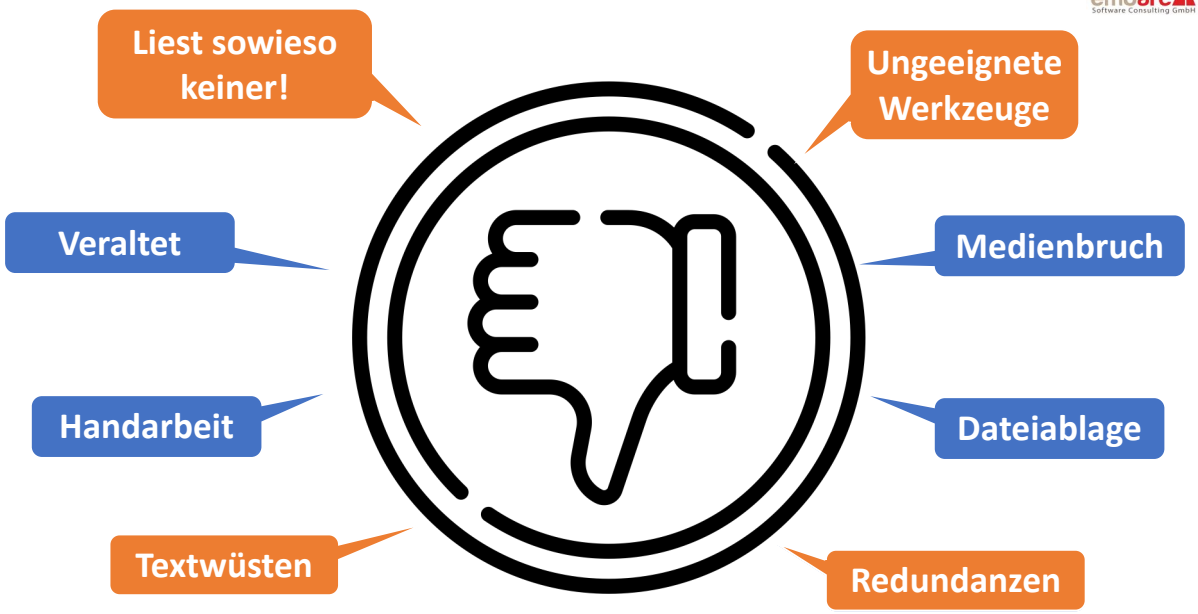
Frage nach dem Warum

Bewertbarkeit

Kommunikation

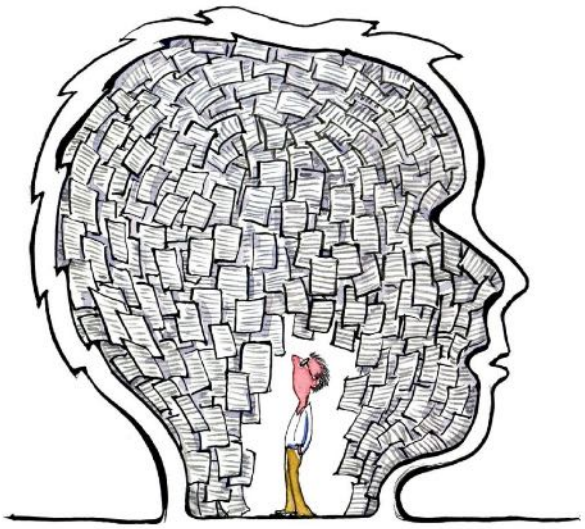
Neue Mitarbeiter

9



10

Ein Bild sagt mehr als 1000 Worte …

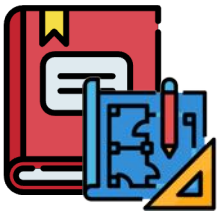


11

Agenda

Gute Diagramme

Werkzeuge



Einbettung in Dokumentation

12

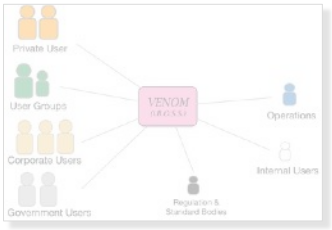
Agenda

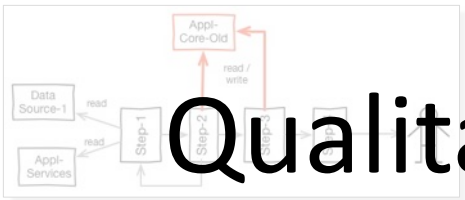
Gute Diagramme

Werkzeuge

Einbettung in Dokumentation

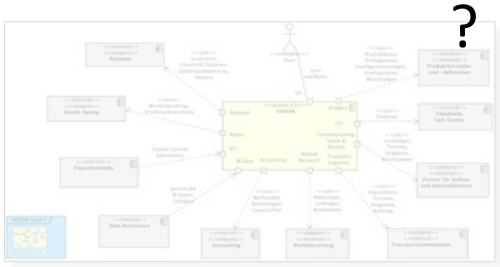
Einleitung

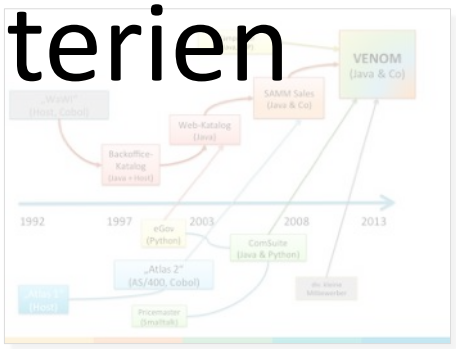




Qualitätskriterien

?





Tipp 1: Qualitätskriterien für Diagramme

Ein Diagramm dient der **Kommunikation**

Darum muss es **lesbar und verständlich** sein

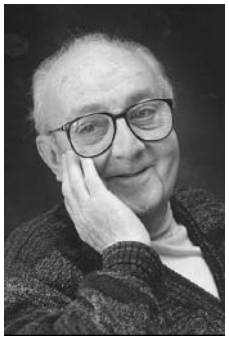
Es muss die gewünschte **Information transportieren**

Denke an einen **Rückkanal**, sonst handelt es sich um einseitige Kommunikation

Denke an die **Wart- und Modifizierbarkeit**

.

„All Models are Wrong but some are Useful“



George Box

Was ist ein Modell?

Bild von Hans Benn auf Pixabay

17

Was ist ein Modell?

Bild von Hans Benn auf Pixabay

https://www.wolfsburgwest.com/cart/DetailsList.cfm?ID=211971013B

18

Tipp 2: Zweck des Diagramms beachten

Ein Diagramm / eine View erfüllt einen **einzigen Zweck**

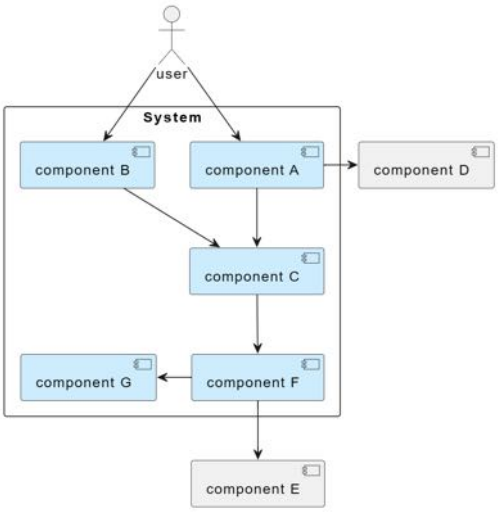
Widerstehe der Versuchung **zu viele Arten von Informationen** in einem Diagramm unterzubringen

Ja, betrachtet man das Diagramm aus der **falschen Sicht, wird es falsch sein**

Darum **beschrifte das Diagramm**, um den Zweck klarzustellen

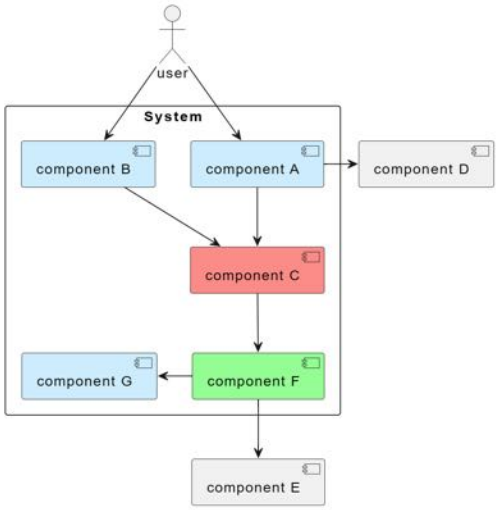
Visual Style

Ein Beispiel



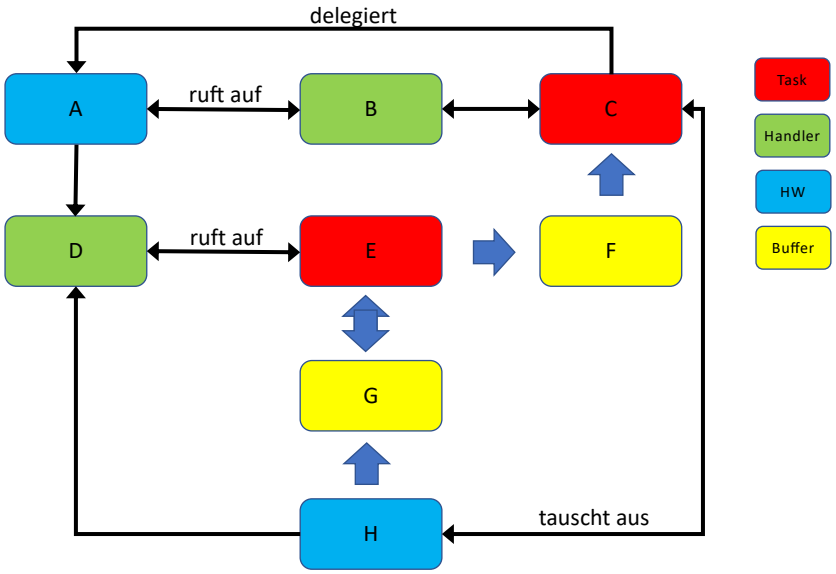
21

Farben



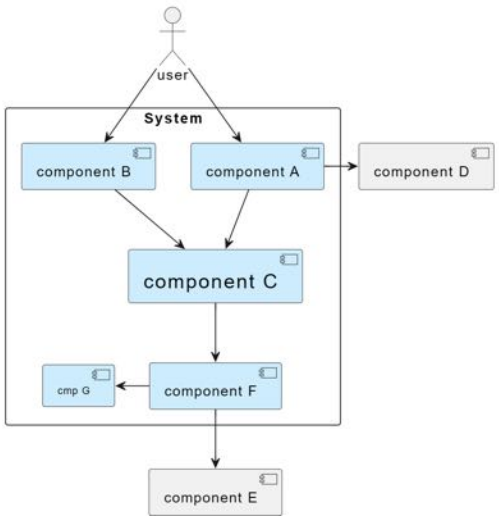
22

So nicht…



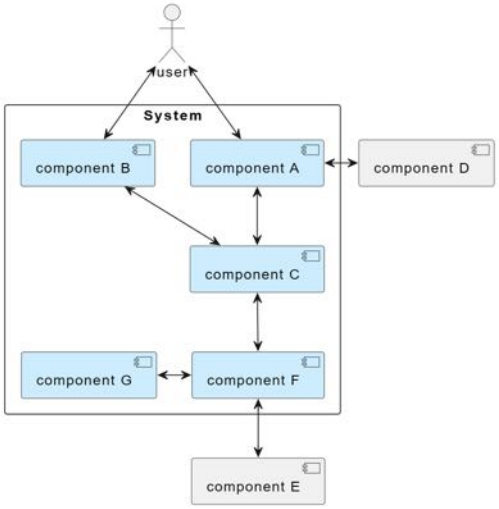
23

Größe



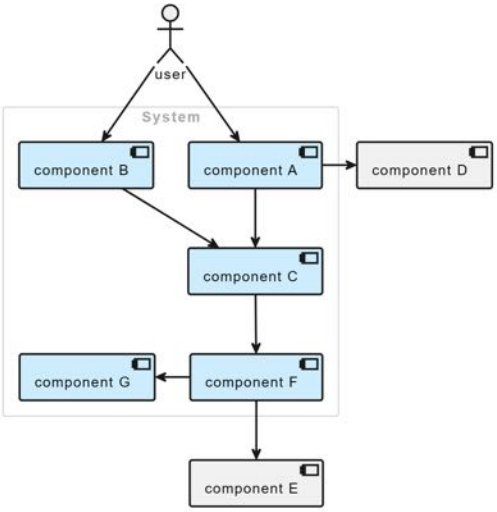
24

Pfeilrichtung



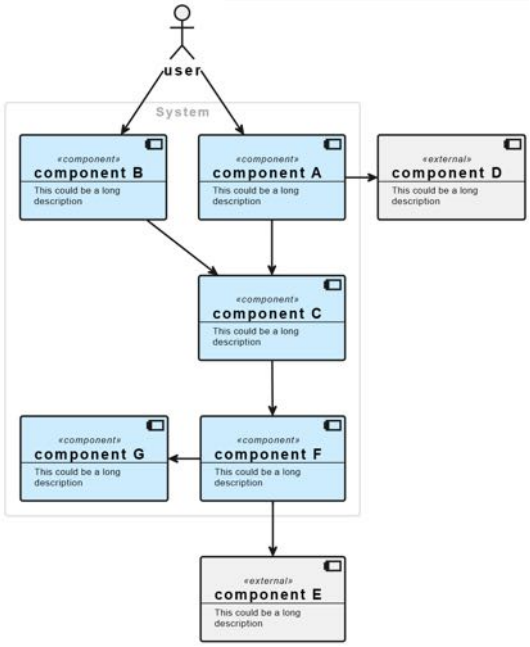
25

Linienstärke



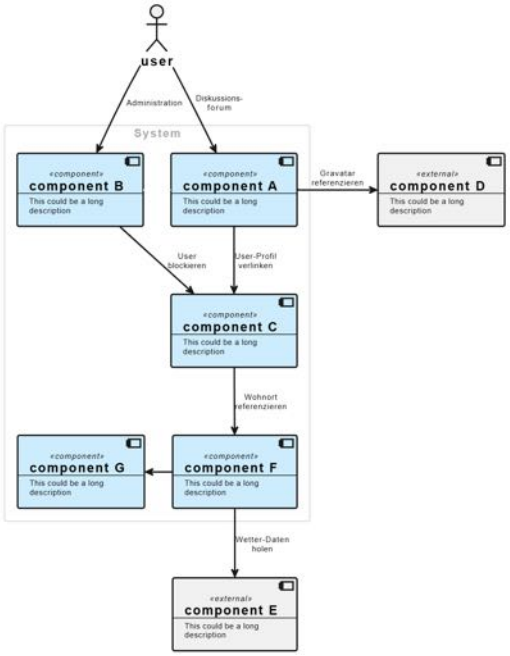
26

Text-Stile



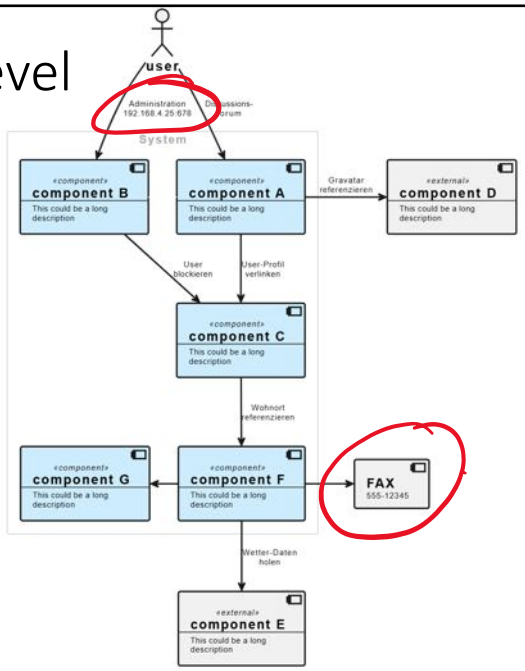
27

Beschriftung

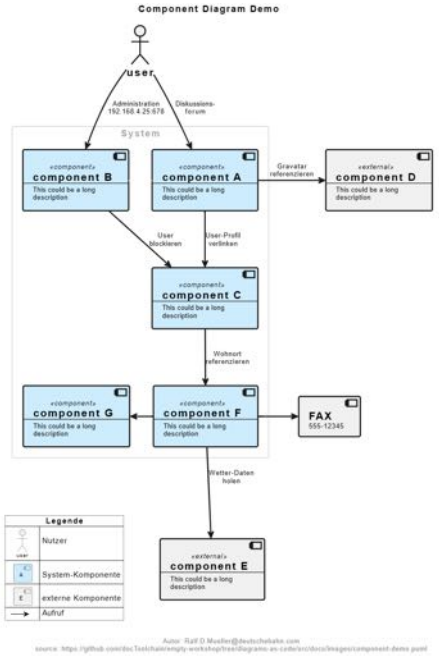


28

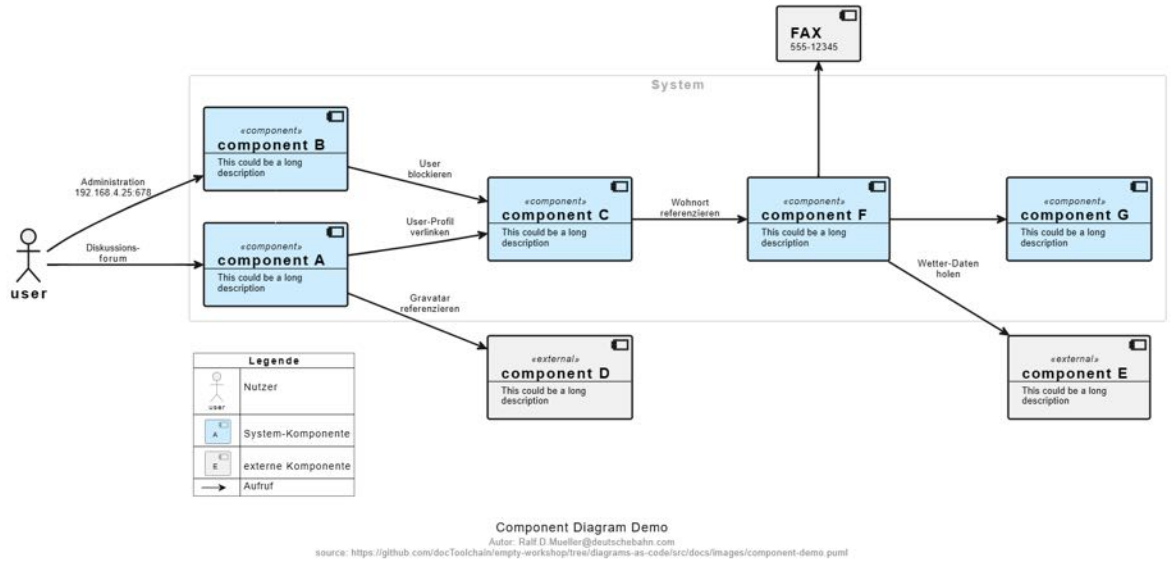
Abstraktionslevel



Legende



Landscape



Tipp 3: Visuelle Stile

Farben sparsam und entsprechend ihrer natürlichen Bedeutung einsetzen

Decorator oder Formen zur Unterscheidung der Elemente einsetzen

A **A** Schriftgrößen und Stile bewusst einsetzen

Verbindungen und Elemente beschriften

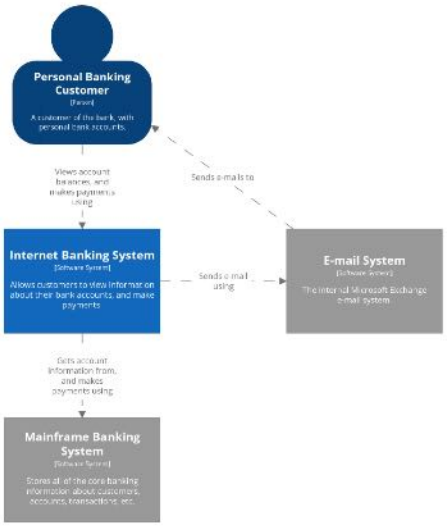
Unterschiedliche Größen bewusst einsetzen

Verbindungen nicht überlappen und nur in eine Richtung

c4model.com

34

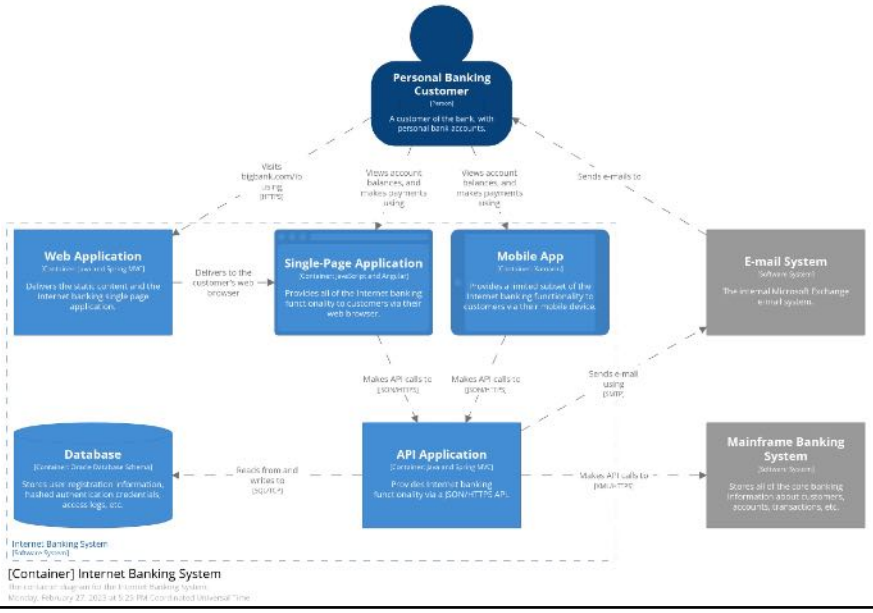
c4model.com: Context



[System Context] Internet Banking System

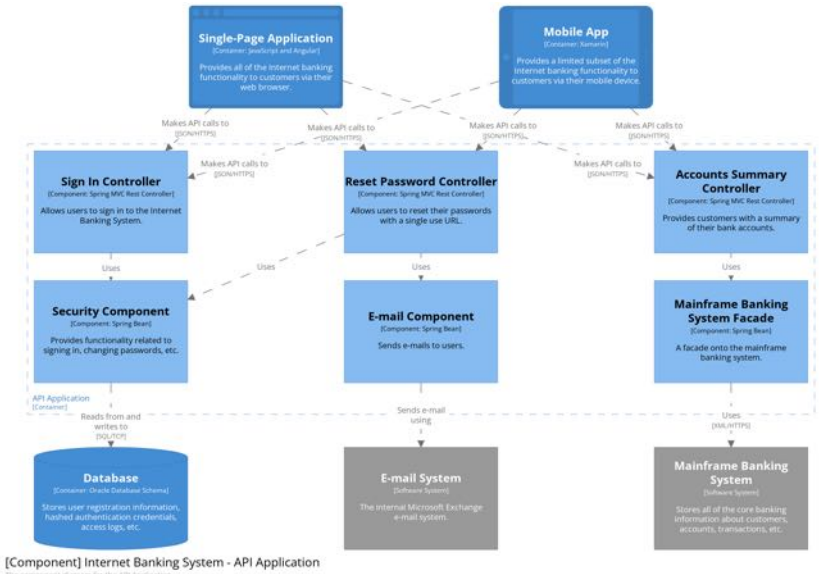
35

c4model.com: Containers



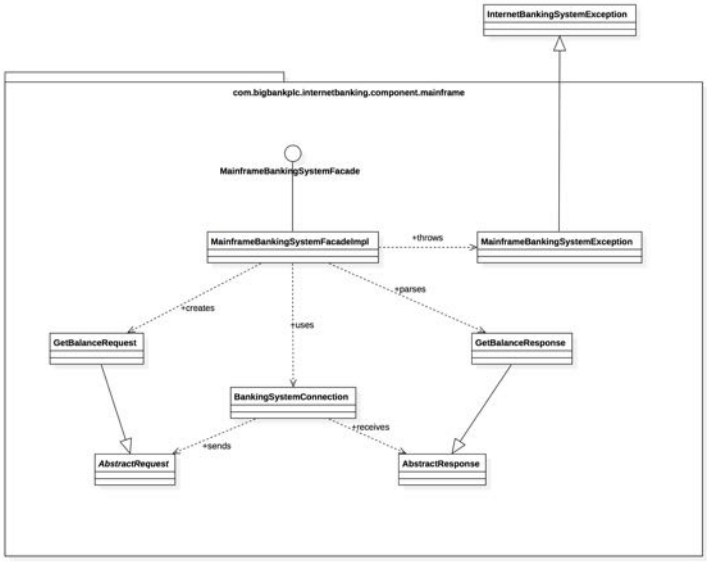
36

c4model.com: Components



37

c4model.com: Classes

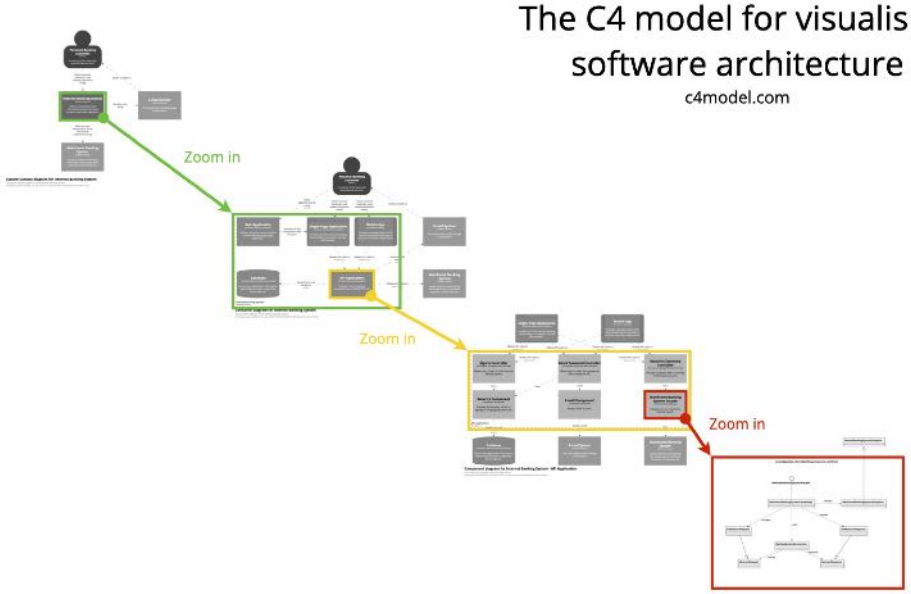


Quelle der C4-Diagramme c4model.com

38

The C4 model for visualising software architecture

c4model.com



Level 1 Context | Level 2 Containers | Level 3 Components | Level 4 Code

39

Agenda

Gute Diagramme

Werkzeuge

Einbettung in Dokumentation

PlantUML

```
1  @startuml
2
3  User ->  Server: via Browser
4    Server ->  Database: fetch profile
5    Server <-- Database
6  User <-- Server
7
8  @enduml
```

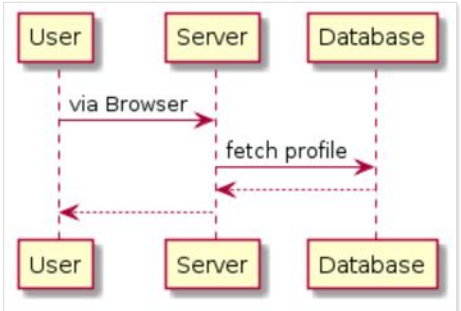
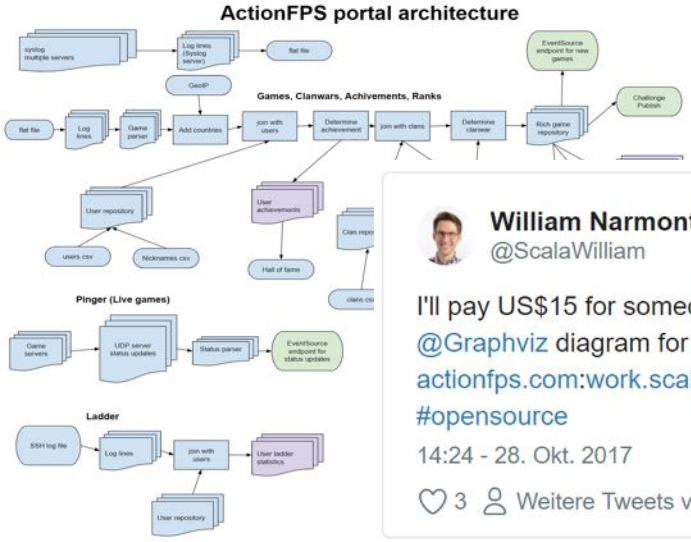


Diagramme: PlantUML



William Narmontas
@ScalaWilliam

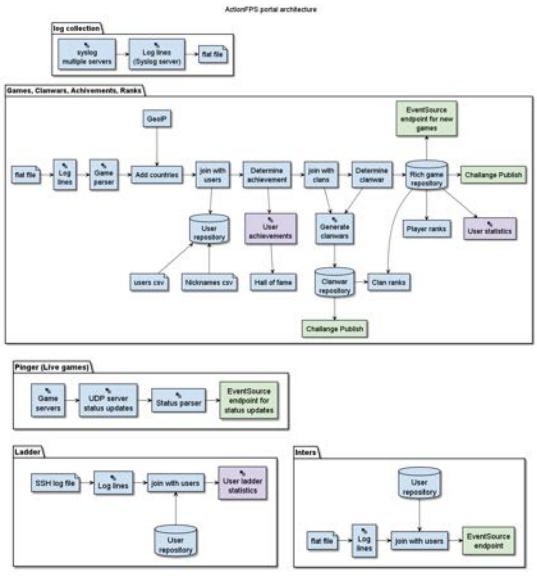
I'll pay US$15 for someone who can do this @PlantUML or @Graphviz diagram for actionfps.com:work.scalawilliam.com/graphviz-plant… #opensource

14:24 - 28. Okt. 2017

♡ 3 Weitere Tweets von William Narmontas ansehen

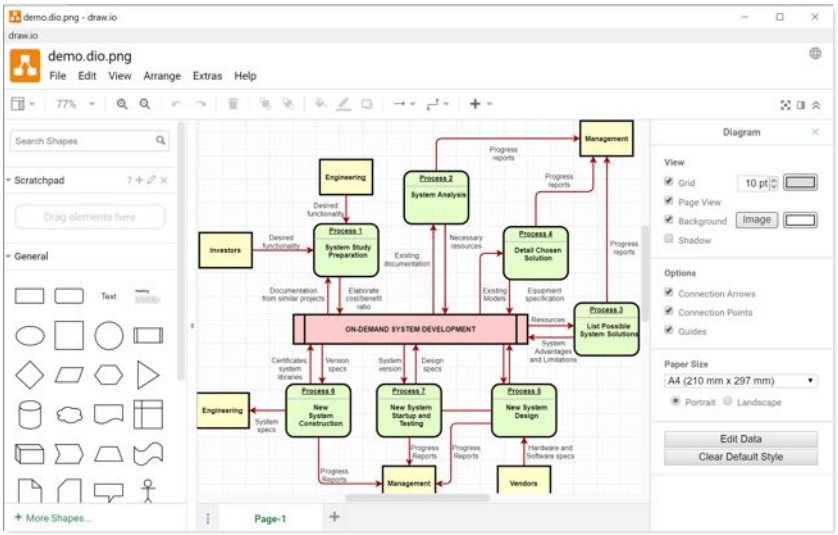
42

Diagramme: PlantUML

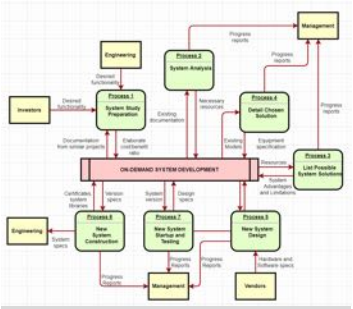


43

draw.io/Diagrams.net



draw.io/Diagrams.net



<xml>
...
</xml>

PNG-Diagramm

Meta-Daten

Kroki

Creates **diagrams** from **textual** descriptions!

Kroki provides a unified API with support for BlockDiag (BlockDiag, SeqDiag, ActDiag, NwDiag, PacketDiag, RackDiag), BPMN, Bytefield, C4 (with PlantUML), Ditaa, Erd, Excalidraw, GraphViz, Mermaid, Nomnoml, Pikchr, SvgBob, UMLet, Vega, Vega-Lite, WaveDrom... and more to come!

#Features

Ready to use

Diagrams libraries are written in a variety of languages: Haskell, Python, JavaScript, Go, PHP, Java... some also have C bindings. Trust us, you have better things to do than install all the requirements to use them. Get started in no time!

Simple

Kroki provides a unified API for all the diagram libraries. Learn once use diagrams anywhere!

Free & Open source

All the code is available on GitHub and our goal is to provide Kroki as a free service.

Fast

Built using a modern architecture, Kroki offers great performance.

25 Diagrams-as-Code Libraries!

https://kroki.io/

46

Vega – A Visualization Grammar

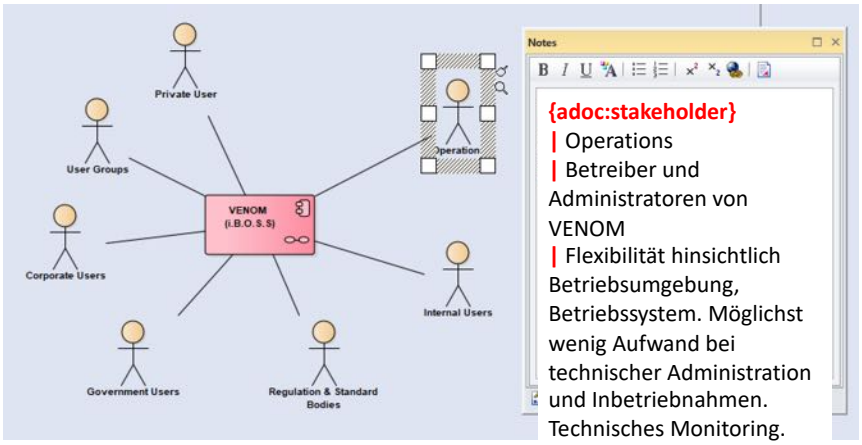
https://vega.github.io/vega/

47

Diagramme: Nicht malen, modellieren!

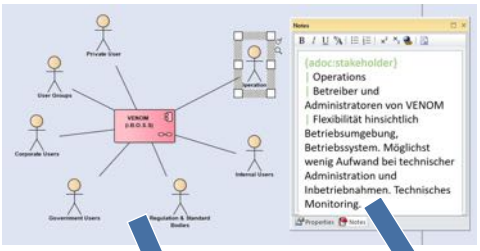
48

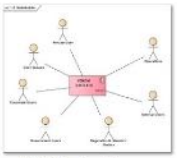
treat Docs-as-Code: automate!



49

treat Docs-as-Code: automate!



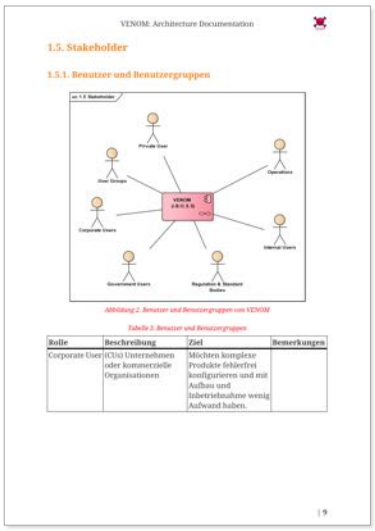


1.5_Stakeholder.png

stakeholder.ad

```
=== Stakeholder

==== Users and Groups of Users


image::ea/1.5_Stakeholder.png[]

[cols="2,3,3,2" options="header"]
.Users and Groups of Users
|===
| Role | Description | Goal  |
Comment

include::../../ea/stakeholder.ad[]

|===
```

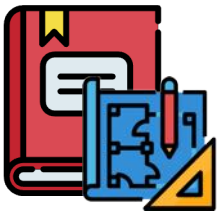

treat Docs-as-Code: automate!



Agenda

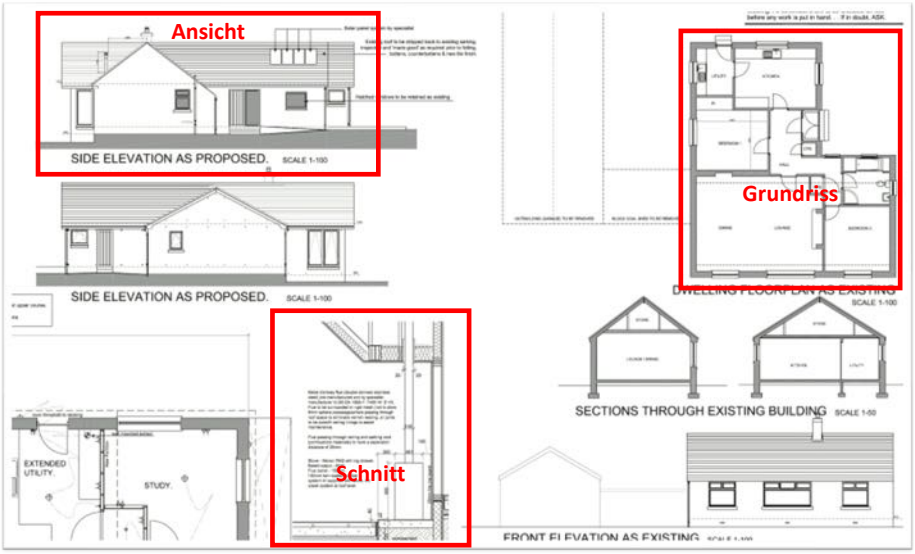
Gute Diagramme

Werkzeuge



Einbettung in Dokumentation

Analogie Bauwesen



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/

arc42
➔https://arc42.de/

Sichten in arc42

Kontextabgrenzung (System Context)
Zeigt die beteiligten Fremdsysteme und Benutzer, mit denen das zu beschreibende System interagiert.

Bausteinsicht (Building Block View)
Zeigt die Struktur des Softwaresystems in Form seiner Bausteine, und wie diese zusammenhängen. Bündelt viele Entscheidungen rund um Verantwortlichkeiten.

Laufzeitsicht (Runtime View)
Zeigt Elemente der Baustein- und/oder Kontextsicht in Aktion, veranschaulicht dynamische Strukturen und Verhalten des beschriebenen Systems.

Verteilungssicht (Deployment View)
Zeigt wie die Software in Betrieb genommen (verteilt) wird, und wie die Zielumgebung aussieht.

Ursprung: 4+1 Sichten

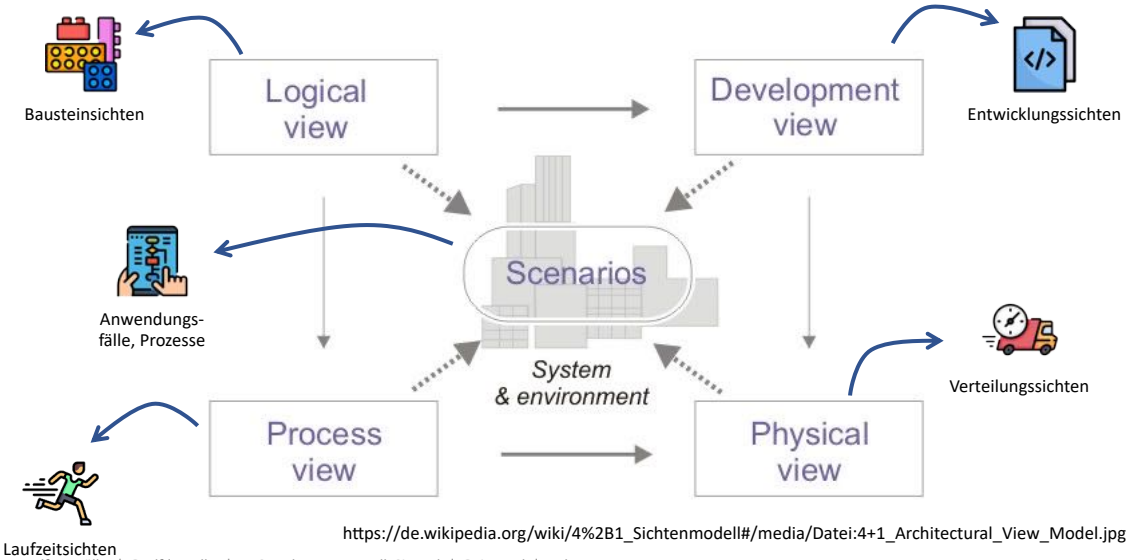
Philippe Kruchten

> *Das **4+1-Sichten-Softwarearchitekturmodell** ist ein weit verbreitetes Modell der Sichten auf ein Softwaresystem, das von Philippe Kruchten zur „**Beschreibung der Architektur eines Software-intensiven Systems auf der Grundlage von mehreren konkurrierenden Sichten**“ entwickelt wurde.*

https://de.wikipedia.org/wiki/4%2B1_Sichtenmodell

56

Elemente des 4+1

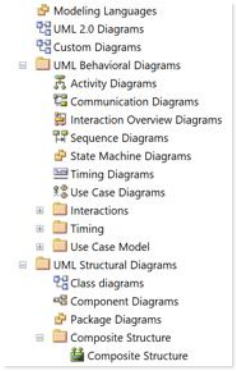


https://de.wikipedia.org/wiki/4%2B1_Sichtenmodell#/media/Datei:4+1_Architectural_View_Model.jpg

57

To UML or to not UML?

> ***"Wir benutzen für unsere Architekturdiagramme UML. Ist standardisiert. Kann jeder lesen."***



-- unbekannte Softwarearchitekt:in

Mögliche UML-Diagrammarten

Klassendiagramm
Kommunikations-diagramm
Sequenzdiagramm

Komponenten-diagramm
Paketdiagramm

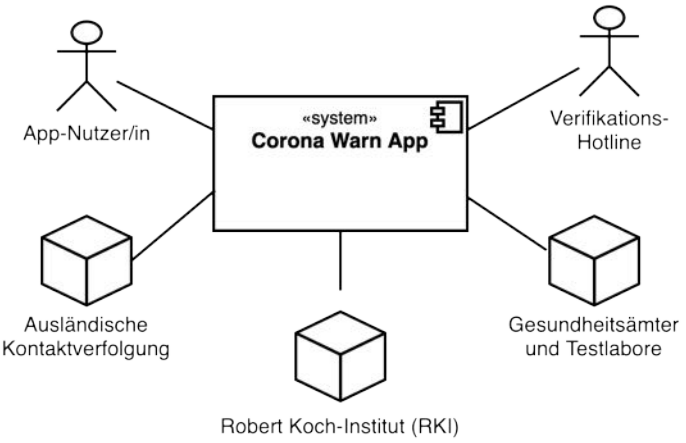
Anwendungsfall-diagramm

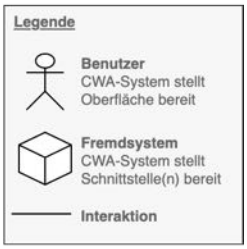
Verteilungs-diagramm

Sequenzdiagramm
Aktivitätsdiagramm

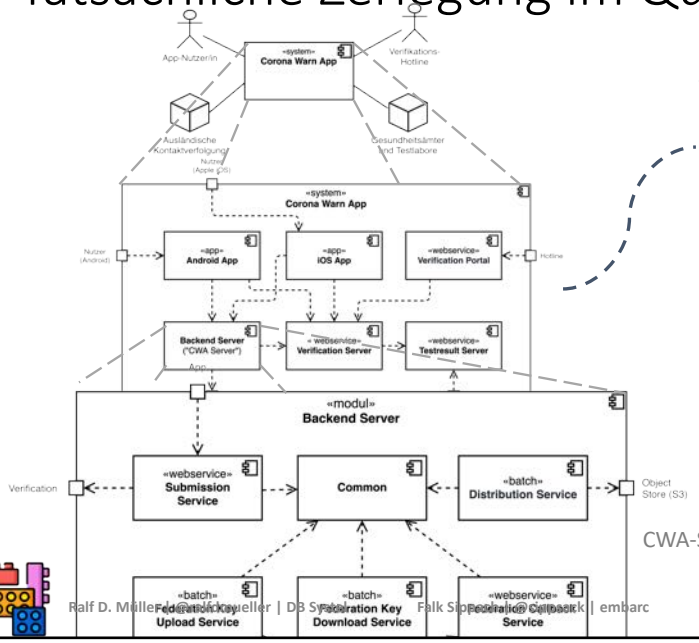
Kontextabgrenzung

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





Tatsächliche Zerlegung im Quelltext



„Bausteinsicht, Ebene 1“

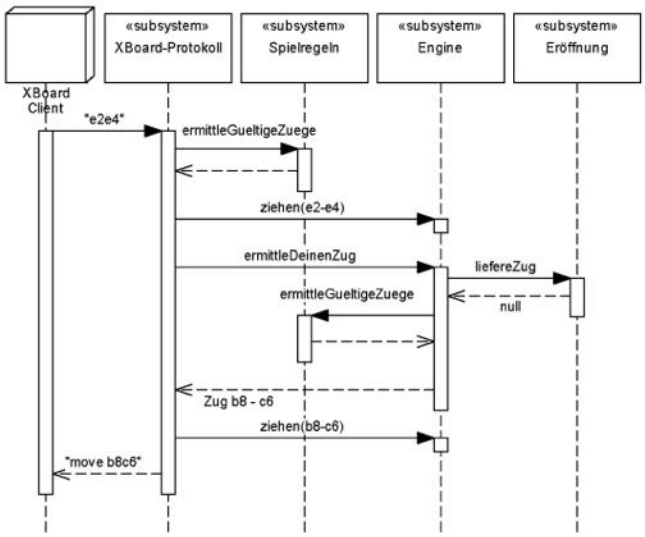
GitHub-Repositories

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

Beispiel eines Ablaufes

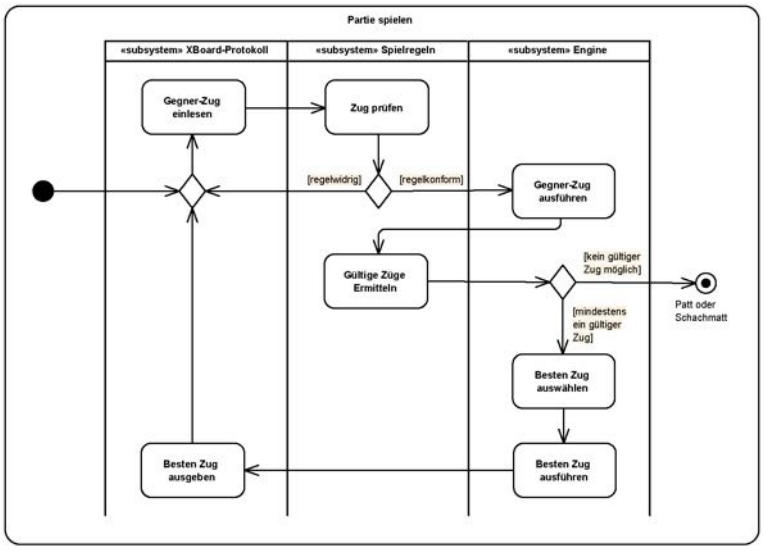


DOKCHESS

https://www.dokchess.de/

62

Beispiel eines Ablaufes

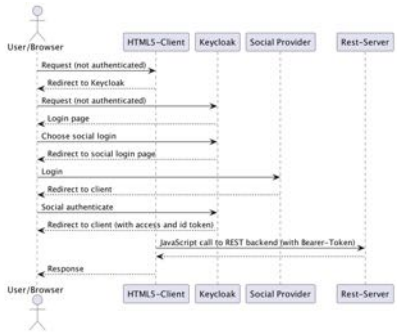


DOKCHESS

https://www.dokchess.de/

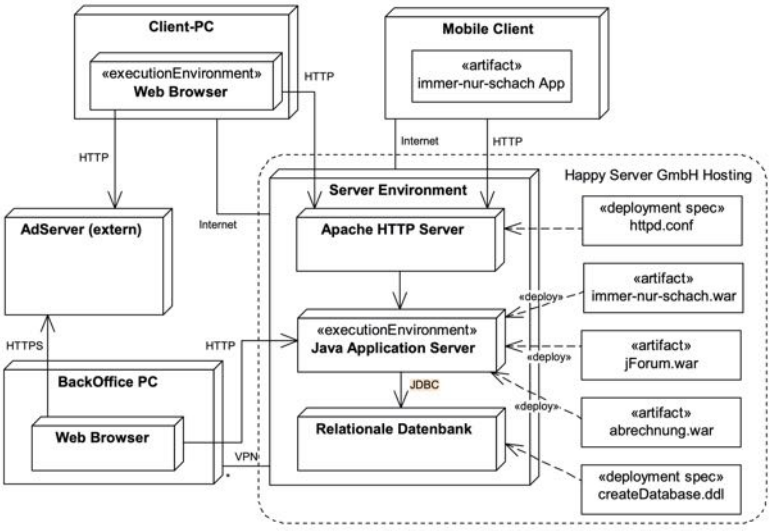
63

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
```

Verteilungssicht "immer-nur-schach"



aus: "Software-Architekturen dokumentieren und kommunizieren" von Stefan Zörner

Waren das schon alle Sichten?

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
...

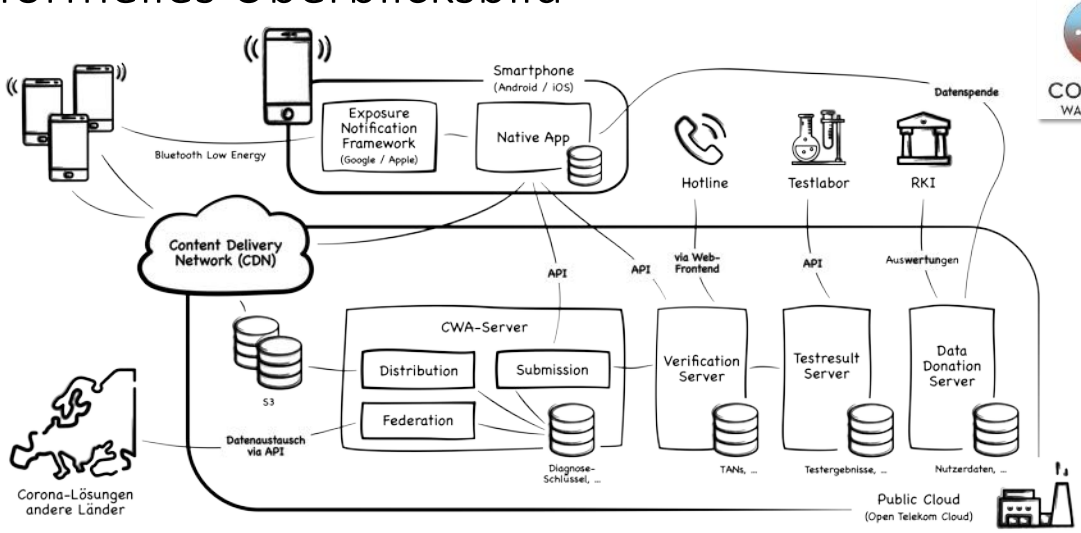
9. Entwurfsentscheidungen
9.1 Entwurfsentscheidung 1
9.2 Entwurfsentscheidung 2
...

10. Qualitätsszenarien
10.1 Qualitätsbaum
10.2 Bewertungsszenarien

11. Risiken

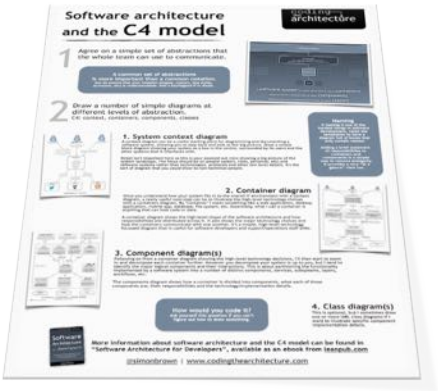
12. Glossar

Informelles Überblicksbild



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

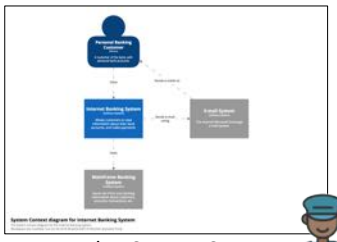
Alternative: C4 Model (Simon Brown)

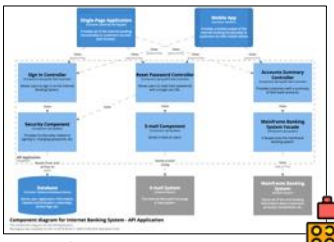


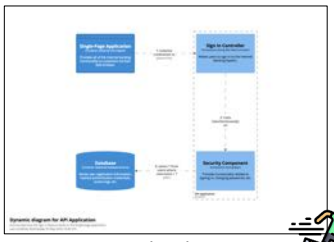
- Agree on a simple set of abstractions that the whole team can use to communicate.
- Draw a number of simple diagrams at different levels of abstraction.
- C4: context, containers, components, classes (code)

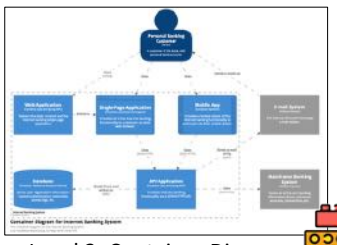
➔ http://static.codingthearchitecture.com/c4.pdf

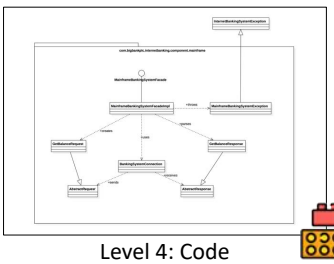
Alternative C4: Inhalte

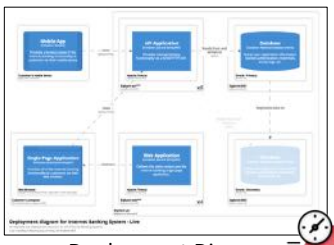

Level 1: System Context


Level 3: Component Diagram


Dynamic Diagram


Level 2: Container Diagram


Level 4: Code


Deployment Diagram

Alle Sichten auf einen Blick

4 + 1 (Kruchten)

Bausteinsichten

Entwicklungssichten

Kontextsichten

Anwendungsfälle/
Prozesse

Informelles
Überblicksbild

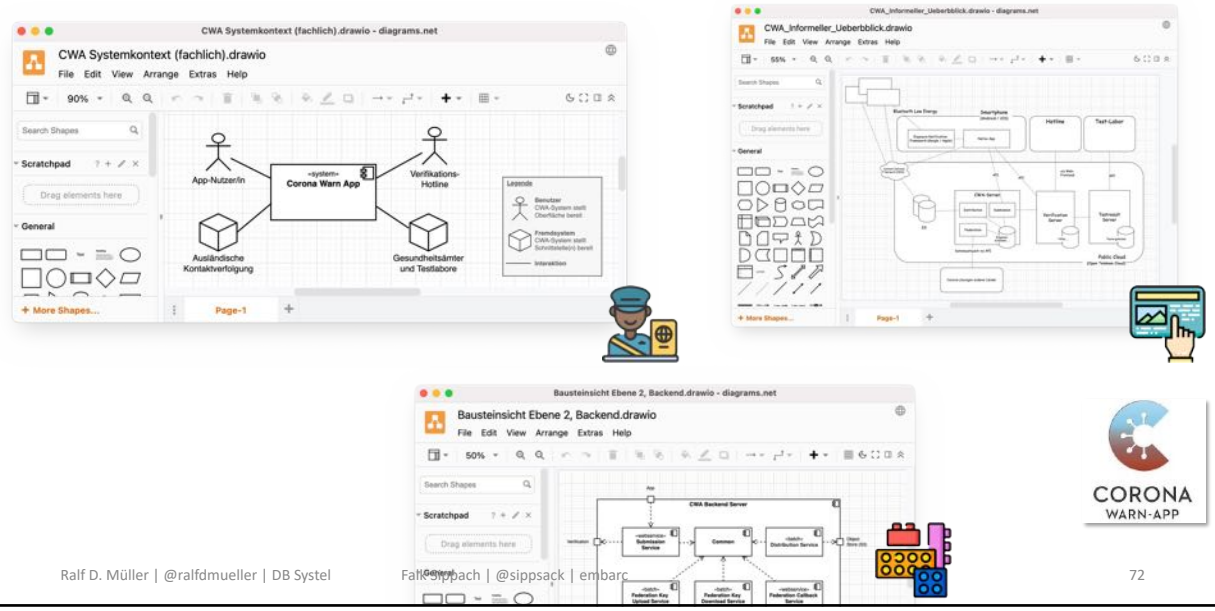
Laufzeitsichten

Verteilungssichten

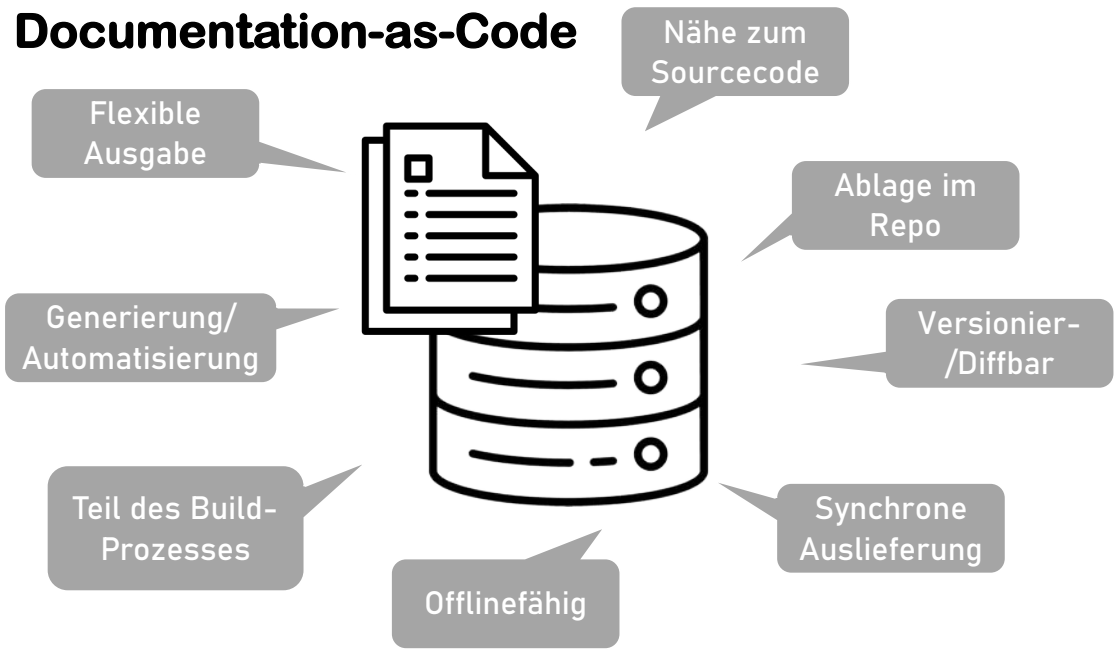
Diagramm-Tools- Empfehlungen

	draw.io/diagrams.net PlantUML (Komponenten-diagramm)	draw.io/diagrams.net
draw.io/diagrams.net	PlantUML (Anwendungsfall-diagramm)	draw.io/diagrams.net
	PlantUML (Sequenzdiagramm, Aktivitätsdiagramme)	PlantUML (Sequenzdiagramm, Aktivitätsdiagramme)

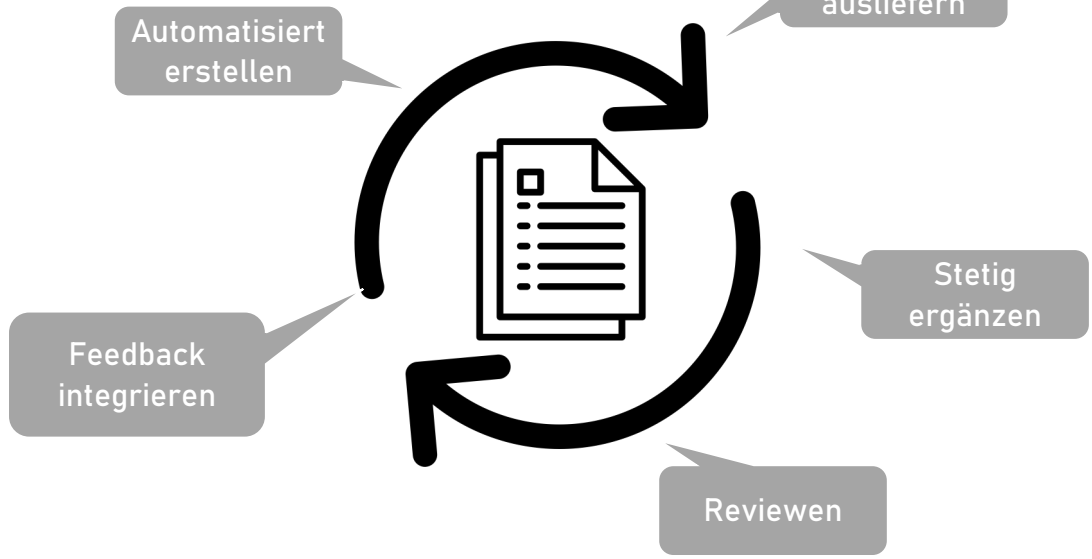
Beispiele Umsetzung in **diagrams.net**



Documentation-as-Code

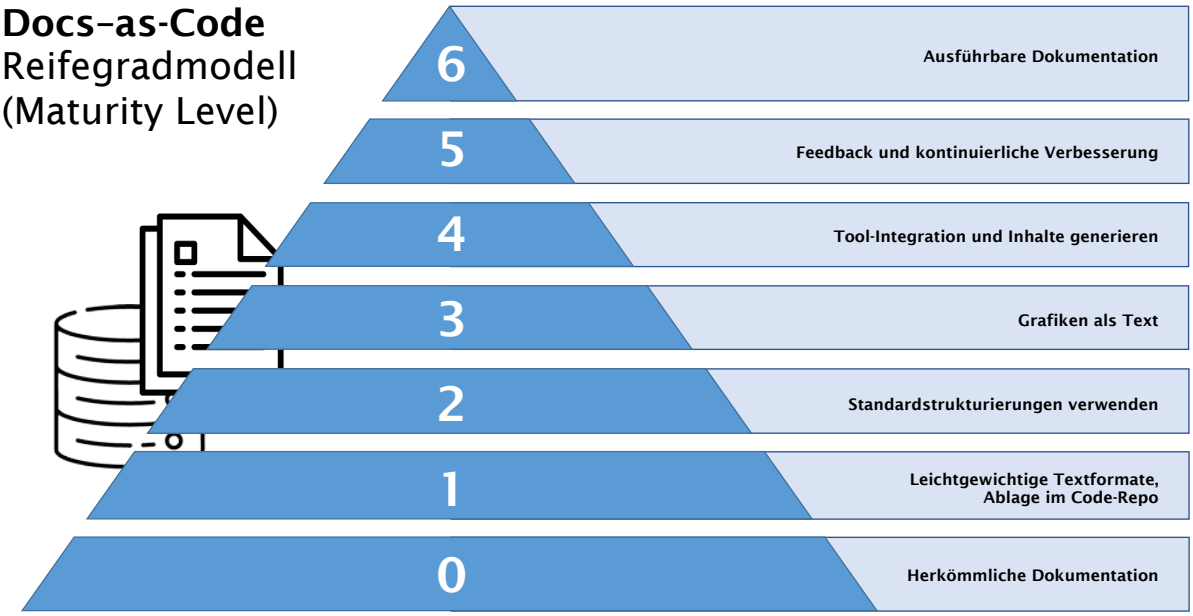


Continuous Documentation



74

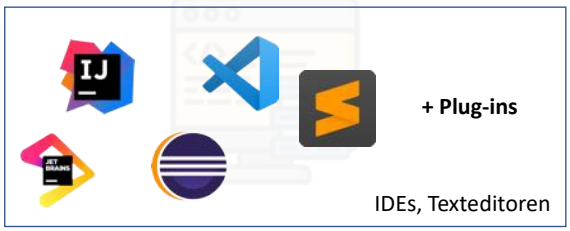
Docs-as-Code Reifegradmodell (Maturity Level)

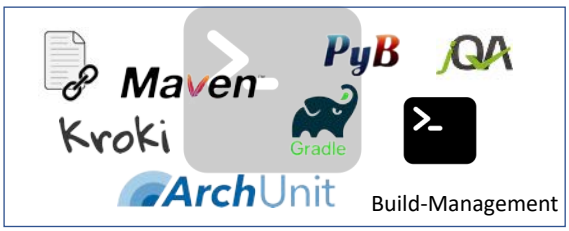


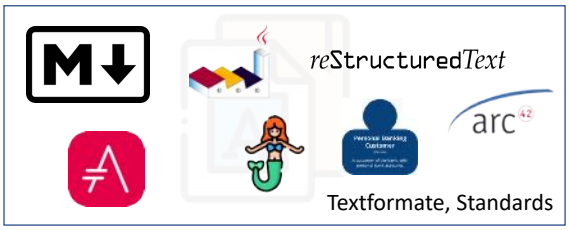
Ralf D. Müller | @ralfdmueller | DB Systel Falk Sippach | @sippsack | embarc 75

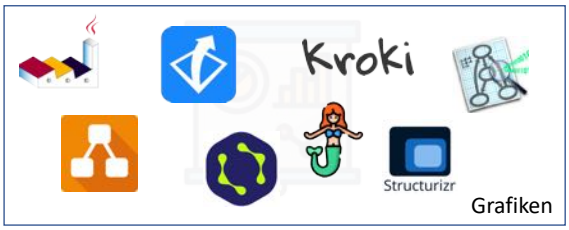
75

Werkzeuge



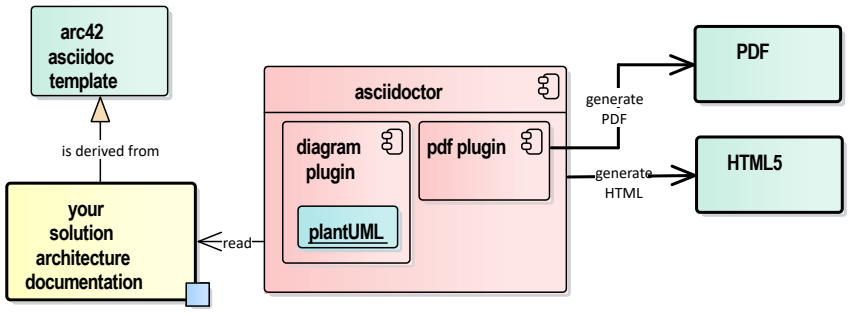






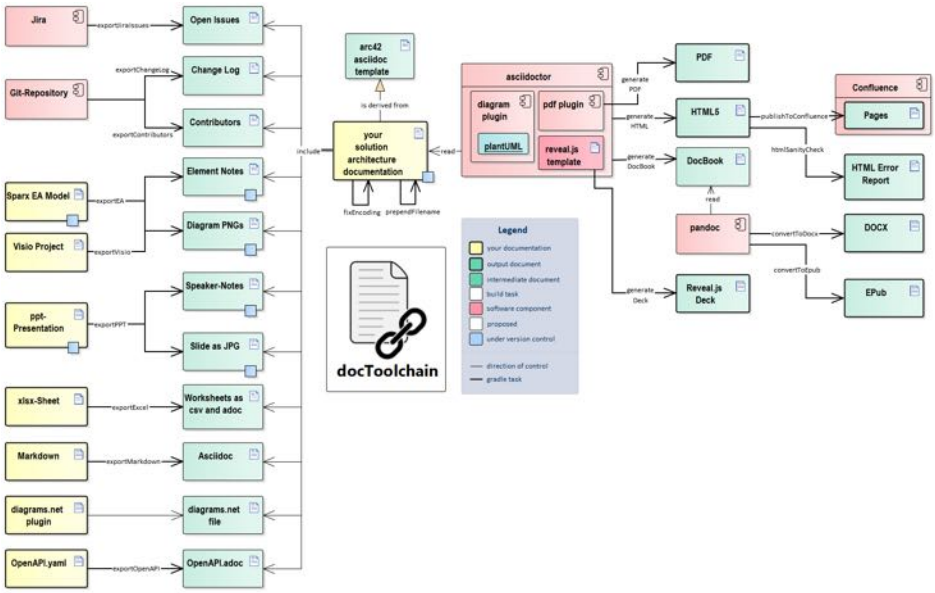
76

Basic Docs-as-Code with AsciiDoc

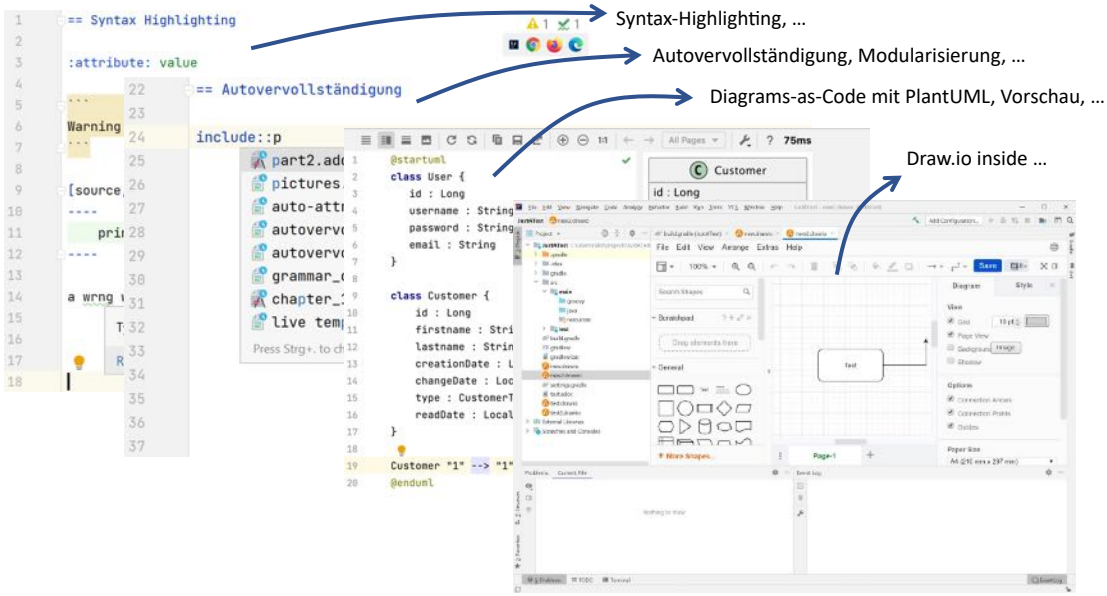


77

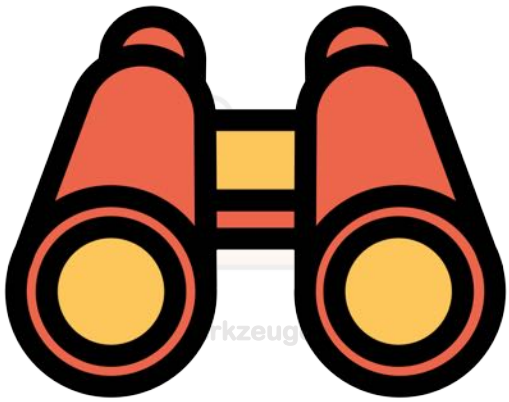
Basic Docs-as-Code with AsciiDoc & docToolchain



Docs-as-Code in der IDE – eine besondere Liebe ❤️🥰



Ausblick



Gute Diagramme

Einbettung in Dokumentation

80

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?

81

Vorteile Diagrams-as-Code (und leichtgewichtige Diagramme)

Jederzeit und von jedem editierbar.

Leicht zu erstellen.

Entwicklertools verwendbar.

Automatisches Layouting und Rendering.

Versionier- und historisierbar.

Direkte Unterstützung der Notation.

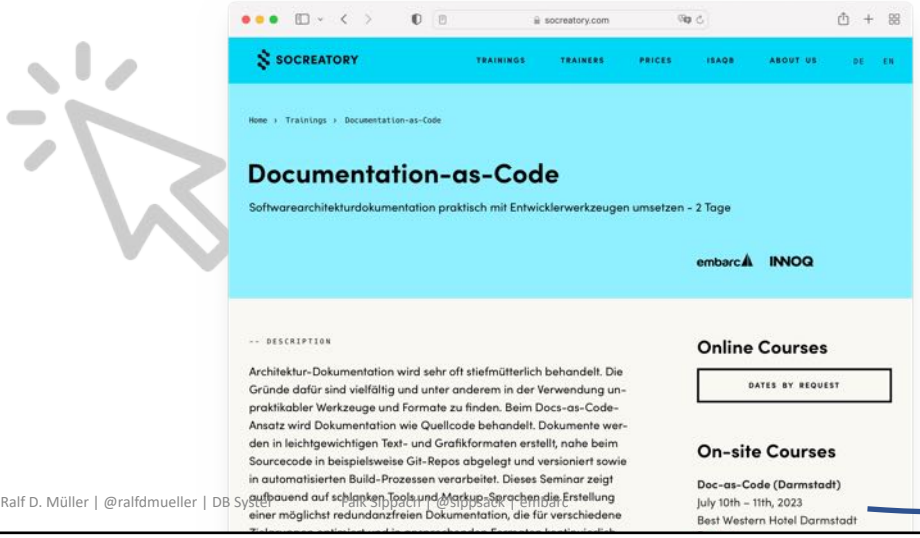
Leicht zu aktualisieren

einheitlicher Stil

Training zu Docs-as-Code

Softwarearchitekturdokumentation praktisch mit Entwicklerwerkzeugen umsetzen

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

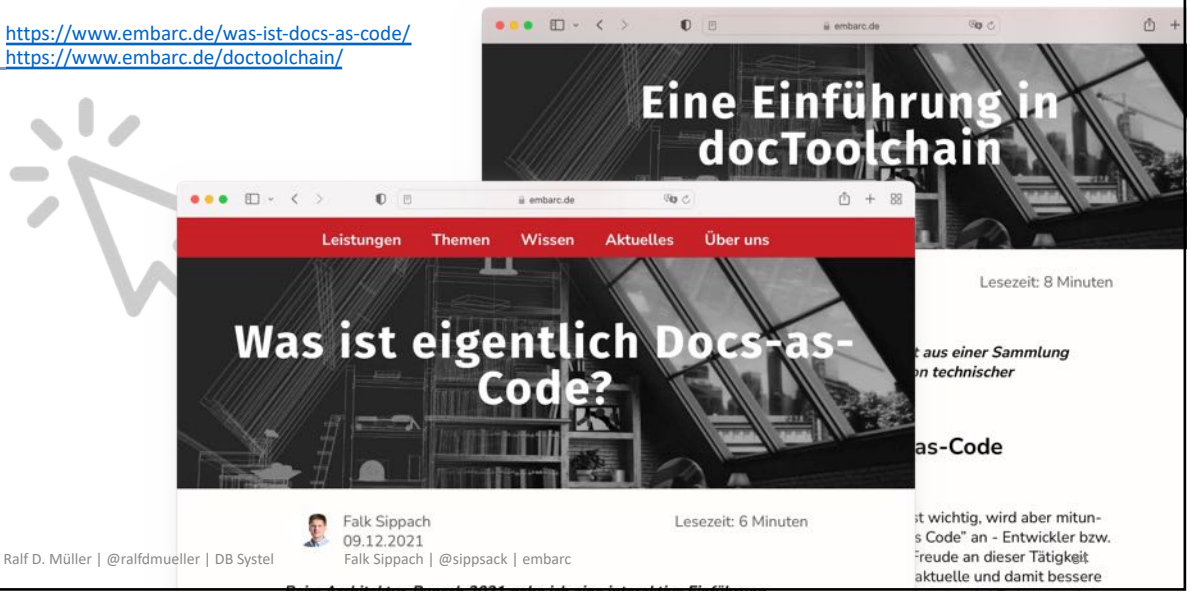


Wir kommen auch zu Euch in die Firma!

Weiterführende Blog-Posts

Was ist eigentlich Docs-as-Code? & **Eine Einführung in docToolchain**

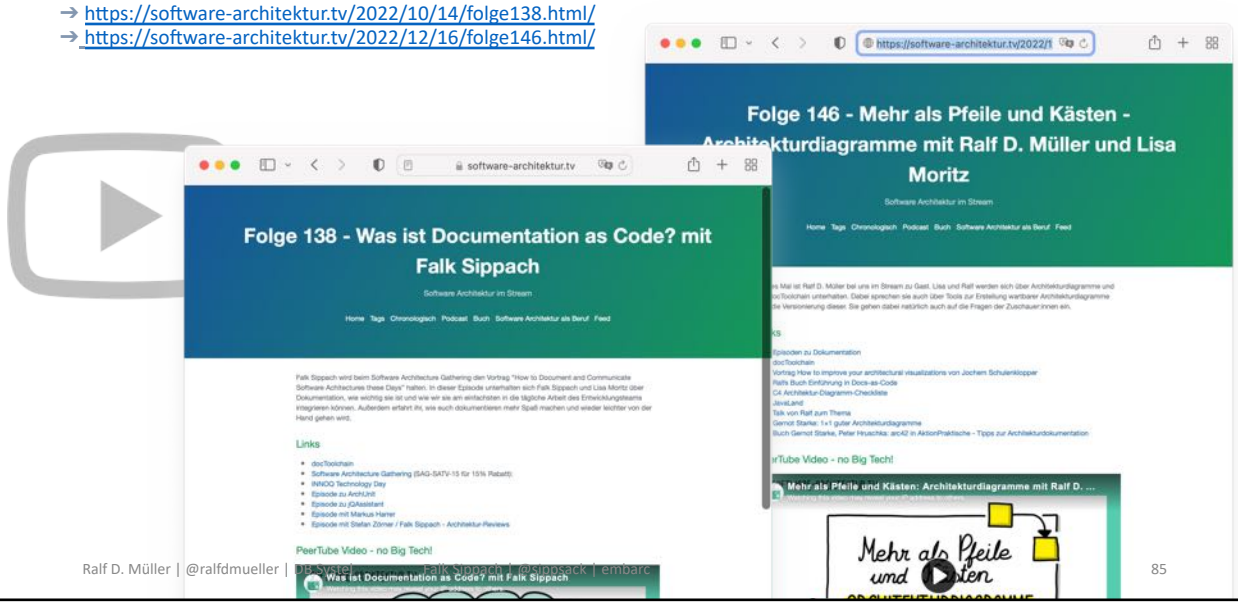
➔ https://www.embarc.de/was-ist-docs-as-code/
➔ https://www.embarc.de/doctoolchain/



84

Weiterführende Videos/Podcasts

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



85

Einführung in Docs-as-Code

https://leanpub.com/praxisbuchdocs-as-code/c/SAS-Munich

Spicken erlaubt!

PDF, 4 Seiten
Kostenloser Download.

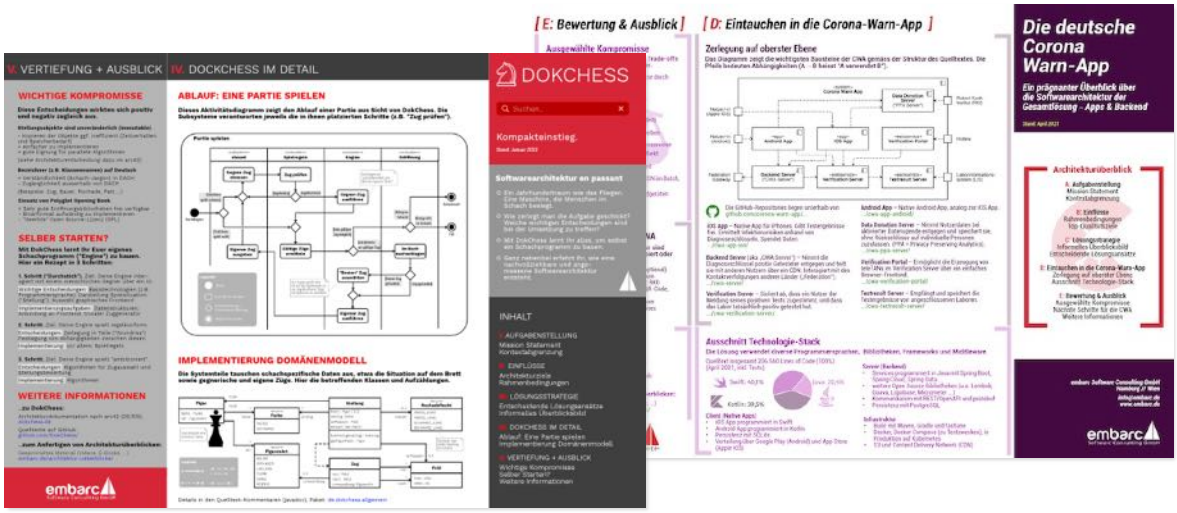
Unsere Architektur-Spicker beleuchten die konzeptionelle Seite der Softwareentwicklung.

Spicker #1:
„Der Architekturüberblick“

- Welche Zutaten gehören in einen Architekturüberblick?
- Welche Formen bewähren sich in welchen Situationen?
- Wie fertigen Sie einen Architekturüberblick an?

➔ embarc.de/architektur-spicker/

Architekturüberblicke als Vorlage



➔ embarc.de/architektur-ueberblicke/

Vielen Dank.

Wir freuen uns auf Eure Fragen!

falk.sippach@embarc.de

@sippsack

ralf.d.mueller@deutschebahn.com

@RalfDMueller

Links/Referenzen (1)

- 1×1 guter Architekturdiagramme: https://www.innoq.com/de/articles/2022/09/better-architecture-diagrams/
- Checkliste Softwarearchitekturdiagramme: https://www.feststelltaste.de/checkliste-softwarearchitekturdiagrammen/
- Praxisbuch Docs-as-Code (vorab) mit erweiterter Checkliste: https://leanpub.com/praxisbuchdocs-as-code/c/SAS-Munich
- Jochem Schulenklopper „ How to improve your architectural visualizations” https://conferences.oreilly.com/software-architecture/sa-eu-2018/public/schedule/detail/68915.html

Links/Referenzen (2)

- Architecture diagrams should be code: https://news.ycombinator.com/item?id=34322130
- render diagrams from the Structurizr DSL: https://twitter.com/simonbrown/status/1586339632828284928?t=QC7vjEZ5ZFcKviUWBbTk_w&s=03
- The Ultimate Guide To Software Architecture Documentation: https://www.workingsoftware.dev/software-architecture-documentation-the-ultimate-guide/?s=03
- Diagrams as code != automatic layout: https://twitter.com/simonbrown/status/1613824847812886528?t=2DA-1BXt0iRbqCl8OyOh1g&s=03

Links/Referenzen (3)

- UML-Diagramme mit PlantUML: https://www.youtube.com/@Randomcode_0/search?query=plantuml
- Alexander Schwartz: Technical Writing with AsciiDoc & IntelliJ https://www.udemy.com/course/technical-writing-with-asciidoc-and-intellij-idea/?couponCode=10E18DB7D7561FBBAC25
- Alexander Schwartz: Asciidoctor Deep Dive https://www.ahus1.de/post/asciidoctor-intro-and-deep-dive