Architekturikonen in Software

Wegweisende Lösungen im Porträt.

Zusammenfassung

Die Architekturkritik bezeichnet bahnbrechende Bauwerke als Architekturikonen. In diesem Vortrag greife ich den Begriff auf und diskutiere die Lösungsstrategien einiger prominenter Softwarelösungen. Ich stelle den Architekturzielen die gewählten Entwurfsentscheidungen gegenüber und mache auf diese Weise das Geheimnis ihres Erfolges sichtbar.

Sie erwartet eine kleine Galerie prägnanter Architektur-Porträts vom Framework bis zum Quelltext-Editor, von 2002 bis 2020. Was ist die Sydney-Oper der Softwarearchitektur?

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 Architekturikonen in Software
1 Porträt #1
2 Porträt #2
3 Porträt #3
4 Porträt #4
5 Porträt #5
6 Zusammenfassung und Ausblick

Agenda

0 **Architekturikonen in Software**

1 Porträt #1

2 Porträt #2

3 Porträt #3

4 Porträt #4

5 Porträt #5

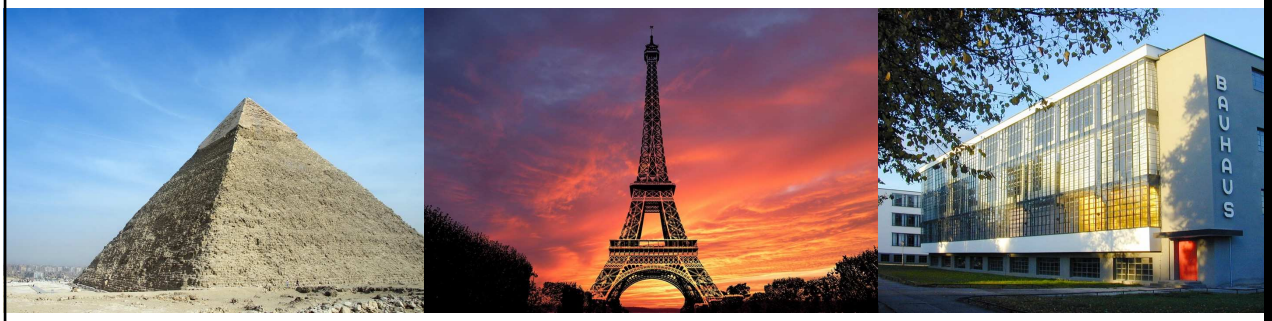
6 Zusammenfassung und Ausblick

0

Architekturikonen bei Gebäuden

„Architekturikone ist ein Begriff aus der Architekturkritik und bezeichnet Bauwerke, die **wegweisend** sind beziehungsweise waren oder aufgrund ihrer Gestaltung **Einzigartigkeit** beanspruchen."

(Wikipedia)



Architekturikonen bei Gebäuden

„… Diese herausragenden Bauwerke, Gebäude und Ensembles erfüllen mehrere der folgenden Kriterien:

- allgemeine Anerkennung
- Popularität
- Originalität
- Symbolwert
- Bedeutung für die Entwicklung der Architektur
- repräsentativ für einen Architekturstil“

Was erwartet Euch in diesem Vortrag?

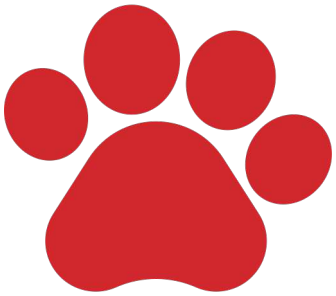
- Fünf **Kurzporträts** prominenter (wegweisender) Softwaresysteme der letzten zwei Jahrzehnte.
- Jeweils eine kurze Einordnung, die **Architekturziele** und die zentralen **Lösungsansätze**. Plus Informelle Architekturüberblicke (Bilder)
- Hinweise, wo Ihr noch mehr zu den Systemen erfahrt, und wo Ihr weitere Porträts findet.

5 Software-Gattungen

Die gezeigten Softwarelösungen sind sehr unterschiedlich …

- Applikationsframework
- Datenbank
- Mobile App
- Programmiersprache
- Quelltexteditor

(alphabetisch sortiert)

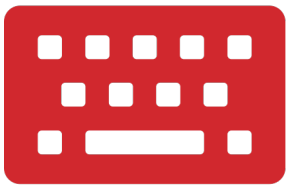


5 Programmiersprachen

… sie sind in unterschiedlichen Programmiersprachen entwickelt:

- C
- Go
- Java
- Swift
- TypeScript

(alphabetisch sortiert)



5 Erscheinungsjahre

… und in unterschiedlichen Zeiten veröffentlicht worden

- 2002
- 2007
- 2012
- 2015
- 2020

(chronologisch sortiert)

Agenda

0 Architekturikonen in Software

1 **Porträt #1: Das Spring Framework**

2 Porträt #2

3 Porträt #3

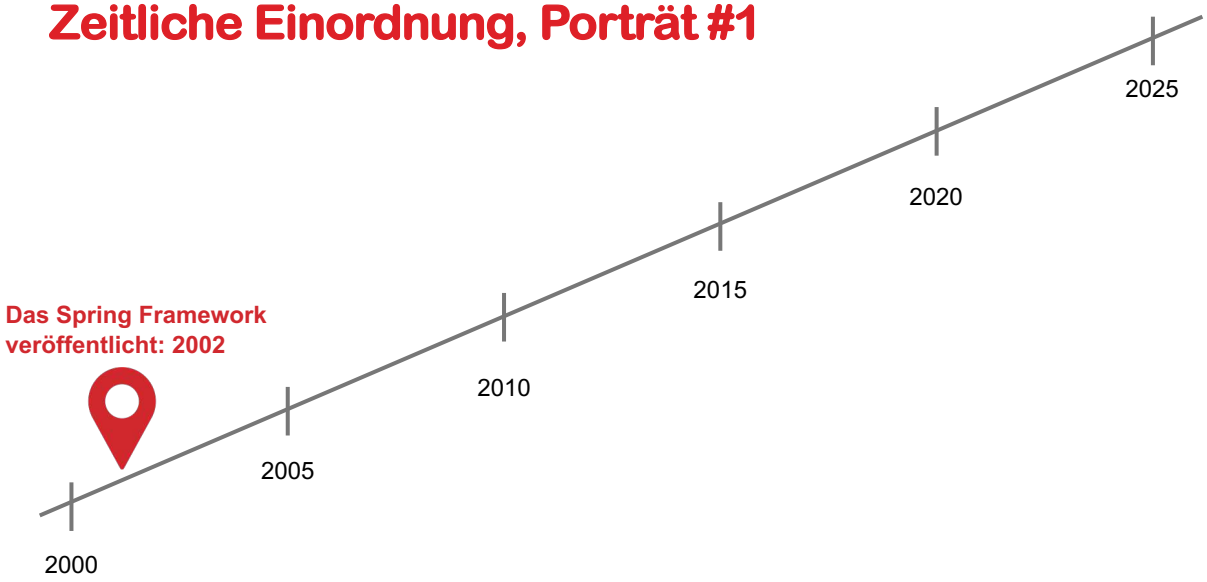
4 Porträt #4

5 Porträt #5

6 Zusammenfassung und Ausblick

1

Zeitliche Einordnung, Porträt #1



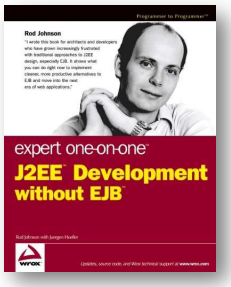
Die Java-Welt 2002 und Kritik an J2EE

Java um die Jahrtausendwende

- Technologie ist ausgewachsen (Sprache seit 1995 verfügbar)
- Unternehmen beginnen geschäftskritische Anwendungen in Java zu realisieren
- Standard-Java dazu: J2EE („Enterprise Edition")

Kritik an J2EE

- Geringe Entwicklerproduktivität
- Einfache Dinge nicht angemessenen leicht, das galt insbesondere für das Komponentenmodell EJB
- Prominente Stimme: Rod Johnson



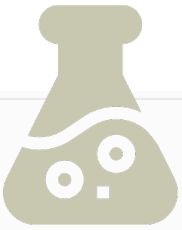
Wichtige Architekturziele des Spring Frameworks

- 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))*

Zentrale Lösungsansätze von Spring

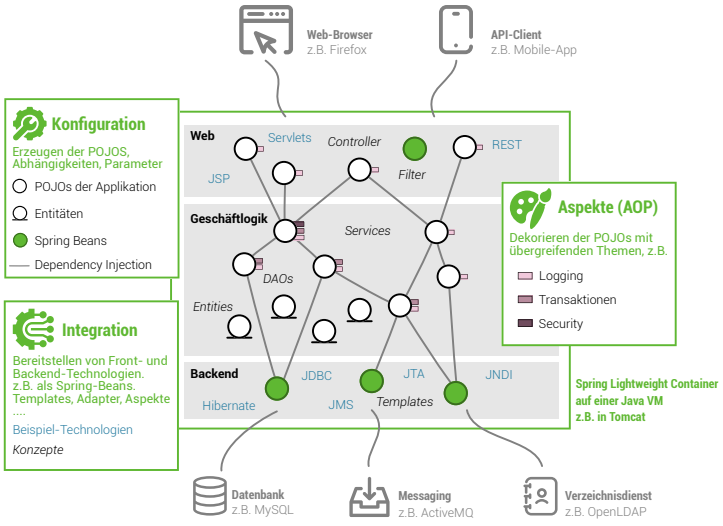


- POJOs (JavaBeans) als "Spring-Komponenten" (keine Abhängigkeit zum Framework, nicht invasiv)
- Dependency Injection zur Entkopplung, Lightweight Container
- Klare Trennung von fachlichem und technischem Code möglich.
- Aspektorientierte Programmierung (AOP) für Transaktionen, Security ... "entrümpelt" 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

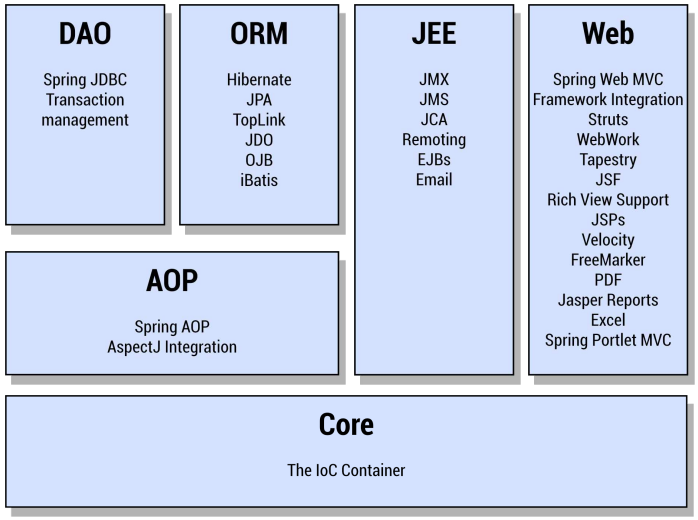
Zuordnung zu Zielen (Farbcode)

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

Eine Web-Anwendung im Rahmen von Spring



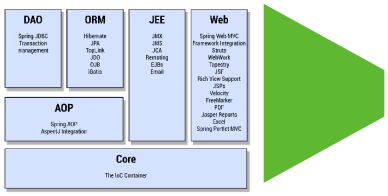
Das Spring Framework ca. 2004



Abkürzungen (Auswahl)

DAO	Data Access Object
ORM	Objekt-/Relationales Mapping
AOP	Aspekt-orientierte Programmierung
JEE	Java Enterprise Edition
JPA	Java Persistence API
JMS	Java Message Service
EJB	Enterprise JavaBeans
MVC	Model View Controller
JSF	JavaServerFaces
JSP	JavaServerPages
IoC	Inversion of Control

von 2004 nach 2022 …



Spring Framework ca. 2004

Projekte des Spring Ökosystems (Sommer 2022)

Spring Framework 2004 ★★★	Spring Batch 2008 ★	Spring REST Docs 2015 ★	Spring CredHub 2018
Spring Web Flow 2006	Spring AMQP 2011	Spring Statemachine 2015 ★	Spring Flo 2018
Spring LDAP 2006	Spring Data 2011 ★★	Spring Cloud 2015 ★★	Spring HATEOAS 2019
Spring Security 2006 ★★	Spring Shell 2012	Spring Cloud Data Flow 2016	Spring for GraphQL 2022 ★
Spring Web Services 2007	Spring Boot 2014 ★★★	Spring for Apache Kafka 2016 ★	
Spring Integration 2008 ★	Spring Session 2015	Spring Vault 2017	

Das Spring Ökosystem 2022

Spring Framework 2004 ★★★	Spring Batch 2008 ★	Spring REST Docs 2015 ★	Spring CredHub 2018
Spring Web Flow 2006	Spring AMQP 2011	Spring Statemachine 2015 ★	Spring Flo 2018
Spring LDAP 2006	Spring Data 2011 ★★	Spring Cloud 2015 ★★	Spring HATEOAS 2019
Spring Security 2006 ★★	Spring Shell 2012	Spring Cloud Data Flow 2016	Spring for GraphQL 2022 ★
Spring Web Services 2007	Spring Boot 2014 ★★★	Spring for Apache Kafka 2016 ★	
Spring Integration 2008 ★	Spring Session 2015	Spring Vault 2017	

Anzahl Sterne bei GitHub
0: < 1000, 1: 1000-4999
2: 5000 - 24999, 3: >= 25000

1.0 GA erschienen 2008 2015

spring

Steckbrief: Spring Framework

- **Software-Gattung** : Applikationsframework
- **Veröffentlicht** : 2002 (Version 1.0 2004)
- **Herkunft / Ursprung** : Interface21 (später VMWare), Open Source (Apache License)
- **Zielplattform** : Java (JVM), plattformunabhängig
- **Programmiersprache(n)** : Java
- **Homepage** : ➔ https://spring.io

Vermächtnis (Warum bemerkenswert?)

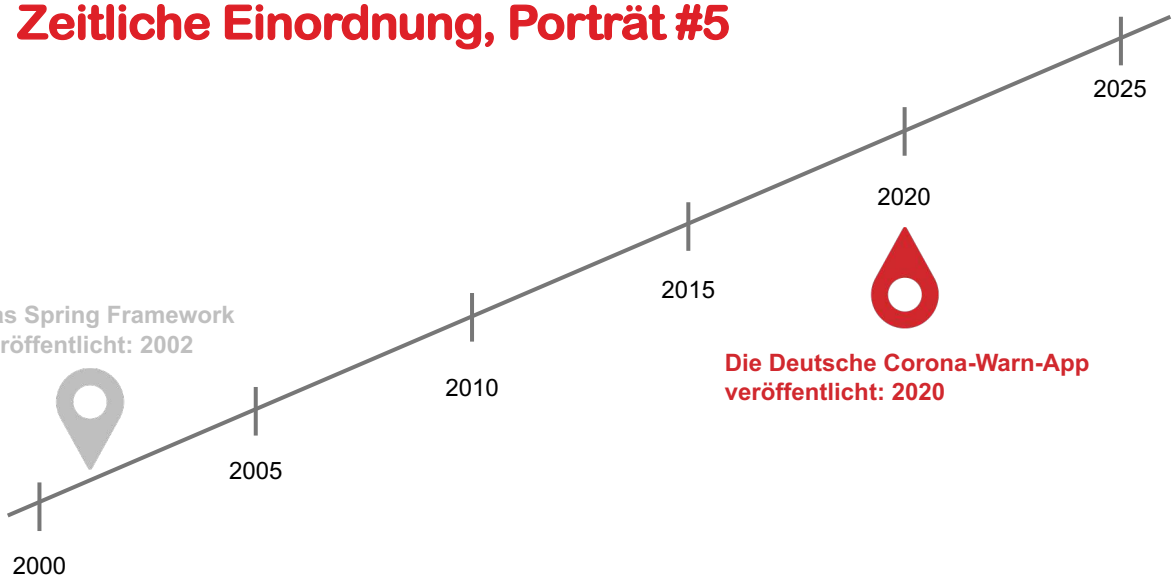
- Stilbildend für Unternehmensanwendungen in Java und Java EE
- über Java hinaus: Etablieren von Dependency Injection und AOP

Agenda

0 Architekturikonen in Software
1 Porträt #1: Spring
2 Porträt #2: Die deutsche Corona-Warn-App
3 Porträt #3
4 Porträt #4
5 Porträt #5
6 Zusammenfassung und Ausblick

2

Zeitliche Einordnung, Porträt #5



Deutschland im Jahr 2020

Der Spiegel, Titelbild 19/2020

Das Virus und die Kontaktverfolgung

- Mit dem ersten Infektionsfall in Bayern beginnt Ende Januar 2020 offiziell die Ausbreitung des neuartigen **Coronavirus** SARS-CoV-2 in Deutschland.
- Schon früh wurden sogenannte **Corona-Apps** als eine Möglichkeit der Kontaktverfolgung und damit als ein Mittel zur Unterbrechung von Infektionsketten diskutiert.
- Der Chaos Computer Club stuft Contact Tracing Apps im April 2020 als **Risikotechnologie** ein.

Die 10 Prüfsteine des Chaos Computer Club

Gesellschaftliche Anforderungen

1. Epidemiologischer Sinn & Zweckgebundenheit
2. Freiwilligkeit & Diskriminierungsfreiheit
3. Grundlegende Privatsphäre
4. Transparenz und Prüfbarkeit

Technische Anforderungen

5. Keine zentrale Entität, der vertraut werden muss
6. Datensparsamkeit
7. Anonymität
8. Kein Aufbau von zentralen Bewegungs- und Kontaktprofilen
9. Unverkettbarkeit
10. Unbeobachtbarkeit der Kommunikation

➔ embarc.de/architekturprinzipien-pruefsteine/

Auftrag der Bundesregierung

Grober Rahmen

- Entwicklung und Betrieb durch ein Konsortium (SAP, Deutsche Telekom)
- Start der Entwicklung 04/2020
- Entwicklung Open Source. Quelltexte und Dokumentation auf GitHub
- Einsatz des Exposure Notification Framework von Google und Apple
- Verfolgen eines dezentralen Ansatzes für die Datenspeicherung

Beitrag Tagesschau vom 26.04.2020, Video (2 min)

„Bundesregierung setzt bei Corona-App auf dezentrale Datenspeicherung“

➔ tagesschau.de/multimedia/video/video-693083.html

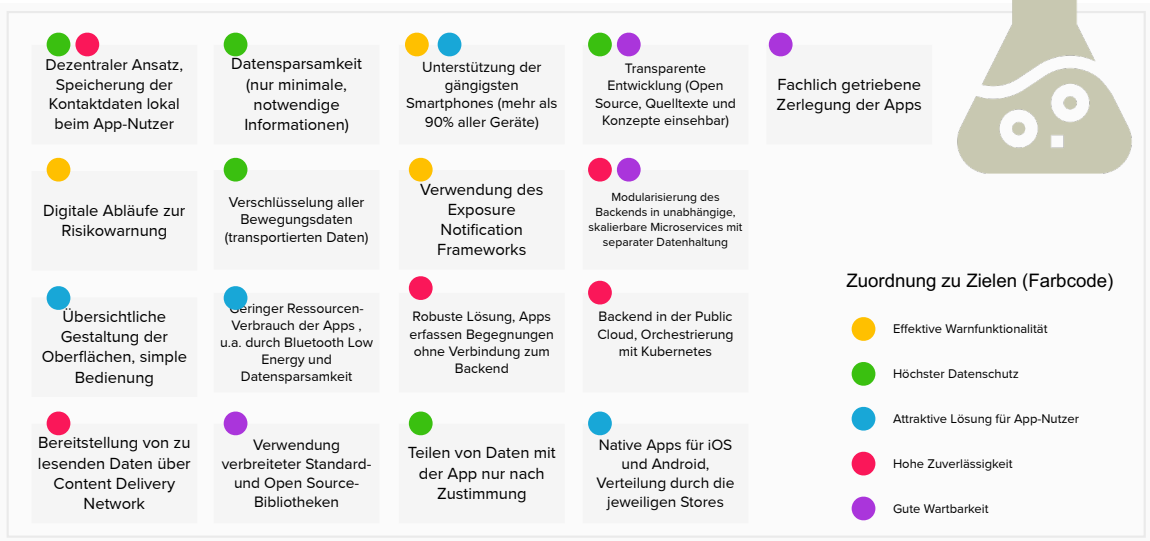
Wichtige Architekturziele der Corona-Warn-App

- Effektive Warnfunktionalität
- Höchster Datenschutz
- Attraktive Lösung für App-Nutzer
- Hohe Zuverlässigkeit
- Gute Wartbarkeit

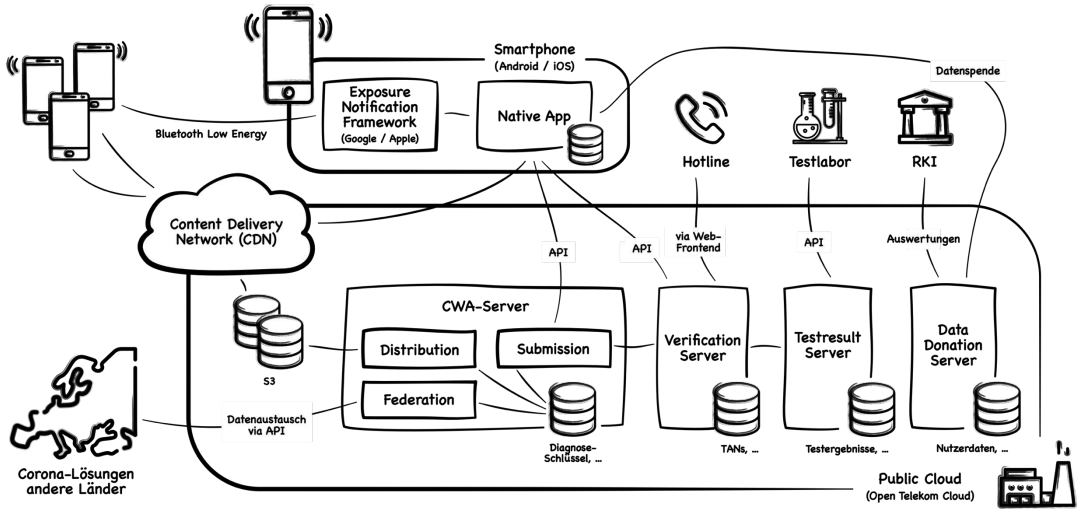
Wichtige Architekturziele der Corona-Warn-App

Ziel	Beschreibung (und zugehöriges Software-Qualitätsmerkmal)
Effektive Warnfunktionalität	Die App ist ein effektiver Baustein bei der Pandemie-Bekämpfung. Kontakte positiv getesteter Personen werden umgehend informiert, Infektionsketten so gebrochen. *(Funktionale Eignung)*
Höchster Datenschutz	Eine IT-gestützte Kontaktverfolgung birgt aufgrund möglicherweise erfasster Kontakt- und Gesundheitsdaten ein hohes Risiko. Der Schutz dieser Daten hat oberste Priorität, Alarmierungen erfolgen grundsätzlich dezentral und anonym. *(Sicherheit)*
Attraktive Lösung für App-Nutzer	Die App ist auch von technisch wenig versierten Personen leicht zu installieren sowie intuitiv und effizient zu bedienen. Ressourcen des Endgerätes wie Bandbreite, Batterie und Speicherplatz nutzt die App sparsam. *(Benutzbarkeit)*
Hohe Zuverlässigkeit	Die Lösung geht mit Lastspitzen wegen hoher Nutzer- oder Infektionszahlen ebenso souverän um, wie mit Störungen und böswilligen Angriffen. Die App arbeitet stets ohne Beeinträchtigungen. *(Zuverlässigkeit)*
Gute Wartbarkeit	Die Software lässt sich leicht anpassen oder um neue Funktionalität erweitern, wenn z. B. Nutzer/-innen, Politik oder neue Forschungsergebnisse es erfordern. Neue Teammitglieder finden sich rasch zurecht. *(Wartbarkeit/Erweiterbarkeit)*

Zentrale Lösungsansätze der Warn-App



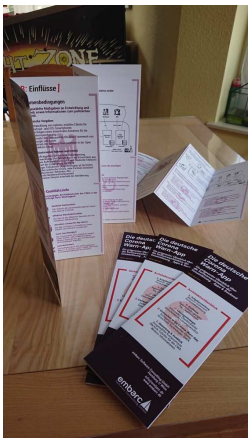
Informeller Architekturüberblick Corona-Warn-App

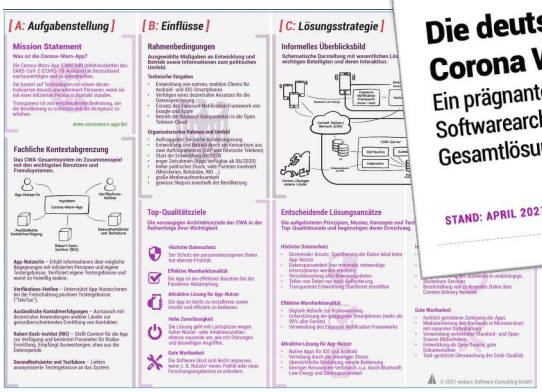


Quelle der Abbildung: S. Zörner, F. Sippach: „So gehen Architektur-Reviews! Entlang der Corona-Warn-App", OOP 2021

Mehr zur Corona-Warn-App

Prägnanter Architekturüberblick als Flyer
(DIN-A3, Treppenfalz) und Folienvortrag





➔embarc.de/architektur-ueberblicke/#corona-warn-app

Steckbrief: Corona-Warn-App

Software-Gattung	: Contact-Tracing
Veröffentlicht	: Juni 2020
Herkunft / Ursprung	: Deutsche Bundesregierung, Open Source
Zielplattform	: Native App auf Android und iOS, Backend in Public Cloud
Programmiersprache(n)	: Kotlin (Android App), Swift (iOS App) Java (Backend)
Homepage	: ➔ coronawarn.app

Warum bemerkenswert? („Vermächtnis“)

- Vorbild für die transparente Entwicklung von Software bei öffentlicher Vergabe

Agenda

0 Architekturikonen in Software

1 Porträt #1: Spring

2 Porträt #2: Go

3 **Porträt #3: Die Graphdatenbank Neo4j**

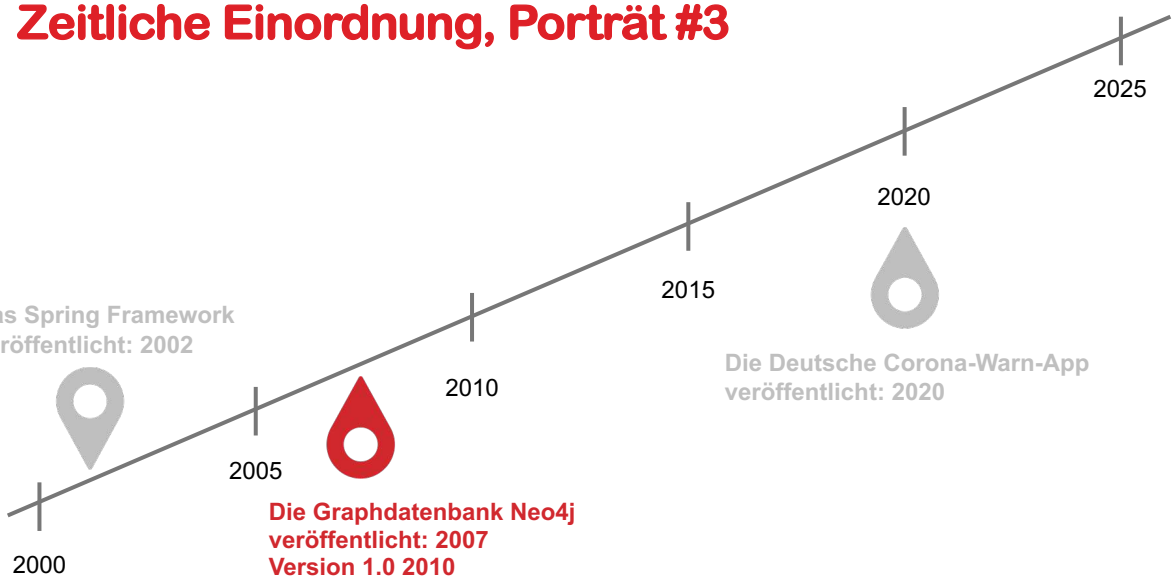
4 Porträt #4

5 Porträt #5

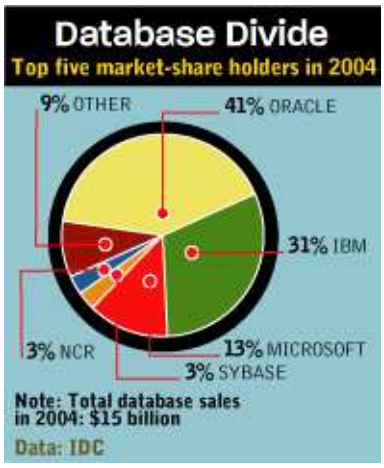
6 Zusammenfassung und Ausblick

3

Zeitliche Einordnung, Porträt #3



Der Datenbank-Markt um 2000



Quelle: IDC 2004

Informationen zu speichern und zu verarbeiten ist eine elementare Aufgabe in der Softwareentwicklung.

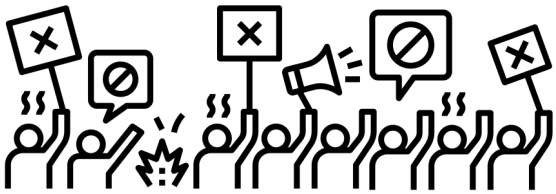
Relationale Datenbanken

- Prägen seit den 80-er Jahren die Systemlandschaften vieler Unternehmen und Organisationen beim Thema Persistenz
- Prominente Vertreter: Oracle, IBM Db2, MySQL
- Pflicht in der IT-Ausbildung und im Studiums, inklusive Abfragesprache SQL

Die NoSQL-“Bewegung“ (ab ca. 2009)

Alternativen zum relationalen Modell

Akzeptanz, dass in bestimmten Anwendungsfällen und unter gewissen Voraussetzungen andere Speicher-techniken deutlich besser geeignet sind.



NoSQL == Not only SQL

Auswahl wichtiger Datenbankmodelle

Relational (RDBMS)

Speicherung strukturierter Daten in Tabellen gemäß eines Schemas. Beziehungen mit Schlüsseln, Zugriffe mit SQL.

- z.B. Bestandssysteme
- Oracle (1980)
- MySQL (1995)

Key-Value Store

Speicherung der Daten als Schlüssel/ Wert-Paare. Effizienter Zugriff auf einen einzelnen Datensatz über den Schlüssel.

- z.B. Caching, Konfiguration
- Redis (2009)
- etcd (2013)

Graphbasiert

Speicherung der Daten als Graph mit Knoten und Kanten. Effiziente Auswertung von Verbindungen der Daten untereinander.

- z.B. Soziale Netzwerke
- Neo4j (2007)
- JanusGraph (2017)

Wide-Column Store

Speicherung der Daten als Zeilen in einer Tabelle, jeder Datensatz kann dabei andere Spalten haben, die sich gruppieren lassen (“Column Family”). Effizienter Zugriff über Zeilen und Spalten.

- z.B. Data-Mining
- Google Bigtable (2005)
- Apache Cassandra (2008)

Zeitreihen (Time Series)

Speicherung der Daten als Zeitpunkt/ Wert-Paare. Effiziente Auswertung inkl. Verdichtung der Inhalte über Zeiträume.

- z.B. Sensordaten, Monitoring
- InfluxDB (2013)
- Prometheus (2015)

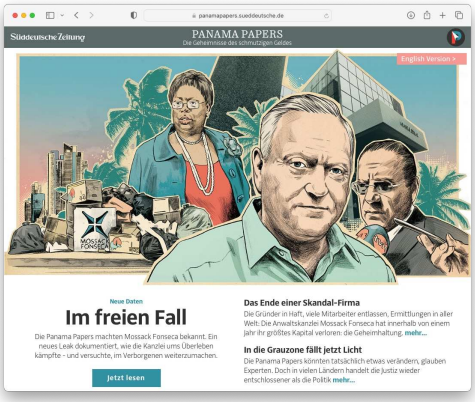
Dokumentenorientiert

Speicherung von Daten unstrukturiert (oder semi-strukturiert) in Dokumenten, Format z.B. XML oder JSON. Zugriff über Schlüssel, Inhalte oder Metadaten

- z.B. Produktbeschreibungen
- Apache CouchDB (2005)
- MongoDB (2009)
- Couchbase (2011)

Legende: = **Typische Anwendung(en)** = **Prominentes Produkt** (in Klammern: erster Release)

Beispiel: Die Daten der Panama Papers



Daten-Leak von 2016

- 11,5 Millionen E-Mails und andere Dokumenten (Urkunden, Passkopien …)
- Ein riesiges Beziehungsgeflecht. Personen und Unternehmen, Beteiligungen, Verwandtschaftsbeziehungen …
- Journalisten suchen nach Verdachtsmomenten für Delikte wie Geldwäsche, Steuerbetrug …

Wichtige Architekturziele von Neo4j

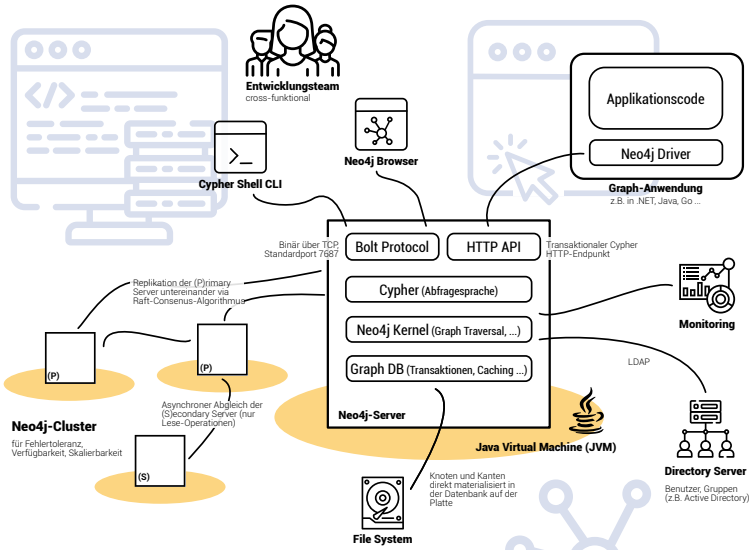
- Einfache Realisierung graphbasierter Anwendungen
- Schnelle Ausführung von Abfragen
- Zuverlässig im Betrieb
- Lauffähig auf vielen Hard- und Softwareplattfomen
- Leicht ausbaubar und erweiterbar um neue Funktionalität

Wichtige Architekturziele von Neo4j

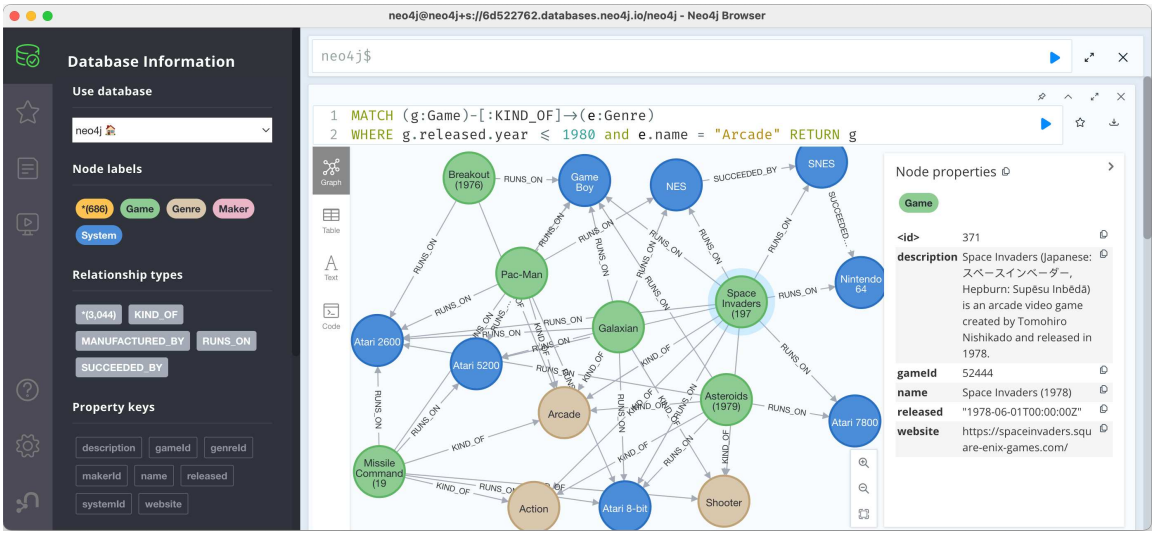
Ziel	Beschreibung (und zugehöriges Software-Qualitätsmerkmal)
Einfache Realisierung graphbasierter Anwendungen	Mit Neo4j lassen sich Graph-gestützte Anwendungen einfach umsetzen. Alle am Entwicklungsprozess Beteiligten erleben eine gute Nutzerfreundlichkeit. Neo4j steht relationalen Datenbanken bezüglich der Lernkurve etwa bei Entwickler:innen in nichts nach. *(Nutzerfreundlichkeit aus Sicht der Entwicklung)*
Schnelle Ausführung von Abfragen	Neo4j führt komplexe Abfragen auch auf sehr großen Datenmengen in kürzester Zeit aus. Insbesondere solche, die sich auf Verknüpfungen zwischen Daten beziehen. Auch die Ausführung aufwändiger Graph-Algorithmen erfolgt sehr effizient. *(Effizienz)*
Zuverlässig im Betrieb	Neo4j speichert Daten verlässlich und sichert deren Konsistenz zu. Auch in unternehmenskritischen Anwendungsfällen kann Neo4j eine hohe Verfügbarkeit darstellen. Gespeicherte Daten sind vor unberechtigtem Zugriff sicher. *(Zuverlässigkeit, Sicherheit)*
Lauffähig auf vielen Hard- und Software-plattfomen	Neo4j lässt sich serverseitig auf allen relevanten Zielsystemen betreiben. Es ist leicht in bestehende Systemlandschaften integrierbar. Bezüglich der Auswahl von Clients, die auf die Daten zugreifen wollen, gibt es keine relevanten Einschränkungen. *(Portierbarkeit, Integrierbarkeit)*
Leicht ausbaubar und erweiterbar um neue Funktionalität	Die Lösung lässt sich gut erweitern, durch das Neo4j-Team ebenso wie durch Anwender aus der Community. Dies betrifft Anpassungen vorhandener Bestandteile ebenso wie neue Funktionalität, die auf den gespeicherten Daten operiert. Neo4j ist für offen für technologische und methodische Trends. *(Erweiterbarkeit)*

Zentrale Lösungsansätze von Neo4j

Informeller Architektur-Überblick Neo4j



Explorieren mit Cypher im Neo4j-Browser



Steckbrief: Neo4j

Software-Gattung	: Graphdatenbank
Veröffentlicht	: 2007 (Version 1.0: Februar 2010)
Herkunft / Ursprung	: Neo4j, Inc., Community Edition ist Open Source (GPL v3)
Zielplattform	: Cross-Plattform (u.a. Linux, Windows …)
Programmiersprache(n)	: implementiert in Java (und Scala), Web-Tools in TypeScript, Cloud-Aspekte in Go
Homepage	: ➔ neo4j.com

Warum bemerkenswert? („Vermächtnis“)

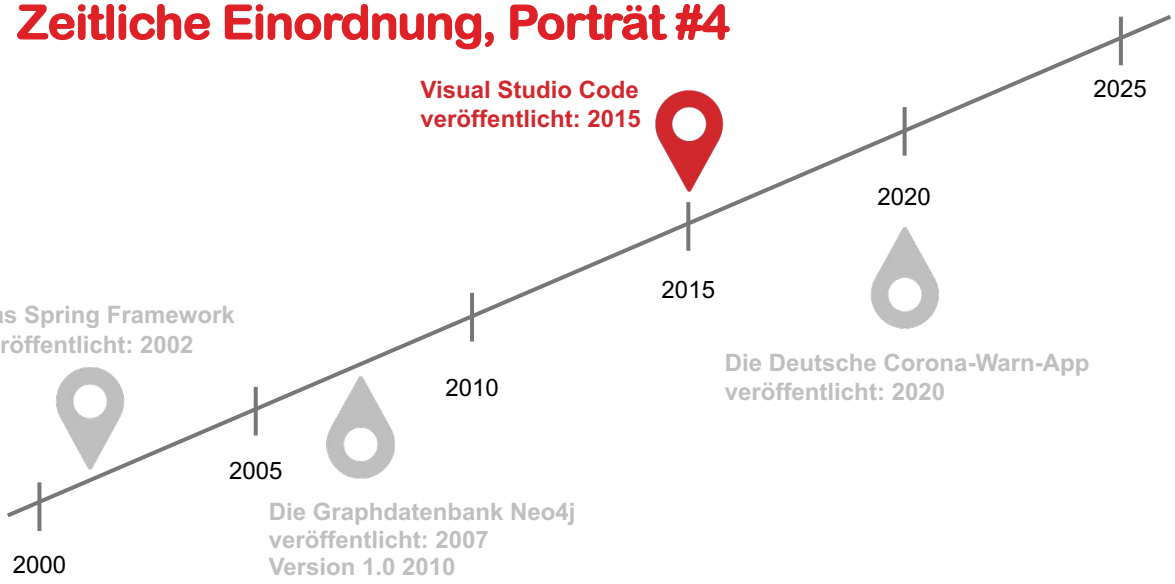
- Wichtiger Vertreter der NoSQL-Bewegung, führend in Graphdatenbanken
- Herausragende Dokumentation

Agenda

4

0 Architekturikonen in Software
1 Porträt #1: Spring
2 Porträt #2: Go
3 Porträt #3: Neo4j
4 **Porträt #4: Visual Studio Code**
5 Porträt #5
6 Zusammenfassung und Ausblick

Zeitliche Einordnung, Porträt #4



Spektrum der Quelltexteditoren

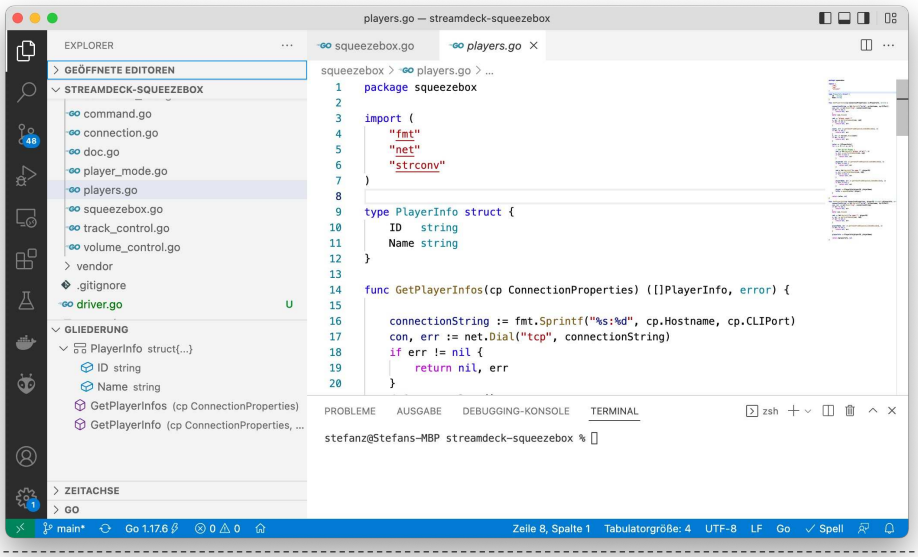
Reine Texteditoren wie **vi** (1976) oder **Sublime** (2008), ggf. mit Programmiersprachen-unterstützenden Features

Ausgewachsenen Entwicklungs-umgebungen (kurz IDEs) wie **Eclipse** (2001) oder **Visual Studio** von Microsoft für .NET.

VS Code: Ein neuer Spieler auf dem Platz

- Im Spektrum näher an reinen Texteditoren
- Ursprünglich als Editor im Browser von einem Team um Erich Gamma bei Microsoft konzipiert

Visual Studio Code



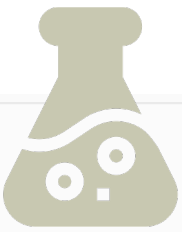
Wichtige Architekturziele von Visual Studio Code

- Überzeugendes Benutzungserlebnis beim Programmieren
- Leicht zu erweitern, auch durch Dritte
- Bestmöglich auf allen relevanten Zielumgebungen
- Verlässlicher Betrieb
- Gute Wartbarkeit der Plattform, auch in Zukunft

Wichtige Architekturziele von Visual Studio Code

	Ziel	Beschreibung (und zugehöriges Software-Qualitätsmerkmal)
⌨	**Überzeugendes Benutzungserlebnis beim Programmieren**	Mit VS Code lässt sich Quelltext intuitiv und effizient bearbeiten. Das Werkzeug punktet bei Entwicklern durch kaum spürbare Start- und Antwortzeiten, etwa beim Refactoring. *(Benutzbarkeit, Effizienz)*
🔌	**Leicht zu erweitern, auch durch Dritte**	VS Code lässt sich leicht um neue Sprachen oder Funktionen erweitern, durch uns und auch die Community oder unsere Partner. Entwicklungsnahe Tools sind bei Bedarf rasch angebunden. *(Erweiterbarkeit)*
💻	**Bestmöglich auf allen relevanten Zielumgebungen**	VS Code läuft anstandslos auf allen verbreiteten Desktop-Betriebssystemen – Windows, macOS, Linux. Und zwar so, wie es Nutzer des jeweiligen Betriebssystems erwarten. Auch remote Deployment-Szenarien sind mit der Lösung darstellbar. *(Portierbarkeit, Kompatibilität)*
❤	**Verlässlicher Betrieb**	VS Code ist ein sehr zuverlässiges Werkzeug. Nutzer setzen darauf, dass ihre Arbeitsergebnisse selbst bei Abstürzen oder anderen Störungen (z.B. Verbindungsabbrüchen) sicher sind. *(Zuverlässigkeit)*
🔧	**Gute Wartbarkeit der Plattform, auch in Zukunft**	VS Code selbst lässt sich auch über einen langen Zeitraum warten und weiterentwickeln. Die Lösung bleibt über viele Jahre zukunftsfähig. *(Wartbarkeit)*

Zentrale Lösungsansätze von VS Code

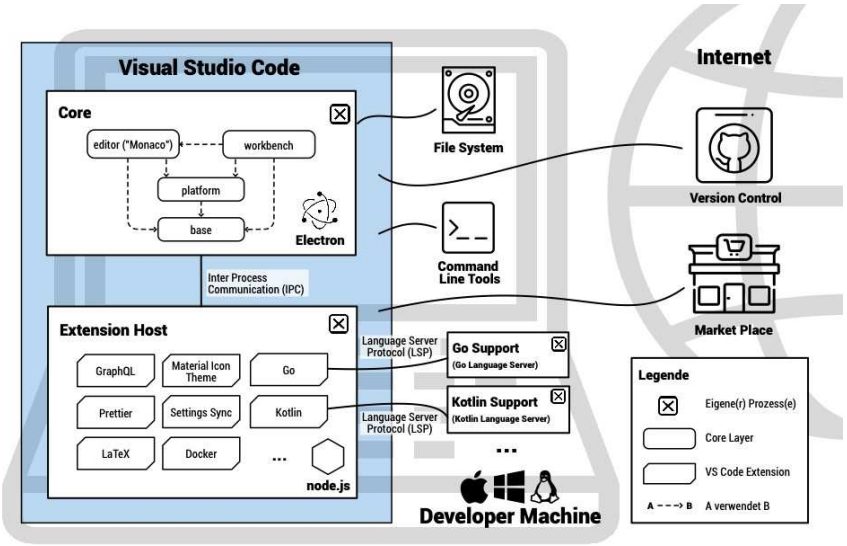


- Verwendung von Electron als UI-Framework
- Entwicklung in TypeScript
- Extension Mechanismus
- Language Server Protocol, ermöglicht Extensions in beliebigen Sprachen
- node.js im Hintergrund, für Zugriff auf Native OS-Ressourcen + Node Ecosystem
- "Extension Isolation" -- Extensions laufen im eigenen Prozess (nicht im Core)
- Zugriff auf Command Line ("Terminal") integriert
- Market Place für Extensions
- Entwicklung Open Source, dadurch rasches Feedback
- Unterstützung für wichtige Werkzeuge wie Git out-of-the-box

Zuordnung zu Zielen (Farbcode)

- Überzeugendes **Benutzungserlebnis** beim Programmieren
- Leicht zu **Erweitern**, auch durch Dritte
- Bestmöglich auf allen relevanten **Zielumgebungen**
- Verlässlicher **Betrieb**
- Gute **Wartbarkeit** der Plattform, auch in Zukunft

Informeller Architekturüberblick



Steckbrief: Visual Studio Code

Software-Gattung	: Quelltext-Editor
Veröffentlicht	: 2015 (Version 1.0 April 2016)
Herkunft / Ursprung	: Microsoft Open Source (MIT License)
Zielplattform	: Desktop (Windows, macOS, Linux) oder Browser-basiert
Programmiersprache(n)	: vorherrschend TypeScript
Homepage	: ➔ code.visualstudio.com

Warum bemerkenswert? („Vermächtnis“)

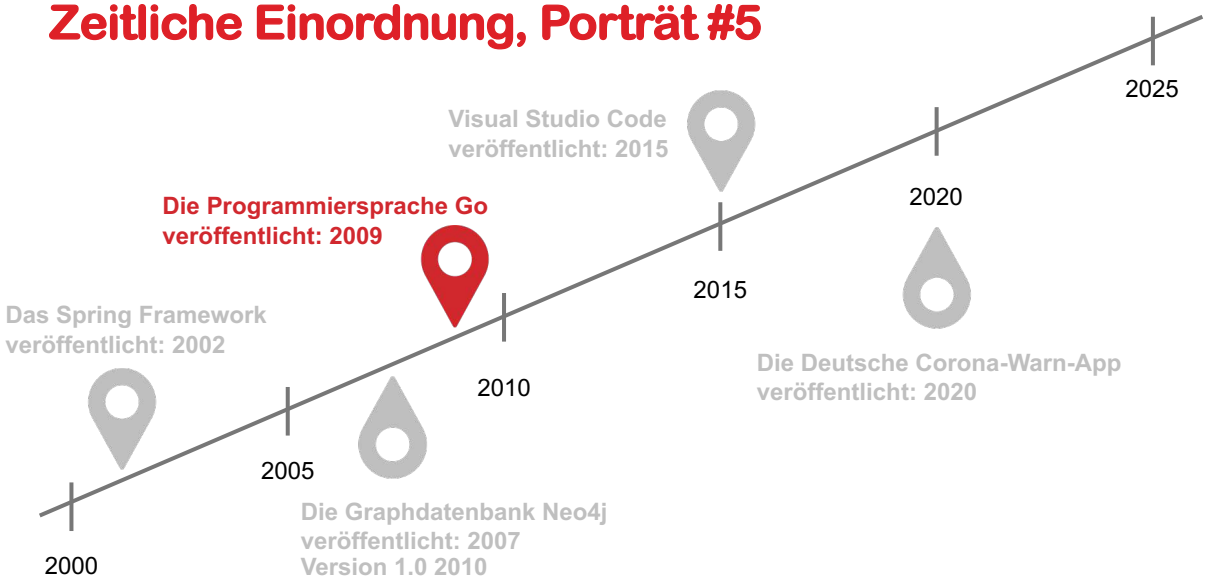
- Dank Erweiterbarkeit hohe Akzeptanz bei Communities auch außerhalb des Microsoft-Universums
- Mit dem Language Server Protocol wegweisend für Programmiersprachen-Unterstützung

Agenda

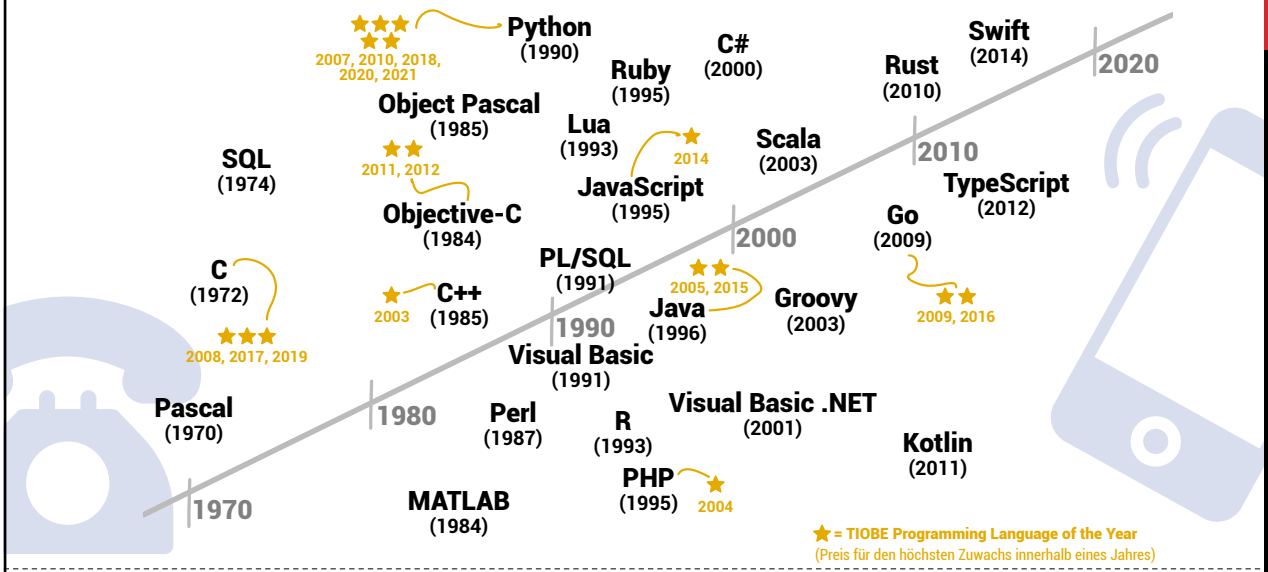
5

0 Architekturikonen in Software
1 Porträt #1: Spring
2 Porträt #2: Corona-Warn-App
3 Porträt #3: Neo4j
4 Porträt #4: Visual Studio Code
5 **Porträt #5: Die Programmiersprache Go**
6 Zusammenfassung und Ausblick

Zeitliche Einordnung, Porträt #5



Programmiersprachen seit 1970 (unvollständige Auswahl)



CNCF – Cloud Native Landscape

Ordnung schaffen in der Cloud.

*"Das Cloud-Native-Landscape-Projekt der CNCF ist als eine Karte durch das bisher unerforschte Terrain der **Cloud-Native-Technologien** gedacht. Damit wird versucht, viele der beliebtesten Projekte und Produktangebote im Cloud-Native-Bereich zu kategorisieren."*

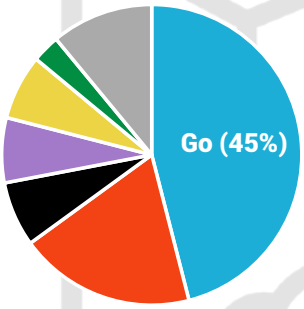
CLOUD NATIVE COMPUTING FOUNDATION

*Wörtlich: "The CNCF Cloud Native Landscape Project is intended as a map through the previously uncharted terrain of **cloud native technologies**. This attempts to categorize many of the most popular projects and product offerings in the cloud native space."*

➔ https://github.com/cncf/landscape

Open Source-Projekte der Cloud Native Landscape

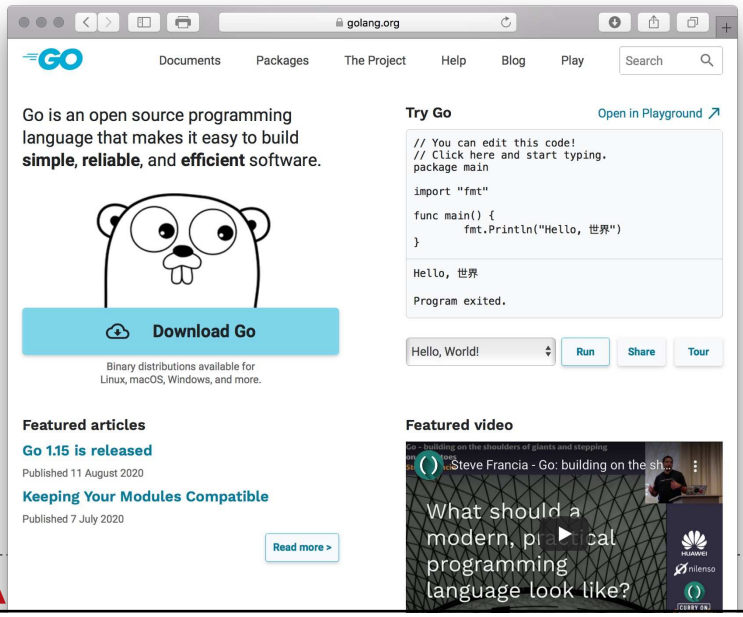
Top 100 Open-Source-Projekte der Cloud Native Landscape
Aufgeschlüsselt nach Programmiersprachen



Farbcode: Primäre Sprache im Projekt

- Go
- Java
- C
- C++
- JavaScript / TypeScript
- Python
- **Sonstige** (Scala Lua, Ruby, Rust, ...)

Was ist Go?

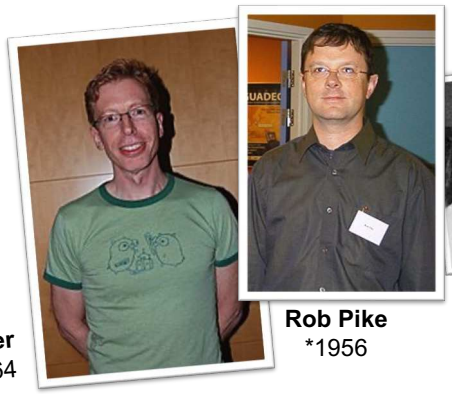


*Go ist eine Open-Source-Programmiersprache, die es leicht macht, **einfache**, **zuverlässige** und **effiziente** Software zu erstellen.*

➔ golang.org

Herkunft

Entwickelt von Google-Mitarbeitern.
Federführend dabei: Robert Griesemer, Rob Pike und Ken Thompson



Robert Griesemer
*1964

Rob Pike
*1956

Ken Thompson (1973)
*1943

(Dennis Ritchie)

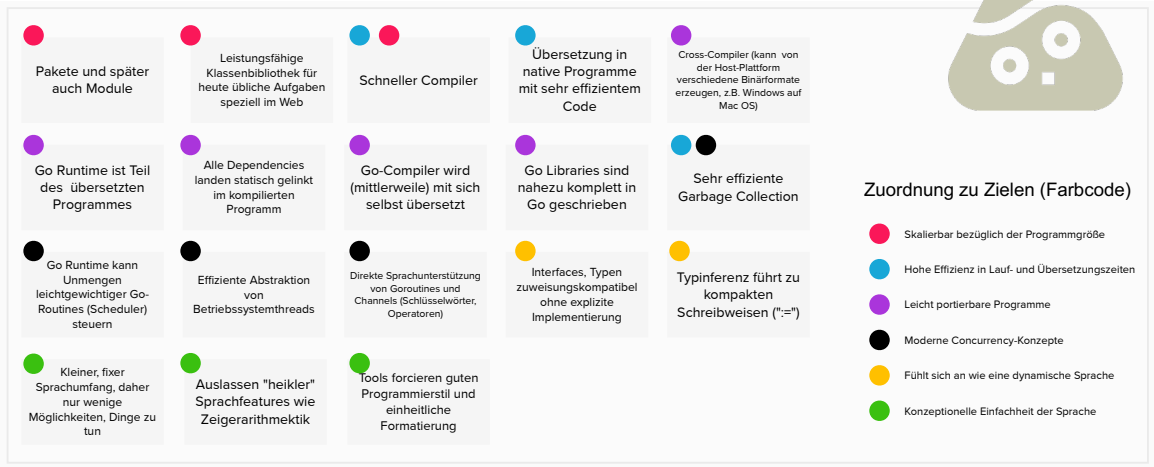
Wichtige Ziele der Programmiersprache Go

- Skalierbar bezüglich der Programmgröße
- Hohe Effizienz in Lauf- und Übersetzungszeiten
- Leicht portierbare Programme
- Moderne Concurrency-Konzepte
- Fühlt sich an wie eine dynamische Sprache
- Konzeptionelle Einfachheit der Sprache

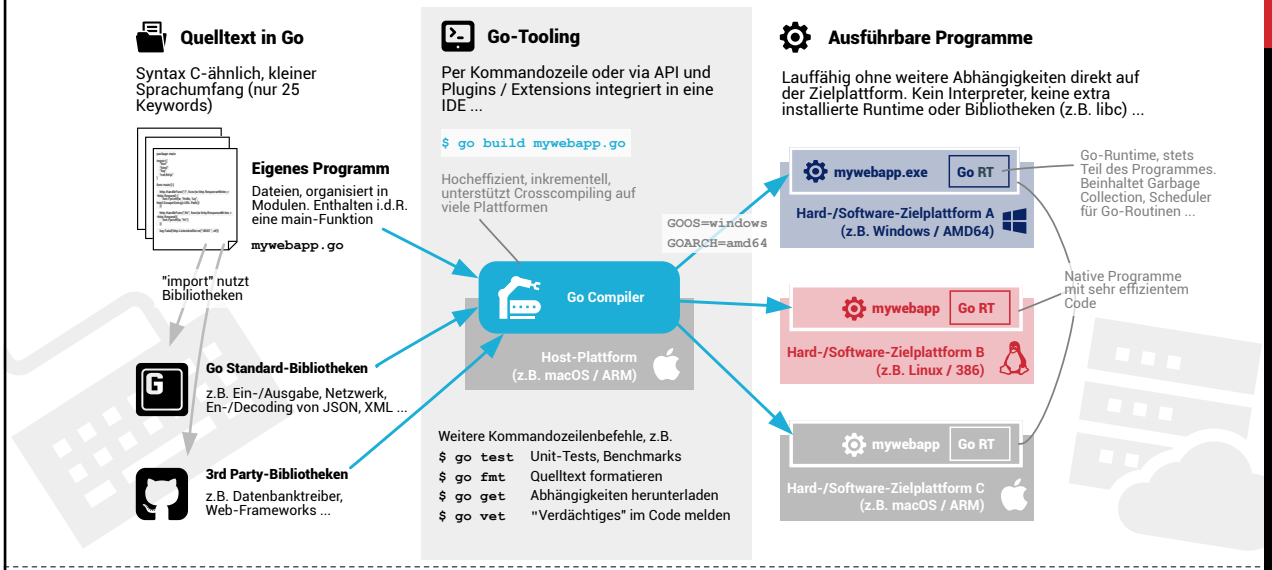
Wichtige Ziele der Programmiersprache Go

Ziel	Beschreibung (und zugehöriges Software-Qualitätsmerkmal)
Skalierbar bezüglich der Programmgröße	Mit Go lassen sich Programme jeder Größe mit angemessenem Aufwand realisieren. Von kleinen Kommandozeilenwerkzeugen bis hin zu sehr großen Systemen.
Hohe Effizienz in Lauf- und Übersetzungszeiten	Go unterstützt die Entwicklung hochperformanter Anwendungen und Dienste für moderne Serverplattformen. Die Programme sind flink übersetzt, ihre Effizienz ist voraussehbar gut. Auch systemnahe Software wie Netzwerkserver oder Datenbanken sind gut realisierbar.
Leicht portierbare Programme	In Go verfasste Programme sind leicht auf andere Hardwareplattformen und Betriebssysteme übertragbar. Das gilt auch für das Go-Tooling, also etwa den Compiler.
Moderne Concurrency-Konzepte	Mit der Sprache ist es vergleichsweise einfach nebenläufige Programme zu schreiben. Die Ausführung auch kleinteilig paralleler Abläufe erfolgt in Go auf effiziente Art und Weise.
Fühlt sich an wie eine dynamische Sprache	In Go zu programmieren geht trotz statischer Typisierung sehr leicht von der Hand. Die Sprache wählt wo möglich selbst "das Richtige" aus.
Konzeptionelle Einfachheit der Sprache	Go besticht durch eine überschaubare und zugleich schlüssige Auswahl an Sprach-Features. Es verzichtet auf "heikle" Programmierkonstrukte und verbirgt nötige Komplexität geschickt. Go ist als Sprache daher vergleichsweise leicht zu erlernen.

Zentrale Lösungsansätze von Go



Go: Ausgewählte Lösungsansätze auf einen Blick



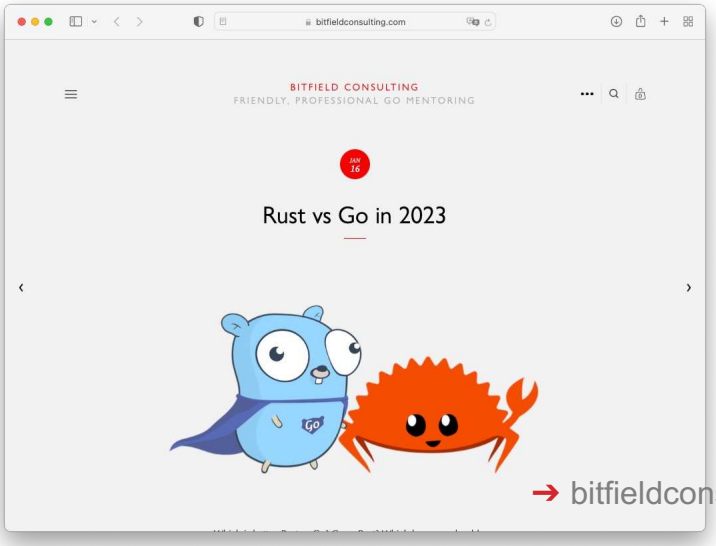
Steckbrief: Go

Software-Gattung	: Programmiersprache
Veröffentlicht	: 2009 (Version 1.0: 2012)
Herkunft / Ursprung	: Google, Open Source
Zielplattform	: Linux (AMD64, PowerPC, …), macOS (x86-64, ARM …), Windows u.v.a.
Programmiersprache(n)	: Go, früher C (Bootstrapping)
Homepage	: ➔ https://go.dev

Vermächtnis (Warum bemerkenswert?)

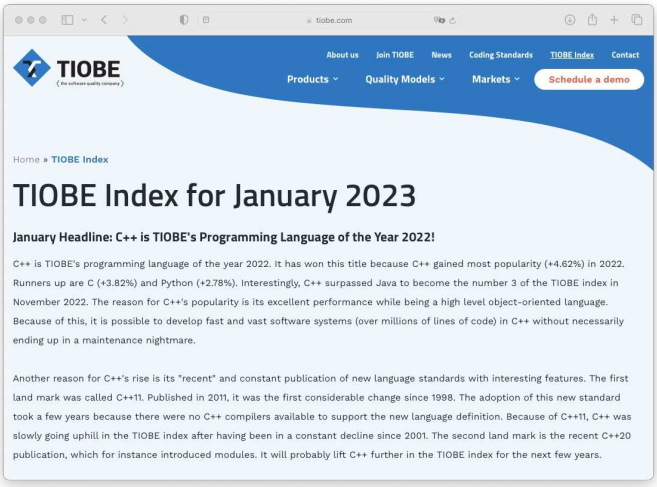
- Moderne Sprache jenseits OO, Nebenläufigkeit an Bord
- Kind der Cloud – die vorherrschende Cloud-Native-Sprache

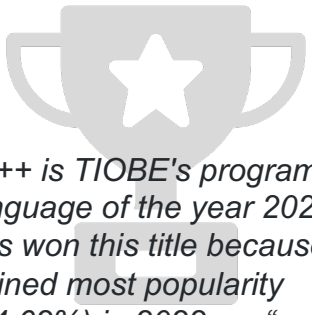
Ausgewogener Blog-Beitrag: Go vs. Rust



➔ bitfieldconsulting.com/golang/rust-vs-go

TIOBE Programming Language of the Year 2022





"C++ is TIOBE's programming language of the year 2022. It has won this title because C++ gained most popularity (+4.62%) in 2022. … "

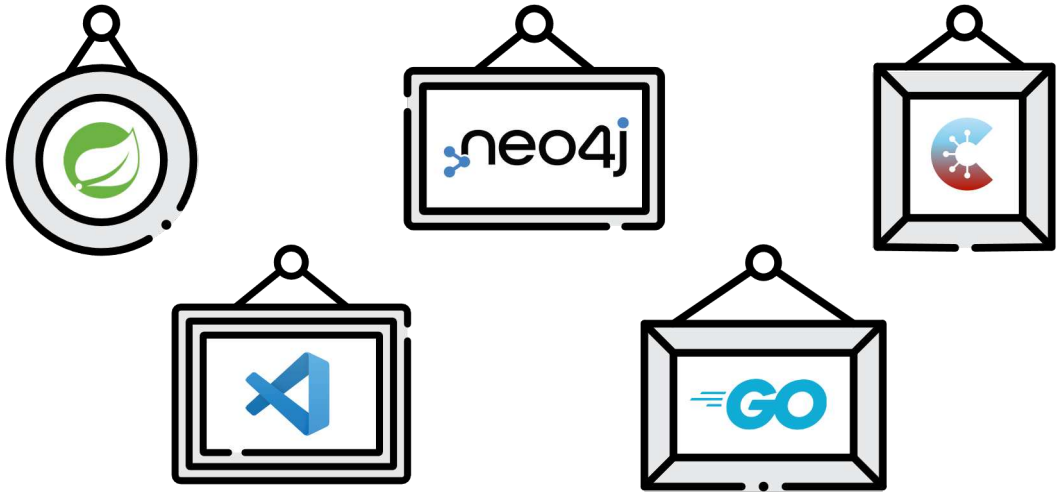
➔ tiobe.com/tiobe-index/

Agenda

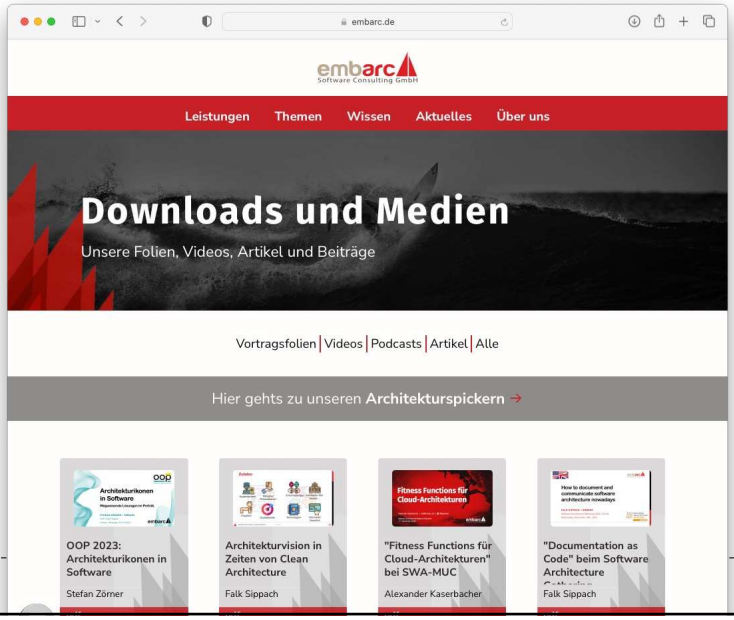
6

0 Architekturikonen in Software
1 Porträt #1: Spring
2 Porträt #2: Corona-Warn-App
3 Porträt #3: Neo4j
4 Porträt #4: Visual Studio Code
5 Porträt #5: Go
6 **Zusammenfassung und Ausblick**

Kleine Galerie der gezeigten Softwarelösungen



Folien als PDF zum Download



➔ embarc.de/download/

71

Artikel-Reihe in IT-Spektrum

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

Zeitversetzt auch online im embarc-Blog

Architektur-Porträt: Die Programmiersprache Go

Zielsetzung, zentrale Ideen und Konzepte

Architektur-Porträt: Corona-Warn-App

Architekturziele und zentrale Lösungsansätze

Architektur-Porträt: Das Spring Framework

Architekturziele und zentrale Lösungsansätze

Architektur-Porträt: Visual Studio Code

Architekturziele und zentrale Lösungsansätze

➔ embarc.de/architektur-portraets/

Unter 4 Augen – Austausch in Zoom

- Du möchtest Dich zu individuellen Fragen, Wünschen oder Ideen im Rahmen der OOP mit unseren Sprechern direkt austauschen?
- Nutze unsere kleinen Sprech(viertel)stunden in den Pausen der Konferenz.
- Deinen Termin suchst Du Dir gern aus - unsere Kollegen freuen sich aufs Treffen in kleiner Zweierrunde!

➔ embarc.de/oop-2023/#unter-vier-augen

Vielen Dank.

Ich freue mich auf Eure Fragen!

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

embarc

oop DIGITAL
software meets business

Vielen Dank.

Ich freue mich auf Eure Fragen!

Folien-Download
embarc.de/download/

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

embarc

oop DIGITAL
software meets business