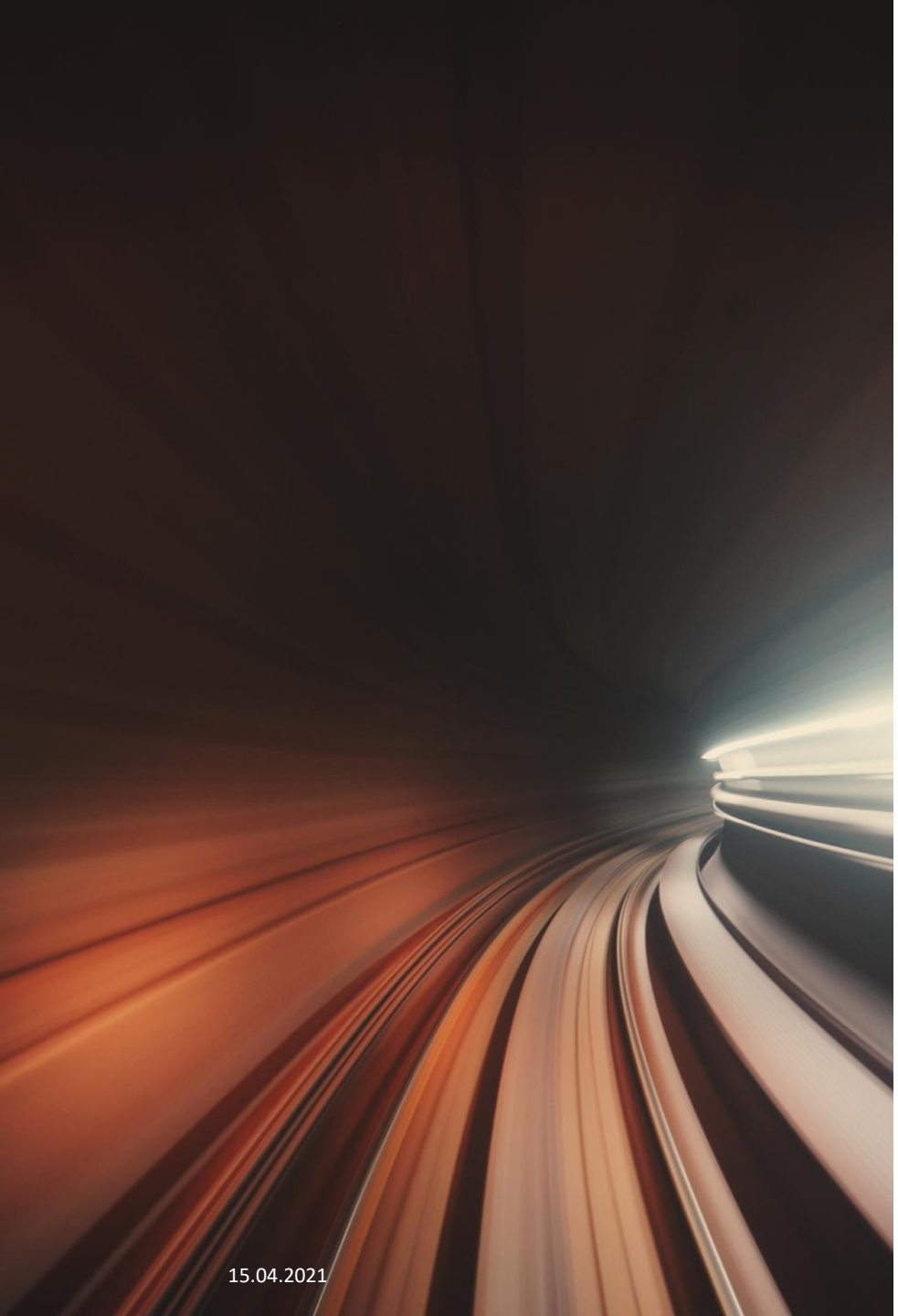
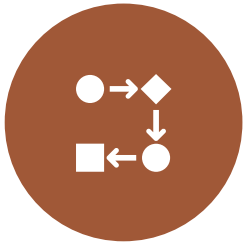
Grundlagen (Komponenten)

PROF. ANDREAS HARTMANN – VERTEILTE ANWENDUNGEN



Agenda

ABGRENZUNG UND DEFINITIONEN



KOMPONENTEN

SCHNITTSTELLEN

Quellenangaben

Die Inhalte sind angelehnt an - und Zitate stammen, wenn nicht anders gekennzeichnet, aus:

- [SP17]Schönbächler, M.; Pfister, C.: IT-Architektur. Grundlagen, Konzepte und Umsetzung. epubli, Berlin, 2017.

Definition Anwendung

„Anwendungen sind abgegrenzte Teile der firmenweiten Software, die einen definierten Teil der gesamten IT-Unterstützung realisieren.

Die Gliederung von Software in Anwendungen ermöglicht, dass IT-Investitionsentscheidungen in einem Gesamtkontext getroffen und priorisiert werden können.“

[SP17] S. 382

Erläuterung zum Begriff

„Eine Anwendung in einem Unternehmen sollte so definiert sein, dass ihre Bedeutung auch nicht-technischen Managern kommuniziert werden kann. D.h., man sollte Anwendungen nicht so fein aufgliedern, dass technisches Know-how nötig wird, um die Gliederung nachzuvollziehen.“

[SP17] S. 381

Konventionen

Verständnis des Begriffs „Anwendung“ sollte auf technischer und Management-Ebene konsistent sein

Sonst besteht Gefahr, dass technische Definition viel feiner gegliedert und nach technischem Fachwissen aufgebaut ist

Investitionsentscheidungen werden jedoch auf Management-Ebene getroffen und das Risiko falscher Entscheidungen steigt

Daher verwenden moderne Konzepte den Begriff „IT-Service“ anstelle Anwendung

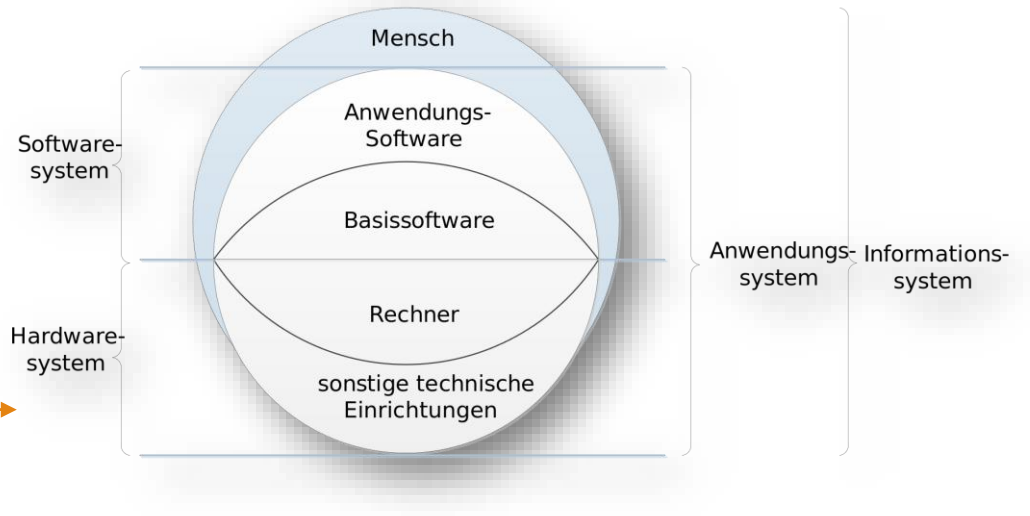
- Womit jedoch eine Abgrenzung von Service <-> Anwendung notwendig wird

Anwendung versus System

Neben der *Anwendung* kommen auch die Begriffe „System“ und „IT-System“ vor

Keine Synonyme!

Ein System ist ein umfassenderes Gebilde als eine Anwendung, bestehend aus Software und Hardware (Gesamtsicht)



Quelle: Von Campan43 - Eigenes Werk, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=22625536

Komponenten

„Eine Komponente ist die kleinste Softwareeinheit, welche der Komposition dient, vertraglich spezifizierte Schnittstellen besitzt und rein explizite Kontextabhängigkeiten aufweist. Eine Komponente kann unabhängig geändert, verteilt und durch Dritte integriert werden.“

[SP17] S. 384

Komponenten

Interagieren ausschließlich über Schnittstellen miteinander und tauschen Daten aus oder lösen Aktionen aus

Schnittstellen sind klar spezifiziert

Anwendung besteht aus einer oder mehreren Komponenten (Komposition)

Komponenten verschiedener Anwendungen können interagieren

Komponenten haben eigenen Lebenszyklus und werden unabhängig voneinander erstellt

Units

„Build Unit“ sind Elemente in einer Komponente, die nicht unabhängig ausgerollt werden sollten

Komponenten werden besser in „Deployment-Units“ transformiert und verteilt

Im IT4IT Standard sprechen wir hierbei von „Build“ und „Build-Package“

Zusammenhang Komponente - Unit

KOMPONENTE

Kann im Gegensatz zur Anwendung aus Nutzersicht nicht alleinstehend verwendet werden

Besitzt einen eigenen Lebens- und Release-Zyklus

BUILD-UNIT

Bilden zusammen den Lebenszyklus der Komponente

Deployment-Unit

Installierbares Softwareauslieferungspaket (vgl. Build Package)

Format wird über Deployment-Prozess (oder Spezifikation) definiert

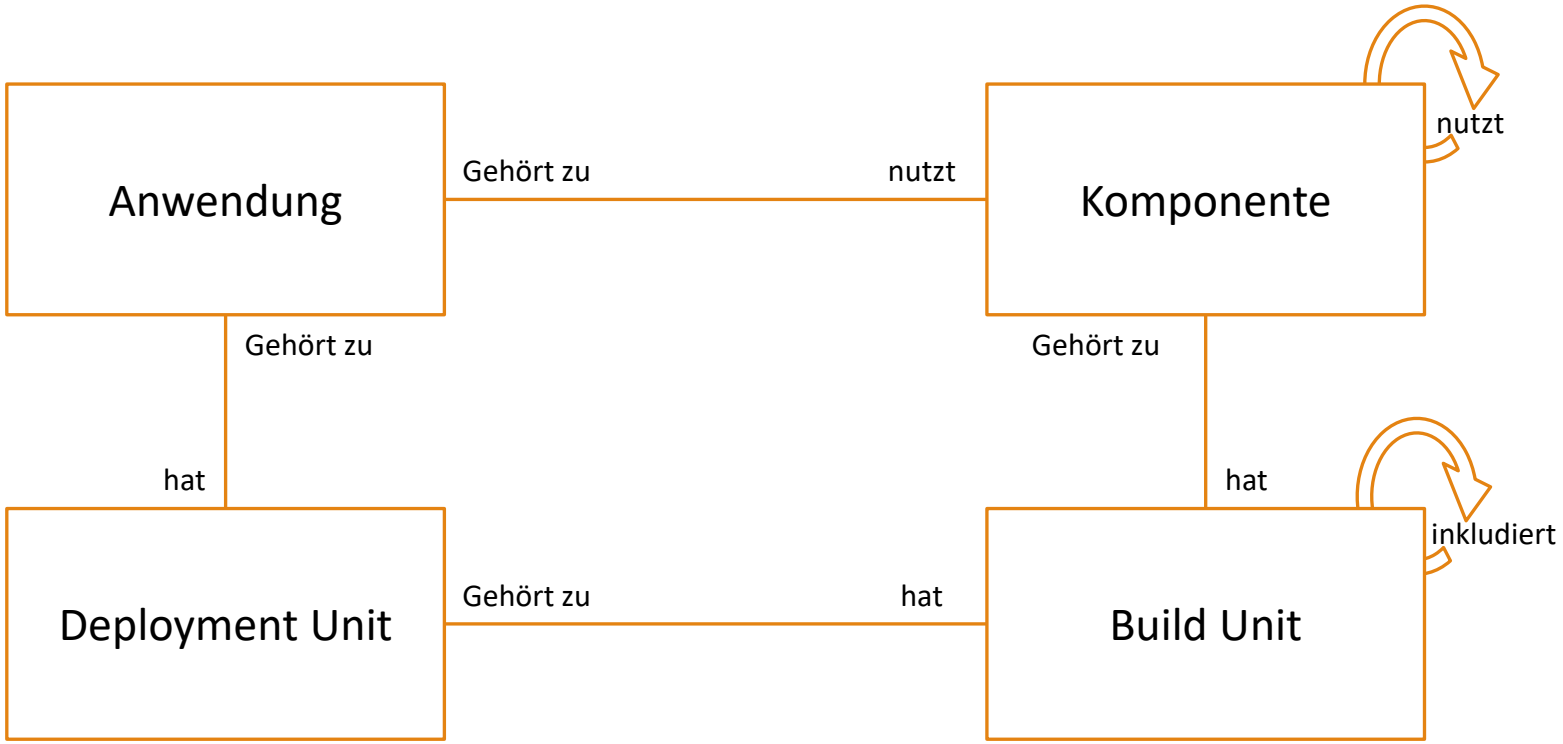
Kann Komponenten anderer Anwendungen enthalten, wenn z.B. Schnittstellen dazu existieren

Gehört immer zu genau einer Anwendung, außer es handelt sich um Framework- oder Plattformkomponenten

Orientiert sich i.d.R. am Installationsziel

Komponentenmodell

Vereinfachte Darstellung nach [SP17] S. 386



Schnittstellen

„Schnittstellen sind Spezifikationen der als legitim betrachteten Annahmen von Komponenten übereinander.“

[SP17] S. 397 bzw. David Parnas

Beschreiben Lieferungen zwischen den Komponenten:

- Was muss geliefert werden?
- Was darf erwartet werden?

Vorbedingungen: davon darf ausgegangen werden

Nachbedingungen: das muss erfüllt werden

Invarianten

Konspirationen

Komponente trifft Annahmen über das Verhalten einer anderen Komponente

Ist in keiner Schnittstelle oder Architekturspezifikation definiert

Annahme ist illegal und wird als „Conspiracy“ bezeichnet

Hohe Auswirkung auf Änderbarkeit von Software (Qualität)

Arten von Schnittstellen

Prozedurale und objektorientiert Schnittstellen (API)

Synchrone oder asynchrone Kommunikationsschnittstellen (RPC, MQ)

Datenbankschnittstellen (SQL)

File-Schnittstellen (bspw. mittels FTP)

Benutzerschnittstellen

…

Schnittstellenkompatibilität

Semantische Ebene

- Fachlichkeit, fachliches Datenmodell

Syntaktische Ebene

- Abbildung der Fachlichkeit in programmatisch korrekter Form

Binäre Ebene

- Repräsentation der technisch abgebildeten Fachlichkeit

Umsetzung der Kommunikation

Zwischen Komponenten *innerhalb des gleichen Betriebssystemprozesses*

- *Vgl. Folien zu Verteilten Systemen / A. Tanenbaum*

Zwischen *physisch verteilten Komponenten zwischen unterschiedlichen Betriebssystemprozesses*

Zwischen *physisch verteilten Komponenten, die über das Netzwerk*, d.h. über einen Netzwerk-Protokoll-Stack verläuft

Einige typischen Schnittstellenstandards schauen wir uns noch genauer an!

Komponentenaufteilung - eine Frage der Größe

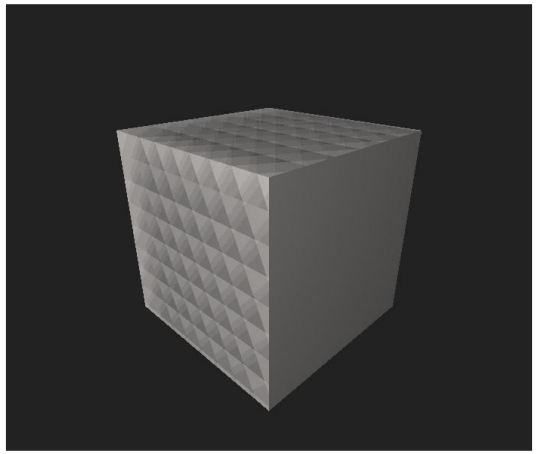
Es geht um die beherrschbare Komplexität einer Komponente (und daher implizit auch um die beherrschbare Komplexität der Anwendung)

Die Komplexität einer Komponente steht in direktem Zusammenhang mit ihrer Größe

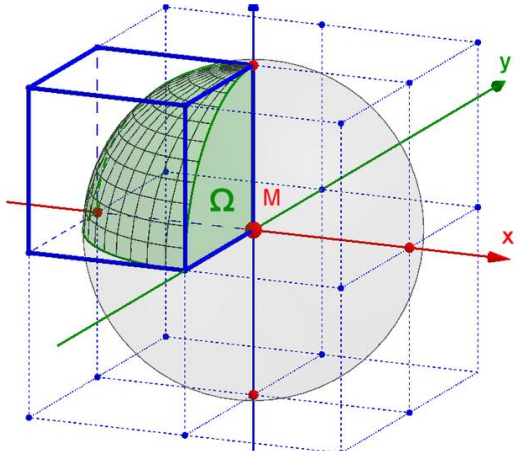
Teile-und-Herrsche Prinzip

Es gibt keine allgemeingültige Definition, sondern nur Muster und Best-Practises

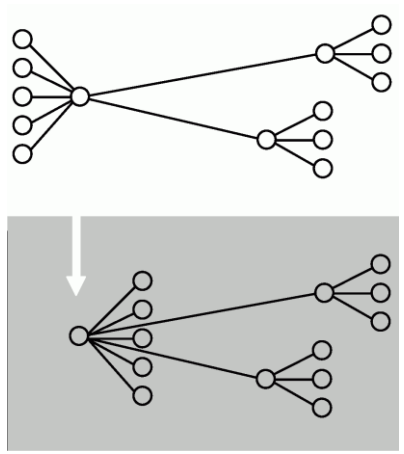
Komponentengrößen



Monolith



Komponenten



„Meer von Klassen“

Zwoelefant, CC0, via Wikimedia Commons

Petrus3743, CC BY-SA 4.0 <https://creativecommons.org/licenses/by-sa/4.0>, via Wikimedia Commons

Gemeinfrei, https://commons.wikimedia.org/w/index.php?curid=44119

Optimale Größe

„Ein Komponente sollte eine Größe haben, bei der sie von einem einzelnen Mitarbeiter noch vollständig verstanden werden kann.“

Alternativ:

- Team anstelle Mitarbeiter – aber wie groß darf dann das Team sein?
- Problem wiederholt sich, vgl. DevOps bzw. Agile Development
- „…eine Pizza sollte zum Essen für ein Team reichen…“
- Siehe auch: https://www.heise.de/newsticker/meldung/Teamzusammensetzung-Eine-grosse-Pizza-sollte-fuer-alle-reichen-4484501.html

Wie strukturieren?

Um komplexe Anwendungen in beherrschbare Komponenten aufzuteilen und diese dann verteilt anzuordnen (und auszuführen), ist eine Strategie zur Strukturierung notwendig!

Möglichkeiten:

- Größe (haben wir eben diskutiert)
- Organisation
- Gemeinsamkeiten
- Datenbewirtschaftung
- Änderungsdrücke
- Einhaltung von Architekturvorgaben

Organisation

Zuständigkeiten und Budget-Hoheiten

Änderungen an der Anwendung durch ein Team durchführbar

Klare Zuordnung dieses Teams

- Damit es zu keinen Streitigkeiten kommt.

Knowhow / Kompetenzen

Prozesse

Hier liegt jedoch eine sehr aktuelle Problematik:

- Digitale Transformation
- Conways‘ Law
- Prozesse unklar, Modelle veraltet
- Zuständigkeiten nicht geklärt
- Budgets veraltet (=Governance veraltet)
- …

Daher strukturiert in vielen Fällen die IT die Organisation (neu) und nicht umgekehrt!

Gemeinsamkeiten

Use Cases, User Stories, User Context Description, User Journeys

Beispielkriterien:

- Benötigtes Wissen
- Prozessuale Aspekte (Datenverarbeitung)
- Nutzerrollen, Nutzergruppen
- Organisatorische Zuständigkeiten
- Geplante Nutzungsdauer

Fachliches und technisches Wissen kapseln → Effektivität und Effizienz steigern

Hohe Kohäsion

Datenbewirtschaftung

Logische Zusammengehörigkeit:

- Erzeugung, Verwendung, Veränderung (~CRUD)

Fachliches Datenmodell → Daten geriebener Ansatz

Problem: fachliche Datenmodelle i.d.R. veraltet!

Erfordert in der Praxis eine Neumodellierung der Geschäftsobjekte und –prozesse

Sehr guter Ansatz für Dokumenten- und Workflow basierte Systeme

Änderungsdruck

Entkopplung bzw. „separation of concerns“

Häufigkeit von Änderungen als Kriterium des „Trennens“

Eine Komponente sollte nicht unabhängigen Änderungsdrücken unterliegen

Dazu müssen Lebenszyklen bekannt sein und verwaltet werden:

- Demand Management
- Configuration Management

In vielen Organisationen nicht vorhanden oder „unreif“!

Ausblick ITAM

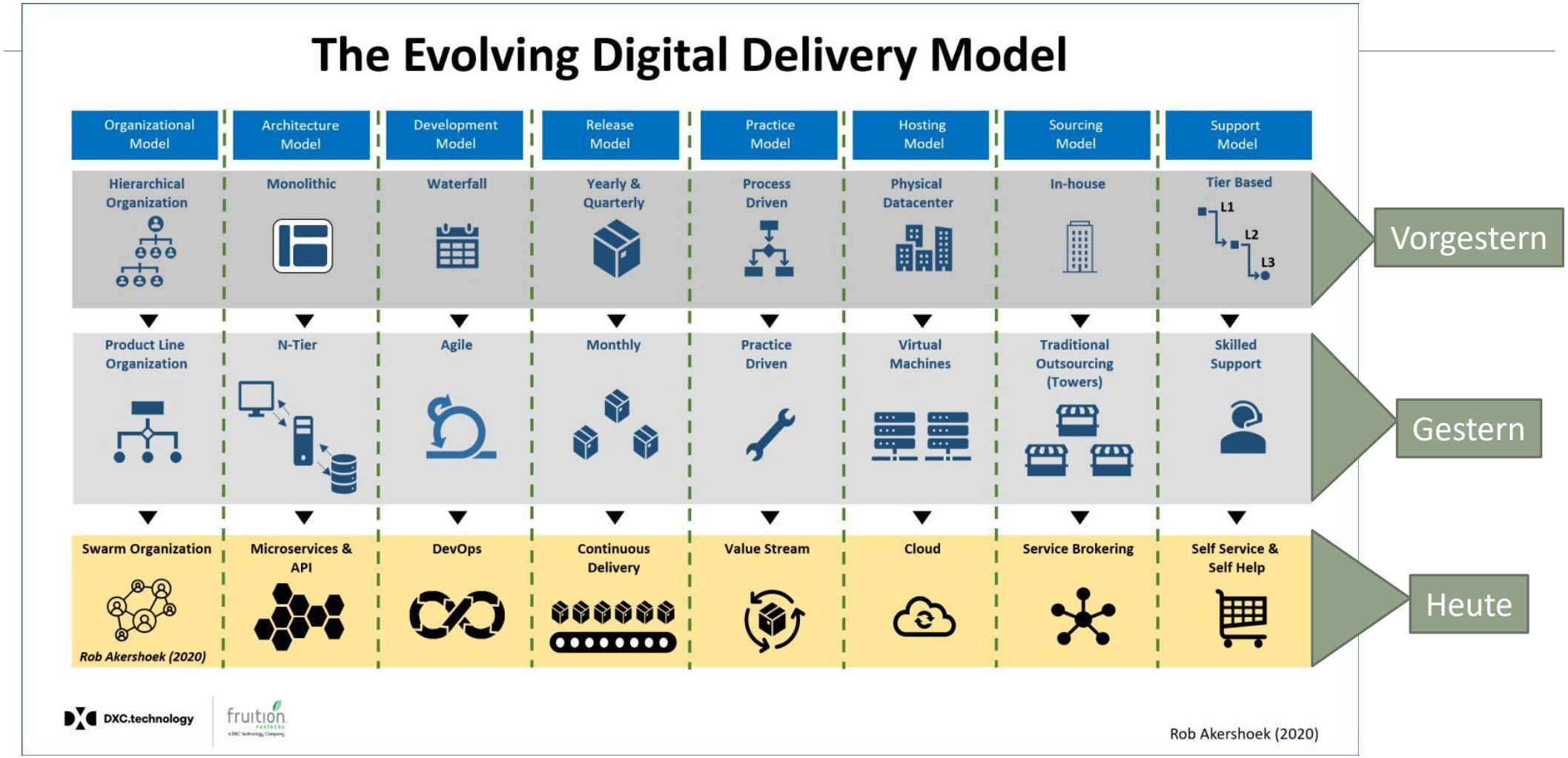
IT Architektur Management

Ja – es ist so komplex!

Frameworks und Referenzarchitekturen

- ITIL v4
- SAFe
- IT4IT
- TOGAF
- COBIT
- …

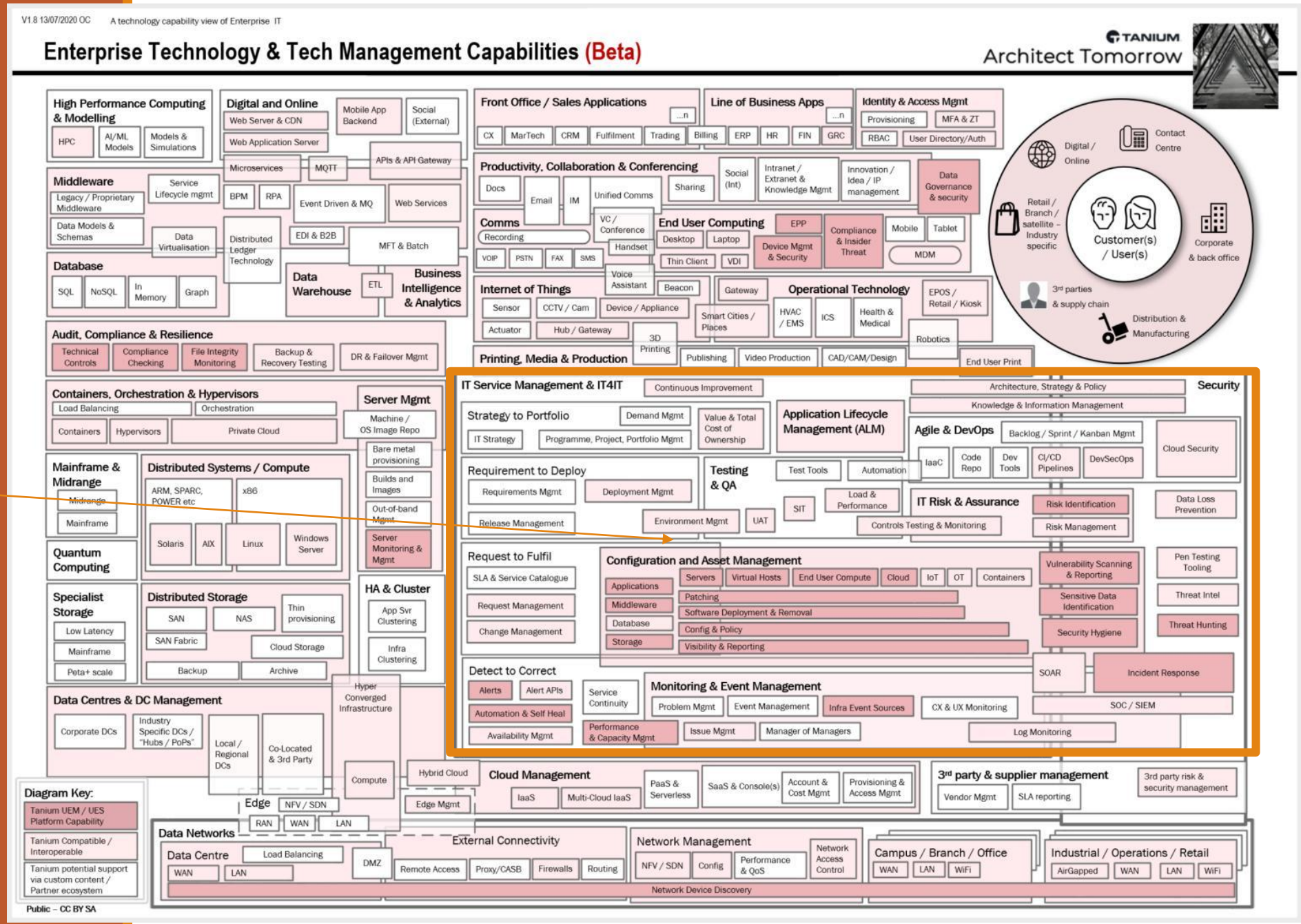
Ausblick – Rob Akershoek „Change“

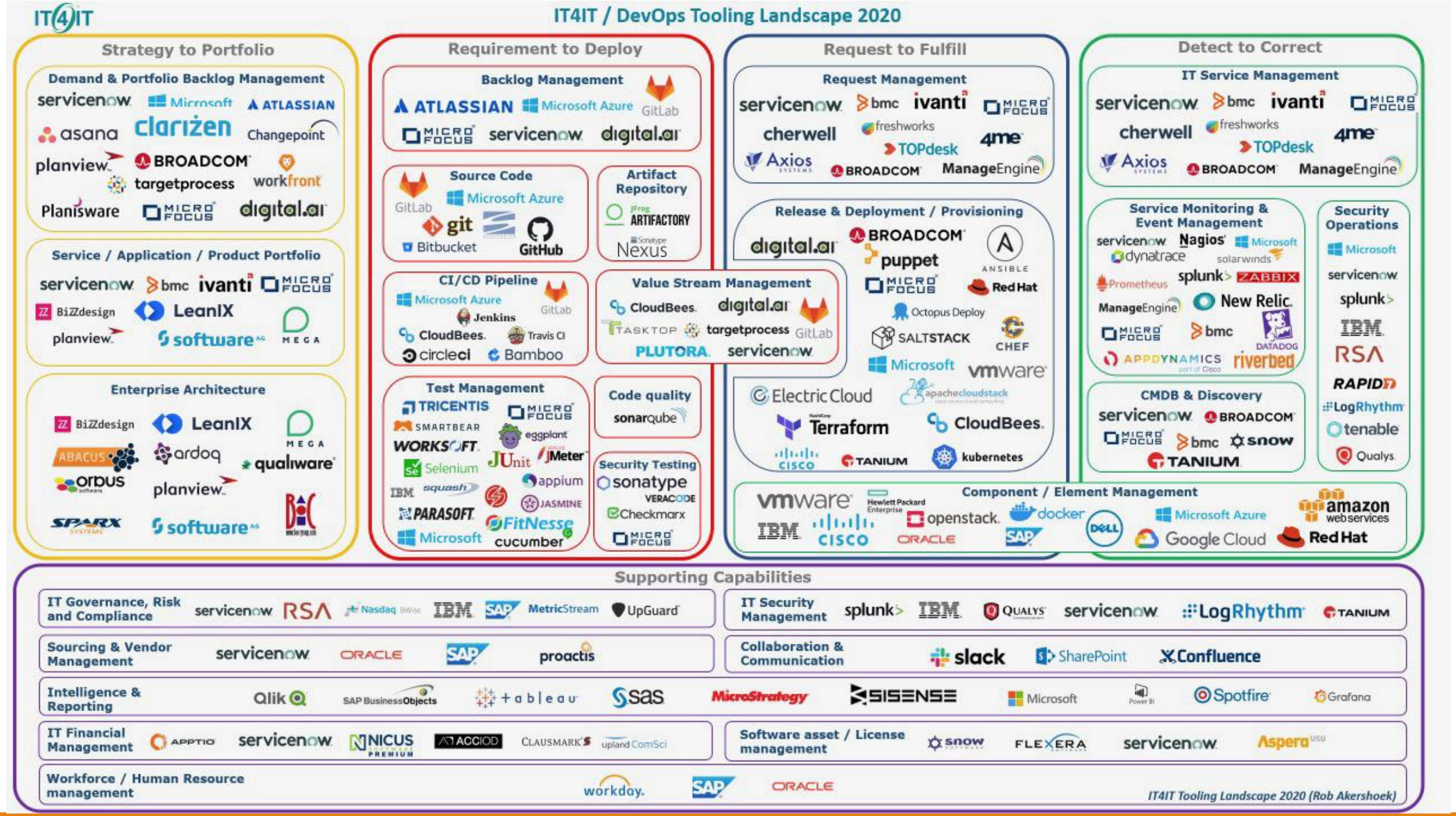


Ausblick

Enterprise Technology Map (Capabilities)

Tools on next slide…




IT4IT
IT4IT / DevOps Tooling Landscape 2020
Strategy to Portfolio
Demand & Portfolio Backlog Management
servicenow
Microsoft
ATLASSIAN
asana
clarizen
Changepoint
planview
BROADCOM
targetprocess
workfront
Planisware
MICRO FOCUS
digital.ai
Service / Application / Product Portfolio
servicenow
bmc
ivanti
MICRO FOCUS
BiZZdesign
LeanIX
planview
software AG
MEGA
Enterprise Architecture
BiZZdesign
LeanIX
MEGA
ABACUS
ardoq
qualiware
orbus
planview
SPARX SYSTEMS
software AG
Requirement to Deploy
Backlog Management
ATLASSIAN
Microsoft Azure
GitLab
MICRO FOCUS
servicenow
digital.ai
Source Code
GitLab
Microsoft Azure
git
Bitbucket
GitHub
Artifact Repository
JFrog ARTIFACTORY
Sonatype Nexus
CI/CD Pipeline
Microsoft Azure
GitLab
Jenkins
CloudBees.
Travis CI
circleci
Bamboo
Value Stream Management
CloudBees.
digital.ai
TASKTOP
targetprocess
GitLab
PLUTORA
servicenow
Test Management
TRICENTIS
MICRO FOCUS
SMARTBEAR
eggplant
WORKSOFT
JUnit
JMeter
Selenium
IBM
squash
appium
JASMINE
PARASOFT
FitNesse
Microsoft
cucumber
Code quality
sonarqube
Security Testing
sonatype
VERACODE
Checkmarx
MICRO FOCUS
Request to Fulfill
Request Management
servicenow
bmc
ivanti
MICRO FOCUS
cherwell
freshworks
TOPdesk
4me
Axios SYSTEMS
BROADCOM
ManageEngine
Release & Deployment / Provisioning
digital.ai
BROADCOM
puppet
ANSIBLE
MICRO FOCUS
Red Hat
Octopus Deploy
SALTSTACK
CHEF
Microsoft
vmware
Electric Cloud
apachecloudstack
Terraform
CloudBees.
CISCO
TANIUM
kubernetes
Component / Element Management
vmware
Hewlett Packard Enterprise
openstack.
docker
IBM
CISCO
ORACLE
SAP
DELL
Microsoft Azure
amazon web services
Google Cloud
Red Hat
Detect to Correct
IT Service Management
servicenow
bmc
ivanti
MICRO FOCUS
cherwell
freshworks
TOPdesk
4me
Axios SYSTEMS
BROADCOM
ManageEngine
Service Monitoring & Event Management
servicenow
Nagios
Microsoft
dynatrace
solarwinds
Prometheus
splunk>
ZABBIX
ManageEngine
New Relic.
MICRO FOCUS
bmc
DATADOG
APPDYNAMICS
riverbed
CMDB & Discovery
servicenow
BROADCOM
MICRO FOCUS
bmc
snow
TANIUM
Security Operations
Microsoft
servicenow
splunk>
IBM
RSA
RAPID7
LogRhythm
tenable
Qualys
Supporting Capabilities
IT Governance, Risk and Compliance
servicenow
RSA
Nasdaq BWise
IBM
SAP
MetricStream
UpGuard
IT Security Management
splunk>
IBM
QUALYS
servicenow
LogRhythm
TANIUM
Sourcing & Vendor Management
servicenow
ORACLE
SAP
proactis
Collaboration & Communication
slack
SharePoint
Confluence
Intelligence & Reporting
Qlik
SAP BusinessObjects
tableau
sas
MicroStrategy
SISENSE
Microsoft
Power BI
Spotfire
Grafana
IT Financial Management
APPTIO
servicenow
NICUS PREMIUM
ACCIOD
CLAUSMARK
uplandComSci
Software asset / License management
snow
FLEXERA
servicenow
Aspera USU
Workforce / Human Resource management
workday.
SAP
ORACLE
IT4IT Tooling Landscape 2020 (Rob Akershoek)

Architekturvorgaben

Vorgeschriebene Muster- und Referenzarchitekturen, z.B. Client-Server

Standards hinsichtlich Plattformen und Technologien

- Dadurch kann die Art der Komponentenbildung maßgeblich beeinflusst werden

Technologische Treiber

- Aktuell: Architekturmuster (MSA), Container (Docker) und Plattformen (Kubernetes)

Achtung!

- Architekturvorgaben nur sinnvoll, wenn ausgereiftes Architekturmanagement vorhanden
- Technologien sehr kurzlebig – häufige Anpassungen können extrem teuer werden
- Kompetenzen oft nicht vorhanden

Entwicklungsansätze

„I know what I want when I see it!“

Klassische Entwicklungsansätze funktionieren heute nur noch begrenzt

Aktueller Fokus auf: agile und iterative Ansätze

Aufteilung in fachliche Komponenten und deren technische Umsetzung

- Erfordert ein fachliches Modell (Prozesse, Organisation, Kommunikation, …)
- Erfordert ein technisches Modell (Technologien, Plattformen, Architekturmuster, …)

Richtig: Nutzer zentrierte Entwicklung, z.B. in ISO 9241-210 (UX)

→ Mehr im Modul Software-Engineering

Frühzeitiger Fokus auf „Service“

Fachliche Services (Geschäftsservices)

Fachliche Leistungen im geschäftlichen Kontext optimal zerlegen

- Prozessfunktionalität
- Prozessgrenzen
- Geschäftsobjekte und deren Bewirtschaftung
- Zuständigkeiten

Ergibt ein neues Bild der Geschäftsarchitektur (Services & Capabilities)

Fähigkeiten werden durch IT-Services implementiert

Wenige und fachlich stabile Schnittstellen

Softwarearchitektur

Strukturierung einer Anwendung

Horizontale und vertikale Aspekte

Softwarearchitektur einer verteilten Anwendung überschneidet sich mit IT-Architektur einer Anwendungslandschaft

→Mehr im Modul Software-Engineering

Horizontal (Schichtenmodell):

- View Layer
- Controller Layer
- Model Layer
- Service Layer
- Business Layer
- Persistence & Integration Layer

Vertikal (schichtübergreifende Funktionalitäten)

- Transaktionsbehandlung
- Systemmeldungen
- Protokollierung
- Internationalisierung
- Sicherheit
- Mandantenfähigkeit
- Frontend-Backend
- Verteilungsanforderungen