Hochschule Konstanz
Fakultät Informatik

Software-Architektur WIN-SSE

Prof. Dr. Johannes C. Schneider

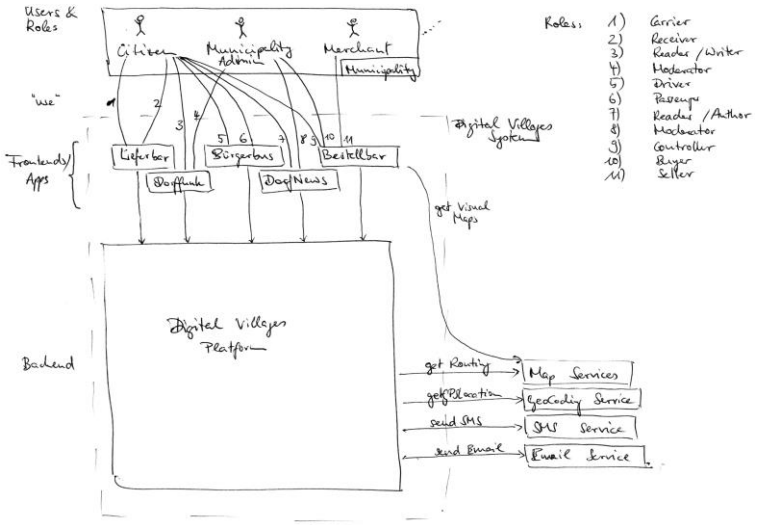
Hallo Architekt:innen!

Über mich

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

DIALOGIKA Wir erfinden Software

Fraunhofer IESE



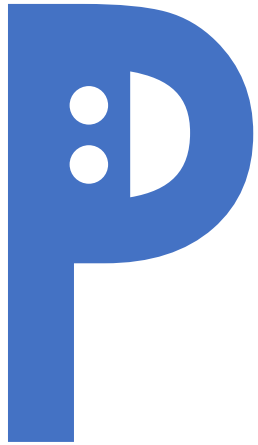
- Studium und Promotion an der TU Kaiserslautern
- Software-Entwickler (Java Enterprise Technologies)
- Teamleiter Webentwicklung (Typescript/React)
- Architekturberater (Industriekunden: Banken, Buchhaltung, Kfz-Kundendienst)
- Fraunhofer-Akademie-Dozent für Architekturseminar
- Einer der Hauptarchitekten der "Digitalen Dörfer"
  - Event-basierte Architektur
  - Multimandanten- und Multiapplikations-Ökosystem
  - AWS-Cloud-Technologien
  - Xamarin Cross-Platform mobile Anwendungen
  - React- und Angular-Webanwendungen
  - Java Spring Backend

Über mich

Hochschule Konstanz
Fakultät Informatik

- Seit September 2020 Professor für Softwareentwicklung an der HTWG.
- Weitere Vorlesungen: Einführung in die Programmierung (WIN/BIT) und Software Engineering 1 (WIN/BIT)
- Forschungsinteressen:
  - **Software-Architektur**: Wie entwirft man Softwaresysteme so, dass sie den Qualitätsanforderungen entsprechen? **Wie dokumentiert man Software-Architektur effizient?**
  - **Cloud-Technologien**: Was sind die Vorteile des Cloud-Deployment? Was sind die Möglichkeiten des Serverless Computing?
  - **Sprachgesteuerte Benutzerschnittstellen (Voice User Interfaces)**: Was sind die möglichen Anwendungsbereiche? Wie gestaltet man Sprachassistenten, die gerne genutzt werden? Was sind die Alternativen zu Google, Alexa und Siri – und wie gut sind sie?
  - **Wortkombinatorik**: Welche Algorithmen können für typische Probleme verwendet werden (z. B. erweiterte reguläre Ausdrücke)? Wie können sie schnell ausgeführt werden? Wie können sie effizient gemacht werden?

Über Sie



- pingo.coactum.de, Code 871775 eingeben
  - Kann auch vom Smartphone aus genutzt werden

- https://pingo.coactum.de/871775

Haftungsausschluss: Lesen Sie die Datenschutzbestimmungen und entscheiden Sie selbst, ob Sie an den Umfragen teilnehmen möchten.

Erwartungen

Was Sie erwartet:

- Grundlagen des Software-Engineering
- In der Praxis erprobte Methode für Softwarearchitekturarbeit (Architecture Decomposition Framework und Fraunhofer ACES)
- Verschiedene Beispiele für Architekturstile
- Erfahrungen eines industrieerfahrenen Softwareentwickler und -architekten, der mit vielen modernen Technologien und Paradigmen gearbeitet hat
- Selbst praktische Erfahrung mit Softwarearchitektur sammeln anhand realistischer Beispielsysteme in den Übungen
- Methoden und Werkzeuge, die Sie bei Ihrem nächsten Projekt (an der Hochschule oder in der Industrie) einsetzen können

Was Sie nicht erwarten sollten:

- ein Programmierkurs für Fortgeschrittene
- Eine schrittweise Erklärung, was gute Architektur ist, anwendbar auf alle Systeme
  - weil es keine "gute" Architektur gibt (nur eine angemessene Architektur, und die ist von System zu System unterschiedlich)
  - denn ein guter Architekt braucht Erfahrung und Kreativität (und dafür gibt es keine Schritt-für-Schritt-Anleitung)
- **Perfekt formulierte Folien**, denn ich habe auf mehrfachen Wunsch die Folien auf dem Englischen in Deutsch übersetzt (und es hakt vielleicht noch an ein paar Stellen)

Inhalt

- Einführung und Organisation
- Die Grundlagen der Software-Engineering
- Software-Prozesse
- Requirements Engineering (Anforderungsanalyse)
- Grundlagen der Software-Architektur
- Architekturtreiber
- Architektursichten
- Architektur-Design
- Architekturstile und -muster
- Architekturdokumentation
- Architekturbewertung
- Architekturarbeit (auch in agilen Projekten)

Grundlagen: Basiswissen Software-Engineering und RE, erste drei Vorlesungen

Softwarearchitektur, Architekturmethoden und Architekturarbeit

Organisatorisches

- **Informationen und Materialien** in moodle
- **Vorlesung**: Dienstag 11:30-13:00 in O-207
- **Übung**: Mittwoch 9:45-11:15 in O-201
  - Die Tutorien finden jede Woche statt, **beginnend morgen**. Es besteht Anwesenheitspflicht, um eine gute Gruppenzusammenarbeit zu gewährleisten.
  - Sie erhalten verschiedene Aufgaben zu einem Beispielsystem.
    - Aufgaben müssen in ausreichendem Maße erledigt werden, System der gelben und roten Karten (ich erkläre das in der Übung..)
    - Teamarbeit, Teamgröße ~3-4 Studenten
    - Es gibt immer wieder Übungen zum Austausche zwischen Gruppen, in denen eigene Ergebnisse anderen Gruppen präsentiert werden
- **Schriftliche Prüfung** (**Klausur** von 90 Minuten), Beispiel-Inhalte in Moodle unter Klausur, gemeinsame Besprechung eines Beispielsytems in letzter Vorlesung

Empfohlene Ressourcen (1/2)

Software-Engineering im Allgemeinen:

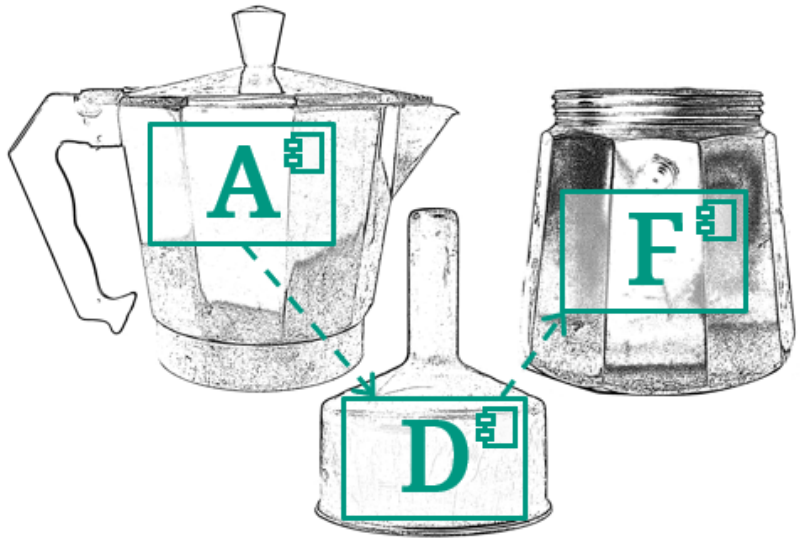
– Ian Sommerville, Software-Engineering

ADF-Architektur-Methode

– Die mit © Fraunhofer IESE gekennzeichneten Folien

– Überblick über Design, Sichten und Dokumentation in https://github.com/architecture-decomposition-framework/welcome-to-adf (auf Englisch, teilweise auch Deutsch)

ansehen!!!



Architecture Decomposition Framewor

Welcome! Good you made it here! This is the entry point for learning more abo **Framework (ADF)**, an architecture view framework invented at Fraunhofer IESE, usable freely by everyone.

Getting started

- Our FAQs
- A article about the ADF in the IESE Blog
  - Designing software architectures more easily and documenting them n (machine-translated to English)
  - Softwarearchitekturen einfacher designen und verständlicher dokumen (German original version)
- Webinar Architekturdokumentation mit dem ADF (in German)

Reference guides

- Architecture Design with the Architecture Decomposition Framework (ADF)

Empfohlene Ressourcen (2/2)

Softwarearchitekturstile und -muster

- Gernot Starke: Effektive Softwarearchitekturen: Ein praktischer Leitfaden

Weitere Lektüre

- Stefan Zörner, "Softwarearchitekturen dokumentieren und kommunizieren"

Empfohlene Tools

- **UI-Prototyping-Werkzeuge**
  - Pencil https://pencil.evolus.vn/Downloads.html (Open Source)
  - Balsamiq https://balsamiq.com/ (Kommerziell)
  - Figma https://www.figma.com/ (Kommerziell, aber in Basisversion kostenlos)
- **Grafische Modellierungswerkzeuge (Architektursichten)**
  - Diagrams.net (ehemals draw.io): https://app.diagrams.net/ (Web-App mit Möglichkeiten zur Zusammenarbeit mit Google Drive oder OneDrive), https://github.com/jgraph/drawio-desktop/releases (Desktop Offline-Version)
  - In der Vorlesung nicht verwendet, aber für die spätere Arbeit an größeren Systemen empfohlen: Enterprise Architect https://www.sparxsystems.de/ mit Fraunhofer ACES Erweiterung
- **Dokumentation**
  - Visual Studio Code https://code.visualstudio.com/ zur Bearbeitung der Dokumentationsvorlage, mit Plugins
    - Markdown All-In-One
    - Draw.io-Integration
    - Marp für VS Code

Hochschule Konstanz
Fakultät Informatik

Grundlagen des Software-Engineering

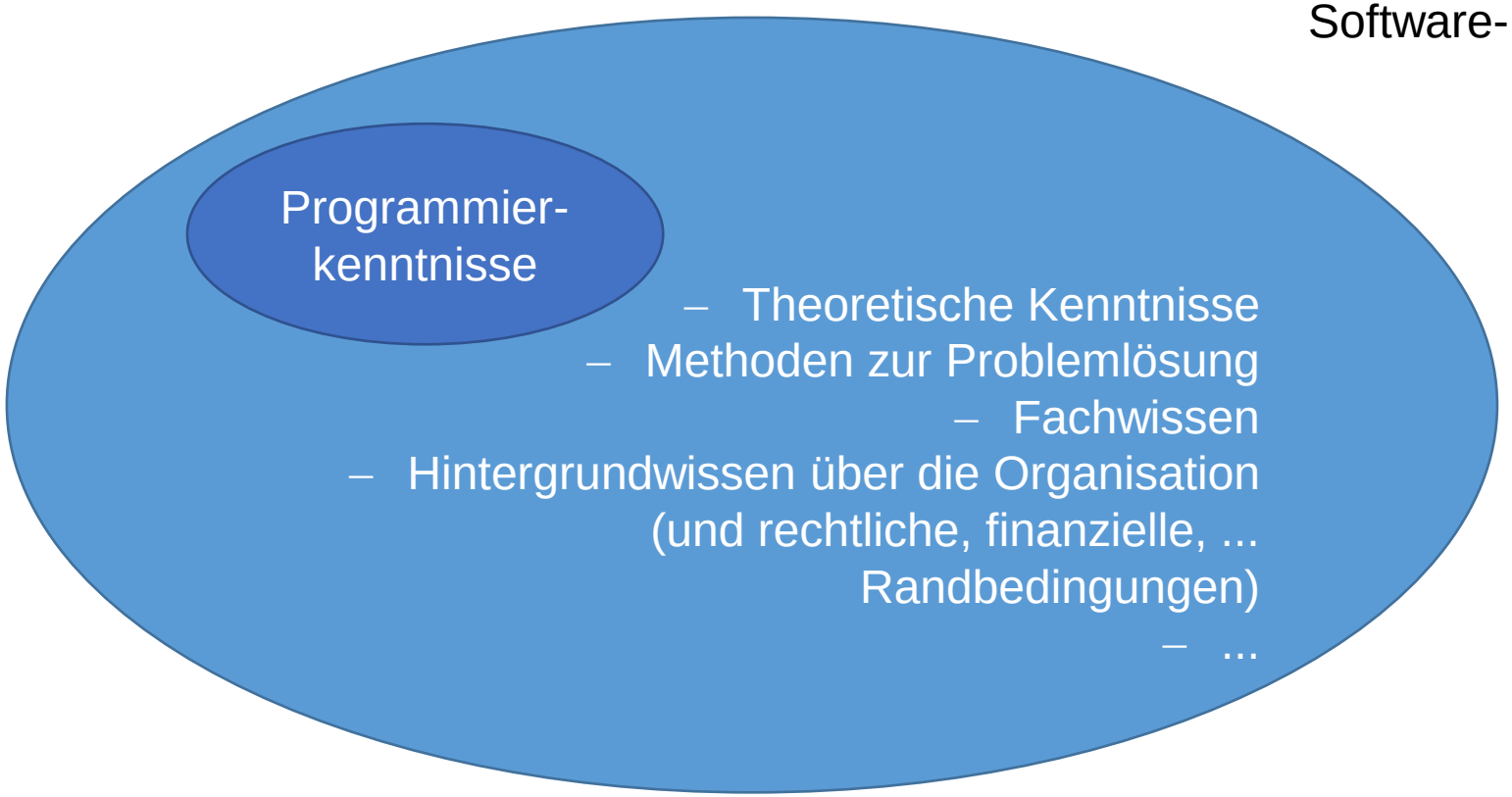
Software-Engineering

Definition von Ian Sommerville:

"*Software engineering is an **engineering discipline** that is concerned with **all aspects of software production** from the early stages of system specification through to maintaining the system after it has gone into use.*"

"Software-Engineering ist eine **Ingenieursdisziplin**, die sich mit **allen** **Aspekten der Softwareherstellung befasst**, von den frühen Phasen der Systemspezifikation bis hin zur Wartung des Systems, nachdem es in Betrieb genommen wurde."

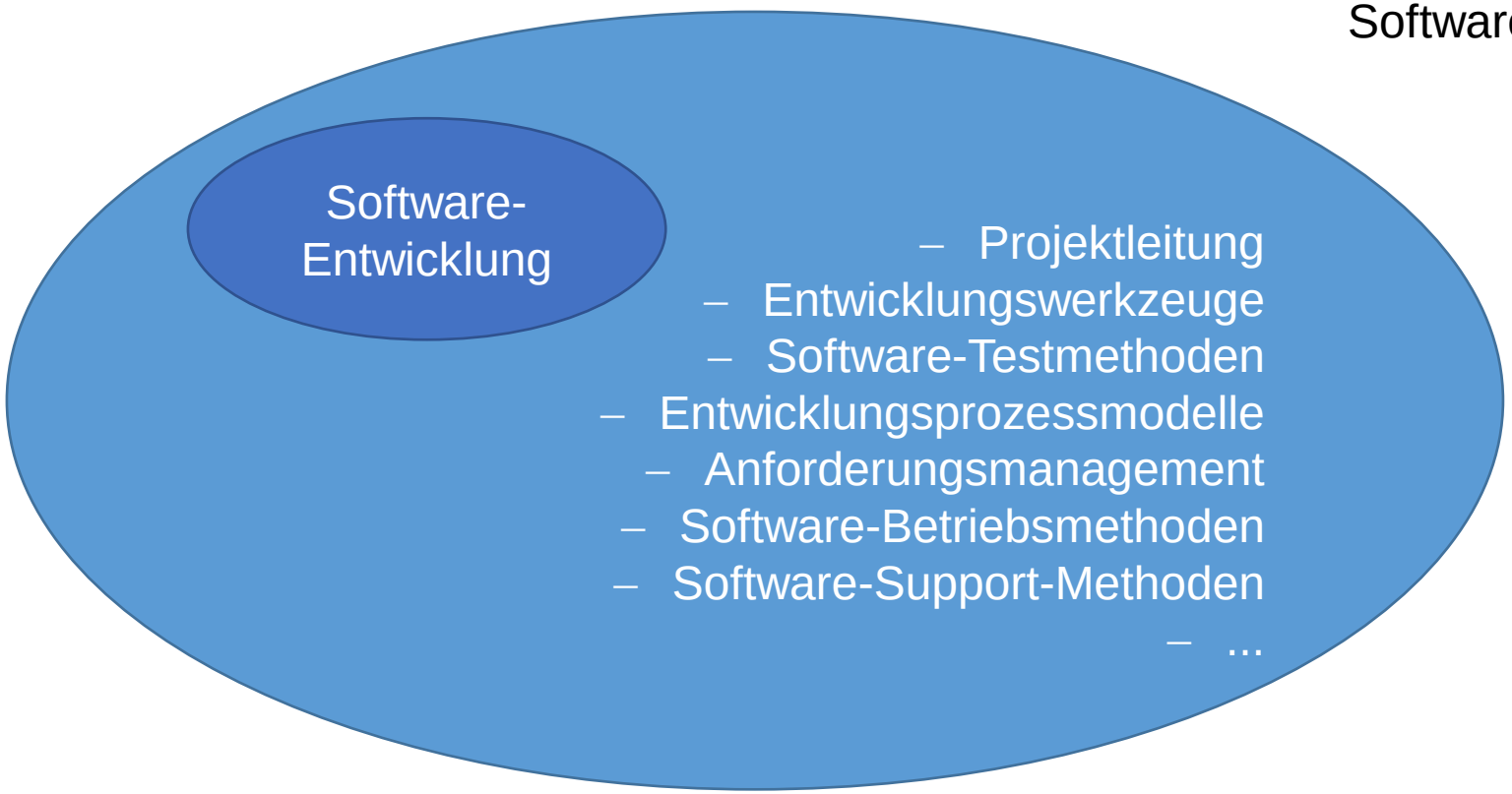
"Ingenieursdisziplin"



Software-Engineering-Fähigkeiten
und -Kenntnisse

Es geht um mehr als die Kenntnis der Programmiersprache!

"Alle Aspekte der Softwareherstellung"



Es geht um mehr als Programmieren/Coden/"Hacken"!

Bedeutung der Softwaretechnik für den Einzelnen

- Die Zahl der Softwaresysteme, die der Einzelne nutzt und **von denen er abhängig ist,** ist **groß** und **nimmt sowohl in Bezug auf die Verbreitung als auch die Komplexität** ständig **zu**:
  - Unterhaltung
  - Smart Home
  - Navigation
  - Fahrassistenzsysteme
  - Unterstützung der Medizin
  - ...

  Wir müssen in der Lage sein, **zuverlässige** und vertrauenswürdige **Systeme kostengünstig und schnell** herzustellen.

- Beispiele für Softwaresysteme, die im täglichen Leben verwendet werden:






https://pixabay.com

Bedeutung der Softwaretechnik für Unternehmen

- Durch die **Digitalisierung** nutzen immer mehr **Unternehmen aller Bereichen** (nicht nur Softwareunternehmen) Software in einem Maße, dass ihr Kerngeschäft **vollständig von der** Zuverlässigkeit der **Software abhängt**

Digitalisierung

https://pixabay.com / https://unsplash.com/photos/2cLrUfdJVIc

Scheitern von Softwareprojekten, Komplexität

- Manche Projekte scheitern allein an der bloßen **Komplexität der Systeme** und ihrer Anforderungen (man denke z.B. an die großen Verzögerungen beim deutschen LKW-Maut-System)

Scheitern von Softwareprojekten, mangelnde SE-Kenntnisse

- Andere Projekte scheitern an **mangelnder Kenntnis von Software-Engineering-Methoden**.
- Ein Grund dafür: Viele **Unternehmen** benötigen im Zuge der Weiterentwicklung ihrer Produkte und Dienstleistungen (auch durch die Digitalisierung) massiv **Software-Lösungen,** haben aber wenig geschultes Personal oder keine soliden SE-Prozesse.

blau: - Einfacher Einstieg - von allen als unbedingt nötig akzeptiert (auch in Nicht-Software-Unternehmen)

grau: - wird auch benötigt, fehlt aber oft

Kosten von Software-Produktion

- **Mythos 1**: **Die Kosten eines Software-Projekts bestehen hauptsächlich aus den Aufwänden für die Implementierung.**
  - **Falsch**: Etwa 60% der Softwarekosten sind Entwicklungskosten, 40% sind Testkosten.

    → Es ist nötig, in systematische Qualitätssicherungsmethoden investieren, da die Software sonst im Betrieb fehlerhaft ist (mit potentiell hohen Folgekosten)
- **Mythos 2**: **Die Haupt-Kosten fallen an, um die erste Kundenversion mit allen Features zu erstellen.**
  - **Falsch**: Die Kosten für die anfängliche Systementwicklung sind wesentlich geringer als die Kosten für eine spätere Änderung der Software - insbesondere bei kundenspezifischen Produkten ("Invididualsoftware")

    → Ein systematischer Software-Engineering-Ansatz zahlt sich sehr bald aus

→ Von Anfang an die Kosten für das Testen und Ändern der Software berücksichtigen!

Ian Sommervilles Software-Prozess-Aktivitäten

- **Software-Spezifikation**
  - Definition der zu erstellenden Software und
  - Randbedingungen für den Software-Betrieb
  - kommt von Kunden und Ingenieuren

- **Software-Entwicklung**
  - **Entwurf** und **Implementierung der** Software

- **Software-Validierung**
  - die Software wird geprüft, um sicherzustellen, dass sie den Anforderungen des Kunden entspricht

- **Software-Weiterentwicklung**
  - Anpassungen an sich ändernde Kunden- und Marktanforderungen.

Eine weitere Aktivität: Betrieb der Software

- **Software-Spezifikation**
  - Definition der zu erstellenden Software und
  - Einschränkungen für seinen Betrieb
  - von Kunden und Ingenieuren

- **Software-Entwicklung**
  - **Entwurf** und **Implementierung der** Software

- **Software-Validierung**
  - die Software wird geprüft, um sicherzustellen, dass sie den Anforderungen des Kunden entspricht

Und was passiert hier dazwischen?

- **Softare-Weiterentwicklung**
  - Anpassungen an sich ändernde Kunden- und Marktanforderungen.

Sommervilles Kategorisierung geht davon aus, dass die Software an den Kunden geliefert und von ihm selbst betrieben wird.
Immer mehr Softwareprodukte werden jedoch als Dienstleistung angeboten und somit vom entwickelnden Unternehmen betrieben.
Oder zumindest werden sie von einem Serviceteam deployt ("bereitgestellt").

Software-Engineering-Aktivitäten

- Requirements Engineering (Erhebung, Spezifikation und Überprüfung von Anforderungen)
- Entwurf und Dokumentation der Softwarearchitektur
- Gestaltung der Benutzeroberfläche (UI-Design)
- Software-Entwicklung (Implementierung)
- Software-Tests (Qualitätssicherung, QA)
- Software-Bereitstellung (Auslieferung des Softwareprodukts, Umsetzung der Software)
- Software-Betrieb (Überwachung und Fehlersuche während der Ausführung des Systems)
- Software-Weiterentwicklung (Erfassung und Priorisierung von Änderungswünschen und neuen Funktionen)

Frage an alle: Beispiel WhatsApp

- Requirements Engineering (Erhebung, Spezifikation und Überprüfung von Anforderungen)
- Entwurf und Dokumentation der Softwarearchitektur
- UI-Design
- Software-Entwicklung (Implementierung)
- Software-Tests (Qualitätssicherung)
- Software-Bereitstellung (Auslieferung des Softwareprodukts, Umsetzung der Software)
- Software-Betrieb (Überwachung und Fehlersuche während der Ausführung des Systems)
- Software-Weiterentwicklung (Erfassung und Priorisierung von Änderungswünschen und neuen Funktionen)

Wir betrachten eine Messaging-App wie WhatsApp:

- Braucht man alle Aktivitäten?
- Was sind Beispiele für Aktivitäten/Ergebnisse aus den Punkten links?
- Welche Teams werden benötigt? Wen würden Sie einstellen?
- Was muss alles spezifiziert werden? Getestet?

Wie lassen sich diese Aktivitäten den Rollen/Teams zuordnen?

- Requirements Engineering (Erhebung, Spezifikation und Überprüfung von Anforderungen)
- Entwurf und Dokumentation der Softwarearchitektur
- UI-Design
- Software-Entwicklung (Implementierung)
- Software-Tests (Qualitätssicherung)
- Software-Bereitstellung (Auslieferung des Softwareprodukts, Umsetzung der Software)
- Software-Betrieb (Überwachung und Fehlersuche während der Ausführung des Systems)
- Software-Weiterentwicklung (Erfassung und Priorisierung von Änderungswünschen und neuen Funktionen)

Foto von Jason Goodman auf Unsplash

"Eine:r macht alles"

- Requirements Engineering (Erhebung, Spezifikation und Überprüfung von Anforderungen)
- Entwurf und Dokumentation der Softwarearchitektur
- UI-Design
- Software-Entwicklung (Implementierung)
- Software-Tests (Qualitätssicherung)
- Software-Bereitstellung (Auslieferung des Softwareprodukts, Umsetzung der Software)
- Software-Betrieb (Überwachung und Fehlersuche während der Ausführung des Systems)
- Software-Weiterentwicklung (Erfassung und Priorisierung von Änderungswünschen und neuen Funktionen)

- blau: gemacht von Chris dem allmächtigen Entwickler
- grau: nicht getan (nur vage oder implizit)

Dies ist häufiger der Fall, als Sie vielleicht denken!

Erfolgreiches Softwareprodukt?
- eher nicht!

Teammitglieder für jede Aktivität

– Requirements Engineering (Erhebung, Spezifikation und Überprüfung von Anforderungen)	– Alex, Anforderungsingenieur:in
– Entwurf und Dokumentation der Softwarearchitektur	– Conny, Architekt:in
– UI-Design	– Charlie, UI-Designer:in
– Software-Entwicklung (Implementierung)	– Kim und Nicci, die Frontend-Entwickler:innen – Patrice und Robin, die Backend-Entwickler:innen
– Software-Tests (Qualitätssicherung)	– Sasha und Tony, die Software-Tester:innen
– Software-Bereitstellung (Auslieferung des Softwareprodukts, Umsetzung der Software) – Software-Betrieb (Überwachung und Fehlersuche während der Ausführung des Systems)	– Uli und Yves, die Software-Betreiber:innen
– Software-Weiterentwicklung (Erfassung und Priorisierung von Änderungswünschen und neuen Funktionen)	– Alex, zusammen mit Chris, Projektleiter:ín

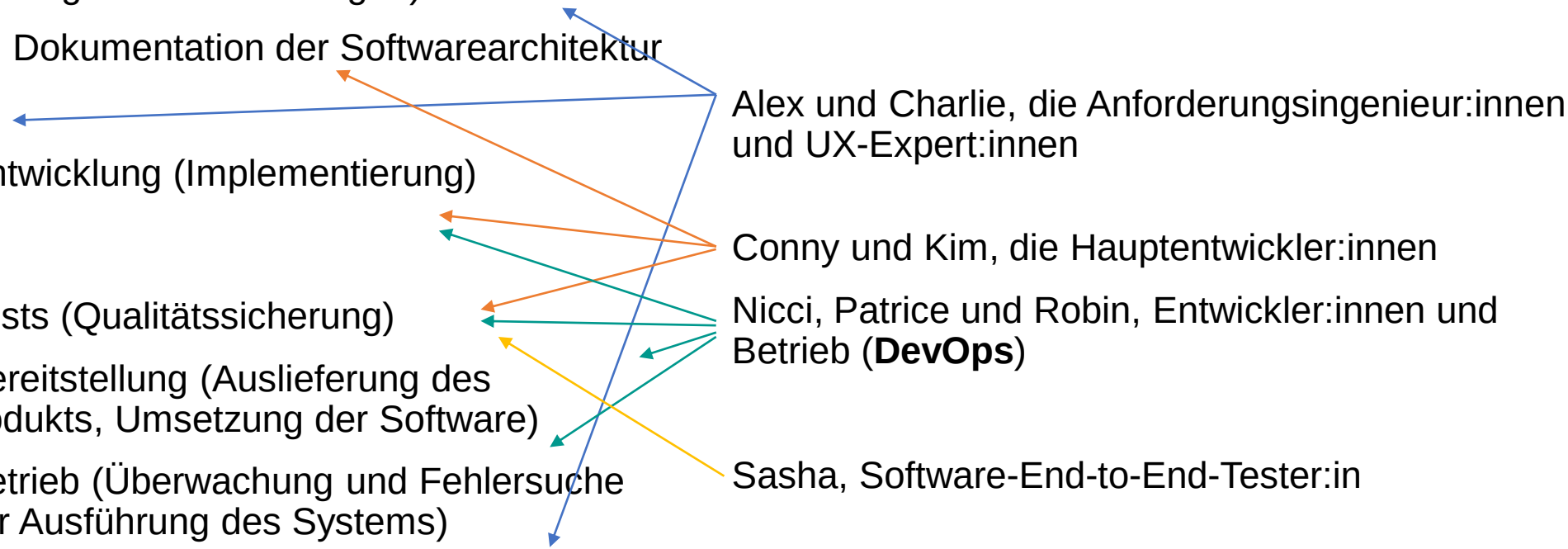
Teammitglieder für jede Aktivität

– Requirements Engineering (Erhebung, Spezifikation und Überprüfung von Anforderungen)	– Alex, Anforderungsingenieur:in
– Entwurf und Dokumentation der Softwarearchitektur	– Conny, Architekt:in
– UI-Design	– Charlie, UI-Designer:in
– Software-Entwicklung (Implementierung)	– Kim und Nicci, die Frontend-Entwickler:innen – Patrice und Robin, die Backend-Entwickler:innen
– Software-Tests (Qualitätssicherung)	– Sasha und Tony, die Software-Tester:innen
– Software-Bereitstellung (Auslieferung des Softwareprodukts, Umsetzung der Software) – Software-Betrieb (Überwachung und Fehlersuche während der Ausführung des Systems)	– Uli und Yves, die Software-Betreiber:innen
– Software-Weiterentwicklung (Erfassung und Priorisierung von Änderungswünschen und neuen Funktionen)	– Alex, zusammen mit Chris, Projektleiter:in

– Viel besser!
– Achtung: Wechsel zur nächsten Aktivität erfordert die Beteiligung einer neuen Person/Rolle.
  – Dadurch entsteht erhöhter Kommunikationsbedarf
  – Es ist leicht, die Verantwortung für bestimmte Dinge abzustreiten ("Das ist nicht mein Problem, der Tester hätte seine Arbeit besser machen müssen")

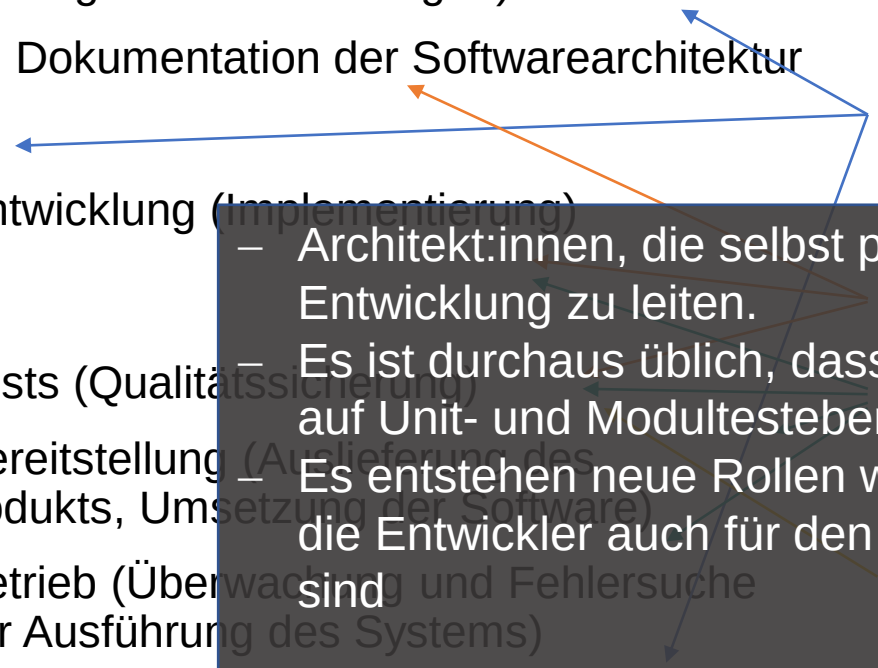
Teammitglieder mit gemischter Verantwortung

- Requirements Engineering (Erhebung, Spezifikation und Überprüfung von Anforderungen)
- Entwurf und Dokumentation der Softwarearchitektur
- UI-Design
- Software-Entwicklung (Implementierung)
- Software-Tests (Qualitätssicherung)
- Software-Bereitstellung (Auslieferung des Softwareprodukts, Umsetzung der Software)
- Software-Betrieb (Überwachung und Fehlersuche während der Ausführung des Systems)
- Weiterentwicklung (Erfassung und Priorisierung von Änderungswünschen und neuen Funktionen)



Teammitglieder mit gemischter Verantwortung

- Requirements Engineering (Erhebung, Spezifikation und Überprüfung von Anforderungen)
- Entwurf und Dokumentation der Softwarearchitektur
- UI-Design
- Software-Entwicklung (Implementierung)
- Software-Tests (Qualitätssicherung)
- Software-Bereitstellung (Auslieferung des Softwareprodukts, Umsetzung der Software)
- Software-Betrieb (Überwachung und Fehlersuche während der Ausführung des Systems)
- Weiterentwicklung (Erfassung von Änderungswünschen und neuen Features)



Alex und Charlie, die Anforderungsingenieur:innen und UX-Expert:innen

Conny und Kim, die Hauptentwickler:innen

Nicci, Patrice und Robin, Entwickler:innen und Betrieb (**DevOps**)

Sasha, Software-End-to-End-Tester:in

- Architekt:innen, die selbst programmieren, sind viel besser geeignet, die Entwicklung zu leiten.
- Es ist durchaus üblich, dass **Entwickler ihren Code auch testen** (zumindest auf Unit- und Modultestebene)
- Es entstehen neue Rollen wie **DevOps** (Entwicklung und Betrieb), bei denen die Entwickler auch für den Betrieb "ihres" Teils der Software verantwortlich sind
- Ausblick: Beim Microservice-Paradigma ist ein kleines Team als Ganzes für jede Aktivität voll verantwortlich (oft nur für einen begrenzten Teil der Funktionen des Systems - ihren "Microservice") - mehr dazu später

Arten von Software

Können Sie Beispiele nennen?

- **Generische Produkte ("allgemeine Software-Lösungen")**
  - Eigenständige Systeme, die an jeden Kunden, der sie kaufen möchte, vermarktet und verkauft werden.
  - Die Entscheidungen über die **Spezifikation** ("Features", Änderungen) werden von der **Softwareentwicklungsfirma** getroffen.
- **Maßgeschneiderte Produkte ("Individual-Software")**
  - Software, die von einem bestimmten Kunden in Auftrag gegeben wird, um seine eigenen Anforderungen zu erfüllen.
  - Die **Spezifikation** dessen, was die Software tun soll, liegt in den Händen des **Kunden,** und er trifft die Entscheidungen über erforderliche Softwareänderungen.
- **Plattformen**
  - **Systeme mit Basisfunktionalität** (Benutzerverwaltung, Vertriebs- und/oder Kommunikationskanäle, Zahlungssysteme, ...), in denen **andere Partner** Produkte, Dienstleistungen oder Software **anbieten können**
  - Kann **offen** (alles, was den Gesetzen folgt, kann angeboten werden) oder **eingeschränkt** sein (jeder Dienst muss ein strenges Annahmeverfahren durchlaufen)
    - z. B. Google Play Store gegenüber Apple App Store

Preismodelle für Softwareprodukte

Können Sie Beispiele nennen?

- Einmalige Zahlung für eine unbegrenzte Lizenz, umfasst
  - jede neue zukünftige Version, oder
  - die aktuelle Version mit Aktualisierungen
  - die aktuelle Version mit Updates für ein Jahr
- Abonnement-Modell
  - z.B. pro Monat zahlen, immer die neueste Version erhalten
- "Freemium"
  - einige grundlegende Funktionen sind kostenlos, für erweiterte Funktionen muss man zahlen
- Werbefinanziert (Werbung in der App)
- "Pay-per-use"
  - je nach Nutzungsdauer oder Anzahl

Bereitstellung von Software (engl. *Deployment*)

Es gibt verschiedene Möglichkeiten, wie verteilte Applikationen bereitgestellt werden können *(deployt* werden können)*:*

- Bereitstellung in der IT-Infrastruktur des Unternehmens ("on premise")
  - dem Unternehmen selbst gehört alles
  - maximale Kontrolle, aber auch maximaler Bedienungsaufwand
- Auslagerung in die Cloud durch das Unternehmen selbst:
  - Unternehmen kontrolliert die Anwendungsdaten
  - das Unternehmen bestimmt, wann die Software aktualisiert werden soll
  - weniger Kontrolle über die Infrastruktur, aber viel weniger Betriebsaufwand (kein Austausch von Servern, Betriebssystem-Updates, ...)
- Bereitstellung für den Kunden durch den Softwareentwickler und Angebot als "Software-as-a-Service"
  - minimale Kontrolle (aus Sicht des Unternehmens)
  - minimaler Aufwand für den Betrieb

mehr Kontrolle,
mehr Aufwand (Betrieb)

weniger Kontrolle,
weniger Aufwand (Betrieb)

Softwarebereitstellung, Beispiele

Können Sie weitere Beispiele nennen?

Es gibt verschiedene Möglichkeiten, wie verteilte Applikationen bereitgestellt werden können (*deployt* werden können):

- Bereitstellung in der IT-Infrastruktur des Unternehmens ("on premise")
  - Moodle im HTWG Rechenzentrum
- Auslagerung in die Cloud durch das Unternehmen selbst:
  - Digitale Dörfer Plattform (Beispiel in den späteren Kapiteln)
- Bereitstellung für den Kunden durch den Softwareentwickler und Angebot als "Software-as-a-Service"
  - Microsoft Office 365

Softwareverteilung, mit einigen Beispielen

engl.: *Distribution*

Die Verteilungsmethode unterscheidet sich stark in Bezug auf

- wie schnell man die Software zum ersten Mal benutzen kann,
- wie schnell Updates bereitgestellt werden,
- wie viel Benutzerfeedback das Softwareentwicklungsunternehmen erhält,
- wie nutzbar die Software ist, wenn der Computer offline ist

Diskutieren Sie diese Fragen für die Beispiele!

Beispiele:

- Software-Installationsprogramm für Desktop-Computer (per Download oder Diskette geliefert)
- Installation der App auf dem Mobiltelefon über Play / App Store
- Webanwendung, die im Browser läuft
- Alexa Skill über den Skill Store
- Cloud-Dienst (z. B. API für Wetterdaten)