10 Jahre Vorgehensmuster

Architektur & Agilität in der Praxis

Stefan Toth

Agile Tour Vienna 2024 – 12. September

embarc

Stefan Toth

CEO, Berater für Agilität
Softwarearchitektur

Stefan.Toth@embarc.de

@st_toth

linkedin.com/in/sto-embarc

www.embarc.de

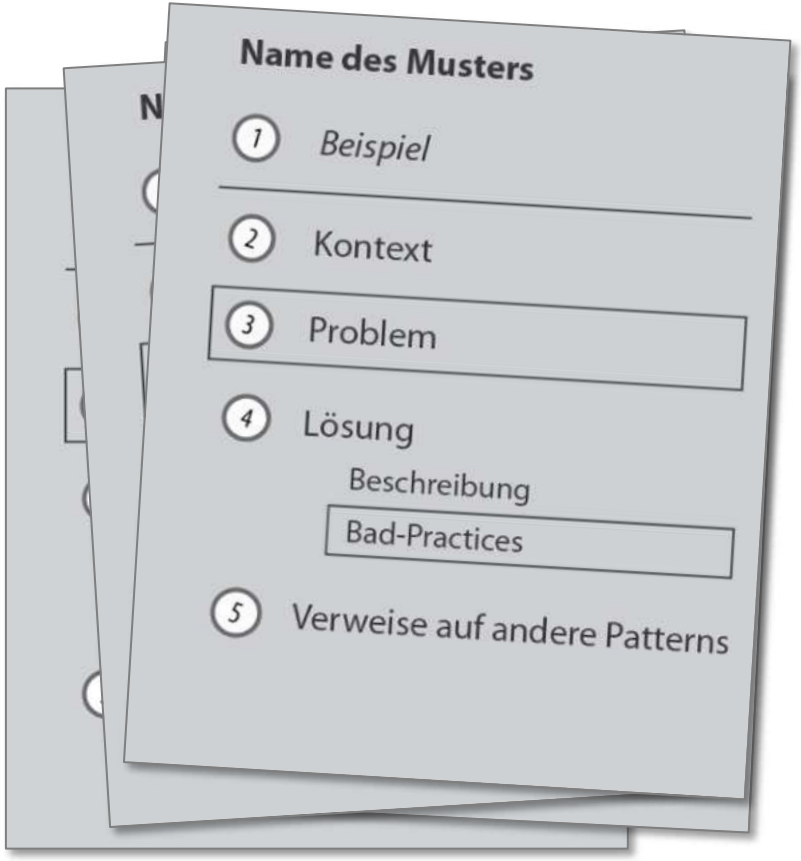
00.

10 Jahre Vorgehensmuster

...

2012 2013 **2014** 2015 2016

Was sind Vorgehensmuster?

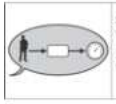


29 Vorgehensmuster

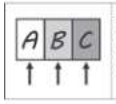
Die Basis für Architekturarbeit

Initialer Anforderungs-Workshop
Wie können Architekturanforderungen effektiv erhoben und kommuniziert werden?
S. 48

Anforderungspflege-Workshops
Wie kann auf Basis einer Anforderungsliste mit architekturrelevanten Inhalten ein stetiger Fluss iterativ verarbeitbarer Aufgaben gewährleistet werden?
S. 53



Szenarien als Architekturanforderungen
Wie drückt man Qualitätsanforderungen aus, um (1) Architekturarbeit sinnvoll zu leiten und (2) Stakeholder-gerecht zu kommunizieren?
S. 57



Szenarien kategorisieren
Wie können Szenarien in iterativen und/oder agilen Prozessen abgearbeitet werden, ohne zu verzögern oder zu behindern?
S. 62

Technische Schulden als Architekturanforderungen *
Wie werden architektonische Probleme und Versäumnisse effizient, transparent und in die Architekturentwicklung integriert behandelt?
S. 66

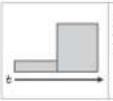
Architekturarbeit im Backlog
Wie kann Architekturarbeit (1) iterativ, (2) stetig priorisiert und (3) mit funktionalen Aufgaben verwoben erledigt werden?
S. 74

Architekturarbeit auf Kanban
Wie kann die Architekturarbeit von der Idee bis zur Auslieferung optimiert werden, so dass ein mit Umsetzungsaufgaben verwobener, stetiger und sichtbarer Fluss von Aufgaben entsteht?
S. 77

Richtig entscheiden

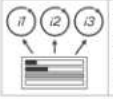
Architekturarbeit vom Rest trennen *
Wie lassen sich jene Aufgaben identifizieren, die (1) Umsicht bei der Entscheidung, (2) evtl. tiefes Architektur- oder Technologieverständnis und (3) breite Kommunikation benötigen?
S. 85

Der letzte vernünftige Moment *
Wann sollte eine architekturelle Fragestellung idealerweise entschieden werden, um (1) das Risiko einer Fehlentscheidung zu minimieren und (2) eine optimale Entscheidung zu treffen?
S. 90



Gerade genug Architektur vorweg
Wie viel Architekturarbeit muss vor dem Einstieg in die Realisierung geleistet sein?
S. 95

Architekturentscheidungen treffen
Welche Tätigkeiten sind nötig, um Architektur-entscheidungen effektiv zu treffen, und wie werden sie zeitlich eingeordnet und wahr-genommen?
S. 101



Release-Planung mit Architekturfragen *
Wie werden Abhängigkeiten, Risiken und die Dringlichkeit von architekturellen Frage-stellungen geplant berücksichtigt, ohne den Prozess unnötig aufzublähen?
S. 110

Risiken aktiv behandeln
Wie sollten Unsicherheiten, die architektur-relevante Auswirkungen haben, gefunden und behandelt werden?
S. 116

Im Prinzip entscheiden
Wie können mehrere Projektmitglieder (Architektur-)entscheidungen bearbeiten, ohne die Konsistenz und Integrität der Software entscheidend zu verringern?
S. 123

Ad-hoc Architekturtreffen
Wie können architektonische Heraus-forderungen oder Unsicherheiten, die während der Umsetzung auftauchen, schnell und trotzdem solide behandelt werden?
S. 127

Zusammenarbeit und Interaktion

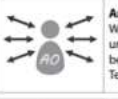
Informativer Arbeitsplatz *
Wie können wichtige Architekturinformationen möglichst breit gestreut werden, um (1) Kontext für die Entwicklung zu geben und (2) für eine gemeinsame Entscheidungsbasis zu sorgen?
S. 135

Gemeinsam entscheiden *
Wie kann eine Entscheidung effektiv in einer Gruppe getroffen werden, wenn (1) ein Entscheider delegiert oder (2) die Gruppe selbst für die Entscheidung verantwortlich ist.
S. 140

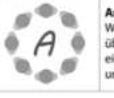
Analog modellieren
Wie kann die Zusammenarbeit auf konzeptioneller Ebene unterstützt werden, um Kreativität, Spontanität und eine zielgerichtete, kollektive Problemlösung zu fördern?
S. 146

Stakeholder involvieren
Wie können Anforderungen und Erwartungen an die Softwarearchitektur effektiv abgeholt und eingeordnet werden, um stetig informierte Architekturarbeit zu leisten?
S. 152

Wiederkehrende Reflexion *
Wie kann nach einer Serie von Entscheidungen (1) Konsistenz und Integrität sichergestellt, (2) das Big Picture im Auge behalten und (3) die Kommunikationslast gesenkt werden?
S. 159



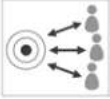
Architecture Owner *
Wie kann Architekturarbeit effektiv, koordiniert und gut erledigt werden, wenn Rahmen-bedingungen keine völlig selbst organisierten Teams zulassen?
S. 166



Architekturcommunities *
Wie können Mitarbeiter eines Projekts (1) effektiv über Architekturthemen nachdenken, (2) die eigenen Fähigkeiten dahingehend ausbauen und (3) ein gemeinsames Bild entwickeln?
S. 172

Abgleich mit der Realität

Frühes Zeigen
Wie kann früh und in möglichst direkter Zusammenarbeit mit dem Kunden überprüft werden, ob sich die Softwarearchitektur ent-sprechend der Ziele und Wünsche entwickelt?
S. 181



Realitätscheck für Architekturziele
Wie können Probleme bei der Erreichung von Architekturzielen frühzeitig erkannt werden?
S. 186

Qualitative Eigenschaften testen *
Wie können Ziele, die die *äußere* Qualität des entwickelten Systems betreffen, objektiv geprüft werden und negative Seiteneffekte späterer Entwicklungstätigkeiten aufgedeckt werden?
S. 191

Qualitätsindikatoren nutzen
Wie können Ziele, die die *innere* Qualität des entwickelten Systems betreffen, objektiv geprüft werden und negative Seiteneffekte späterer Entwicklungstätigkeiten aufgedeckt werden?
S. 201

Code und Architektur verbinden *
Wie können Architektur und Code verbunden werden, so dass (1) die Facharchitektur nicht verwässert, (2) Konzeptfehler klar werden und (3) Architekturprüfungen im Code gültig bleiben?
S. 212

Kontinuierlich integrieren und ausliefern *
Wie können Ergebnisse von Metriken, Prüfungen oder Tests schnell zurück zum Entwickler fließen, um (1) direktes Feedback zu ermöglichen und (2) die Qualität stetig hoch zu halten?
S. 220

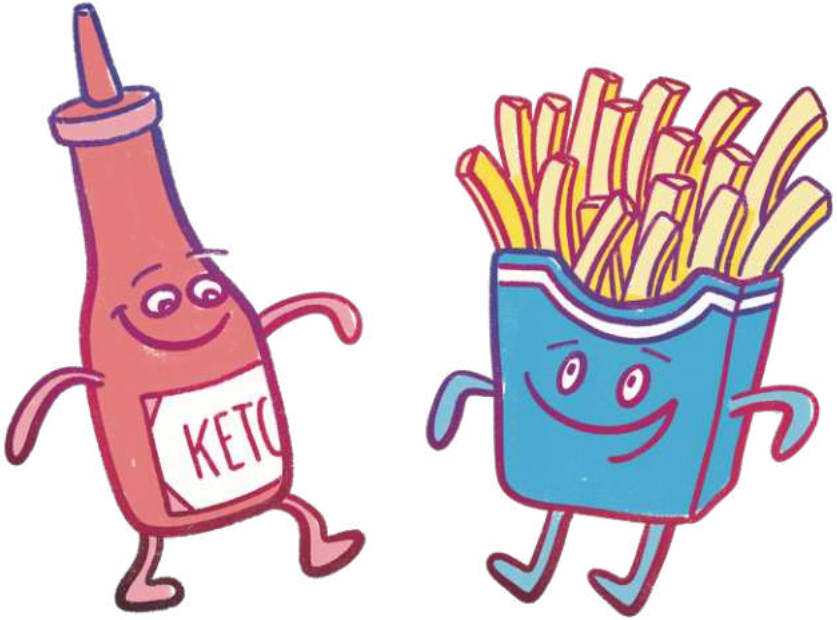
Problemen auf den Grund gehen
Wie können für die Architektur erkannte Probleme oder Risiken analysiert werden, um Verschwendung und Ineffektivität bei deren Beseitigung zu vermeiden?
S. 225

* die Problembeschreibung ist hier aus Platzgründen leicht gekürzt wiedergegeben

01.

Architektur und Agilität

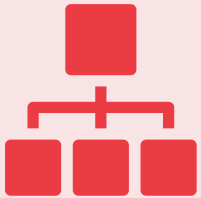
Welches Problem lösen wir überhaupt?



Architektur-Themen

Zerlegung

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

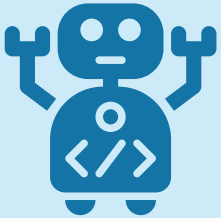


Zielumgebung

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

Technologie-Stack

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

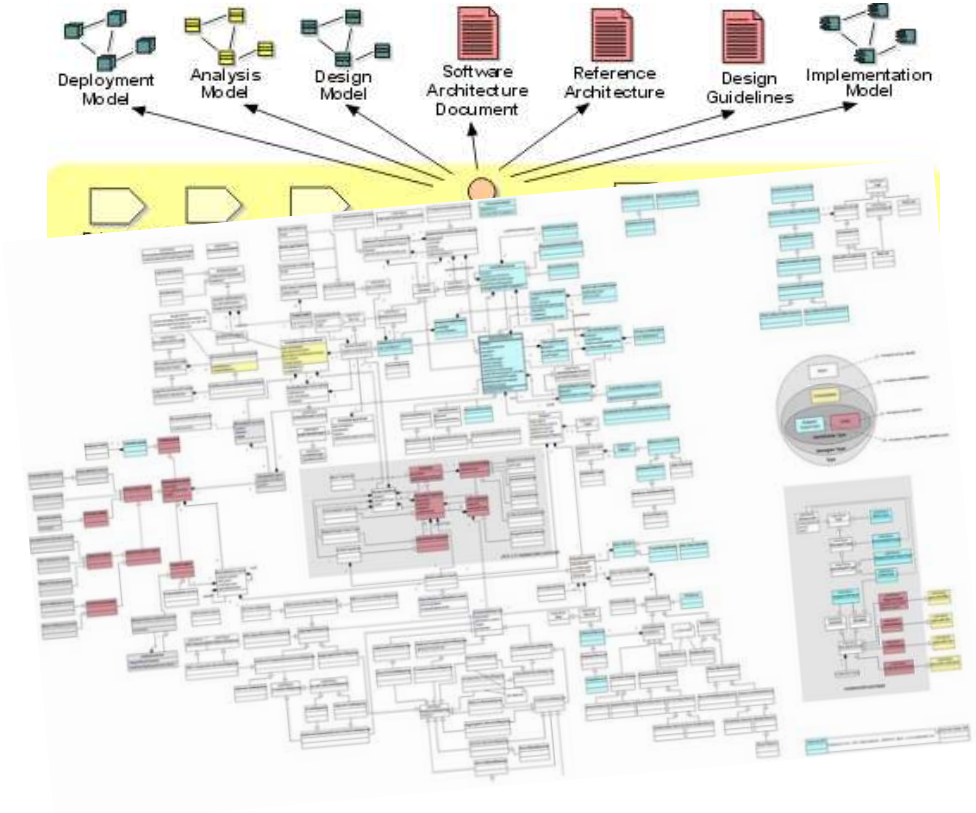


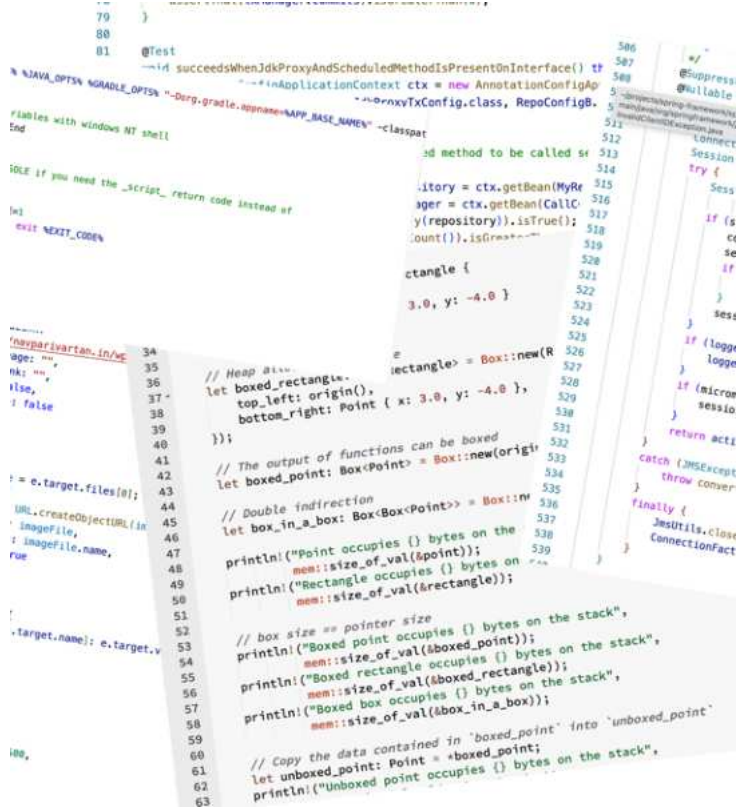
Vorgehen

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

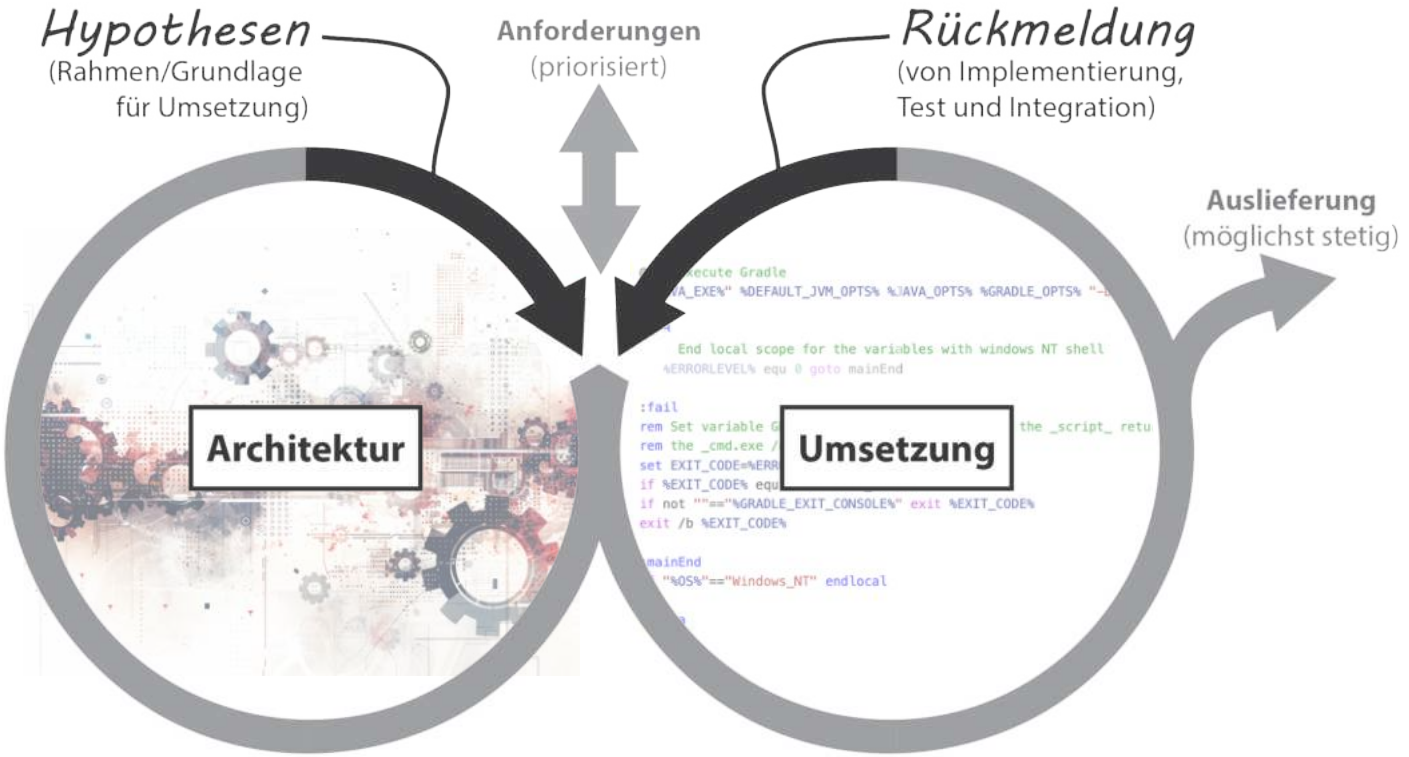

„Accidential Architecture"

Architektur als eigene Disziplin



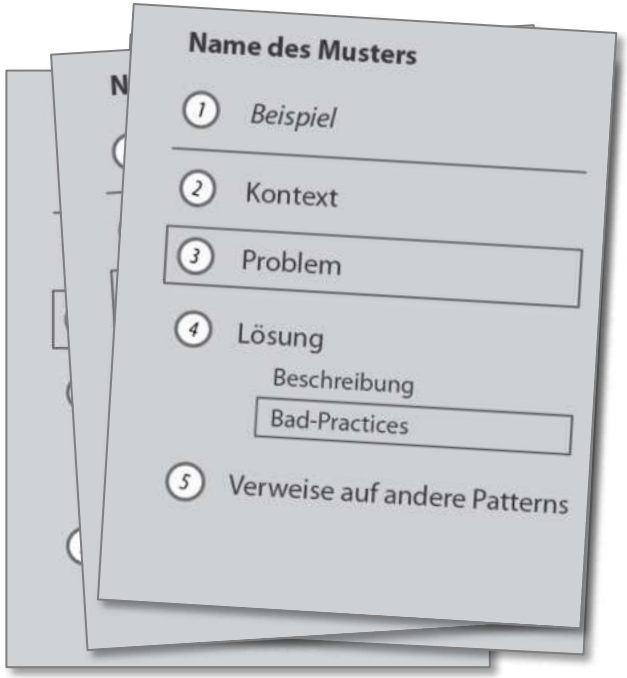


Architektur als Entwicklungsmodus



Was macht „agile Architektur“ aus?

- **Iterativ**:
  Nicht alles auf einmal, kein „BUFD“
- **Feedbackorientiert:**
  Inkl. Zielen, Tests und Flexibilität
- **Mit der Entwicklung verzahnt**:
  Nutzt Backlogs und Erkenntnisse
- **Gemeinschaftlich**:
  Keine abgehobenen Tools, Transparenz
- …



02.

Vorgehensmuster

10 Jahre später...

2022 2023 **2024** 2025 2026

Iterationen

2014

2015

2019

2025

Vorgehensmuster

Die Basis für Architekturarbeit

Anforderungspflege-Workshops
Wie kann auf Basis einer Anforderungsliste mit [illegible] Inhalten ein stetiger Fluss [illegible] bearbeitbarer Aufgaben gewährleistet werden?
S. 48 S. 53

Szenarien als Architekturanforderungen
Wie drückt man Qualitätsanforderungen aus, um (1) Architekturarbeit sinnvoll zu leiten und (2) Stakeholder-gerecht zu kommunizieren?
S. 57

Szenarien kategorisieren
Wie können Szenarien in iterativen und/oder agilen Prozessen abgearbeitet werden, ohne zu verzögern oder zu behindern?
S. 62

Technische Schulden als Architekturanforderungen *
Wie werden architektonische Probleme und Versäumnisse effizient, transparent und in die Architekturentwicklung integriert behandelt?
S. 66

Architekturarbeit im Backlog
Wie kann Architekturarbeit (1) iterativ, (2) stetig priorisiert und (3) mit funktionalen Aufgaben verwoben erledigt werden?
S. 74

Architekturarbeit auf Kanban
Wie kann die Architekturarbeit von der Idee bis zur Auslieferung optimiert werden, so dass ein mit Umsetzungsaufgaben verwobener, stetiger und sichtbarer Fluss von Aufgaben entsteht?
S. 77

Richtig entscheiden

Architekturarbeit vom Rest trennen *
Wie lassen sich jene Aufgaben identifizieren, die (1) Umsicht bei der Entscheidung, (2) evtl. tiefes Architektur- oder Technologieverständnis und (3) breite Kommunikation benötigen?
S. 85

Der letzte vernünftige Moment *
Wann sollte eine architekturelle Fragestellung idealerweise entschieden werden, um (1) das Risiko einer Fehlentscheidung zu minimieren und (2) eine optimale Entscheidung zu treffen?
S. 90

Gerade ge[illegible]chitektur vorweg
Wie viel [illegible]chitektu[illegible]rbeit muss vor dem Einstieg in die Re[illegible]lisiert sein?
S. 95

Release-Planung mit Architekturfragen
Wie werden Abhängigkeiten, Risiken und die Dringlichkeit von architekturellen Fragestellungen geplant berücksichtigt, ohne den Prozess unnötig aufzublähen?
S. 110

Risiken aktiv behandeln
[illegible] Unsicherheiten, [illegible]tektur-relevante [illegible] haben, gefunden und [illegible] werden?
S. 116

Im Prinzip entscheiden
Wie können mehrere Projektmitglieder (Architektur-)entscheidungen bearbeiten, ohne die Konsistenz und Integrität der Software entscheidend zu verringern?
S. 127

Zusammenarbeit und Interaktion

Informativer Arbeitsplatz *
Wie können [illegible]chitekturinformationen möglichst b[illegible]estreut werden, um (1) Kontext für die Entw[illegible]ng zu geben und (2) für eine gemeinsame Entscheidungsbasis zu sorgen?
S. 135

Gemeinsam entscheiden *
Wie kann eine Entscheidung effektiv in einer Gruppe getroffen werden, wenn (1) ein Entscheider delegiert oder (2) die Gruppe selbst für die Entscheidung verantwortlich ist.
S. 140

Analog modellieren
[illegible] die Zusammenarbeit auf konzeptionelle[illegible]erstützt werden, um Kreativität, [illegible]manität [illegible] zielgerichtete, [illegible]ktive Problemlösung zu förd[illegible]
S. 146

Pfad des geringsten Widerstands

Architecture Owner *
Wie kann Architekturarbeit effektiv, koordiniert und gut erledigt werden, wenn Rahmenbedingungen keine völlig selbst organisierten Teams zulassen?
S. 166

Architekturcommunities *
Wie können Mitarbeiter eines Projekts (1) effektiv über Architekturthemen nachdenken, (2) die eigenen Fähigkeiten dahingehend ausbauen und (3) ein gemeinsames Bild entwickeln?
S. 159

Abgleich mit der Realität

Realitätscheck für Architekturziele
Wie können Probleme bei der Erreichung von Architekturzielen frühzeitig erkannt werden?
S. 181 S. 186

Qualitative Eigenschaften testen *
Wie können Ziele, die die *äußere* Qualität des entwickelten Systems betreffen, objektiv geprüft werden und negative Seiteneffekte späterer Entwicklungstätigkeiten aufgedeckt werden?
S. 191

Qualitätsindikatoren nutzen
Wie können Ziele, die die *innere* Qualität des entwickelten Systems betreffen, objektiv geprüft werden und negative Seiteneffekte späterer Entwicklungstätigkeiten aufgedeckt werden?
S. 201

Code und Architektur verbinden *
Wie können Architektur und Code verbunden werden, so dass (1) die Facharchitektur nicht verwässert, (2) Konzeptfehler klar werden und (3) Architekturprüfungen im Code gültig bleiben?
S. 212

Kontinuierlich integrieren und ausliefern *
Wie können Ergeb[illegible]on Metriken, Prüfungen oder Tests schn[illegible]zurück[illegible]um Entwickler fließen, um (1) direktes [illegible]back zu ermöglichen und (2) die Qualit[illegible]och zu halten?
S. 220

Problemen auf den Grund gehen
Wie können für die Architektur erkannte Probleme oder Risiken analysiert werden, um Verschwendung und Ineffektivität bei deren Beseitigung zu vermeiden?
S. 225

* die Problembeschreibung ist hier aus Platzgründen leicht gekürzt wiedergegeben

Top-5-Challenger

Walking Skeleton

Architektur-Radar

Gesundheits-Check

Innovationspotentiale

2-speed Architecture

Architektur-Kata

03.

Erfolgreiche Muster

Die Top-Muster der ursprünglichen 29...

3+

- Letzter vernünftiger Moment
- Konsensentscheidungen

#2

Qualitative Tests

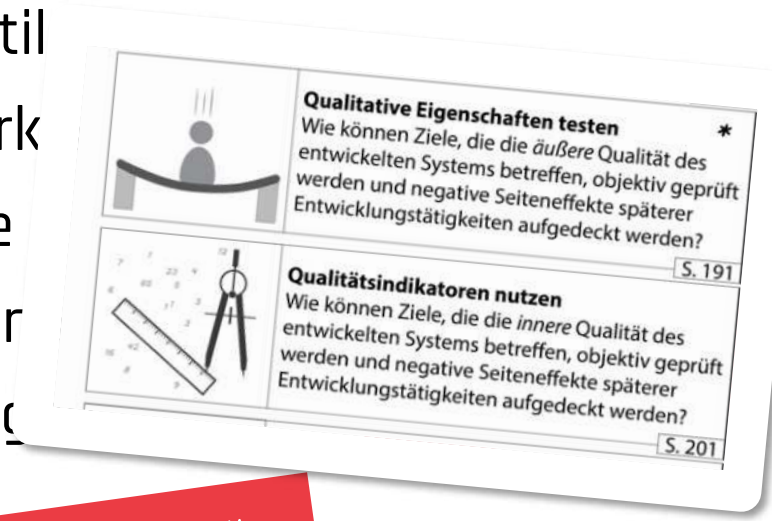
Elemente von Architekturarbeit

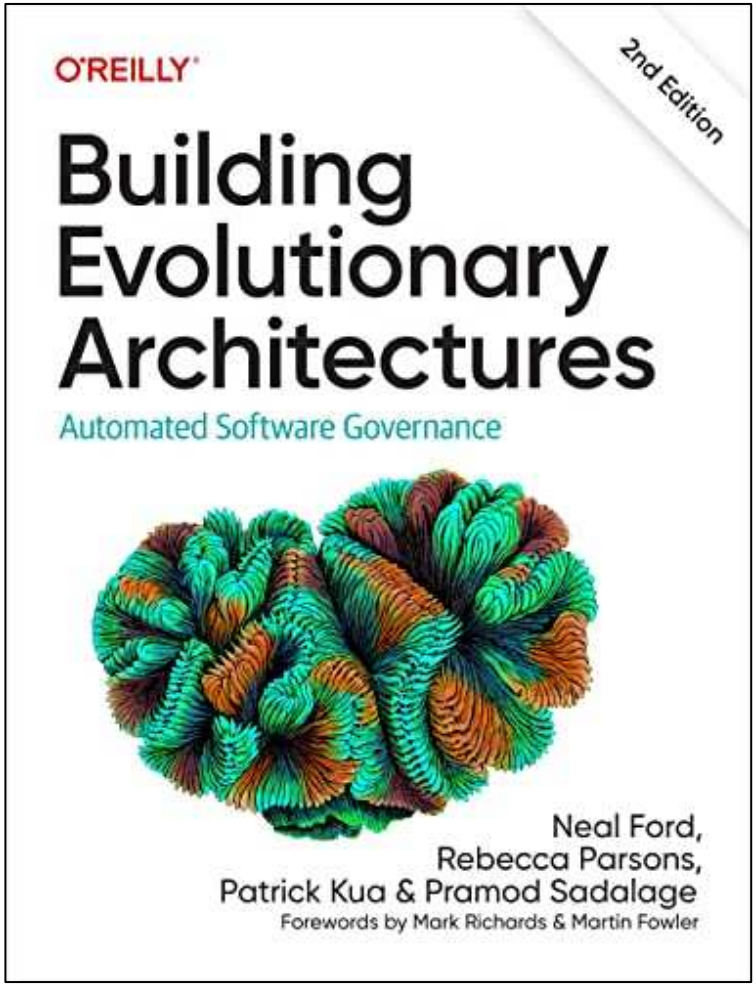
- Effizienz
- Sicherheit
- Zuverlässigkeit
- Skalierbarkeit
- Wartbarkeit
- Portierbarkeit
- ...

„Qualitätsmerkmale“

- Strukturierung
- Muster, Stil
- Framework
- Protokolle
- Basistechn
- Umgebung
- ...

„Entscheidungen“

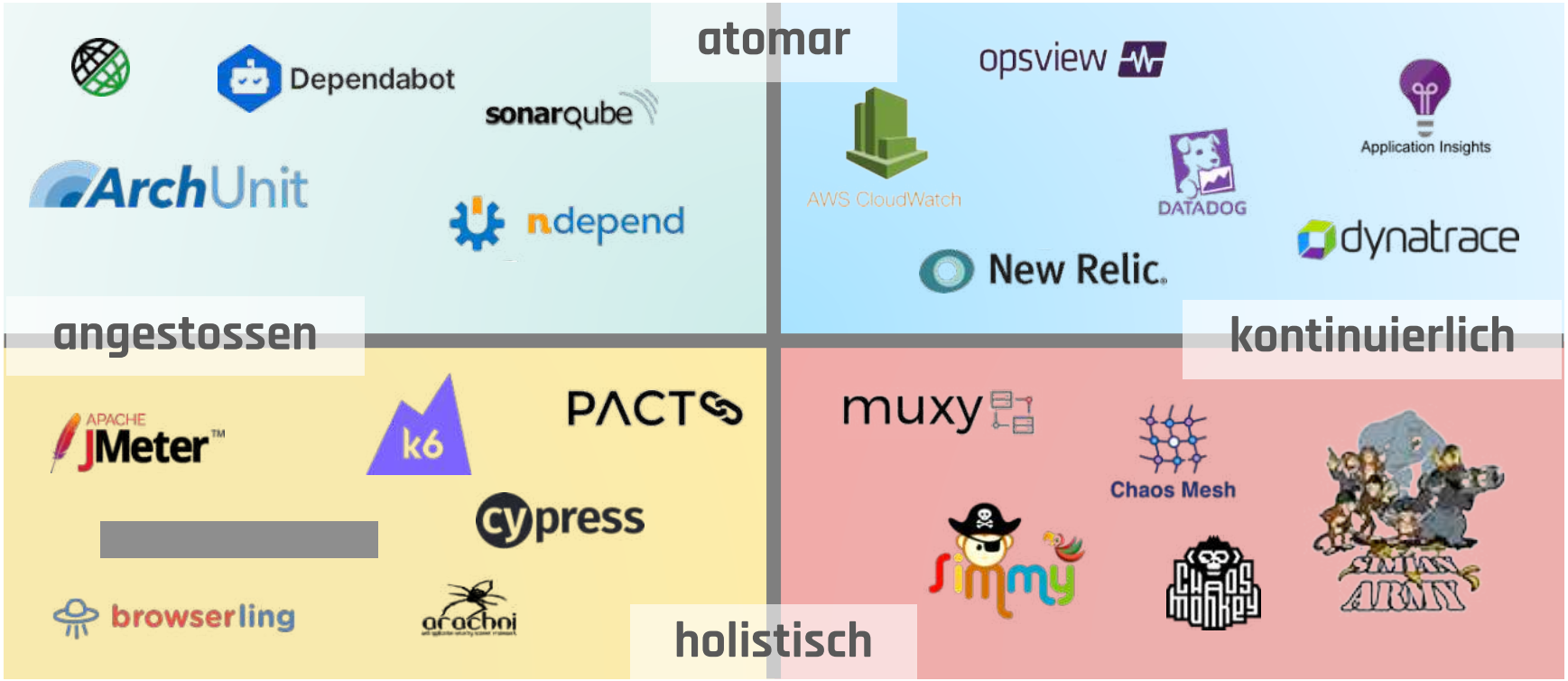


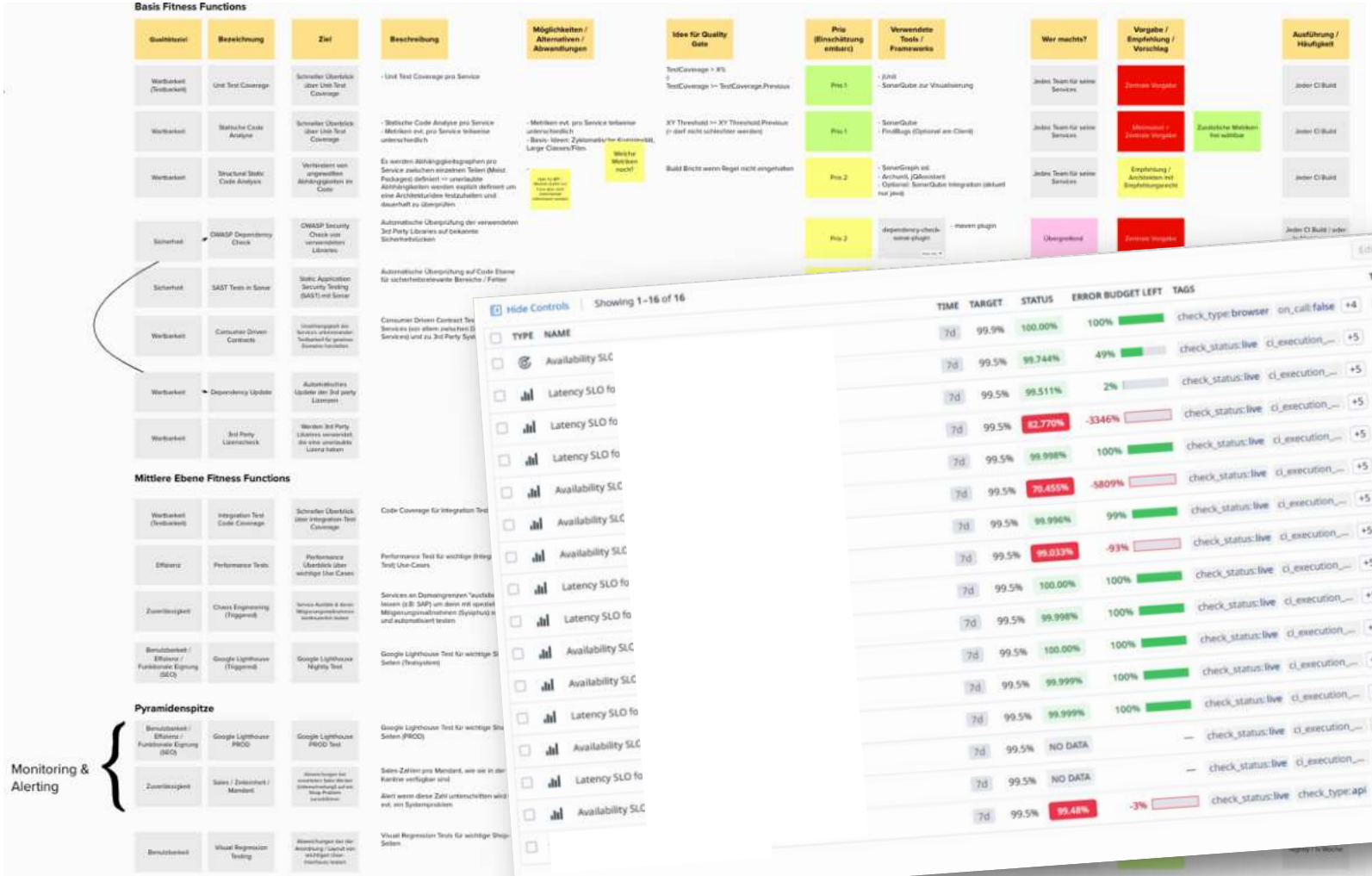


Eine **Fitness Function** misst objektiv, wie gut eine Lösung die an sie gesetzten Ziele erreicht.

- Building Evolutionary Architectures – Ford, Parsons, Kua, Sadalage, O'Reilly 2017

Fitness Function Quadrant

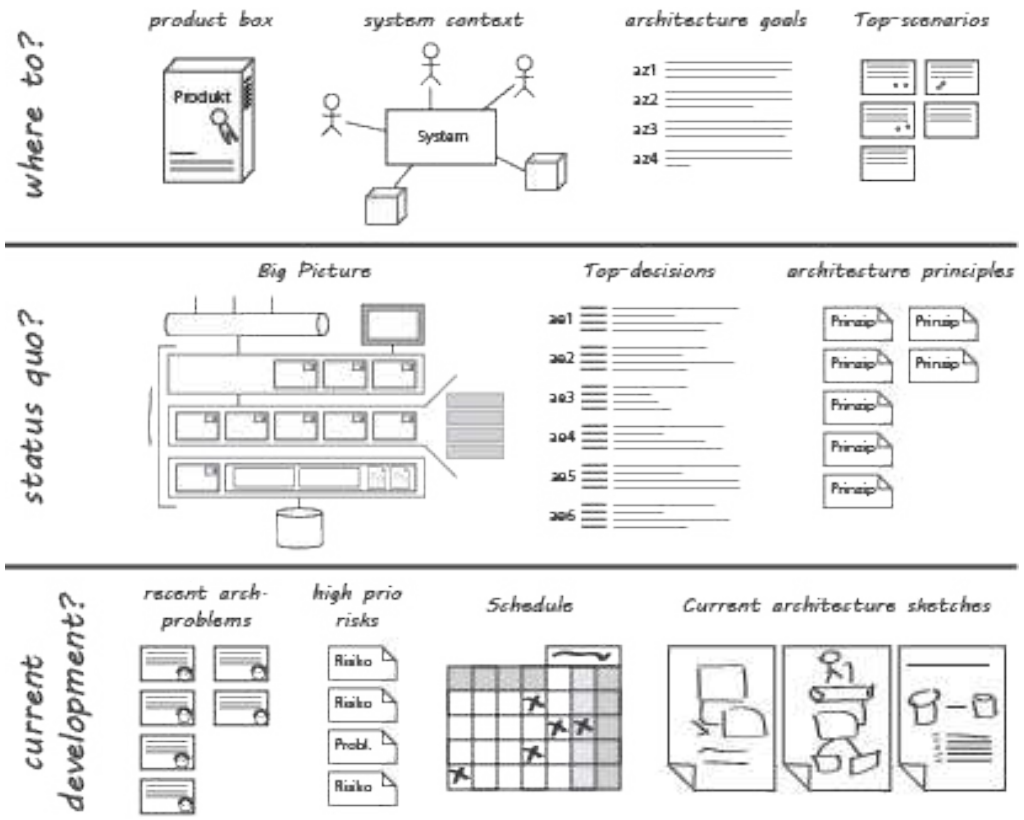



Basis Fitness Functions
Qualitätsziel
Bezeichnung
Ziel
Beschreibung
Möglichkeiten / Alternativen / Abwandlungen
Idee für Quality Gate
Prio (Einschätzung embarc)
Verwendete Tools / Frameworks
Wer machts?
Vorgabe / Empfehlung / Vorschlag
Ausführung / Häufigkeit
Wartbarkeit (Testbarkeit)
Unit Test Coverage
Schneller Überblick über Unit-Test Coverage
- Unit Test Coverage pro Service
TestCoverage > X%
TestCoverage >= TestCoverage.Previous
Prio 1
- JUnit
- SonarQube zur Visualisierung
Jedes Team für seine Services
Zentrale Vorgabe
Jeder CI Build
Wartbarkeit
Statische Code Analyse
Schneller Überblick über Unit-Test Coverage
- Statische Code Analyse pro Service
- Metriken evt. pro Service teilweise unterschiedlich
- Basis- Ideen: Zyklomatische Komplexität, Large Classes/Files
XY Threshold >= XY Threshold.Previous
(= darf nicht schlechter werden)
Prio 1
- SonarQube
- FindBugs (Optional am Client)
Jedes Team für seine Services
Metrikset + Zentrale Vorgabe
Zusätzliche Metriken frei wählbar
Jeder CI Build
Wartbarkeit
Structural Static Code Analysis
Verhindern von ungewollten Abhängigkeiten im Code
Es werden Abhängigkeitsgraphen pro Service zwischen einzelnen Teilen (Meist Packages) definiert => unerlaubte Abhängigkeiten werden explizit definiert um eine Architekturidee festzuhalten und dauerhaft zu überprüfen
Welche Metriken noch?
Build Bricht wenn Regel nicht eingehalten
Prio 2
- SonarGraph od.
- Archunit, (QAssistant
- Optional: SonarQube Integration (aktuell nur java)
Jedes Team für seine Services
Empfehlung / Architekten mit Empfehlungsrecht
Jeder CI Build
Sicherheit
OWASP Dependency Check
OWASP Security Check von verwendeten Libraries
Automatische Überprüfung der verwendeten 3rd Party Libraries auf bekannte Sicherheitslücken
Prio 2
dependency-check-sonar-plugin
- maven plugin
Übergreifend
Zentrale Vorgabe
Jeder CI Build / oder
Sicherheit
SAST Tests in Sonar
Static Application Security Testing (SAST) mit Sonar
Automatische Überprüfung auf Code Ebene für sicherheitsrelevante Bereiche / Fehler
Wartbarkeit
Consumer Driven Contracts
Consumer Driven Contract Tes
Services (von allem zwischen D
Services) und zu 3rd Party Syst
Wartbarkeit
Dependency Update
Automatisches Update der 3rd party Lizenzen
Wartbarkeit
3rd Party Lizenzcheck
Werden 3rd Party Libaries verwendet die eine unerlaubte Lizenz haben
Mittlere Ebene Fitness Functions
Wartbarkeit (Testbarkeit)
Integration Test Code Coverage
Schneller Überblick über Integration-Test Coverage
Code Coverage für Integration Test
Effizienz
Performance Tests
Performance Überblick über wichtige Use Cases
Performance Test für wichtige (Integ
Test) Use-Cases
Zuverlässigkeit
Chaos Engineering (Triggered)
Services an Domaingrenzen "ausfalle
lassen (z.B: SAP) um dann mit spezie
Mitigierungsmaßnahmen (Sysiphus) e
und automatisiert testen
Benutzbarkeit / Effizienz / Funktionale Eignung (SEO)
Google Lighthouse (Triggered)
Google Lighthouse Nightly Test
Google Lighthouse Test für wichtige Si
Seiten (Testsystem)
Pyramidenspitze
Monitoring & Alerting
Benutzbarkeit / Effizienz / Funktionale Eignung (SEO)
Google Lighthouse PROD
Google Lighthouse PROD Test
Google Lighthouse Test für wichtige Sho
Seiten (PROD)
Zuverlässigkeit
Sales / Zeiterfahrt / Mandant
Sales-Zahlen pro Mandant, wie sie in de
Kentnie verfügbar sind
Alert wenn diese Zahl unterschritten wird
evt. ein Systemproblem
Benutzbarkeit
Visual Regression Testing
Abweichungen bei der Anzeigung / Layout von wichtigen User-Interfaces testen
Visual Regression Tests für wichtige Shop-Seiten
Hide Controls
Showing 1–16 of 16
TYPE
NAME
TIME
TARGET
STATUS
ERROR BUDGET LEFT
TAGS
TEAMS
Availability SLO
Latency SLO fo
7d 99.9% 100.00% 100% check_type:browser on_call:false +4 ateam benutzerservice +1
7d 99.5% 99.744% 49% check_status:live ci_execution_... +5
7d 99.5% 99.511% 2% check_status:live ci_execution_... +5
7d 99.5% 82.770% -3346% check_status:live ci_execution_... +5
7d 99.5% 99.998% 100% check_status:live ci_execution_... +5
7d 99.5% 70.455% -5809% check_status:live ci_execution_... +5
7d 99.5% 99.996% 99% check_status:live ci_execution_... +5
7d 99.5% 99.033% -93% check_status:live ci_execution_... +5
7d 99.5% 100.00% 100% check_status:live ci_execution_... +5
7d 99.5% 99.998% 100% check_status:live ci_execution_... +5
7d 99.5% 100.00% 100% check_status:live ci_execution_... +5
7d 99.5% 99.999% 100% check_status:live ci_execution_... +5
7d 99.5% 99.999% 100% check_status:live ci_execution_... +5
7d 99.5% NO DATA — check_status:live ci_execution_... +5
7d 99.5% NO DATA — check_status:live ci_execution_... +5
7d 99.5% 99.48% -3% check_status:live check_type:api +8
Results per

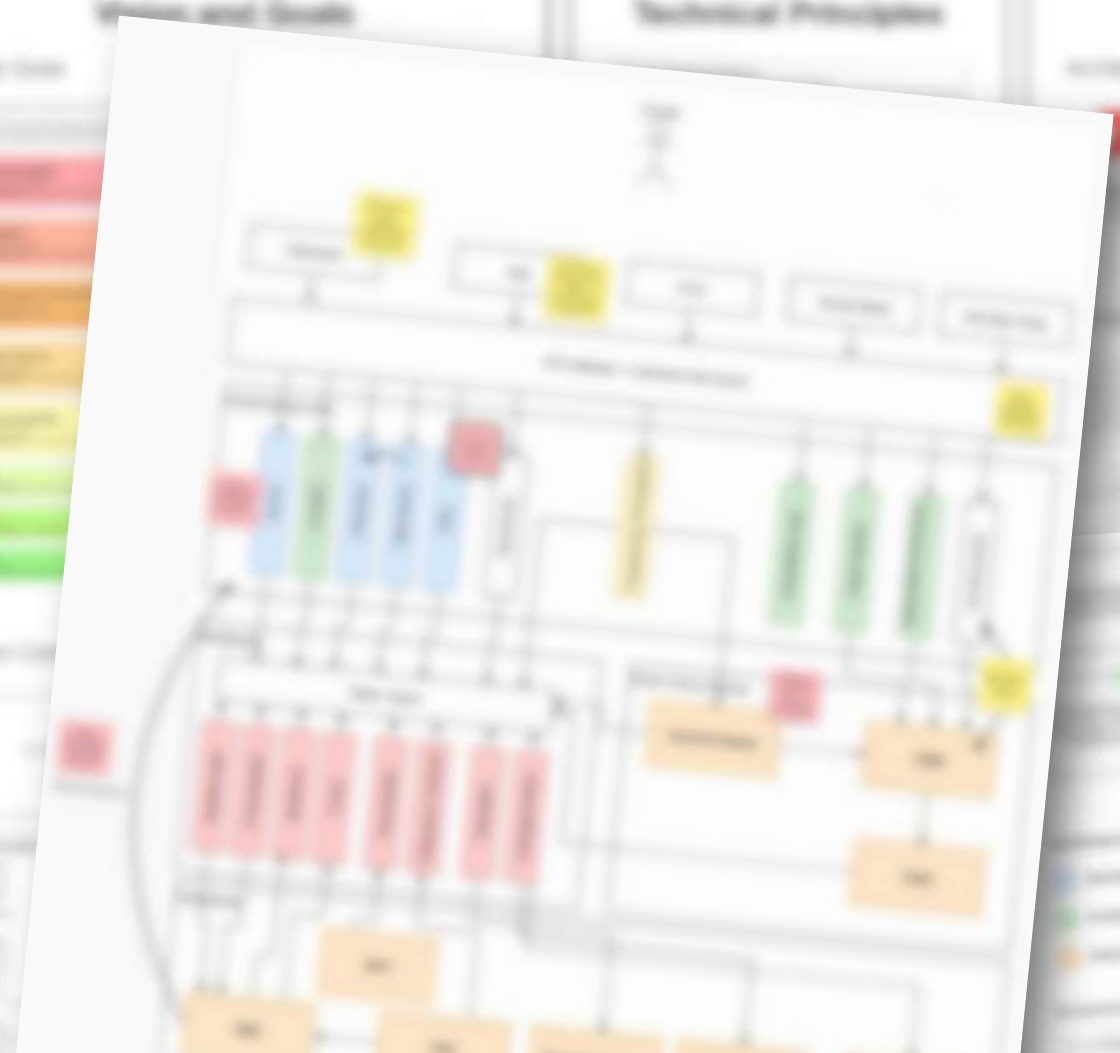
#1

Architekturwand

Vision und Zustand der Architektur zeigen



- Knapper Überblick
- Änderungshäufigkeit: unten mehr
- Physisch oder digital
- Erhöht Transparenz
- Ermöglicht Post-It Sessions



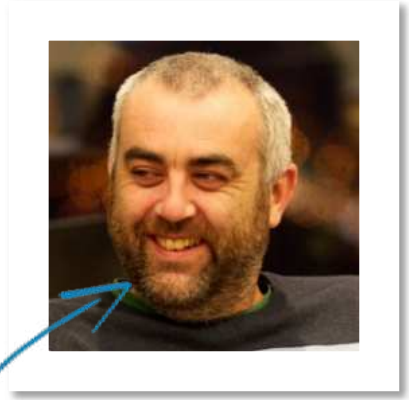
04.

Neue Vorgehensmuster

Die Top-3/4 der hinzugekommenen Praktiken

Microservices...

> *"In short, the microservice architectural style is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API."*



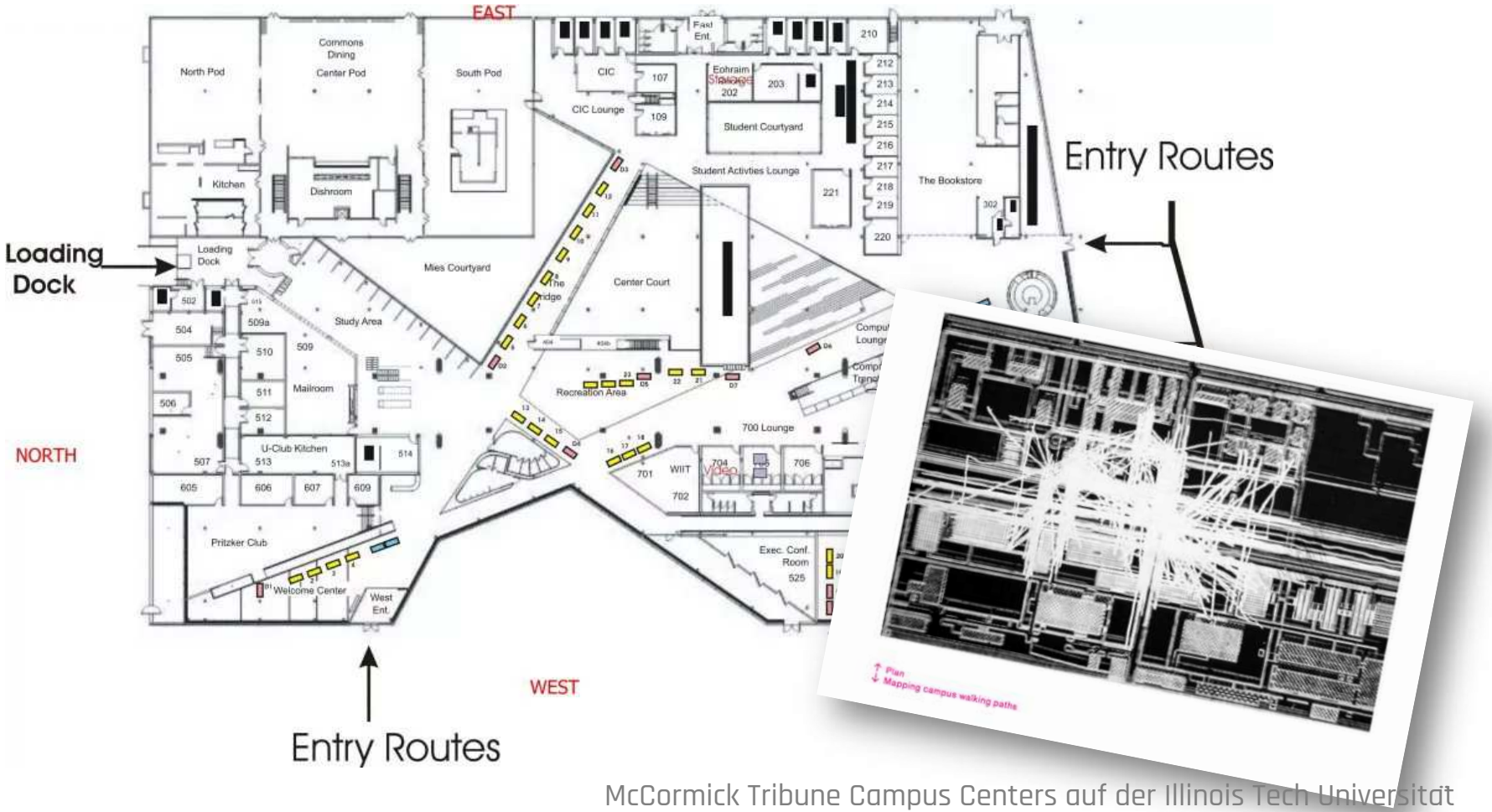
(James Lewis, Martin Fowler)

Einige Netflix Ingineure zitiert…

"[…] we quickly see that a distributed system of any meaningful size becomes too complex for a human [to understand]. There are simply too many parts, changing and innovating too quickly, interacting in too many unplanned and uncoordinated ways for a human to hold those patterns in their head."

"The same is true in other complex systems, including monoliths (usually with many, often unknown, downstream dependencies) that become so large that no single architect can understand the implications of a new feature on the entire application."

Aus "Chaos Engineering" von Casey Rosenthal, Lorin Hochstein, Aaron Blohowiak, Nora Jones, Ali Basiri



McCormick Tribune Campus Centers auf der Illinois Tech Universität

3+

- 2-speed Architecture
- Architektur-Radar

Zalando Tech Radar

2024.06

Datastores

ADOPT

1. Amazon ElastiCache
2. AWS DynamoDB
3. AWS S3
4. Elasticsearch
5. Exasol
6. PostgreSQL

TRIAL

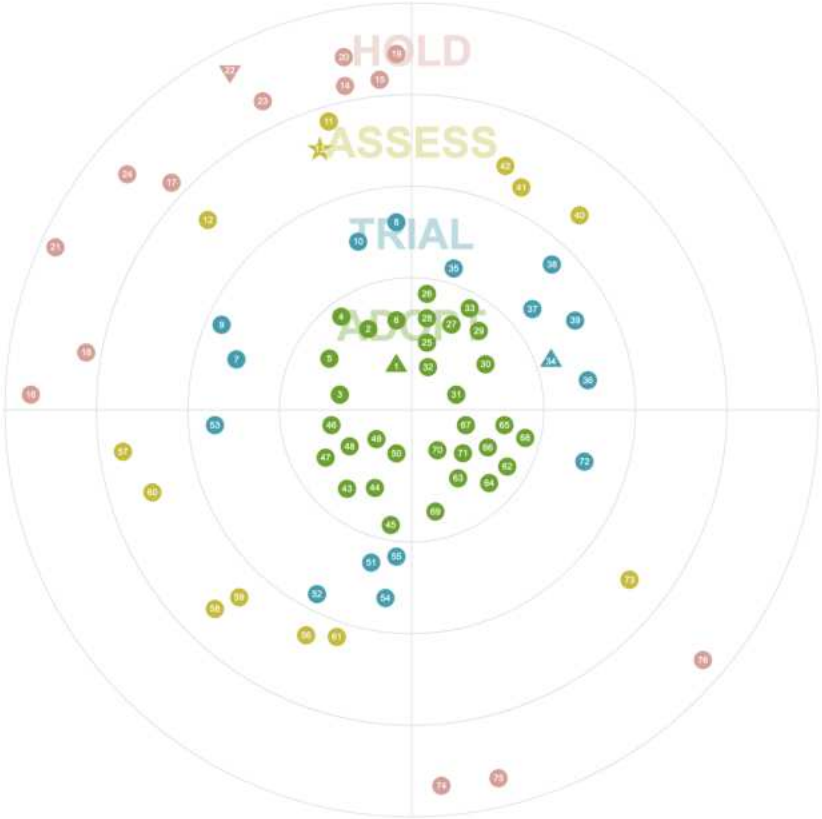
7. Amazon Feature Store
8. Amazon Redshift
9. Druid
10. HDFS

ASSESS

11. Amazon MemoryDB
12. RocksDB
13. Valkey

HOLD

14. Apache Cassandra
15. Consul
16. Hazelcast
17. HBase
18. Memcached
19. MongoDB
20. MySQL
21. Oracle DB
22. Redis
23. Solr
24. ZooKeeper



▲ moved up ▼ moved down ★ new ○ no change

Data Management

ADOPT

25. Airflow
26. AWS EMR
27. AWS Kinesis
28. AWS SNS
29. AWS SQS
30. Databricks
31. Kafka
32. Nakadi
33. Spark

TRIAL

34. AWS Athena
35. dbt
36. Flink
37. Google BigQuery
38. Presto
39. RabbitMQ

ASSESS

40. AWS Glue
41. AWS Lake Formation
42. Streamlit

HOLD

Infrastructure

ADOPT

43. Amazon SageMaker
44. AWS CloudFormation
45. AWS CloudFront
46. Docker
47. Kubernetes
48. OpenTelemetry
49. Skipper
50. ZMON

TRIAL

51. AWS Elemental MediaConvert
52. AWS Lambda
53. AWS Step Functions
54. Open Policy Agent
55. WebAssembly

ASSESS

56. Amazon Bedrock
57. Amazon Pinpoint
58. AWS Service Catalog
59. GraalVM
60. Kotlin Multiplatform
61. Slurm

HOLD

Languages

ADOPT

62. Go
63. GraphQL
64. Java
65. JavaScript
66. Kotlin
67. OpenAPI (Swagger)
68. Python
69. Scala
70. Swift
71. TypeScript

TRIAL

72. Dart

ASSESS

73. R

HOLD

74. Clojure
75. Haskell
76. Rust

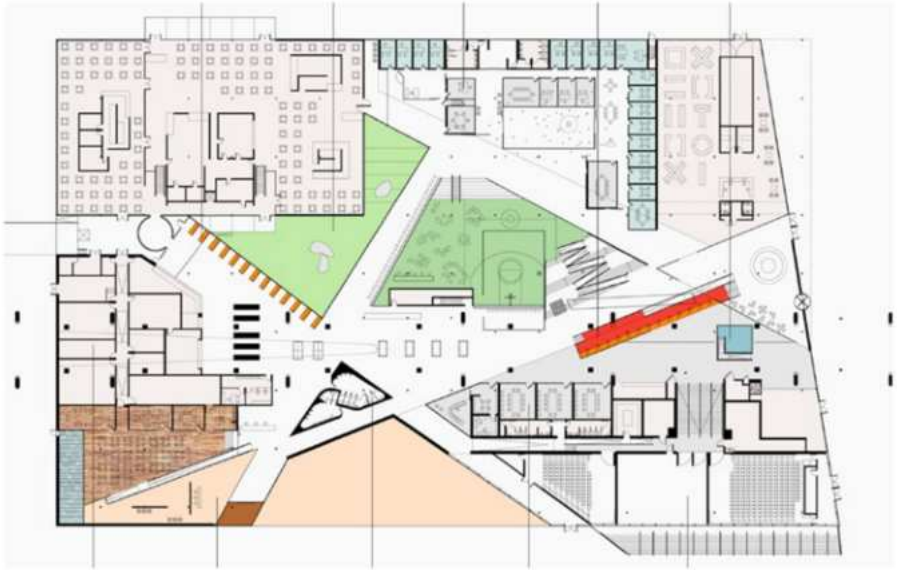
#2

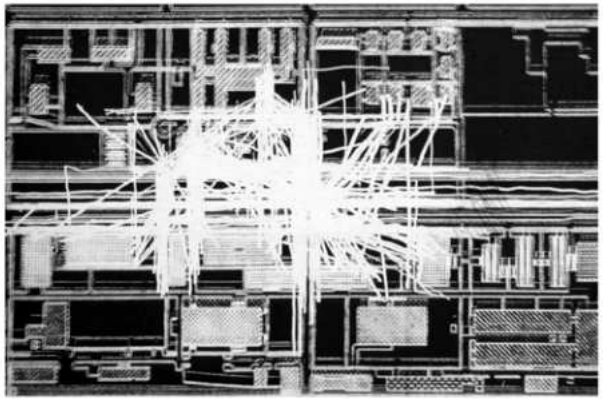
Pfad des geringsten Widerstands

Bottom-Up...

- **Ziele** von Nutzerinnen und Designer sind **homogen**
- Es gibt **Offenheit** aber auch einen **starken Rahmen**
  (Stundenpläne, Gebäudezwecke, ...)

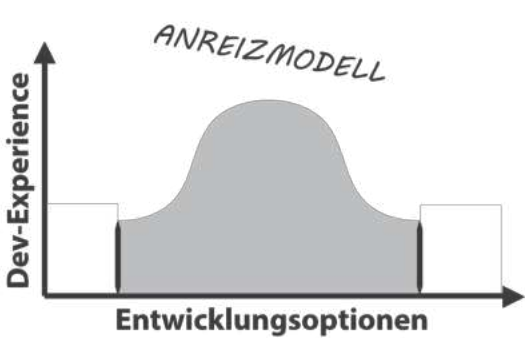
- Was wenn der Kontext anders wäre?

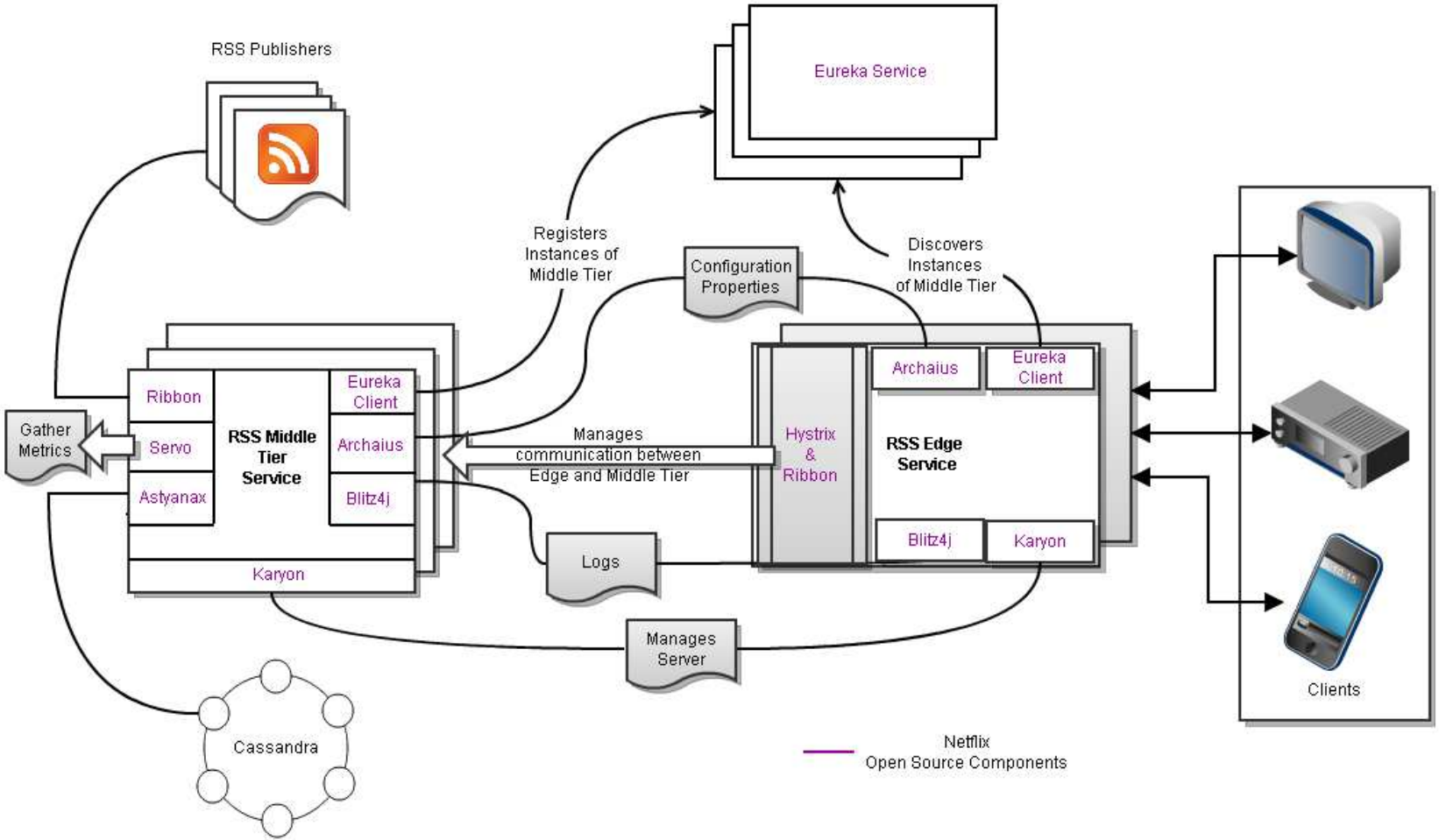


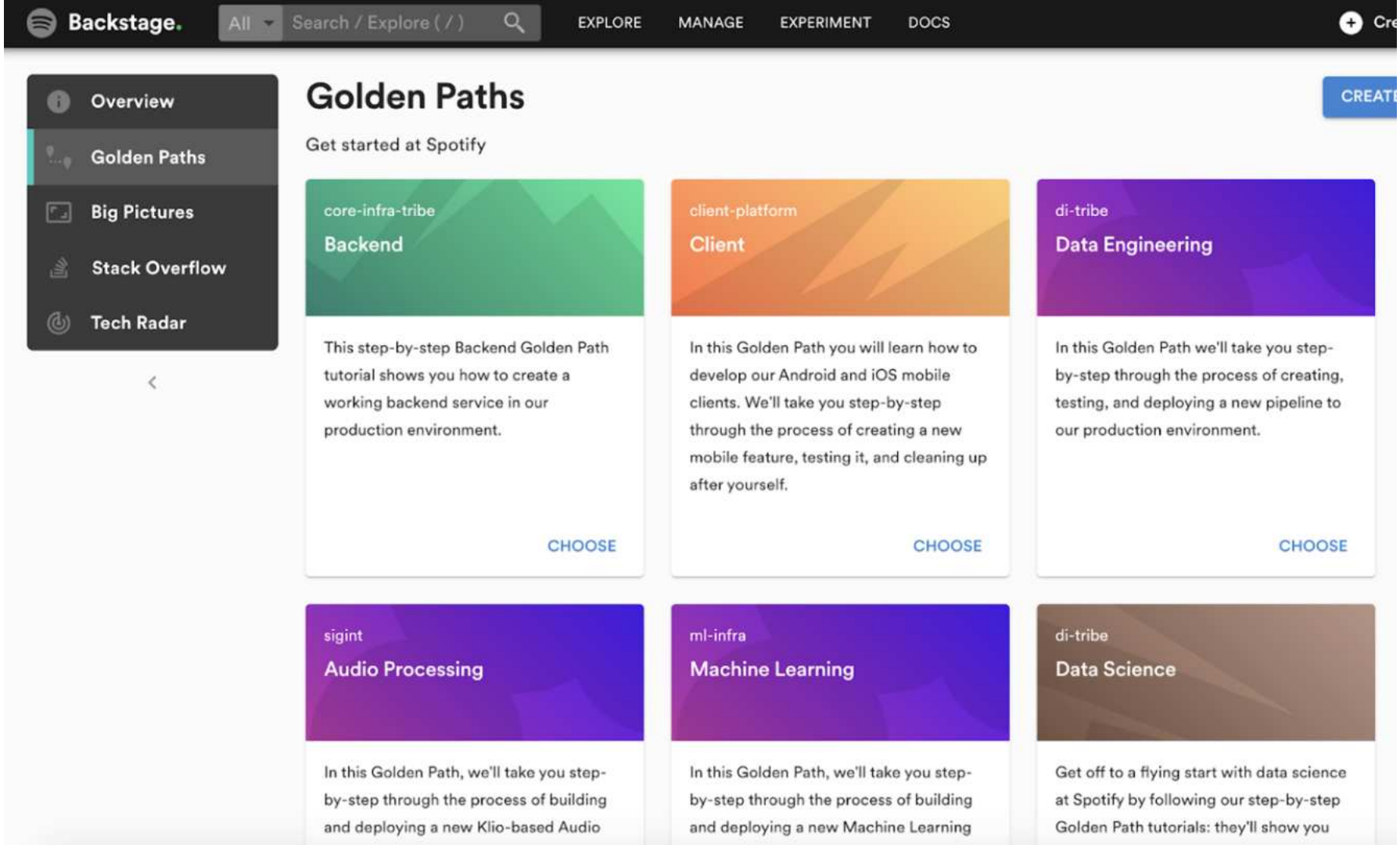


Plan
Mapping campus walking paths

Anreize für höhere Designziele?




RSS Publishers
Eureka Service
Registers Instances of Middle Tier
Configuration Properties
Discovers Instances of Middle Tier
Ribbon
Servo
Astyanax
RSS Middle Tier Service
Eureka Client
Archaius
Blitz4j
Karyon
Gather Metrics
Manages communication between Edge and Middle Tier
Hystrix & Ribbon
Archaius
Eureka Client
RSS Edge Service
Blitz4j
Karyon
Logs
Manages Server
Cassandra
Clients
Netflix Open Source Components


Backstage.
All
Search / Explore (/)
EXPLORE
MANAGE
EXPERIMENT
DOCS
Overview
Golden Paths
Big Pictures
Stack Overflow
Tech Radar
Golden Paths
Get started at Spotify
core-infra-tribe
Backend
This step-by-step Backend Golden Path tutorial shows you how to create a working backend service in our production environment.
CHOOSE
client-platform
Client
In this Golden Path you will learn how to develop our Android and iOS mobile clients. We'll take you step-by-step through the process of creating a new mobile feature, testing it, and cleaning up after yourself.
CHOOSE
di-tribe
Data Engineering
In this Golden Path we'll take you step-by-step through the process of creating, testing, and deploying a new pipeline to our production environment.
CHOOSE
sigint
Audio Processing
In this Golden Path, we'll take you step-by-step through the process of building and deploying a new Klio-based Audio
ml-infra
Machine Learning
In this Golden Path, we'll take you step-by-step through the process of building and deploying a new Machine Learning
di-tribe
Data Science
Get off to a flying start with data science at Spotify by following our step-by-step Golden Path tutorials: they'll show you

Path of Least Resistance

Wildwuchs und Governance-Aufwände durch geringen Widerstand bekämpfen

Top-5 Challenger

Basis: 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? ...

LASR-Kartenset inkl. Qualitätszielkarten

LASR - **L**ightweight **A**pproach for **S**oftware **R**eviews

➔ https://www.embarc.de/lasr-kartenset-bestellen/

14 Qualitätsmerkmale als Zielsetzung

Funktionale Eignung

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

- Angemessenheit
- Korrektheit
- Vollständigkeit

Zuverlässigkeit

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

- Verfügbarkeit
- Wiederherstellbarkeit
- Fehlertoleranz

Performanz

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

- Zeitverhalten
- Verbrauchsverhalten
- Kapazität

Benutzbarkeit

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

- Bedienbarkeit
- Erlernbarkeit
- Barrierefreiheit

Sicherheit

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

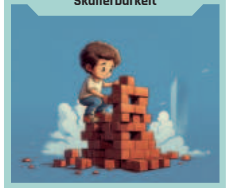
- Vertraulichkeit
- Authentizität
- Datenintegrität

Betreibbarkeit

Lässt sich die Software gut betreuen? Sind Produktionsprobleme leicht auffindbar und behebbar?

- Beobachtbarkeit
- Aktualisierbarkeit
- Reaktionseffizienz

Skalierbarkeit



Kann die Software auf Lastschwankungen und Wachstum etwa in den Mengengerüsten angemessen reagieren?

- Deployment-Flexibilität
- Elastizität
- Wachstumseffizienz

Nachhaltigkeit

Ist die Software umweltverträglich? Lässt sie sich ressourcenschonend nutzen?

- Energieeffizienz
- Emissionsarmut
- Langlebigkeit

Auditierbarkeit

Ist die Software aus einer juristischen, finanziellen oder Sicherheitsperspektive gut zu überprüfen?

- Nachvollziehbarkeit
- Nachweisbarkeit
- Zurechenbarkeit

Safety

Sind Personen, Tiere, Sachen oder Umwelt vor Schäden durch die Software geschützt?

- Betriebssicherheit
- Verifizierbarkeit
- Regulatorische Compliance

Kosteneffizienz

Ist der Betrieb der Software wirtschaftlich optimiert und lässt sich ihr Einsatz gut planen?

- Sparsamkeit
- Schätzbarkeit
- Kostentransparenz

Wartbarkeit

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

- Analysierbarkeit
- Änderbarkeit
- Erweiterbarkeit

Kompatibilität

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

- Koexistenz
- Interoperabilität
- Versionierungsfähigkeit

Portierbarkeit

Ist die Software leicht auf andere Zielumgebungen übertragbar?

- Übertragbarkeit
- Installierbarkeit
- Austauschbarkeit

Top-5-Challenger (aus 14 Karten die 5 wichtigsten wählen)

Start Top-5

Kandidaten

Nach 2 Runden, Beispiel (Beispiel)

Start Top-5

Ablagestapel

Kandidaten

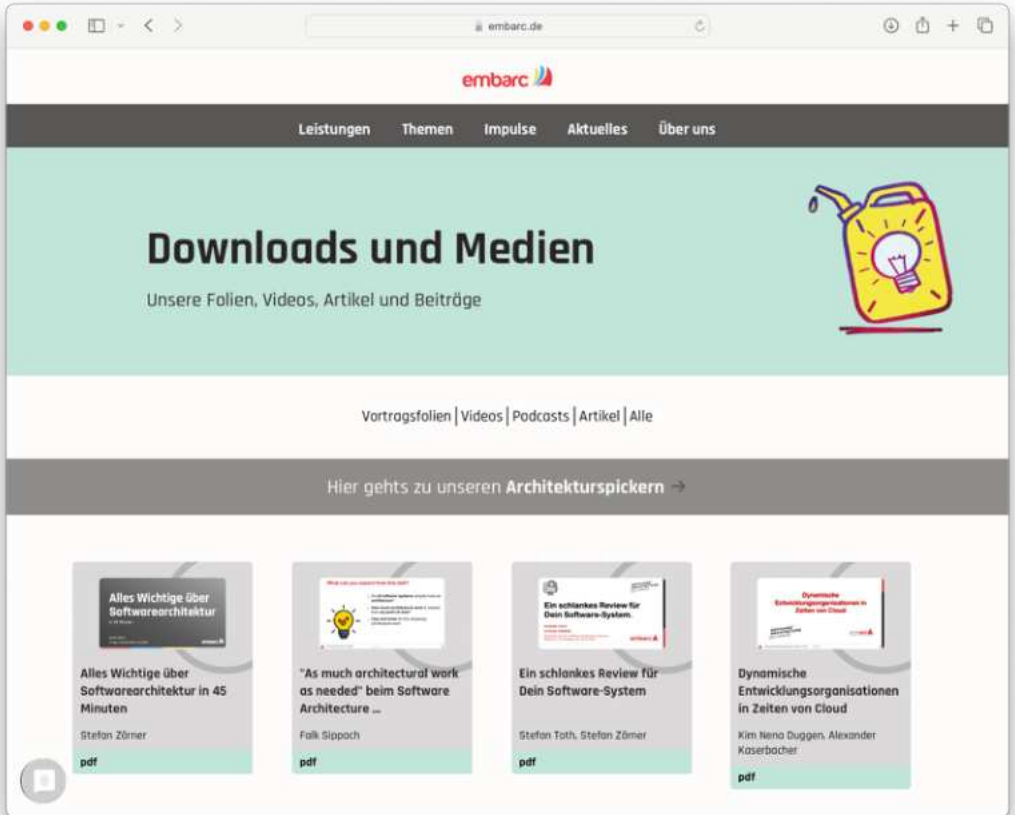

Challenger
Ablage
Nach Wichtigkeit sortiert ...
Wichtigkeit
Beschreibung
Notizen
Warum ist das Z
Warum a
Notizen zur B
Warum ist das Ziel in
Warum an die
Notizen zur Begrü
Warum ist das Ziel in den
Warum an dieser
Notizen zur Begründung
Warum ist das Ziel in den Top-5?
Warum an dieser Stelle?
Notizen zur Begründung:
Warum ist das Ziel in den Top-5?
Warum an dieser Stelle?

05.

Abschluss

Weitere Informationen...

Folien als PDF zum Download



→ embarc.de/download/

Feedback & Fragen?

Ich freue mich auf Fragen, Diskussionen, Pizza!