die Mobiliar

Die Evolution zu Microservices

Forum Bank IT zum Thema Microservices, Self contained Services und Containerisierung 13.09.2023

Andreas Grütter| Leiter IT Architektur

Ausblick auf die nächsten 30 Minuten ....

1

Einleitung

Digitale und agile Transformation in der Mobiliar

2

Die neue Zielarchitektur

Aus der Performance Krise hin zur neuen Zielarchitektur der IT-Landschaft, was waren die Herausforderungen

3

Joint Application Plattform

Der nächste Schritt: Continuous Delivery, Docker und kleine Microservices im Backend

4

Demo der Werkzeuge zum Führen einer Microservice Architektur

Demo

Das Verständnis von Microservices

Chris Richardson https://microservices.io/index.html

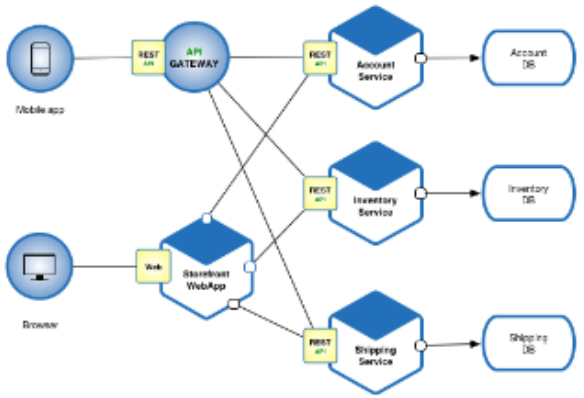
- *Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of loosely coupled services, which implement business capabilities. The microservice architecture enables the continuous delivery/deployment of large, complex applications. It also enables an organization to evolve its technology stack*

Martin Fowler: https://martinfowler.com/articles/microservices.html

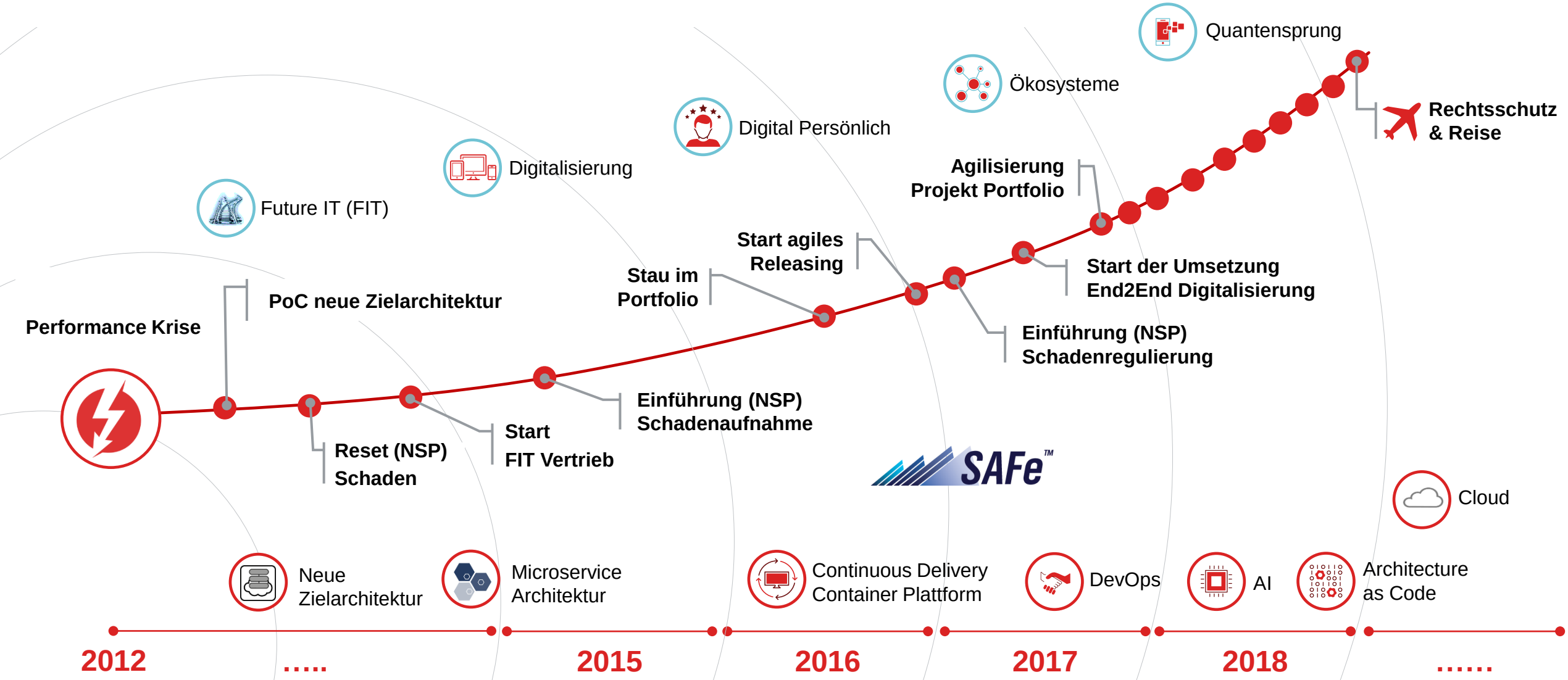
- *The term "Microservice Architecture" has sprung up over the last few years to describe a particular way of designing software applications as suites of independently deployable services. While there is no precise definition of this architectural style, there are certain common characteristics around organization around business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data.*

Die gemeinsamen Eigenschaften

- Zerlegung in **fachlich kleine Services**
- Services sind sehr **lose gekoppelt**
- **Reduktion der Abhängigkeiten**
  (Services können einzeln installiert, aktualisiert und verwaltet werden)
- Ermöglicht die **dezentrale Datenhaltung**
- Höherer Freiheitsgrad bei der Technologiewahl



Beginn digitale und agile Transformation in der Mobiliar



2. Die neue Zielarchitektur

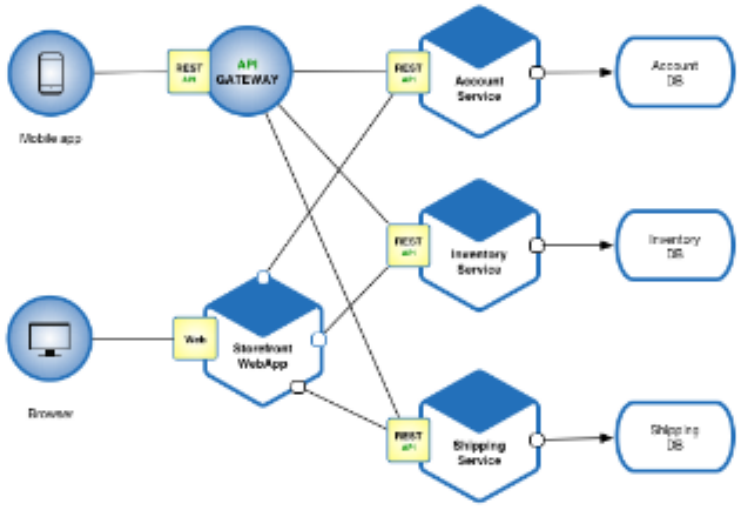
Aus der Performance Krise hin zur neuen Zielarchitektur

Flexibles Gesamtsystem in Bezug auf

- Neue fachliche Anforderungen schnell umsetzen
  ⇨ Agiler werden
  - Klein (inklusive Datenhaltung) als Gegenmodell zu Monolith
  - Lose Kopplung, leichte Austauschbarkeit (build for rebuild before reuse)
  - "Standard" Technologie Schnittstellen: REST
- Neue technologische Möglichkeiten schneller nutzen
  - Technologieauswahl auf aktuelle Versionen und Konzepte heben (verabschieden von den alten J2EE Patterns)
  - Frontend Technologie auf Internet ausrichten
- Schwankungen in der Last auffangen
  - Einzelne Services horizontal Skalieren (billiger)
  - Services schnell restarten (LifeCycle einführen)

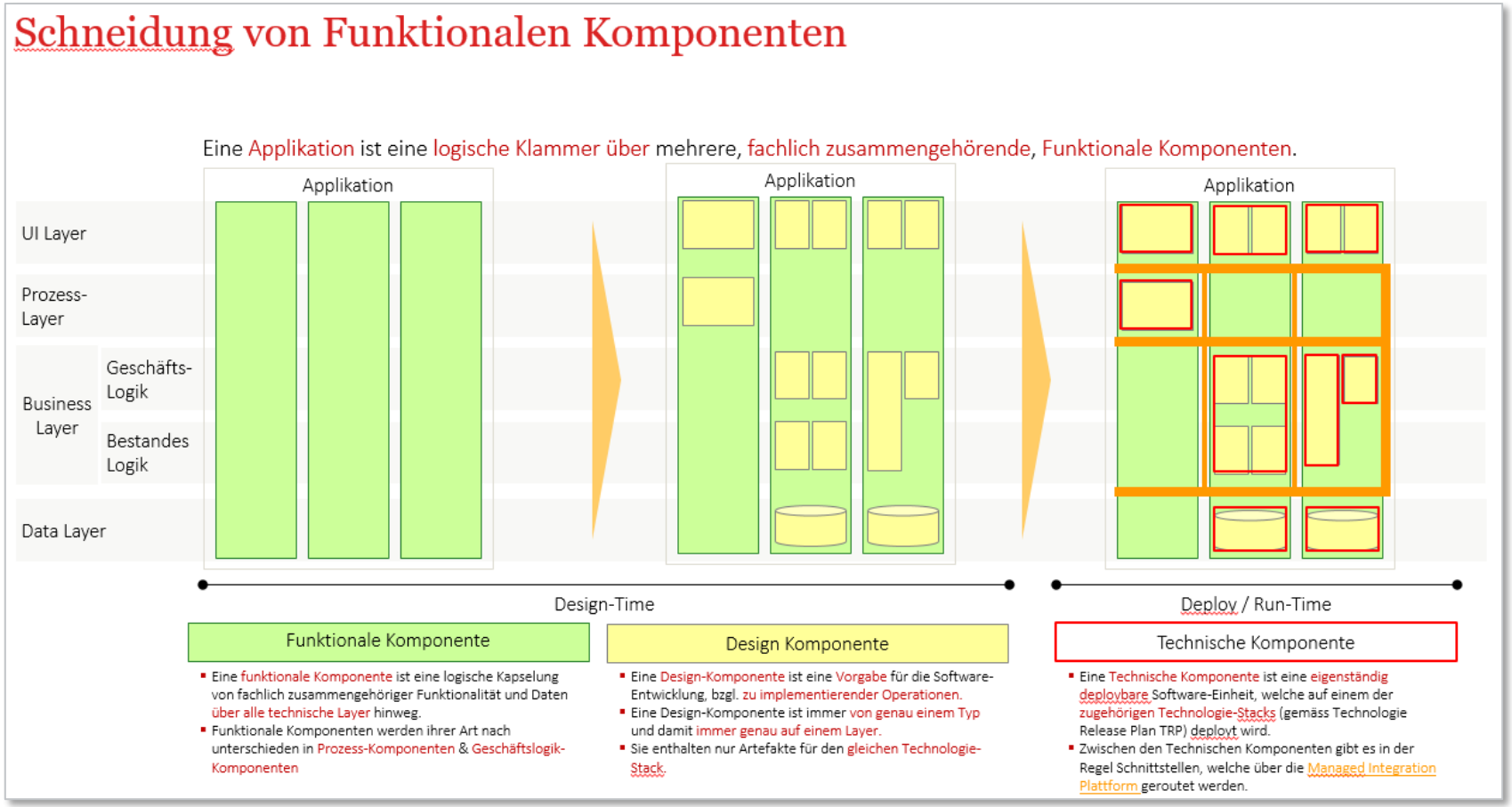
Die gemeinsamen Eigenschaften

- Zerlegung in **fachlich kleine Services**
- Services sind sehr **lose gekoppelt**
- **Reduktion der Abhängigkeiten**
  (Services können einzeln installiert, aktualisiert und verwaltet werden)
- Ermöglicht die **dezentrale Datenhaltung**
- Höherer Freiheitsgrad bei der Technologiewahl



Die neue Zielarchitektur und deren Widerstände

- Vertikale Schneidung der Komponenten entlang der fachlichen Abhängigkeiten und nicht der technischen Layers
- Decompose by Business Capability

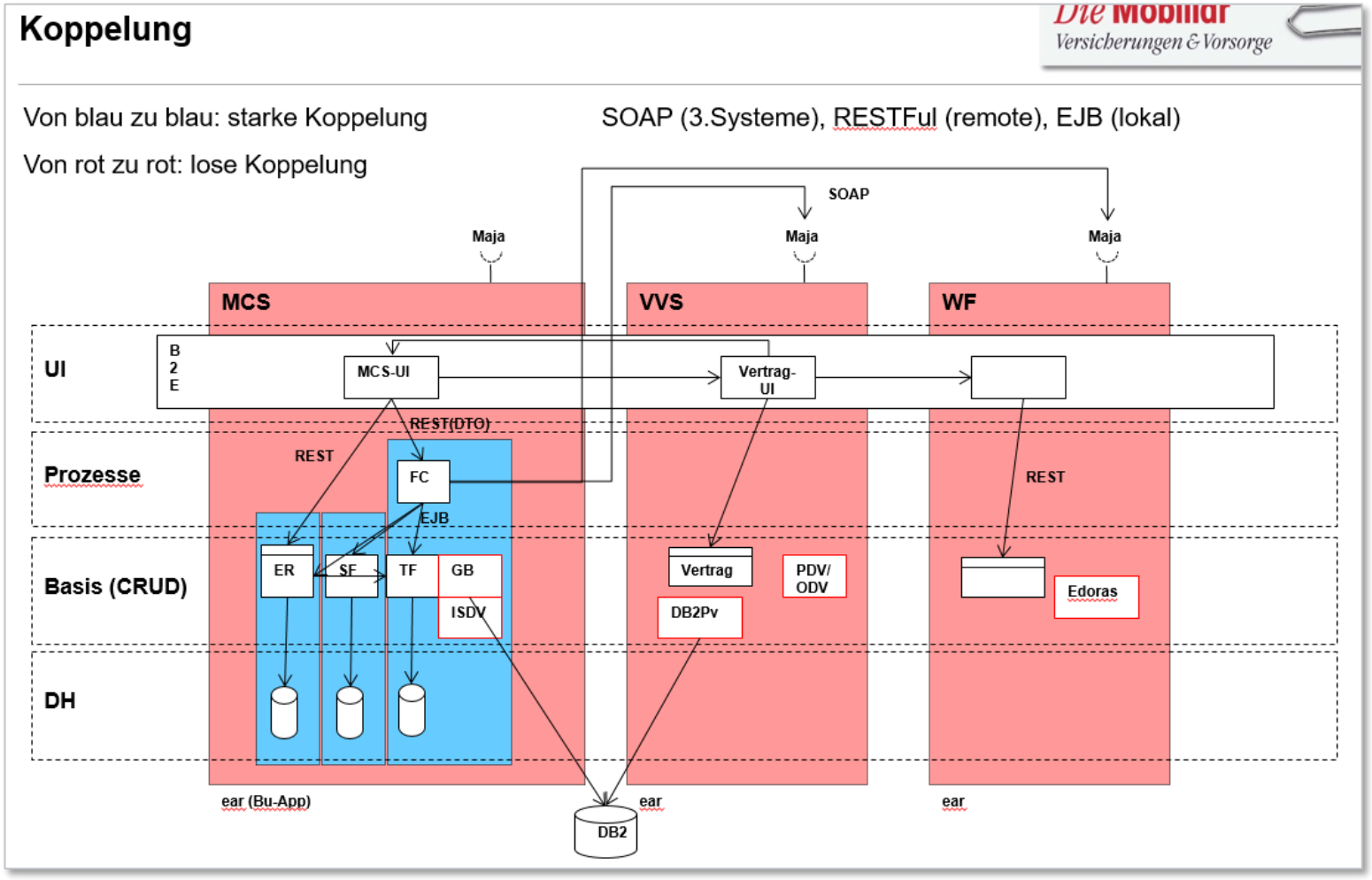


Ein einfaches, verständliches Modell zur Schneidung von Prozessen, Daten und der Managementinformationssysteme (in Komponenten) ist nötig!

Die neue Zielarchitektur und deren Widerstände

Grosse Diskussionen über die Integrationsarten

- Oracle Schema versus DB2
- MAIA MDA versus REST
- SOAP versus REST
- ESB versus wenig Middleware

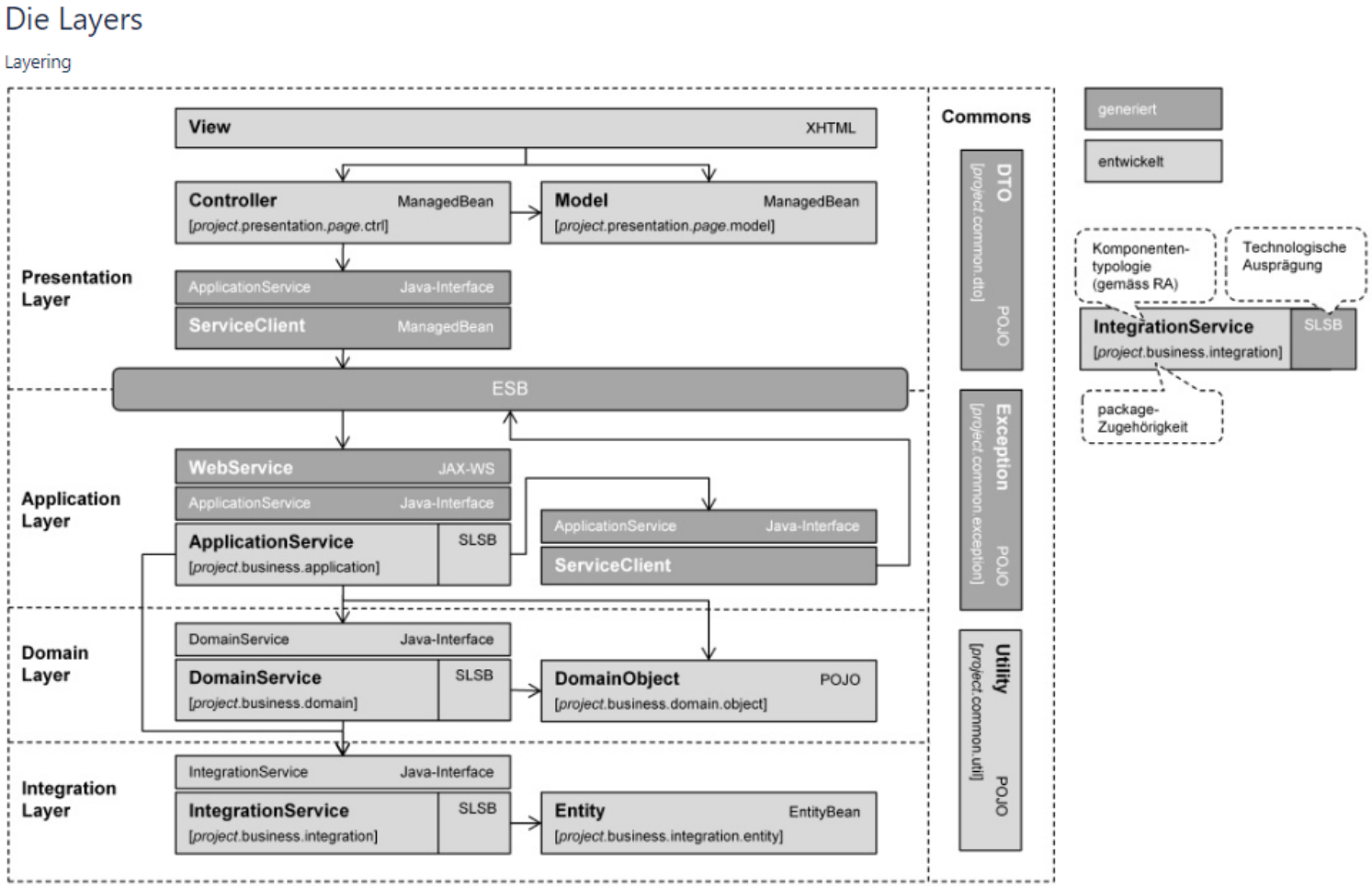


Je eine Implementation als Integrationsart reicht:
Beispiel: Für Remote Procedure Invocation REST und für Messaging Kafka

Die neue Zielarchitektur und deren Widerstände

Interne Architektur

- DTO, versus, VO versus generiete Klassen um Schnittstellen sicher und versionierbar zu machen
- Abstraktionen innerhalb eines Service (Layering)

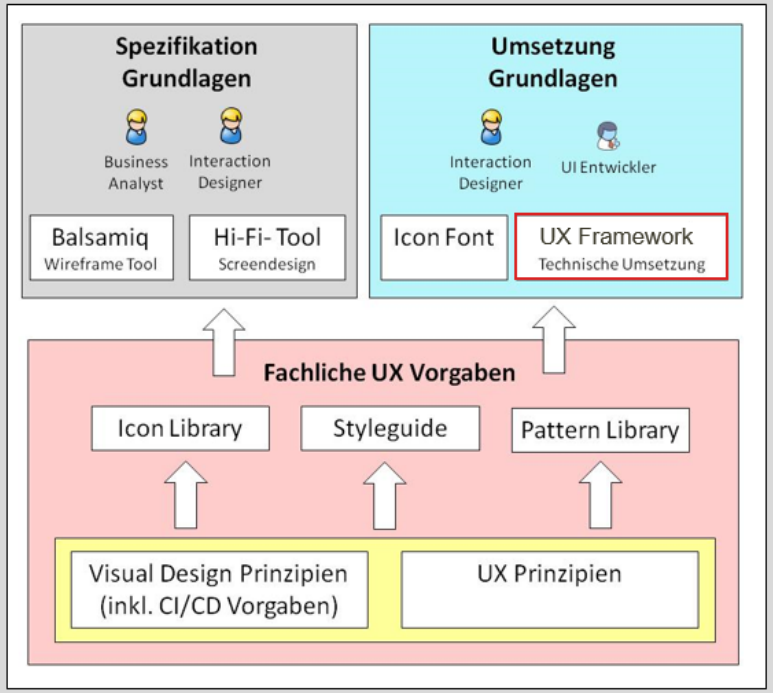


Mit Microservice ist die Innenarchitektur nicht mehr so wichtig wie früher, aber es müssen alle Anforderungen automatisch geprüft werden (automatische Governance)

Die neue Zielarchitektur und deren Widerstände

Ergonomische Oberflächen die einheitlich aussehen, um die manuellen Prozessbrüche zu vereinfachen

Mobiliar UX Vorgaben & Werkzeuge für eine einheitliche und ergonomisch bedienbare Benutzeroberfläche



Grundlage & Erweiterung
Fachliche UX Vorgaben sind die Grundlage von „Lego-Steinen“ zur effizienten und einheitlichen

- **Spezifikation** (Lego-Steine in UI-Spezifikations-Tools), sowie
- **Implementierung** (technische Umsetzung in einem UX Framework) von Benutzeroberflächen.

Die Werkzeuge werden anhand des Einsatzes in Projekten laufend angepasst und erweitert.

Einsatz
Zur Zeit findet das UX-Framework im B2E-Portal und den darin integrierten Benutzeroberflächen Verwendung. Weiter wird es im Rahmen des Postzugangs (B2B) eingesetzt.

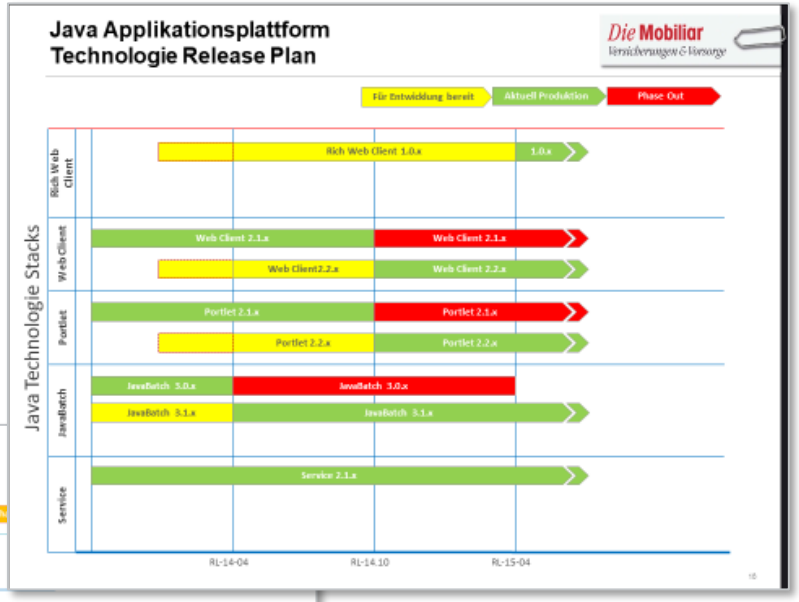
Die UX Vorgaben gelten auch im B2C Portal (ohne Einsatz des UX Frameworks).

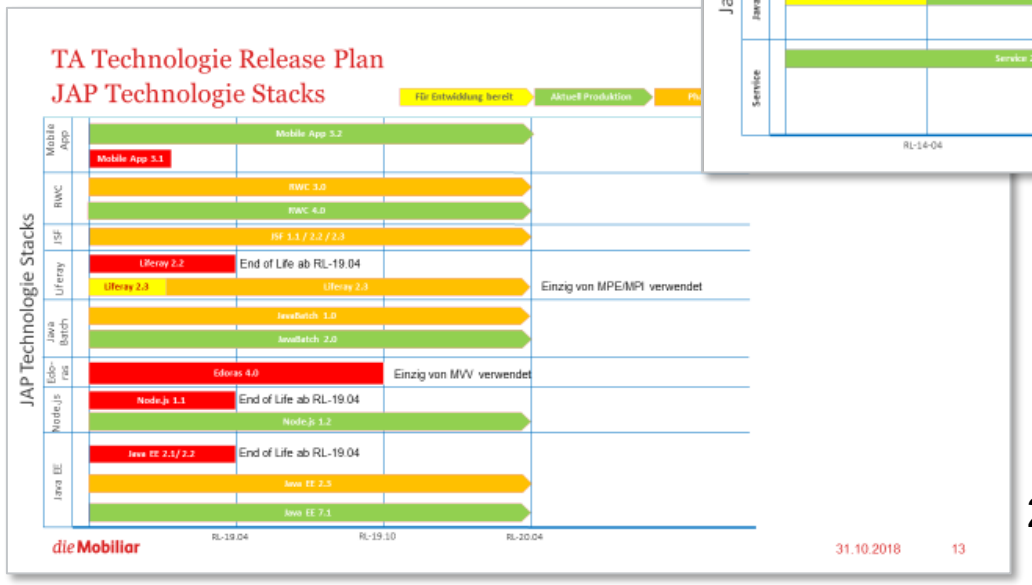
Einführen von einem einheitlichen Design ist aufwändig, aber nützlich um eine Microservice Architektur im Frontend umzusetzen

Ordnen der Entwicklungsplattformen und Umsetzung der Zielarchitektur

- Zielplattform für Eigenentwicklungen festlegen und priorisieren
- Verantwortlichkeiten der Entwicklungsplattformen festlegen und regeln
- Roadmaps der einzelnen Plattformen regeln

2014





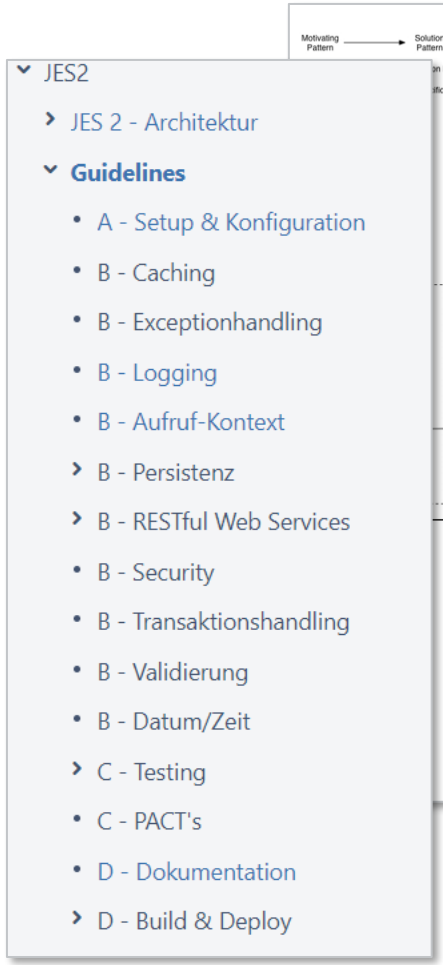
2017

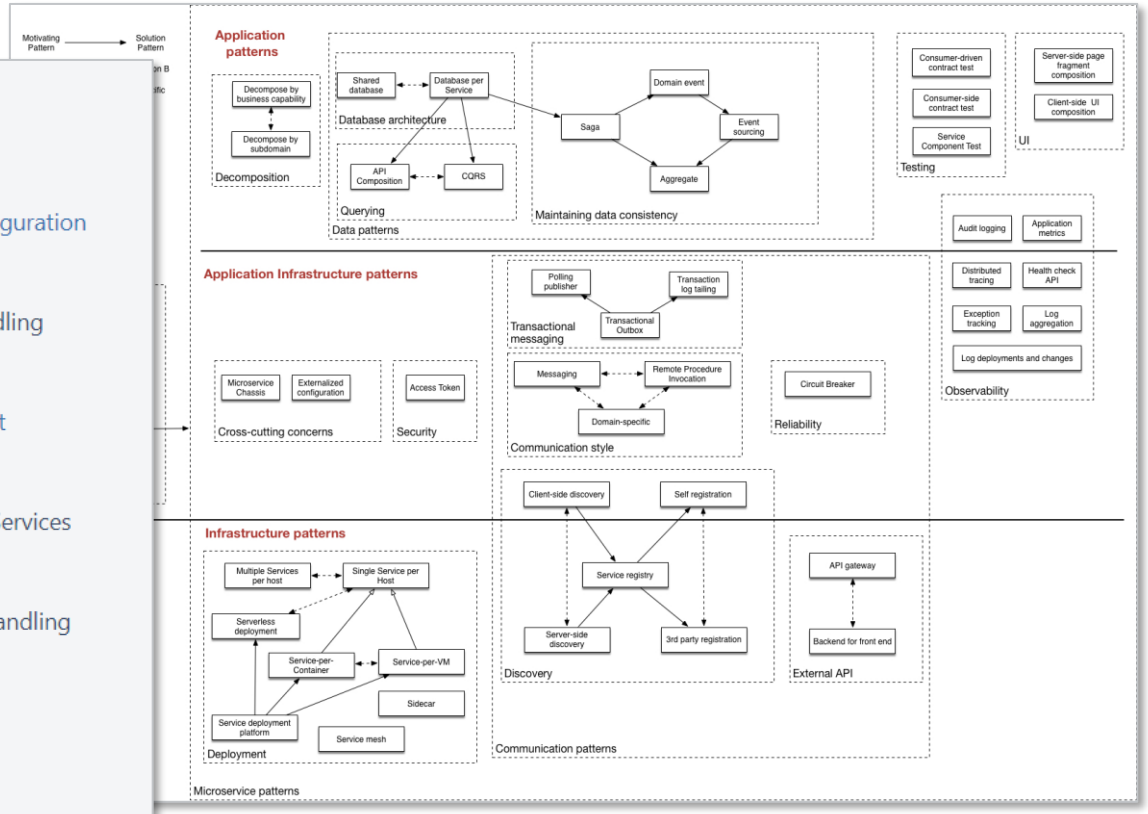
Applikationsentwicklungsplattformen definieren und managen.
Nur eine präferierte Variante zulassen!

Umsetzung der Zielarchitektur und priorisieren der Entwicklungsplattformen

Eine erste Version der JAP Plattform in einem Projekt erfolgreich umsetzen und nutzen

- Decompose by Business Capability
- Database per Service
- API Composition
- Service Component Testing
- Audit Logging
- Application Metrics
- Microservice Chassis
- External Configuration
- Remote Procedure Invocation
- Distributet Tracing
- etc





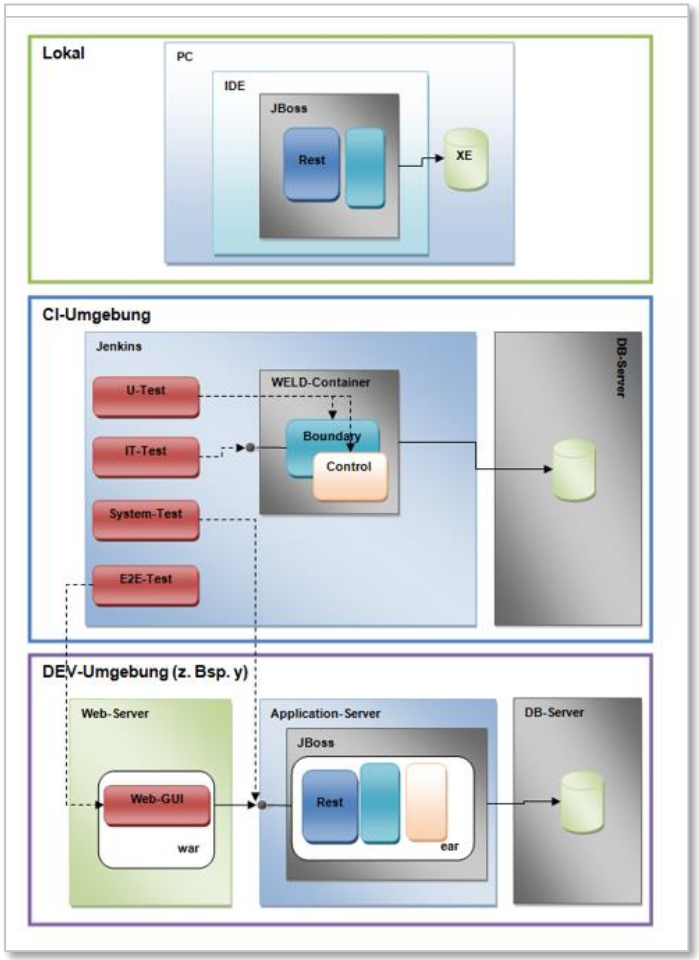
https://microservices.io/patterns/index.html

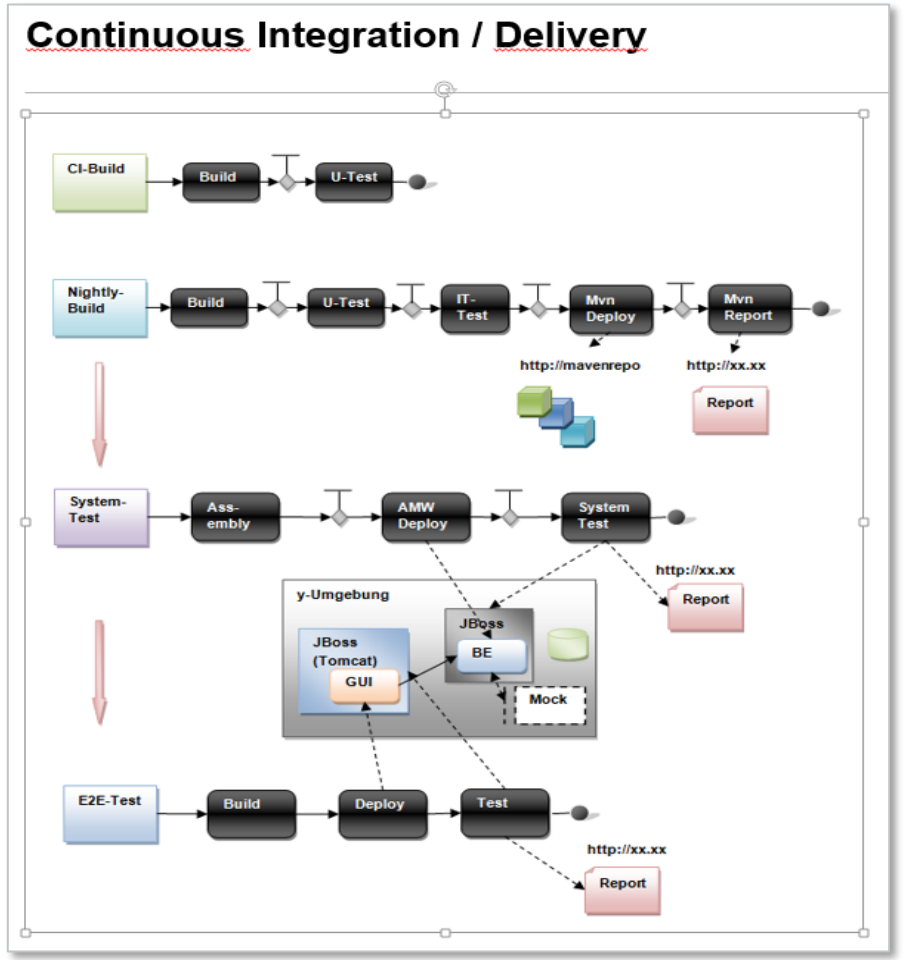
Standardisieren der nicht funktionalen Qualitätsanforderungen über Industriestandards; wo nichts vorhanden selber umsetzen!

Umsetzung der Zielarchitektur und priorisieren der Entwicklungsplattformen

Eine erste Version von Continuous Integration und Continuous Deployment einführen

- Bis auf die Testsysteme
- Automatisierung minimal



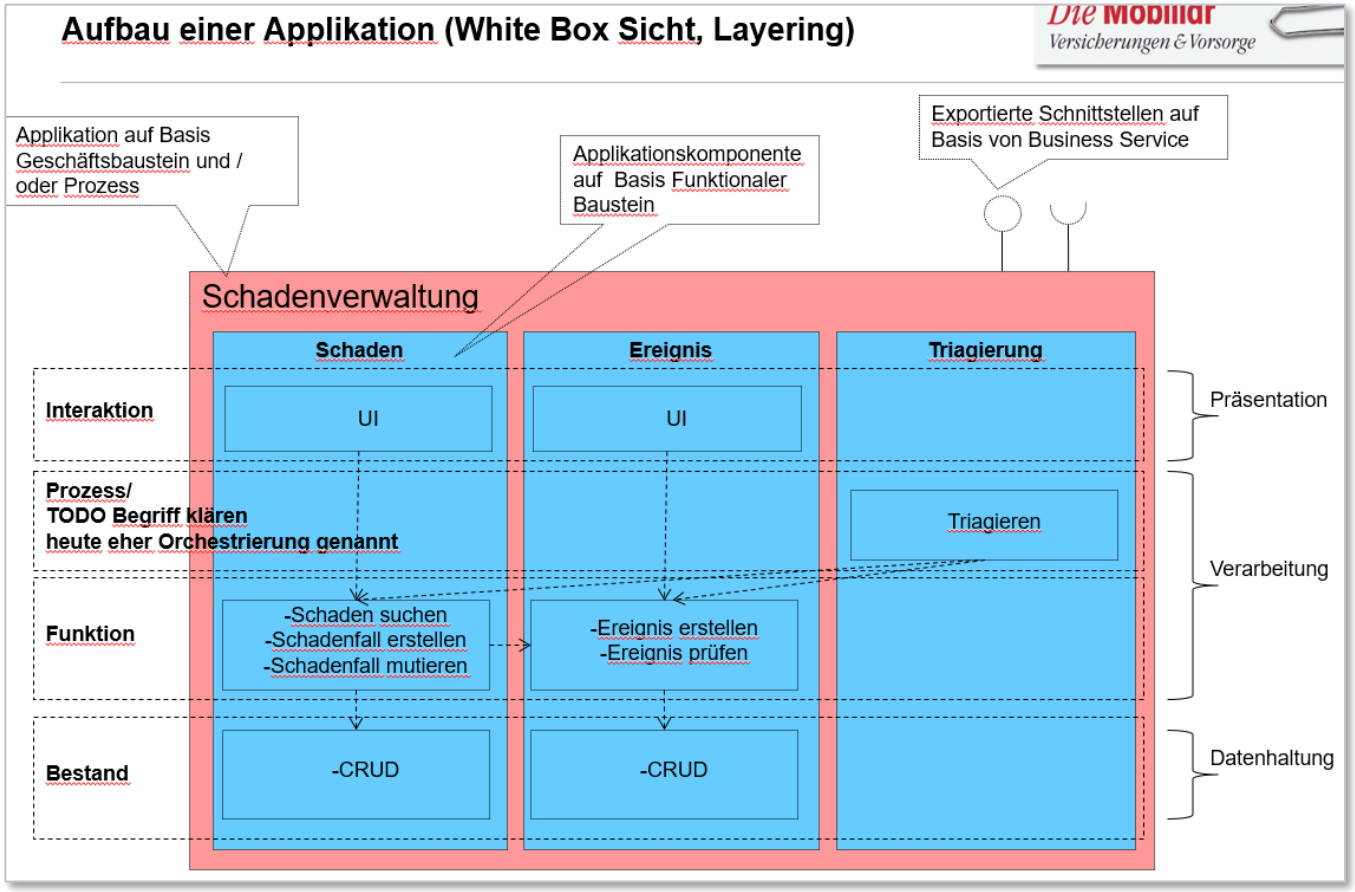


Die 3 Grundprinzipien: Versionierung, Continuous Testing und Continuous Integration müssen verankert werden, um die Produktivität zu steigern!

Herausforderungen bei der Umsetzung der Zielarchitektur

Aus Sicht Applikationsarchitektur

- Wie klein / gross sind die Microservices?
  ⇨ *Decompose by Business Capability muss man lernen*
- MDA versus Microservices



Decompose by Business Capability muss man lernen

Herausforderungen bei der Umsetzung der Zielarchitektur

Aus Sicht Technologiearchitektur

- Wie integrieren wir die "alten" Services und Applikationen (inkl. Fehlerhandling)?
  ⇨ *viel Aufwand hineingesteckt, damit möglichst transparent und einfach für die Anwender*
- DB welche DB2 versus Oracle Schemas
  ⇨ *Automatisieren der Bestellungen*
- Trunked Bases Development
  ⇨ *durchsetzen*
- Continuous Integration und Testing
  ⇨ *messen und vorgeben*
- Fachliche Shared Libraries
  ⇨ *Kampf am Anfang verloren: nach gewisser Zeit von selbst durchgesetzt*

Anforderungen

Die Mobiliar Versicherungen & Vorsorge

- JAP basierte Applikationen basierend auf
  - JEEService
  - RichWebClient

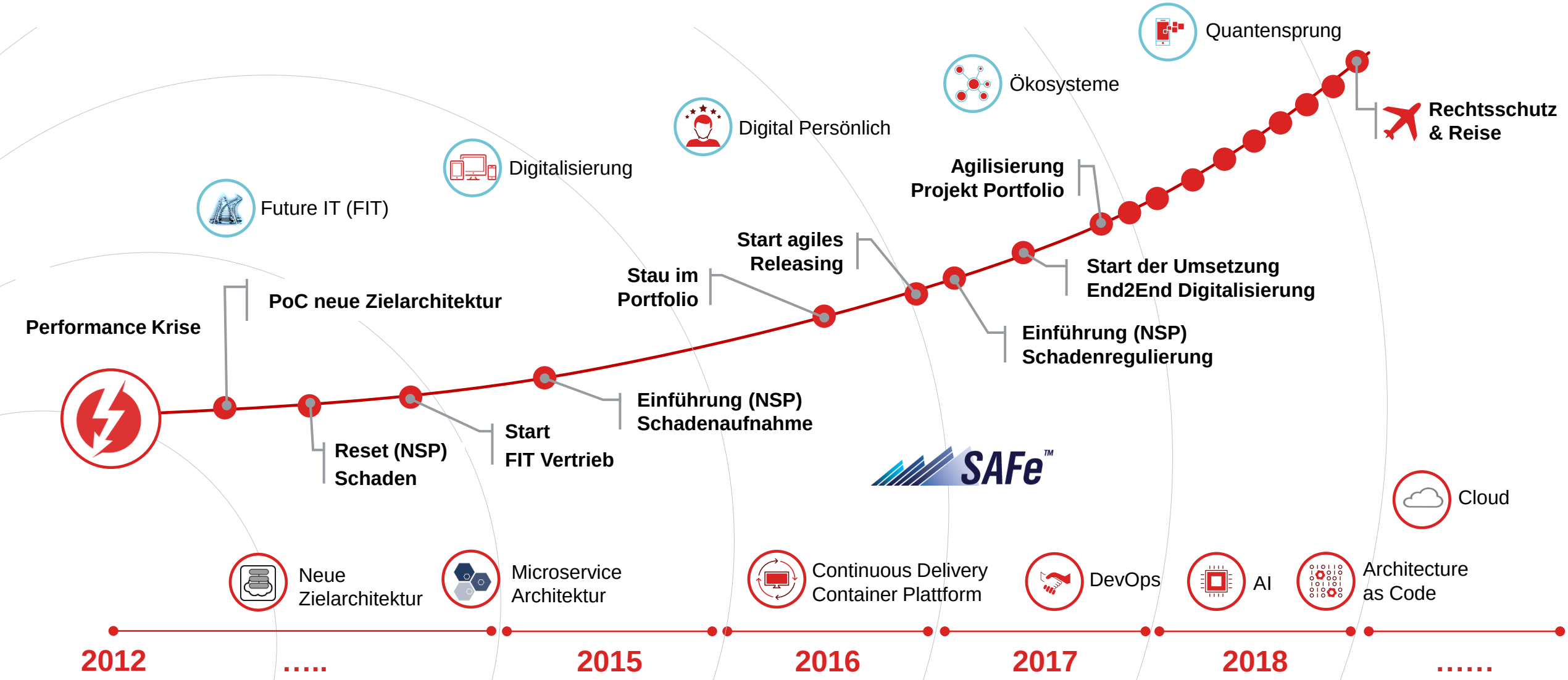
  führen Trunk Based Development und Continous Delivery ein

- Was heisst das:
  - Um sich in der Entwicklung nicht gegenseitig zu blockieren, dürfen Commits nur auf trunk landen wenn die Unit- und Integrationtests grün sind
  - Der **CI Integriert die Softwareabhänigkeiten** des Trunks
  - Testdaten in Verantwortlichkeit der Entwicklung
  - Entwicklung ist für CI-Umgebung verantwortlich (bzw. die Applikationsowner)

Die 3 Grundprinzipien: Versionierung, Continuous Testing und Continuous Integration müssen verankert werden, um die Produktivität zu steigern!

3. Joint Application Plattform

Beginn digitale und agile Transformation in der Mobiliar



Continuous Delivery, Docker und kleine Microservices im Backend

Reduktion Aufwände für die Continuous Delivery Pipeline (Effizient werden)

- Datenbank
- Incident Management
- Deployment Freigaben

Etablieren von automatischer Governance

- Build
- Codeverletzlichkeit und Coverage Analyse
- Testpyramide
- LifeCycle Methoden
- API Dokumentation
- DB Versionierung

Mobitor Funktionen

Der Mobitor führt Informationen aus verschiedenen Quellen zusammen. Derzeit unterstützt sind folgende:

Quelle	Voraussetzungen	Notwendigkeit	Qualitäts-Regeln	
AMW	Server-Name, Applikations-Name, Umgebung	muss	• keine SNAPSHOT Version • AMW Deployment erfolgreich	
Sonar	Sonar-Projekt-Key, Sonar-Analyse	optional	• keine Blocker • keine Criticals • für strikte Regeln (agiler Release): Unit Test Coverage > 70% • für Legacy Regeln: Coverage > 70%	
TeamCity	Build in TeamCity (mit configurationId)	optional	• Failed Build führt zu Fehler-Hinweis	
NexusIQ	Analyse des Projektes im NexusIQ (früher Nexus CLM)	optional	• Critical Vioaltions führen zu Fehler-Hinweis	
REST /ping	swagger.json (alle .../ping)	optional	• Status-Code: 200..299	
REST /shakedowntest	swagger.json (alle .../shakedowntest	optional	• Status-Code: 200..299 • ShakedownTestResultDto Ergebnis ohne Fehler-Einträge	
Health-Status	swagger.json mit /health/status Resource (enthalten in JES Version)	optional	• Status-Code: 200-299 • Telemetrie ist enabled	
Jira	• Filter im JIRA der für den Benutzer mobi_rest sichtbar ist (Filter-Id) • Release tags in version control must match the pattern "release/<version>" e.g. release/1.2.3	optional	• Priorität Critical -> Fehler • Priorität Major -> Warning	
Flyway Schema Version	swagger.json mit /health/flyway/current Resource (enthalten in JES Version)	optional	• Status der Migration auf Success	
Kubernetes	AMW schickt die notwendigen Informationen (Servername, Applikationsname) als Annotations an Kubernetes (korrektes AMW Template resp. Util nutzen, deployment.yaml)	optional	• Restart-Count < 1 führt zu Warn-Hinweis • Restart-Count > 5 führt zu Fehler-Hinweis	
HP ALM	Ergebnisse für Test-Läufe (Runs für Test-Sets) die im ALM erstellt wurden	optional	• Run "Failed" Tip: klick auf "sit@alm" zeigt Details, klick auf die Anzahl springt ins ALM	
BitBucket, Commit-Datum	Tage seit dem Commit der diese Version erstellt hat Tag im Git-Repository das die Version enthält. Z.B.: • release/1.20.156 • release-1.20.156 • release-1.20.0-156 • release-156 • release/156 **Empfohlen** ist im Git-Tag die gesammte Version zu nutzen, nicht nur die Build-Nummer: **release/1.20.156**	optional	• keine	

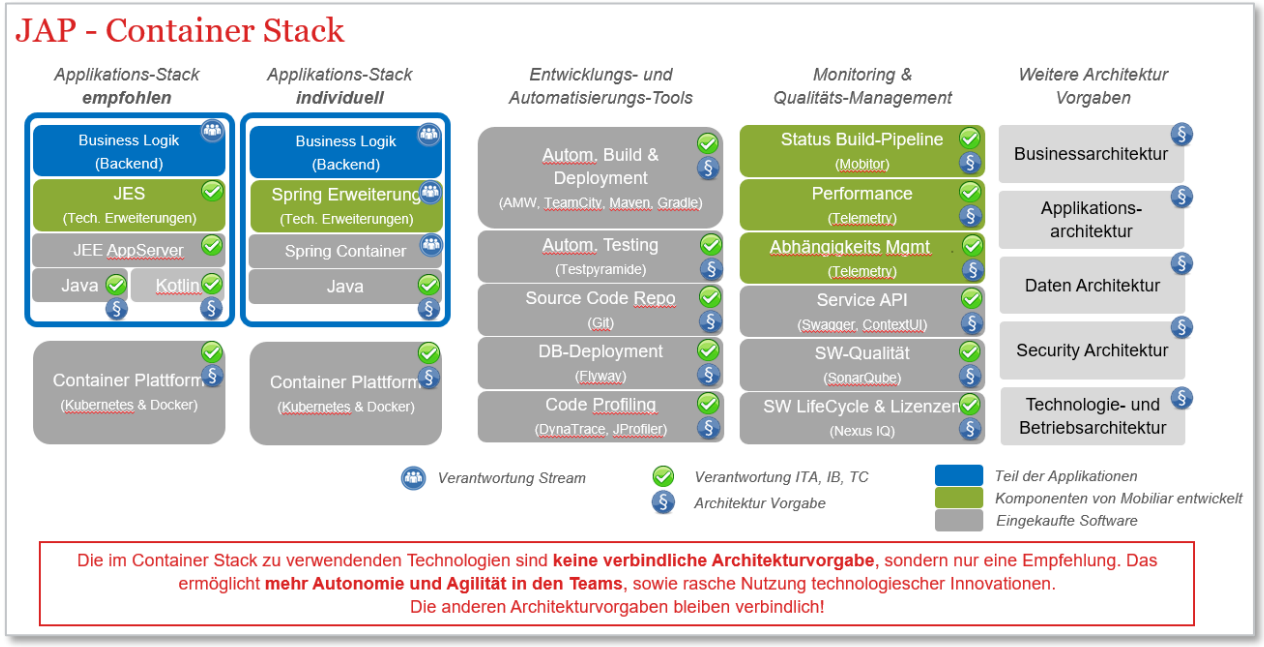
Automatisieren der Governance und erhöhen der Automatisierung Continuous Pipeline führen zu effizienteren Teams

Continuous Delivery, Docker und kleine Microservices im Backend

Druck von den Entwickler für mehr Technologie in den Services

JAP Plattform (Basis JEE)

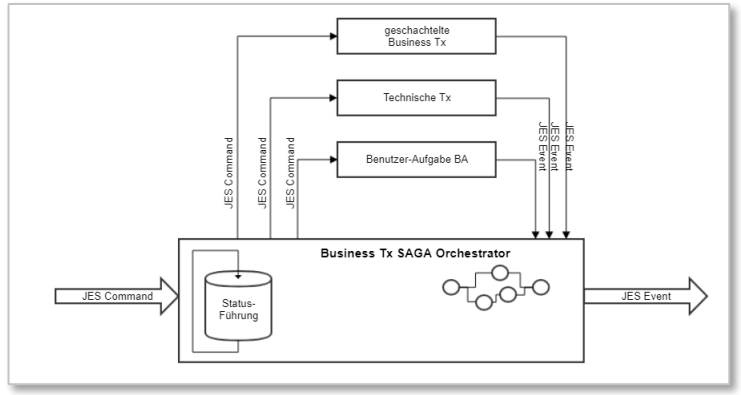
- Ersatz Single Service per VM durch Single Service per Container
- Erweiterungen in allen bisherigen Patterns und neue zur Verfügung stellen
  - API Gateway
  - Messaging
  - Consumer Driven Contract Testing
  - viel selbstgeschriebener Code entfernt



Plattform weiter standardisieren, gleichzeitig Freiheiten für die Teams erarbeiten

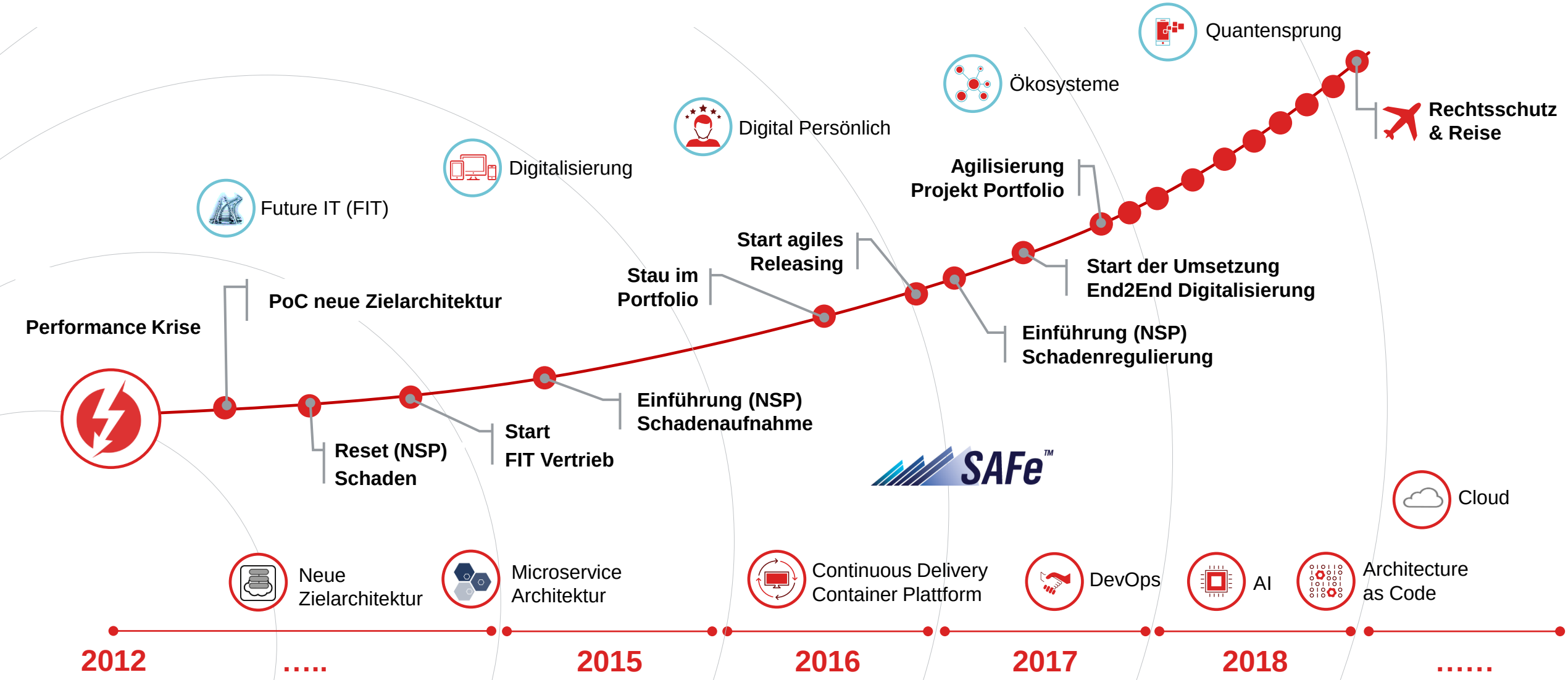
Herausforderungen bei der Umsetzung der Joint Application Plattform

- **Aus Sicht Applikationsarchitektur**
  - Der Zoo an Microservices wächst und wächst
  - Mit den aktuellen Tools (EA und Casewise) halten wir nicht Schritt
  - Die Grösse der Microservices hat sich entlang der Business Capabilities stabilisiert
  - Frontend vom Monolith zu Micro Frontends
  - Transaktionshandling (BASE, SAGA Pattern)
- Gewaltentrennung Dev / Ops
- **Aus Sicht Technologie Architektur**
  - Neue Datenbanken ElasticSearch, Mongo
  - Umbauen der gossen MS und Migration des Patterns (viel Aufwand)



Konsequent über alle Layer Microservices nutzen und Pain-Points entfernen

Beginn digitale und agile Transformation in der Mobiliar



Demo der Ist Werkzeuge

Aus Sicht:

- Applikationsarchitektur / Solutionarchitektur
- Teams
- Technologiearchitektur (Plattformarchitektur)

Zusammen-fassung

1. Konsequent über alle Layer Microservies nutzen und Painpoints entfernen
2. Präferierte Entwicklungsplattform weiter standardisieren, aber gleichzeitig Freiheiten für die Teams erarbeiten
3. Automatisieren der Governance
4. Versionierung, Continuous Testing und Continuous Integration müssen verankert werden, um Continuous Delivery zu leben