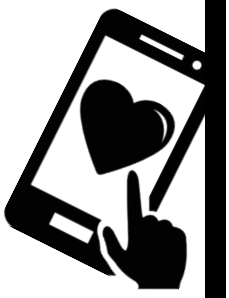
Monolith sucht Resilienz.
Softwarearchitektur Speed-Dating



STEFAN ZÖRNER, EMBARC

JUG Saxony, Leipzig

Donnerstag, 14.09.2023, zu Gast bei der Spread Group

Softwarearchitektur Speed-Dating

Zusammenfassung

Zeitgemäße Softwarearchitektur ist nicht das Werk einzelner. Architekturansätze und Ideen entstehen im Team und werden gemeinsam reflektiert. Alle an der Entwicklung beteiligten müssen sie zumindest verstehen und mittragen können. Aber was genau müsst Ihr vermitteln? Und reicht aufschreiben?

Stefan Zörner zeigt auf lebendige Weise, wie Ihr Eure Softwarearchitektur wirkungsvoll kommunizieren könnt. Nach kurzen theoretischen Inputs rund um Architekturdokumentation und -bewertung probiert Ihr das Gehörte gleich aus. Ihr lernt die Lösungen anderer Teilnehmenden kennen und erfahrt Schritt für Schritt, welche Zutaten in einem Architekturüberblick keinesfalls fehlen sollten, egal wie kurz er ist. Ihr lernt die richtigen Fragen zu stellen und passende Antworten parat zu haben.

Bringt bitte die Bereitschaft mit, Euch über Eure Produkte, Projekte und Softwarelösungen auszutauschen, und Anderen Feedback zu geben.

Stefan Zörner

- Softwareentwickler + -architekt bei embarc in Hamburg
- Vorher oose, IBM, Mummert + Partner, Bayer AG, …

Schwerpunkte:

- Softwarearchitektur (Entwurf, Bewertung, Dokumentation)
- Cloud- und Java-Technologien

Stefan.Zoerner@embarc.de
@StefanZoerner
linkedin.com/in/stefan-zoerner
xing.to/szr
@StefanZoerner@mastodon.social

embarc

Agenda

0 Einstieg
1 Runde #1
2 Runde #2
3 Runde #3
4 Runde #4
5 Weitere Informationen

Agenda

0 **Einstieg**

1 Runde #1

2 Runde #2

3 Runde #3

4 Runde #4

5 Weitere Informationen

0

Typische Situationen … (Findet Ihr Euch wieder?)

Situation 1 („Neue im Team“)

Neue Leute starten in Eurem Vorhaben und möchten mitarbeiten.

Leitfragen für die Zielgruppe (das neue Teammitglied):
Was muss ich wissen?
Wie finde ich mich hier zurecht?

Typische Situationen … (Findet Ihr Euch wieder?)

Situation 2 („Makro-Architektur“)

Eure Lösung beinhaltet Vorgaben für neue fachliche Module, Komponenten, Services, Applikationen … (z.B. Microservices, SCS).

Leitfragen für die Zielgruppe (z.B. andere Teams):
Wie bauen wir architekturkonform ein neues „Teil“?
Was sind Vorgaben, Vorschläge, Freiheiten …?

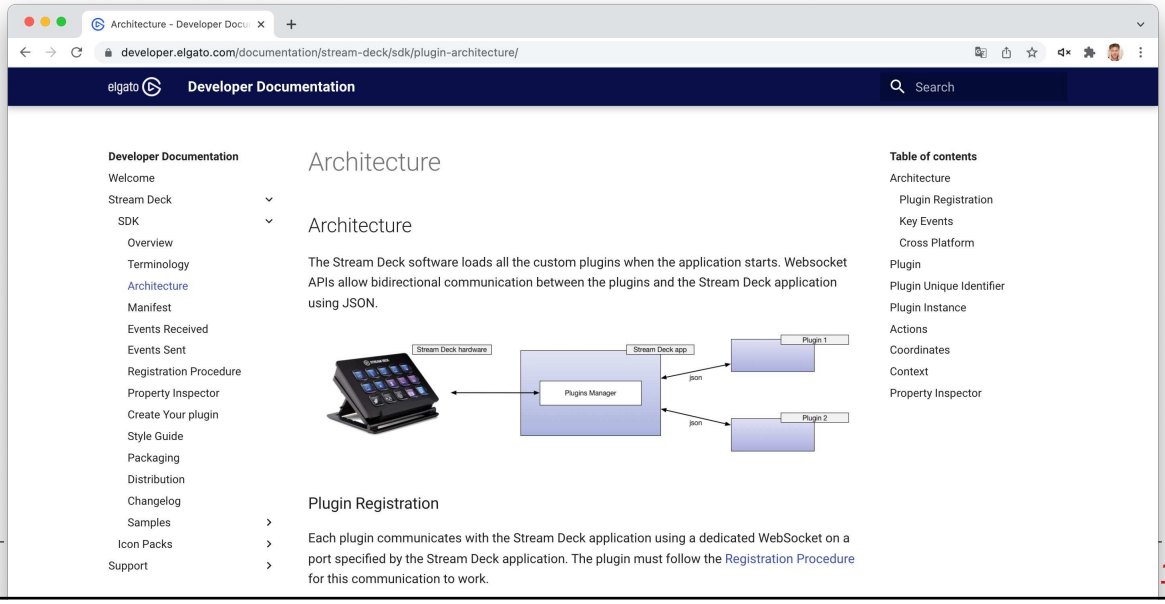
Typische Situationen … (Findet Ihr Euch wieder?)

Situation 3 („Plugin-Architektur“)

Eure Lösung besitzt geplante Erweiterungspunkte für Außenstehende.
Im Extremfall ist es ein Framework.

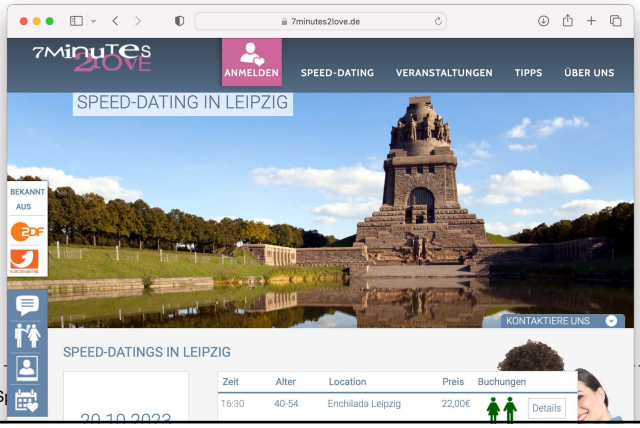
Leitfrage für die Zielgruppe (Außenstehende):
Wie funktioniert der Rahmen?
Wie baue, teste (ggf. veröffentliche) ich eine Erweiterung?

Stream Deck SDK (zur Entwicklung eigener Plugins)



Speed-Dating

> *„Unter Speed-Dating versteht man eine ursprünglich aus den USA stammende Methode, schnell neue Flirt- oder Beziehungspartner, aber auch Geschäftskontakte zu finden.“*
>
> (Wikipedia)



Speed-Dating (Original)

- Jeder männliche Single lernt jeden weiblichen Single kennen und umgekehrt.
- Veranstaltung aufgeteilt in Runden, Dauer jeweils circa sieben bis acht Minuten
- In dieser eng bemessenen Zeit haben die Singles die Gelegenheit, sich gegenseitig ein wenig kennenzulernen.

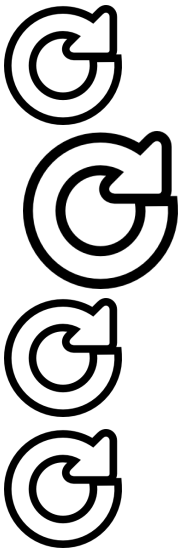
Speed-Dating (Original)

- Nach Ablauf der Zeit ertönt ein Gong als Zeichen, das zum Partnerwechsel auffordert.
- Gleichzeitig notieren die Singles auf ihnen vorher ausgehändigten Zetteln, ob sie ihr Gegenüber gerne wiedersehen wollen oder nicht.

Speed-Dating (Softwarearchitektur)

- Veranstaltung aufgeteilt in Runden, Dauer jeweils circa sieben bis acht Minuten
- Jede/r lernt nützliche Zutaten für eine Architekturdokumentation und spannende Softwarevorhaben etwas genauer kennen.

Speed-Dating (Softwarearchitektur)



Ablauf einer Runde:

- Theoretischer Input zu einem Werkzeug
  - (kurz, alle gemeinsam)
- Vorbereitungszeit
  - (jeder für sich)
- Gegenseitiges Vorstellen/Kennenlernen
  - (paarweise, im Wechsel)

Agenda

0 Einstieg

1 **Runde #1: Kontextabgrenzung**

2 Runde #2

3 Runde #3

4 Runde #4

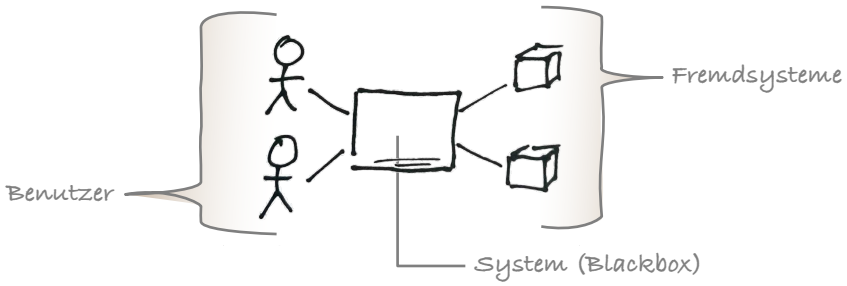
5 Weitere Informationen

1

Kontextabgrenzung

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

Form: Graphik, ergänzt um kurze Beschreibungen.



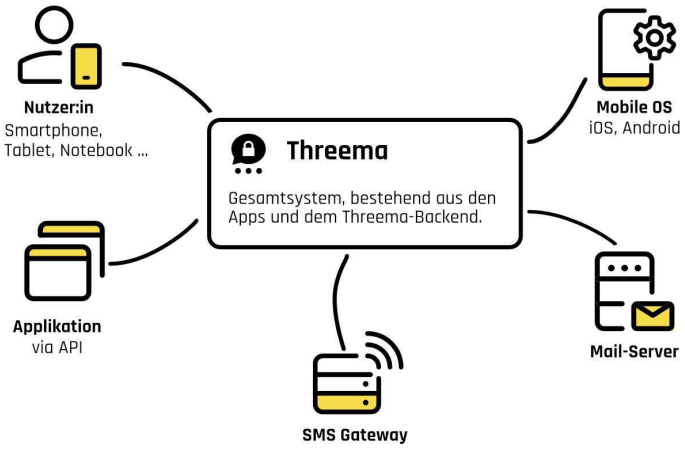
Beispiel: Threema

- Threema ist ein **mobiler Instant-Messenger**, bei dem der Schutz der Privatsphäre ganz oben auf der Agenda steht.
- Die Software erlaubt seinen Nutzer:innen den sicheren, zuverlässigen und bequemen **Austausch von Nachrichten** und Medien **ohne die Angabe persönlicher Informationen** wie Rufnummer oder E-Mail-Adresse.
- Die **Identität eines Kommunikationspartners** lässt sich – bei vorherigem Kontakt – auch ohne Verwendung des Adressbuchs zweifelsfrei feststellen.
- Auch für **Unternehmen** bietet Threema eine attraktive und zugleich gesetzeskonforme Lösung zum Schutz der **Kommunikation** innerhalb der Organisation und mit Partnern.

Threema.

Kontextabgrenzung Threema

Mit wem und was interagiert Threema? Die Lösung im Zusammenspiel mit wichtigen anderen Akteuren.



Speed-Dating, Runde #1

Zu zweit oder (bei Bedarf) dritt.

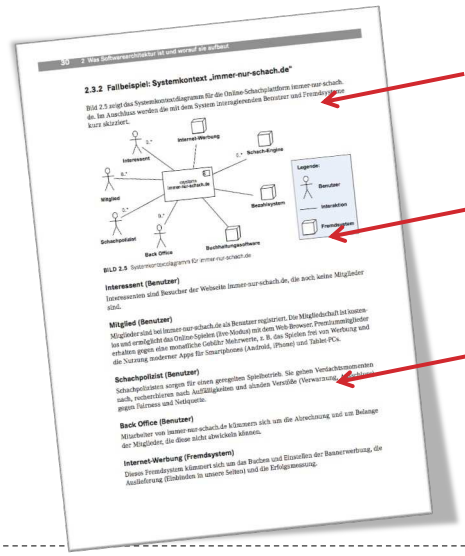
Zeit: 7 Minuten

Take Away …

Ein guter Einstieg in ein System grenzt es zunächst von der Umgebung ab.

- Mit wem interagieren wir, und warum?

Ein Bild ist nicht genug …



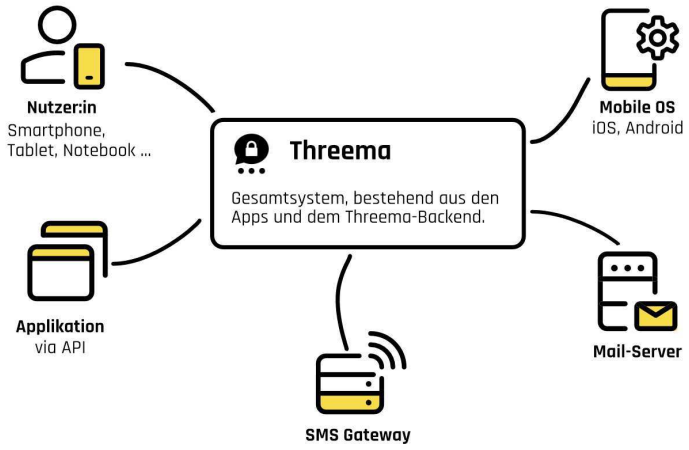
Einleitender Text

Diagramm mit Legende

Erläuterungen zu den Akteuren (z.B. tabellarisch)

Kontextabgrenzung Threema

Mit wem und was interagiert Threema? Die Lösung im Zusammenspiel mit wichtigen anderen Akteuren.



Kontextabgrenzung, Akteure bei Threema

Kurze Erläuterungen zu den Benutzern und Fremdsystemen:

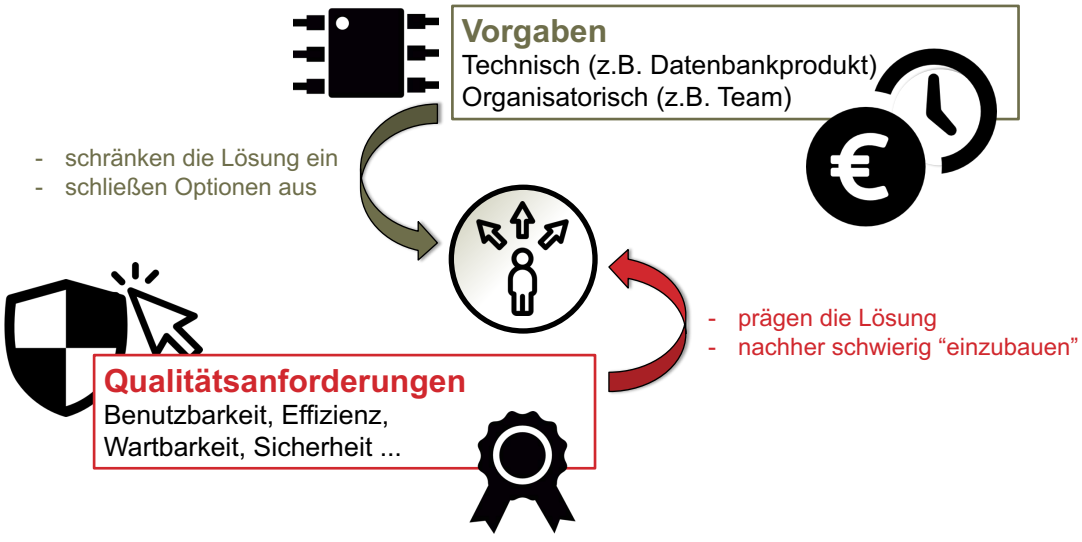
Akteur	Beschreibung
Nutzer:in	Teilt Textnachrichten und andere Inhalte (z.B. Fotos) mit Kontakten. Direkt oder innerhalb von Gruppen.
Mobiles OS (Operating System)	Stellt Sicherheitstechnologien und Funktionalität bereit. Zum Beispiel zur Interaktion mit Medien und anderen Apps oder für Benachrichtigungen.
Applikation	Neben App-Nutzer:innen können auch durch dritte geschriebene Anwendungen Nachrichten senden und empfangen. Über eine API und Threemas Gateway.
SMS Gateway	Threema versendet Codes per SMS für die Verifikation von mobilen Rufnummern, falls Nutzer:innen diese mit ihrer Threema-ID verknüpfen möchten.
Mail-Server	Ermöglicht die Verifikation einer E-Mail-Adresse zur Verknüpfung mit einer Threema-ID (ebenfalls optional).

Agenda

2

0 Einstieg
1 Runde #1: Kontextabgrenzung
2 **Runde #2: Wichtige Einflüsse**
3 Runde #3
4 Runde #4
5 Weitere Informationen

Einflüsse auf Entscheidungen



Qualitätsmerkmale

Begriffe nach ISO 25010

Benutzbarkeit (Usability)

Ist die Software intuitiv zu bedienen, leicht zu erlernen, attraktiv?

Portabilität (Portability)

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

Funktionale Eignung (Functional Suitability)

Sind die berechneten Ergebnisse genau genug / exakt, ist die Funktionalität angemessen? ...

Effizienz (Performance)

Antwortet die Software schnell, hat sie einen hohen Durchsatz, einen geringen Ressourcenverbrauch? ...

Kompatibilität (Compatibility)

Ist die Software konform zu Standards, arbeitet sie gut mit anderen zusammen?

Zuverlässigkeit (Reliability)

Ist das System verfügbar, tolerant gegenüber Fehlern, nach Abstürzen schnell wieder hergestellt? ...

Sicherheit (Security)

Ist das System sicher vor Angriffen? Sind Daten und Funktion vor unberechtigtem Zugriff geschützt? ...

Wartbarkeit (Maintainability)

Ist die Software leicht zu ändern, erweitern, testen, verstehen? Lassen sich Teile wiederverwenden? ...

Eingabe eines Kennwortes

Username

StefanZ

Password

••••••••••

Username

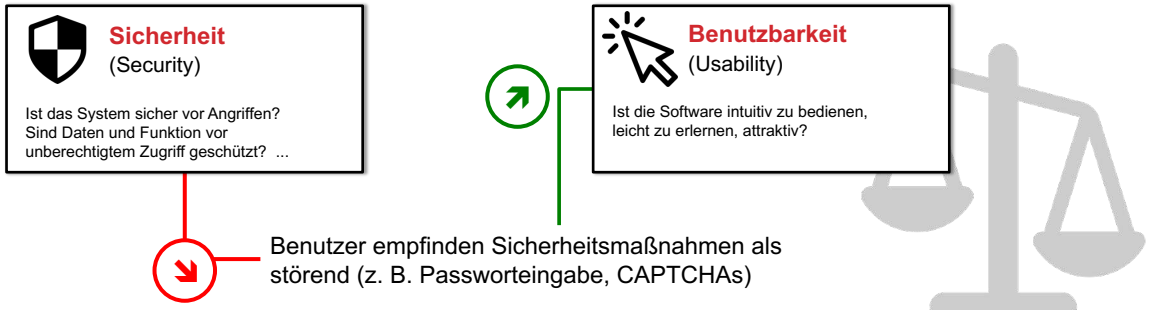
StefanZ

Password

geheim!123

Typische Wechselwirkung

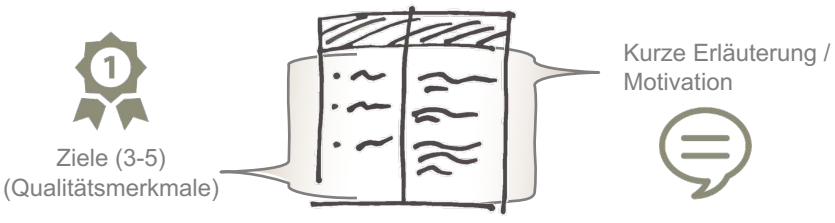
Qualitätsmerkmale hängen voneinander ab.
Entscheidungen führen zu Kompromissen (engl. „Trade-off“).



Architekturziele

Die wichtigsten (Top 3-5) an das System gestellten Qualitätsanforderungen („-ilities“), inkl. Motivation

Form: Stichpunkte oder (besser) Tabelle.



Beispiel: Architekturziele Confluence

(aka „Architekturtreiber“ oder „Qualitätsziele“)

Ziel	Beschreibung
Leicht zu betreiben	Für ein Team ist es einfach mit Confluence zu starten.
Gute Benutzbarkeit	Confluence ist effizient und intuitiv von allen Team-Mitgliedern zu verwenden.
Hohe Zuverlässigkeit	Das System steht den Anwendern jederzeit zur Verfügung.
Sicherheit der Inhalte	Inhalte sind vor unberechtigtem Zugriff und Veränderung geschützt.
Gute Wartbarkeit	Confluence ist leicht zu ändern und um Funktionalität zu erweitern.

Die Reihenfolge gibt eine Orientierung bezüglich der Wichtigkeit.

CONFLUENCE

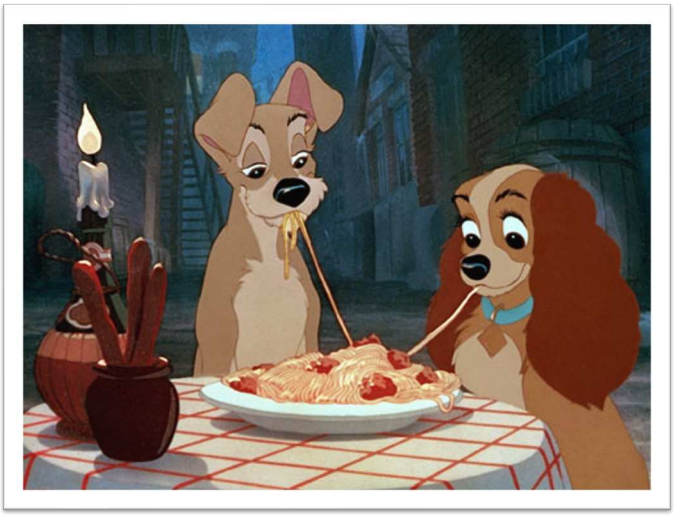
Wichtige Architekturziele des Spring Frameworks

- Hohe Entwicklerproduktivität
- Unternehmenskritische Anwendungen ermöglichen
- Auf lange Sicht wartbare Lösungen
- Portable und kompatible Arbeitsergebnisse
- Zukunftsfähiges Framework

Wichtige Architekturziele des Spring Frameworks

Ziel	Beschreibung (und zugehöriges Software-Qualitätsmerkmal)
Hohe Entwickler-produktivität	Anwendungen auf Basis von Spring lassen sich effizient entwickeln. Java-Entwickler kommen gut mit dem Framework zurecht. Die dahinterliegenden Konzepte sind schnell verstanden und kein Hexenwerk. Entwickler konzentrieren sich auf die Fachlogik. *(Wartbarkeit, Benutzbarkeit (aus Entwicklersicht))*
Unternehmens-kritische Anwendungen ermöglichen	Spring erlaubt die Entwicklung von zuverlässigen und sicheren Applikationen, wie sie Unternehmen und Organisationen zur Unterstützung wichtiger Geschäftsfähigkeiten erwarten. *(Zuverlässigkeit, Sicherheit)*
Auf lange Sicht wartbare Lösungen	Mit Spring realisierte Anwendungen sind leicht erweiterbar und änderbar. Genutzte Technologien (z.B. Bibliotheken, Middleware, Persistenz) lassen sich einfach aktualisieren und zu einem gewissen Grad auch austauschen. Das gilt insbesondere für Spring selbst. *(Erweiterbarkeit. Änderbarkeit)*
Portable und kompatible Arbeitsergebnisse	Die Anwendungen sind auf allem relevanten Zielplattformen lauffähig (Applikationsserver wie z.B. WebSphere, Web-App-Server wie Tomcat, Standalone als Java-Prozess ...). Neue Java- oder Spring-Versionen brechen eine vorhandene Anwendung nicht. *(Kompatibilität, Portierbarkeit)*
Zukunftsfähiges Framework	Spring kann dem technologischen Fortschritt folgen. Neue Produkte, Entwicklungsmethoden und Zielumgebungen sind rasch adaptiert. Die Entscheidung für Spring ist keine Sackgasse. *(Wartbarkeit (des Frameworks selbst))*

Speed-Dating, Runde #2



Zu zweit oder (bei Bedarf) dritt.

Zeit: 7 Minuten

Take Away …

Der wichtigste Einflussfaktor beim Treffen von Architekturentscheidungen ist die geforderte Qualität.

- Es ist gleichzeitig der Schlüssel jeder Architekturbewertung.

Agenda

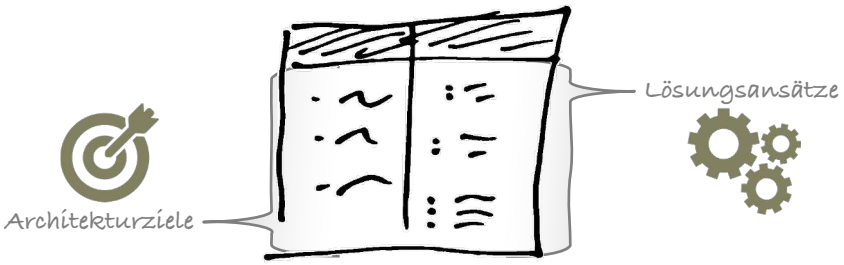
0 Einstieg
1 Runde #1: Kontextabgrenzung
2 Runde #2: Wichtige Einflüsse
3 **Runde #3: Lösungsstrategie**
4 Runde #4
5 Weitere Informationen

3

Lösungsstrategie (Tabelle)

Architekturziele und zugeordnete high-level Lösungsansätze.

Form: Tabelle ([Ziele | Lösungsansätze]).



Zentrale Lösungsansätze von Spring

- POJOs (JavaBeans) als "Spring-Komponenten" (keine Abhängigkeit zum Framework, nicht invasiv)
- Dependency Injection zur Entkopplung, Lightweight Container
- Klare Trennung von fachlichem und technischem Code möglich.
- Aspektorientierte Programmierung (AOP) für Transaktionen, Security ... "entrümpelt" Fachlogik
- Betonung von Java und "klassischem" OO und Vermeidung server-spezifischen Komponenten-modellen wie EJB
- Einsatz bekannter Entwurfsmuster, inkl. entsprechender Benennung (z.B. Prototype, Adapter ...)
- Durchgängige Konzepte wie Templates, Repositories ...
- Modularer Aufbau des Frameworks
- Konfiguration / Meta-Daten externalisiert (XML, u.a., später u.a. auch mit Annotationen)
- Entkopplung des Frameworks von spezifischen Java- und Server-Versionen
- Gute Unterstützung für Unit- und Integrationstests
- Nutzung und Ergänzung verbreiteter (Java-) Standards wie Servlets, JDBC, JTA, ...
- Abstraktion von Technologien, z.B. über Templates, vereinfachte APIs
- Integration etablierter Open-Source-Bibliotheken und Frameworks

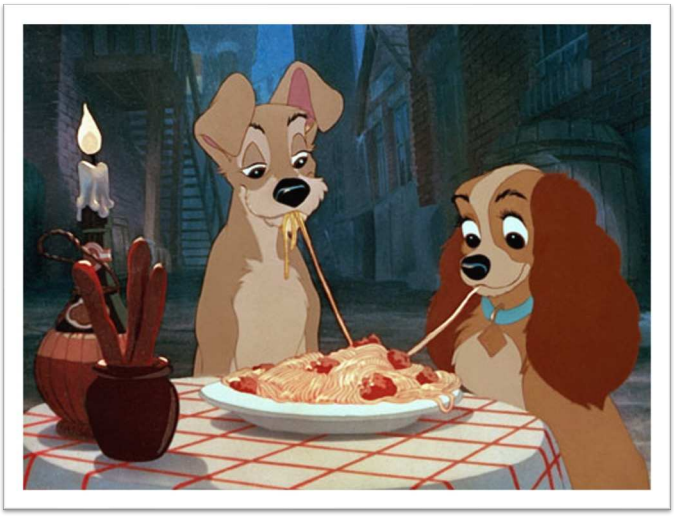
Zuordnung zu Zielen (Farbcode)

- Hohe Entwicklerproduktivität
- Unternehmenskritische Anwendungen ermöglichen
- Auf lange Sicht wartbare Lösungen
- Portable und kompatible Arbeitsergebnisse
- Zukunftsfähiges Framework

Lösungsstrategie Confluence

Architekturziel	passende Lösungsansätze in Confluence
Leicht zu betreiben	• üblicher **Web-Browser** als Client genügt • unterstützt diverse Java (Web-)Applikationsserver, u.a. **Apache Tomcat** • **Deployment-Monolith** (ein *.war-File) oder Standalone-Installation • unterstützt gängige Datenbanken
Gute Benutzbarkeit	• Intuitives, **modernes** Web UI • Rich Text Editor und einfaches Wiki Markup • Leistungsfähige, **effiziente Suche** mit Apache Lucene
Hohe Zuverlässigkeit	• Java (Web-)Applikationsserver, darüber Monitoring, **Cluster**-Unterstützung • RDBMS für Persistenz • Job Scheduling mit **Quartz**
Gute Wartbarkeit	• **Modulare** Implementierung in Java, Dependency Injection mit **Spring** IoC • Durchgängige Konzepte (z.B. Manager, Permissions) • **Plugins** für eigene Erweiterungen • Einsatz gängiger Java **Open Source-Lösungen**
Sicherheit der Inhalte	• Seraph Security Framework , eigenes **Berechtigungskonzept** • Speicherung der Inhalte in RDBMS • Authentifizierung über Anbindung an Verzeichnisdienste (**LDAP**) möglich

Speed-Dating, Runde #3



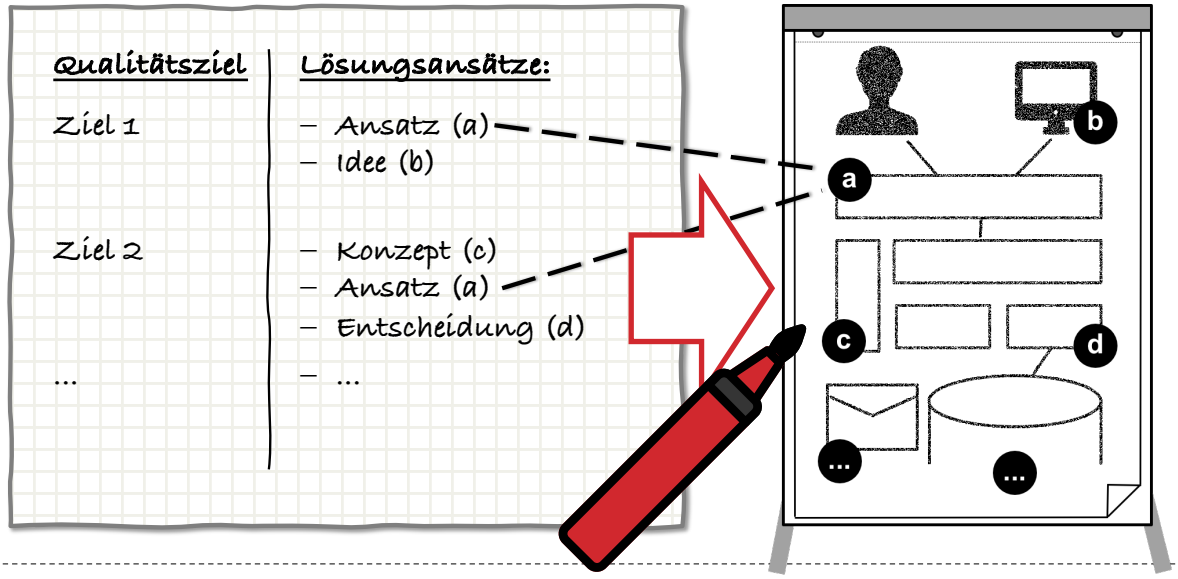
Zu zweit oder
(bei Bedarf) dritt.

Zeit: 7 Minuten

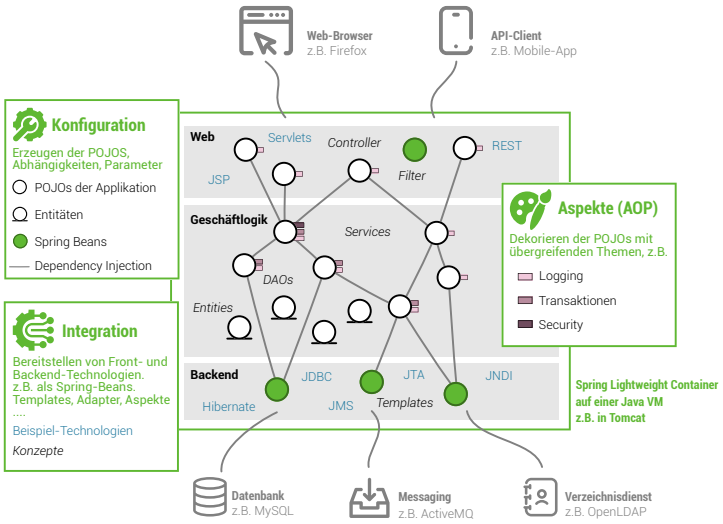
Take Away …

Die Lösungsstrategie schlägt die Brücke zwischen architektur-relevanten Anforderungen und Eurer Lösung.

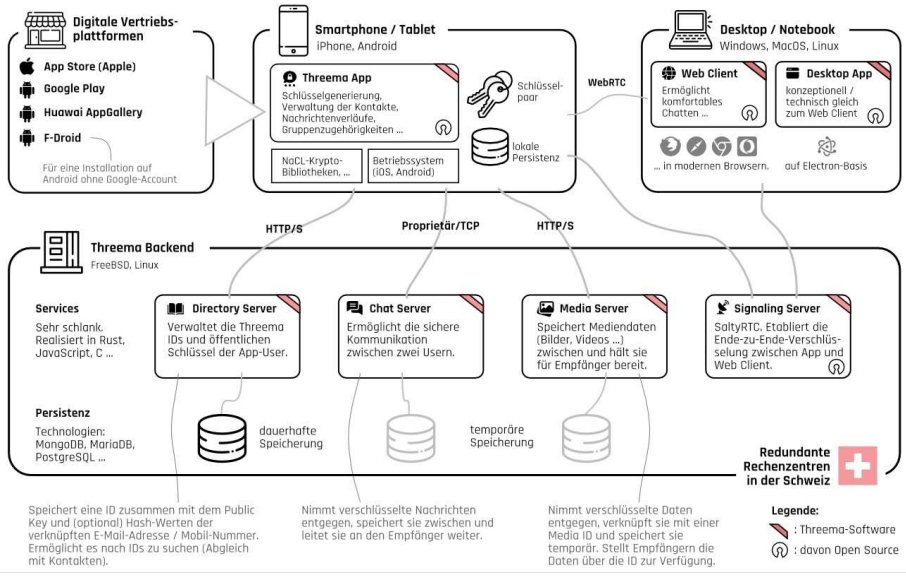
Überblicksbild aus Lösungsstrategie ableiten



Eine Web-Anwendung im Rahmen von Spring



Ein informeller Überblick über Threema



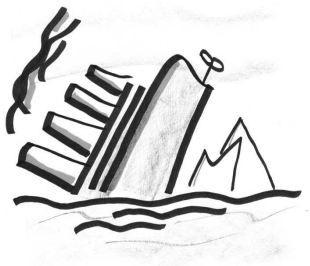
Agenda

0 Einstieg
1 Runde #1: Kontextabgrenzung
2 Runde #2: Wichtige Einflüsse
3 Runde #3: Lösungsstrategie
4 **Runde #4: Eine konkrete Entscheidung**
5 Weitere Informationen

4

Softwarearchitektur

eine (!) Definition



*"Softwarearchitektur ist die Menge der Entwurfsentscheidungen, die, wenn falsch getroffen, Dein Projekt zum Scheitern bringen kann."**

(Eoin Woods)

* Im Englischen eigentlich: "Software architecture is the set of design decisions which, if made incorrectly, may cause your project to be cancelled."

Themen für Entscheidungen

Zerlegung

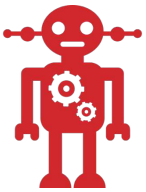
Welcher Architekturstil?
Wie zerfällt die Anwendung?
Teilsysteme, Module,
Komponentenbildung,
Abhängigkeiten ...

Zielumgebung

Wo läuft die Software?
Beim Endanwender, im
eigenen Rechenzentrum,
Cloud, Verteilung,
Virtualisierung ...

Technologie-Stack

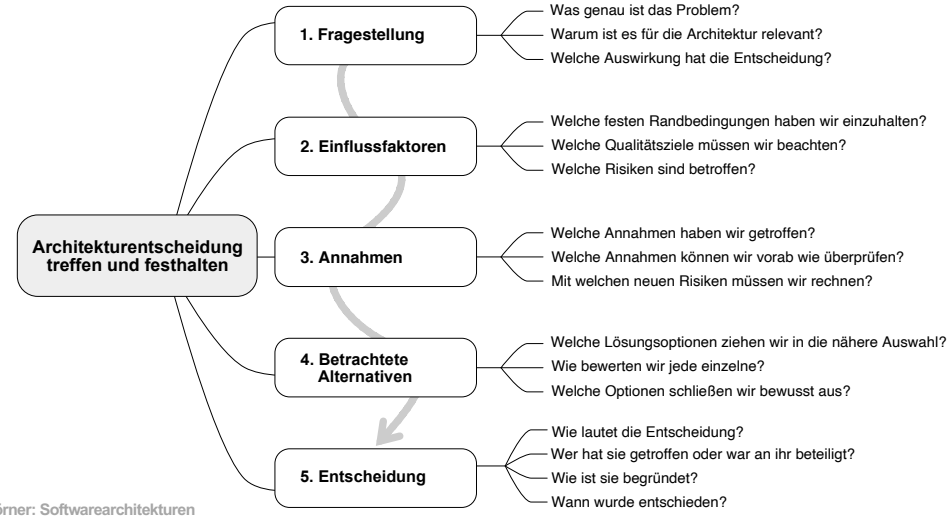
Was setzen wir ein?
Programmiersprache(n)
Bibliotheken, Frameworks,
Middleware,
Querschnittsthemen ....



Vorgehen

Wie arbeiten wir?
Planen, Entwickeln, Testen,
Bauen, Dokumentieren,
Ausliefern, Nachjustieren,
...

Entscheidungen treffen. Ein Werkzeug

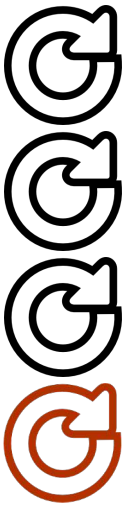


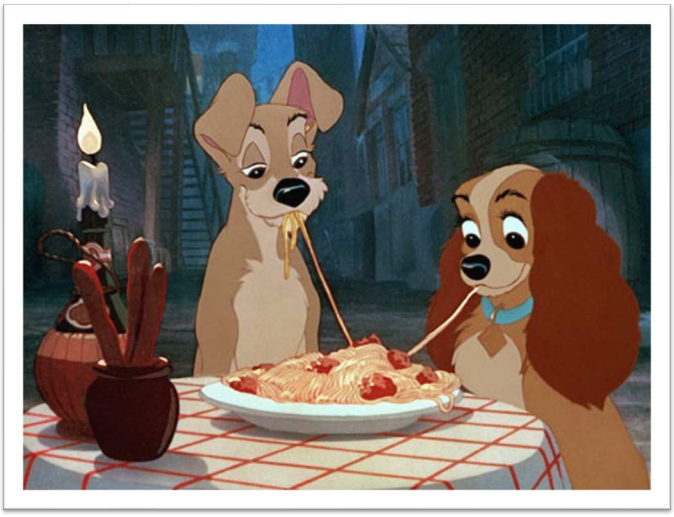
Quelle: Stefan Zörner: Softwarearchitekturen dokumentieren und kommunizieren

Der Leitfaden in verdichtet …

- Was genau war das **Problem?** (idealerweise formuliert als Frage)
- Welche **Optionen** haben wir betrachtet (oder wären denkbar)?
- Wie haben wir **entschieden** (oder was ist aktuell unsere **Präferenz?**)
- Was waren **Einflüsse** auf die Entscheidung?
- Sind wir **Kompromisse** sind wir mit der Entscheidung eingegangen?

Speed-Dating, Runde #4





Zu zweit oder (bei Bedarf) dritt.

Zeit: 7 Minuten

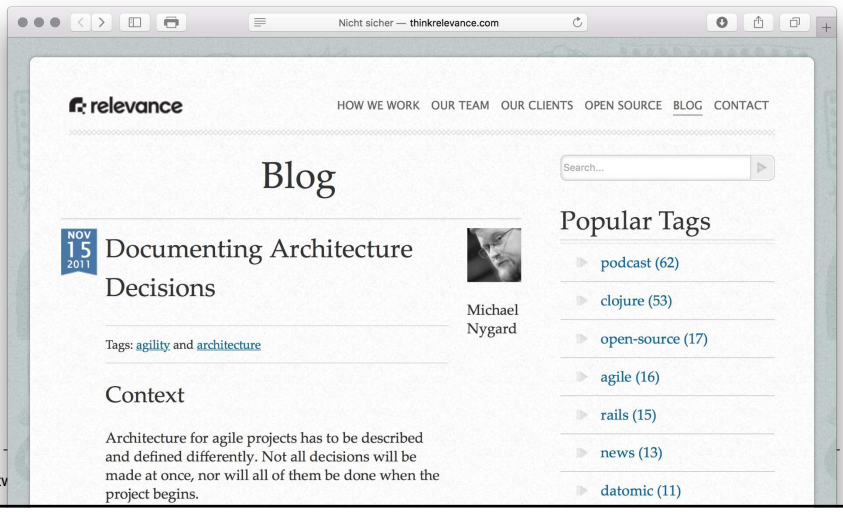
Take Away …

Zentrale Entscheidungen nachvollziehbar festhalten ist der Schlüssel, um bessere Antworten zu haben als „Historisch gewachsen“.

Anderer Vorschlag: ADRs

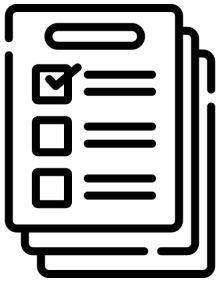
Architecture Decision Records (Michael Nygard, 2011)

➔ http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions



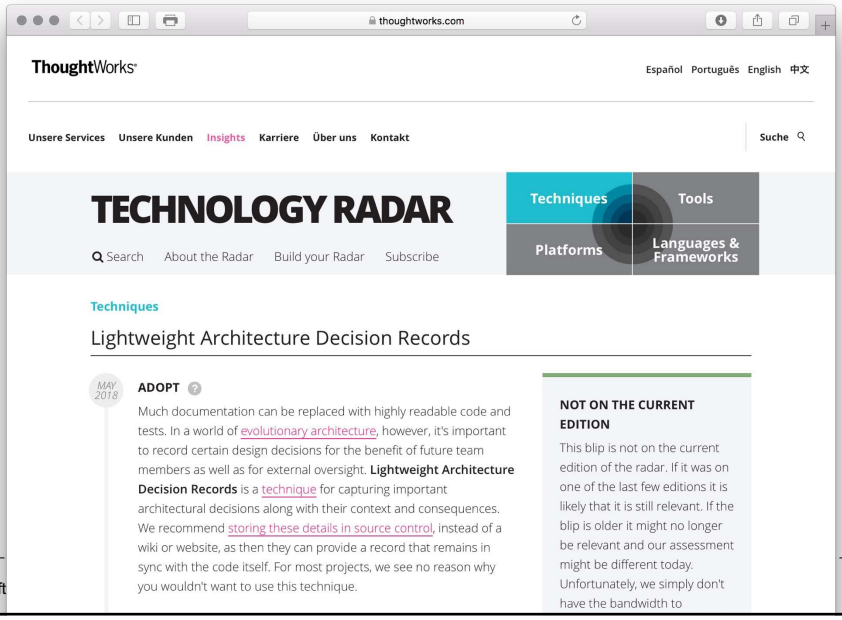
Abschnitte eines ADRs

Architecture Decision Record



- **Titel** (title): kurze, den Inhalt des ADR wiedergebende Beschreibung der Entscheidung.
- **Kontext** (context): Beschreibung der Bedingungen, die zu der Entscheidung geführt haben.
- **Entscheidung** (decision): Beschreibung der Entscheidung, die im Rahmen des vorher beschriebenen Kontexts erfolgt ist.
- **Status** (status): Position im Lebenszyklus der Entscheidung wie "proposed", "accepted", "discarded", "deprecated" oder "superseeded".
- **Konsequenzen** (consequences): Auflistung aller positiven und negativen Konsequenzen.

Mehr Informationen zu ADRs



Abheften in arc42

1. Einführung und Ziele
1.1 Aufgabenstellung
1.2 Qualitätsziele
1.3 Stakeholder

2. Randbedingungen
2.1 Technische Randbedingungen
2.2 Organisatorische Randbedingungen
2.3 Konventionen

3. Kontextabgrenzung
3.1 Fachlicher Kontext
3.2 Technischer- oder Verteilungskontext

4. Lösungsstrategie

5. Bausteinsicht
5.1 Ebene 1
5.2 Ebene 2
...

6. Laufzeitsicht
6.1 Laufzeitszenario 1
6.2 Laufzeitszenario 2
...

7. Verteilungssicht
7.1 Infrastruktur Ebene 1
7.2 Infrastruktur Ebene 2
...

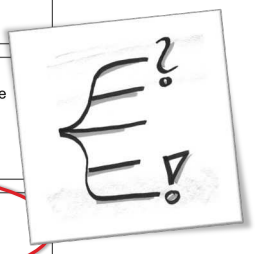
8. Konzepte
8.1 Fachliche Strukturen und Modelle
8.2 Typische Muster und Strukturen
8.3 Persistenz
8.4 Benutzungsoberfläche
...

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

10. Qualitätsszenarien
10.1 Qualitätsbaum
10.2 Bewertungsszenarien

11. Risiken

12. Glossar



arc42

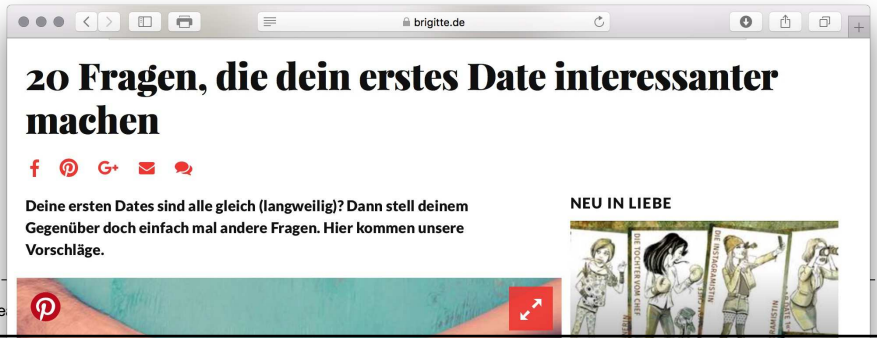
Agenda

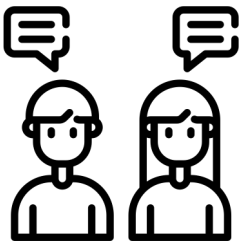
5

0 Einstieg
1 Runde #1: Kontextabgrenzung
2 Runde #2: Wichtige Einflüsse
3 Runde #3: Lösungsstrategie
4 Runde #4: Eine konkrete Entscheidung
5 **Weitere Informationen**

Date-Infos im Internet

- „Warum quatschen Männer bei ersten Dates so viel?“
- „Diese Dinge schrecken Männer beim Dating ab“
- „Männer aufgepasst: Darauf achten Frauen beim ersten Date“
- „3 Warnzeichen beim ersten Date, die verraten, ob die Beziehung hält“
- ...





Hast du jemals mit Drogen experimentiert?

Wenn du dir eine Superkraft aussuchen könntest, welche wäre das?

Sollte ich etwas über dich wissen, dass ich niemals fragen würde?

Was ist dein Lieblingseis? Und warum?

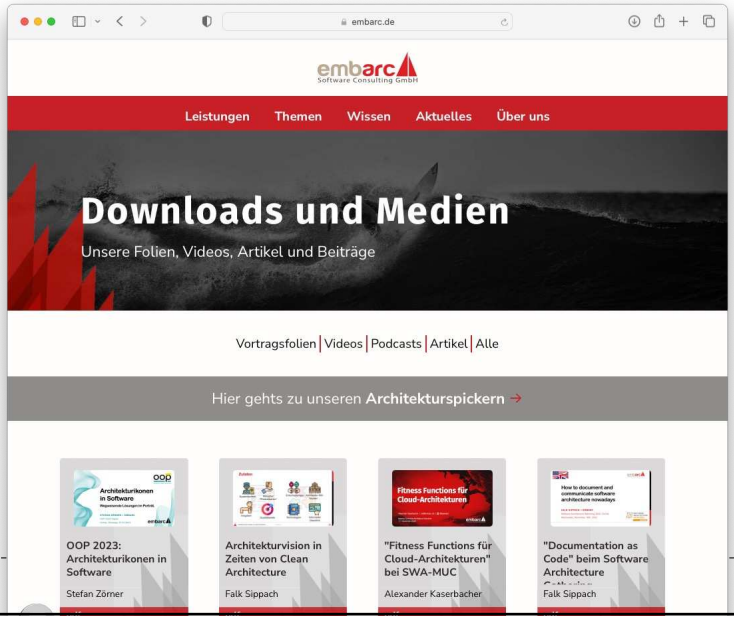
Was baut Ihr eigentlich? Wem nützt es?

Was sind die wesentlichen Features des Systems?

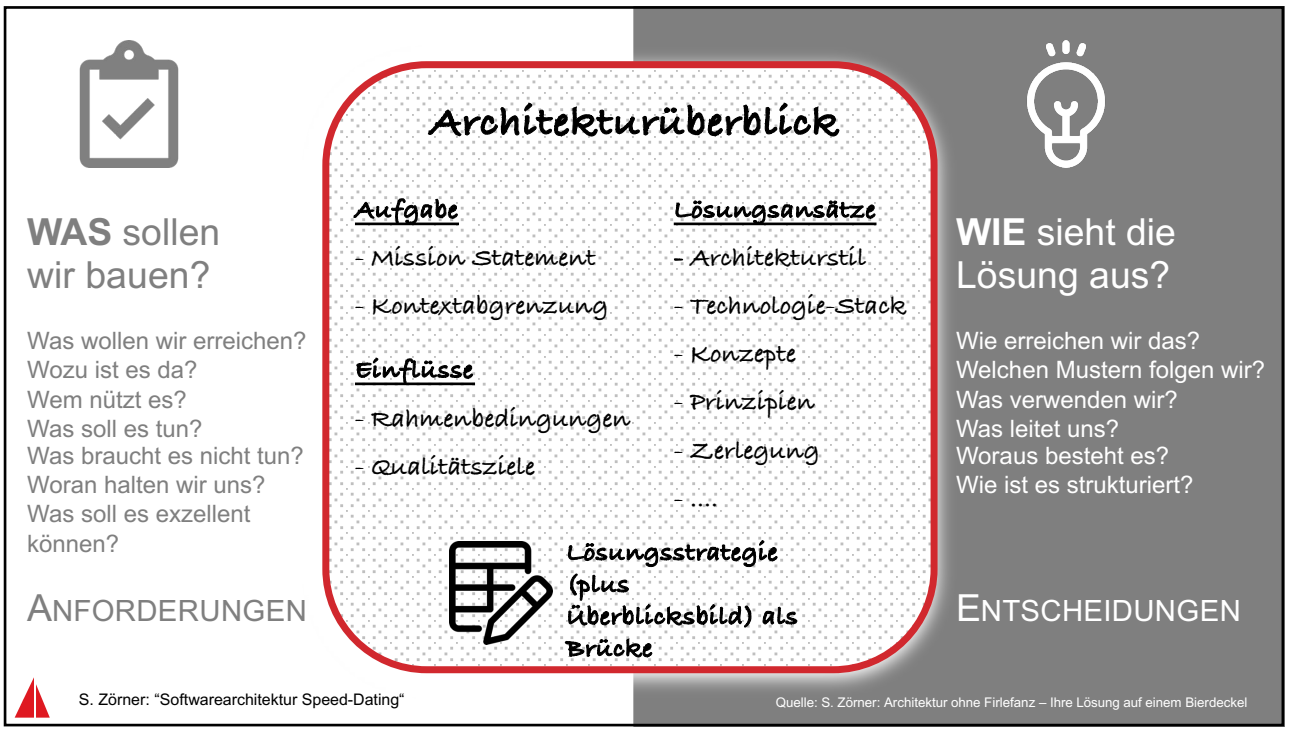
Sollte ich etwas über das System wissen, dass ich niemals fragen würde?

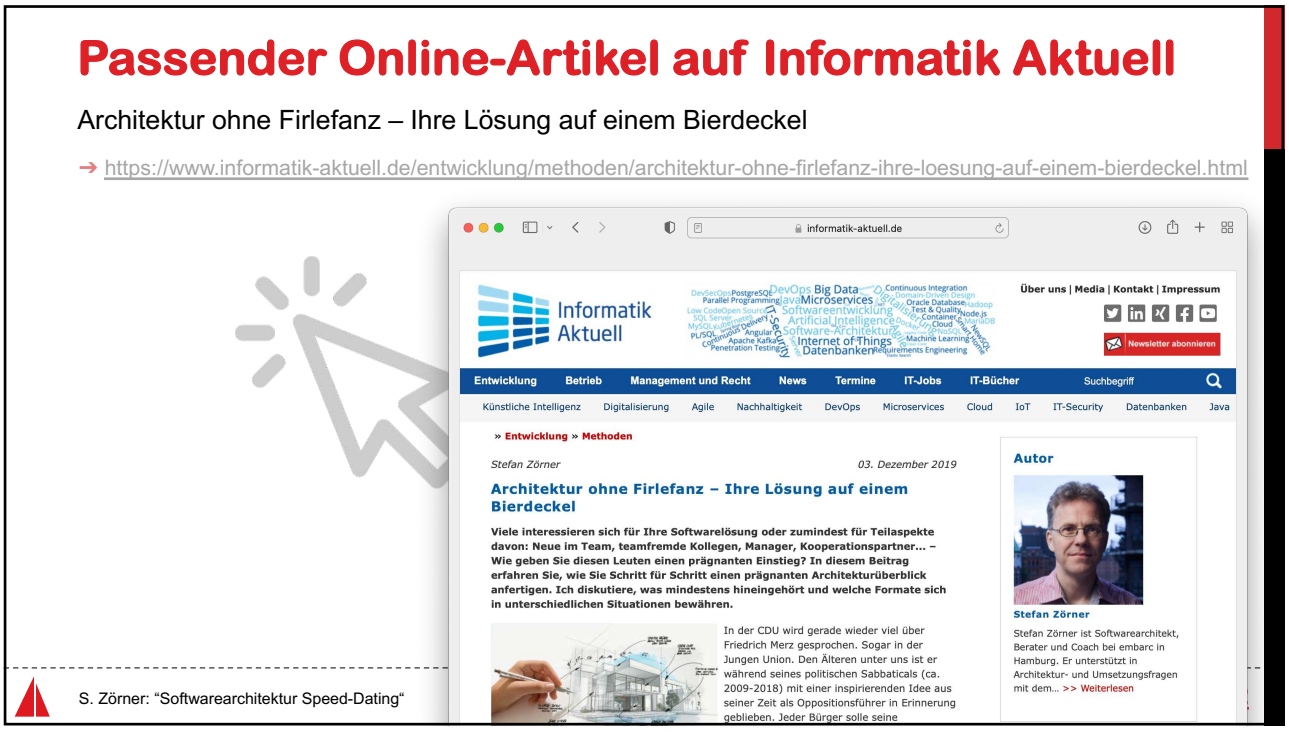
Wie unterscheidet es sich von Produkten der Mitbewerber, oder der Vorgängerversion?

Folien als PDF zum Download



➔ embarc.de/download/


WAS sollen wir bauen?
Was wollen wir erreichen?
Wozu ist es da?
Wem nützt es?
Was soll es tun?
Was braucht es nicht tun?
Woran halten wir uns?
Was soll es exzellent können?
ANFORDERUNGEN
Architekturüberblick
Aufgabe
- Mission Statement
- Kontextabgrenzung
Einflüsse
- Rahmenbedingungen
- Qualitätsziele
Lösungsansätze
- Architekturstil
- Technologie-Stack
- Konzepte
- Prinzipien
- Zerlegung
- ....
Lösungsstrategie (plus Überblicksbild) als Brücke
WIE sieht die Lösung aus?
Wie erreichen wir das?
Welchen Mustern folgen wir?
Was verwenden wir?
Was leitet uns?
Woraus besteht es?
Wie ist es strukturiert?
ENTSCHEIDUNGEN
S. Zörner: "Softwarearchitektur Speed-Dating"
Quelle: S. Zörner: Architektur ohne Firlefanz – Ihre Lösung auf einem Bierdeckel


Passender Online-Artikel auf Informatik Aktuell
Architektur ohne Firlefanz – Ihre Lösung auf einem Bierdeckel
➔ https://www.informatik-aktuell.de/entwicklung/methoden/architektur-ohne-firlefanz-ihre-loesung-auf-einem-bierdeckel.html
informatik-aktuell.de
Informatik Aktuell
Über uns | Media | Kontakt | Impressum
Newsletter abonnieren
Entwicklung Betrieb Management und Recht News Termine IT-Jobs IT-Bücher Suchbegriff
Künstliche Intelligenz Digitalisierung Agile Nachhaltigkeit DevOps Microservices Cloud IoT IT-Security Datenbanken Java
» Entwicklung » Methoden
Stefan Zörner
03. Dezember 2019
Architektur ohne Firlefanz – Ihre Lösung auf einem Bierdeckel
Viele interessieren sich für Ihre Softwarelösung oder zumindest für Teilaspekte davon: Neue im Team, teamfremde Kollegen, Manager, Kooperationspartner... – Wie geben Sie diesen Leuten einen prägnanten Einstieg? In diesem Beitrag erfahren Sie, wie Sie Schritt für Schritt einen prägnanten Architekturüberblick anfertigen. Ich diskutiere, was mindestens hineingehört und welche Formate sich in unterschiedlichen Situationen bewähren.
In der CDU wird gerade wieder viel über Friedrich Merz gesprochen. Sogar in der Jungen Union. Den Älteren unter uns ist er während seines politischen Sabbaticals (ca. 2009-2018) mit einer inspirierenden Idee aus seiner Zeit als Oppositionsführer in Erinnerung geblieben. Jeder Bürger solle seine
Autor
Stefan Zörner
Stefan Zörner ist Softwarearchitekt, Berater und Coach bei embarc in Hamburg. Er unterstützt in Architektur- und Umsetzungsfragen mit dem... >> Weiterlesen
S. Zörner: "Softwarearchitektur Speed-Dating"

Das Buch zum Film ;-)

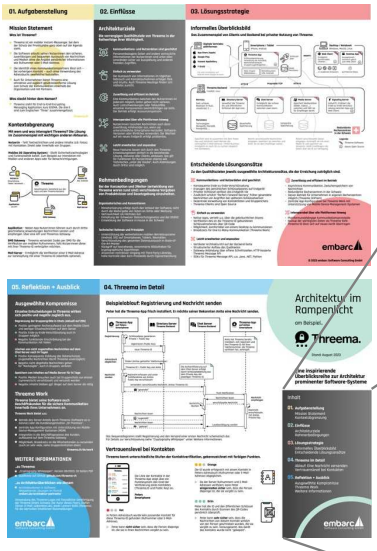
Stefan Zörner
Softwarearchitekturen dokumentieren und kommunizieren

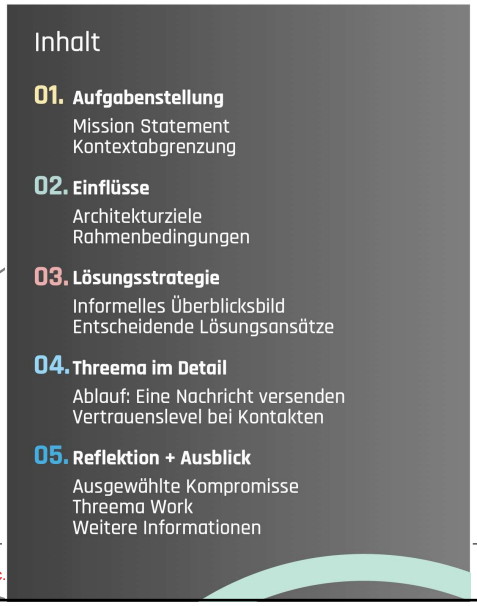
Entwürfe, Entscheidungen und Lösungen nachvollziehbar und wirkungsvoll festhalten

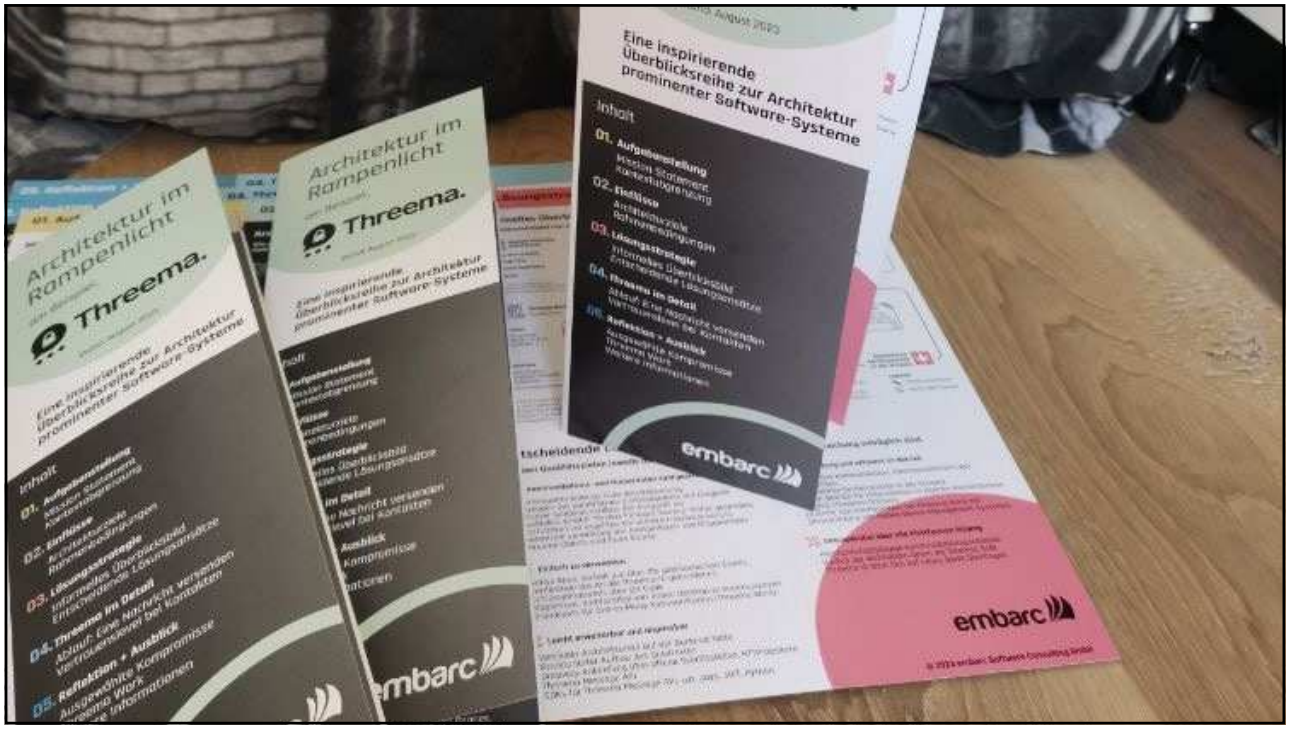
Hanser Fachbuch
3., überarbeitete und erweiterte Auflage (Dez. 2021)
309 Seiten. Fester Einband
ISBN: 978-3-446-46928-0

➔ https://www.swadok.de

Flyer: Instant-Messenger Threema

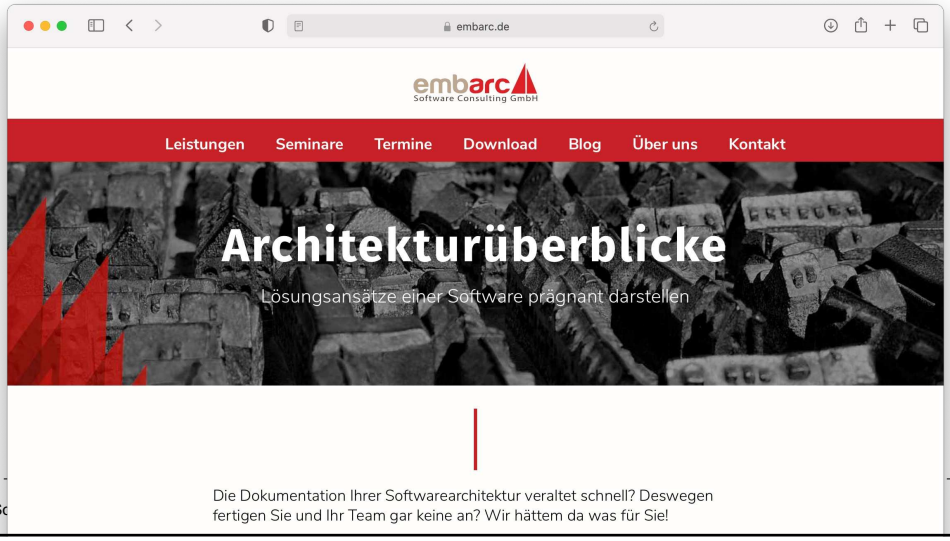






Auch als Download (PDF, zum Selberdrucken ;-)

➔ https://www.embarc.de/architektur-ueberblicke/



S. Zörner: "So

66

Artikel-Reihe in IT-Spektrum

Alle zwei Monate ein neues Porträt seit Ausgabe 04 | 2022 …

Zeitversetzt auch online im embarc-Blog

➔ embarc.de/architektur-portraets/

Vielen Dank.

Ich freue mich auf Eure Fragen!

Stefan.Zoerner@embarc.de

@StefanZoerner

linkedin.com/in/stefan-zoerner

xing.to/szr

@StefanZoerner@mastodon.social