

› EINHEIT 01: KOMPLEXITÄT

LERNZIELE UND KOMPETENZEN

Komplexität in **technischen** und **organisatorischen** Systemen
erkennen, verstehen und **einschätzen** können.

EINFÜHRUNG

*„Komplexe Systeme enthalten eine **Vielzahl von Komponenten**, die zusammenwirken und damit eine Funktionseinheit bilden. Dabei selbstorganisiert sich die »emergente« Funktion des Systems **ohne jede übergeordnete Kontrollinstanz**.“*

Forschungsperspektiven der Max-Planck-Gesellschaft, 2010, S. 56ff.

*„Ein komplexes System trägt die **Anlage zur chaotischen Entartung** in sich.“*

Unbekannt

*„Komplexe Systeme sind Systeme, welche **sich der Vereinfachung verwehren** und **vielschichtig bleiben**.“*

Wikipedia, https://de.wikipedia.org/wiki/Komplexes_System

WOZU!?

> Wozu behandeln wir komplexe Systeme in diesem Kurs?

Sie haben im Grundstudium unterschiedlichste Werkzeuge, Methoden und Vorgehen kennen gelernt

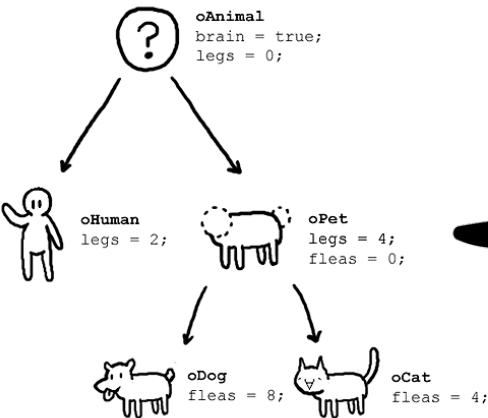
Am Ende dieser Vorlesung sollen Sie in der Lage sein, das bisher Gelernte aber auch alles, was Sie zukünftig lernen werden, auf ein ganzheitliches Software-Projekt anzuwenden um Komplexität zu reduzieren.

Warum? Weil Software-Entwickler dazu neigen komplexe Systeme zu entwerfen!

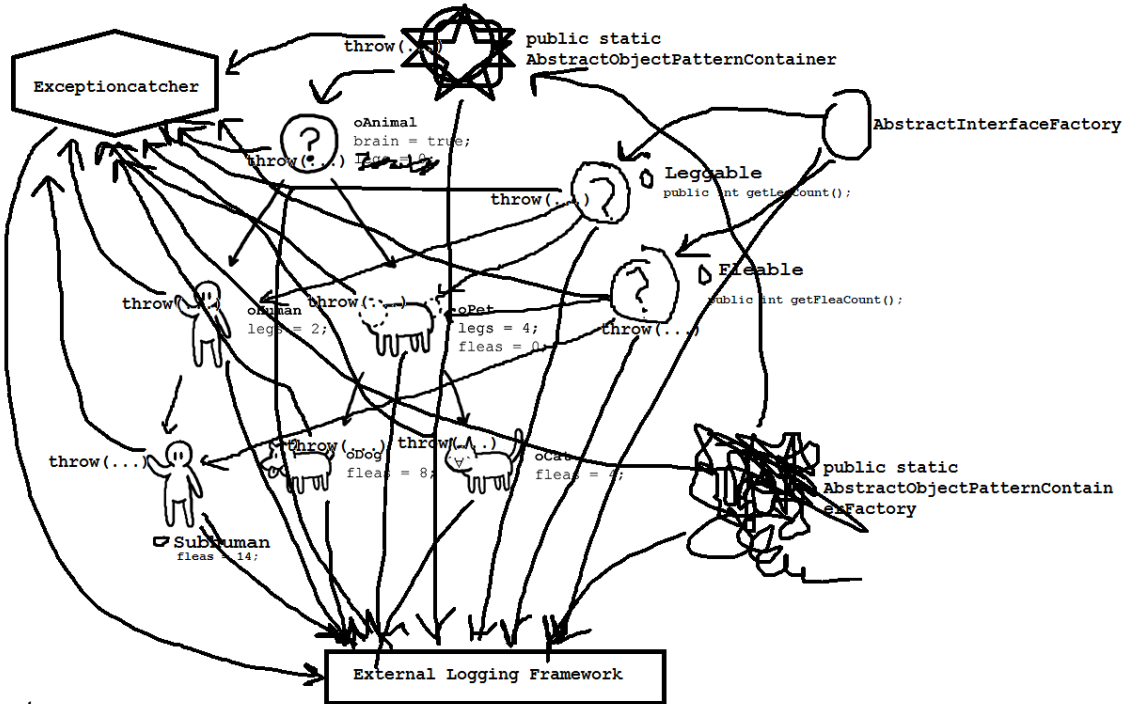


BEISPIEL OOP

Was uns OOP verspricht...



Was wir am Ende erhalten...



Quelle: unbekannt

ÜBUNG

Aufgabe

Sammeln Sie mit Ihrem Banknachbarn Sprachkonstrukte aus möglichst objektorientierten Programmiersprachen (Java, C#, C++, JavaScript, Python etc.), die Ihnen einfallen.

Dauer

3 Minuten

Abschluss

Gemeinsame Auswertung an der Tafel



AUFGABE

Verschaffen Sie sich (als Wiederholung) nochmals einen Überblick über die verschiedenen Aspekte der objektorientierten Programmierung.

Lesen Sie bis zur nächsten Einheit:

[https://de.wikipedia.org/wiki/Objektorientierte Programmierung](https://de.wikipedia.org/wiki/Objektorientierte_Programmierung)

Bearbeiten Sie die Übungsaufgaben im ILIAS Kurs in Einheit 1 zum Thema objektorientierte Programmierung.



Bildquelle: By Wikimedia Foundation, CC BY-SA 3.0,
<https://commons.wikimedia.org/w/index.php?curid=10310220>

BEISPIEL UML

Wie UML die Software beschreibt...



Wie der Code am Ende aussieht...



Bildquelle: Richard Campbell, via Twitter @richcampbell











DISKUSSION

Aus welchen Gründen entsteht die Diskrepanz zwischen UML Design und tatsächlichem Code?



BEISPIEL

STATE-OF-THE-ART SINGLE PAGE APPLICATION

 Eine Angular Anwendung
 mit Bootstrap und
 einem SpringBoot Backend
 gehostet in Apache mit
 einer MariaDB und
 einem Redis Key-Value Store
 alles in Docker Containern
 in einem Kubernetes Cluster
 auf Amazon AWS gehostet und
 mit Ansible verwaltete werden.

> Das könnte man erhalten, wenn man heute eine »einfache« Single Page Application bei einem Dienstanbieter in Auftrag gibt.

AUFGABE

Erstellen Sie ein Architekturdiagramm der zuvor vorgestellten Anwendung!

Darstellung: frei

Gruppengröße: maximal drei (3) Teilnehmer

Erstellen Sie Ihre Gruppe und laden Sie das Diagramm in ILIAS Kurs in Einheit 1 Single Page Application Architektur hoch.

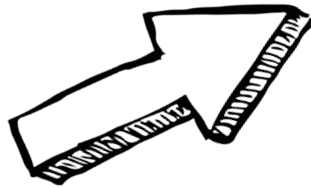
Zugelassene Formate: PDF, PNG oder SVG

BEISPIEL TWITTER MICROSERVICE ARCHITEKTUR

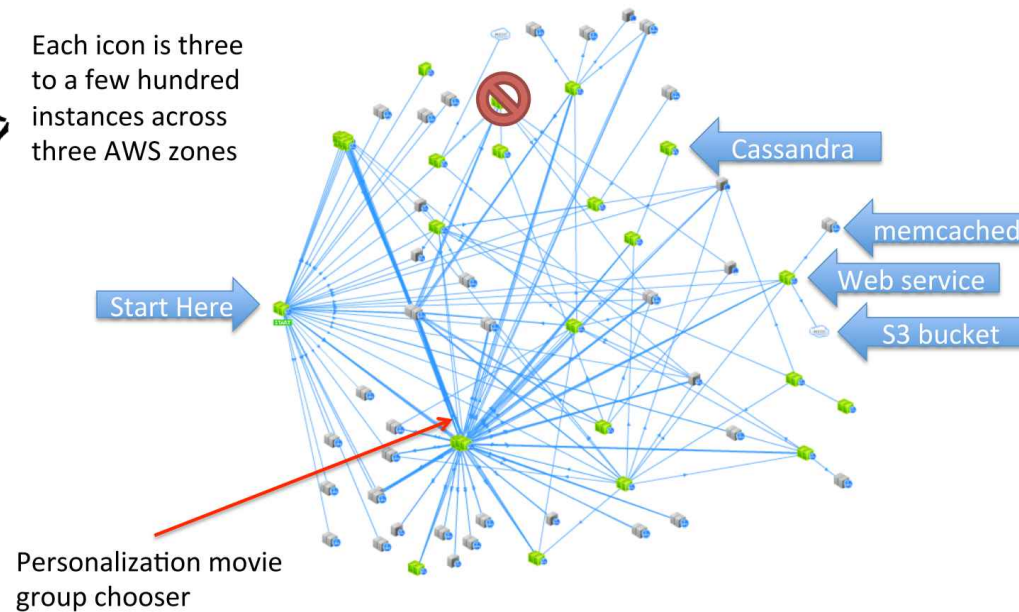
Web Server Dependencies Flow

(Home page business transaction as seen by AppDynamics)

Each icon...
few hundred instances...
across **three AWS zones**...



Each icon is three
to a few hundred
instances across
three AWS zones



Bildquelle: Adrian Cockcroft, via Twitter @adrianco

BEGRIFFSDEFINITION

Software
Engineering

Schon einmal gehört...

Grundlagen Software Engineering 1
Grundlagen Software Engineering 2

komplexer

Unterschied zwischen

einfach, kompliziert und
komplex?

Systeme

System?

BEGRIFFSDEFINITION SOFTWARE ENGINEERING

> IEEE Standard Glossary of Software Engineering Terminology:

*“... the application of a **systematic, disciplined, quantifiable** approach to the **development, operation, and maintenance of software.**”*

> Software Engineering, I. Sommerville:

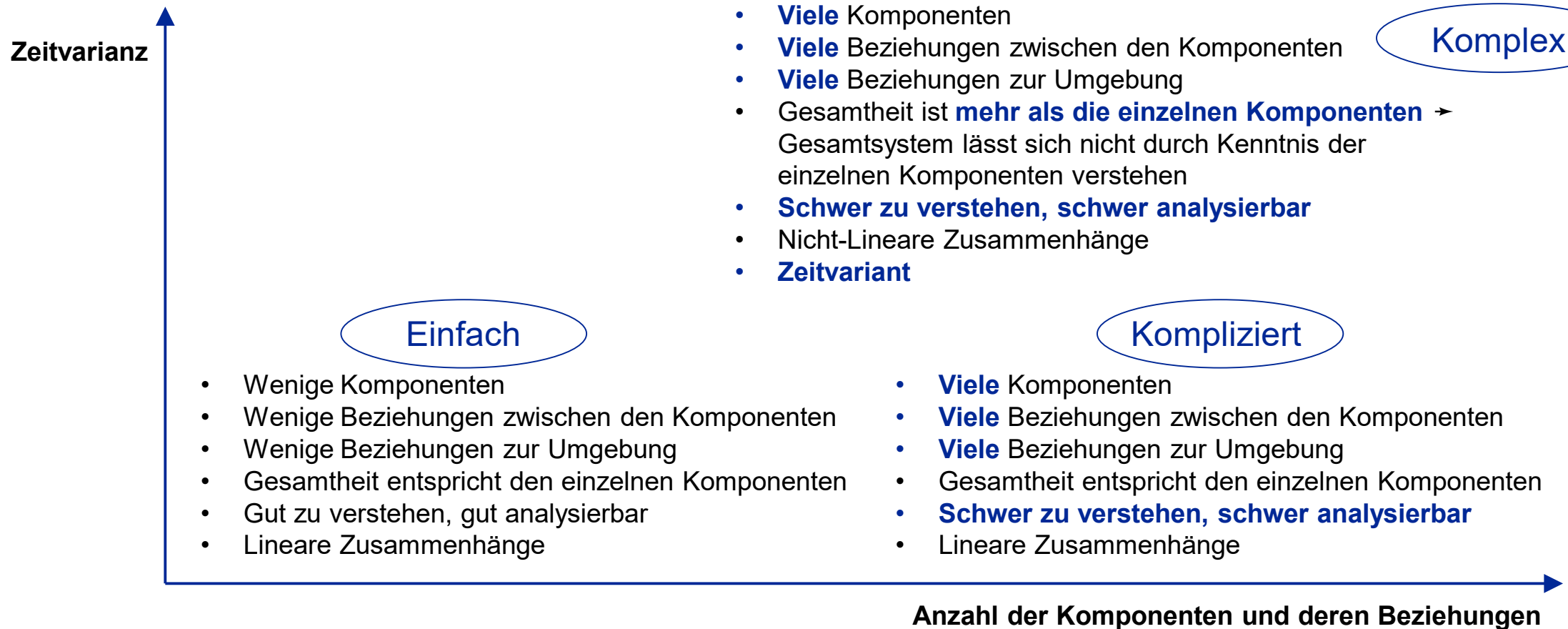
*“Software engineering is an engineering discipline that is concerned with **all aspects of software production.**”*

BEGRIFFSDEFINITION SOFTWARE ENGINEERING

- > Software Engineering ist eine empirische Disziplin
Beruht auf Beobachtungen und Erfahrungen
- > Weiterentwicklung des Software Engineerings
Ist durch die Praxis getrieben
- > Achtung
Es gibt in den meisten Fällen kein „Richtig“ oder „Falsch“
Was in dem eine Projekt oder Team funktioniert, kann in einem anderen Projekt oder Team Probleme verursachen
- > Die Wahl der Methode, Architektur, Projektmanagement
Immer abhängig vom Kontext

BEGRIFFSDEFINITION

EINFACH ➔ KOMPLEX ➔ KOMPLIZIERT



AUFGABE

Sehen Sie sich das Video »Wie reagieren Menschen auf wachsende Komplexität« in der Lerneinheit 1 ILIAS Kurs an und beantworten Sie die Frage(n) zum Video!

AUFGABE

Mit JavaScript lassen sich ohne Zweifel komplizierte Programme schreiben.

Erstellen Sie nun ein komplexes Programm unter der Zuhilfenahme der `async/await` Syntax und Promises, welches sich zeitinvariant verhält.

Studieren Sie hierzu zunächst Promises <https://javascript.info/promise-basics> und die `async/await` Funktionalität unter <https://javascript.info/async-await>.

Laden Sie das Diagramm in ILIAS Kurs in Einheit 1 JavaScript als Textdatei (*.txt) hoch.

Der Code soll auf <https://playcode.io/> ausführbar sein.

BEGRIFFSDEFINITION SYSTEM

Eine **Gesamtheit von Komponenten**, die so aufeinander bezogen oder miteinander verbunden sind und in einer Weise **interagieren**, dass sie als eine **aufgabe-, sinn-, oder zweckgebundene Einheit** angesehen werden können.



WIE ENTSTEHEN KOMPLEXE SYSTEME

Idee



Projekt



Software



Organisatorisches System =
organisatorische Komplexität

Technisches System =
technische Komplexität

ÜBUNG

Aufgabe

Überlegen Sie sich mit Ihrem Banknachbarn jeweils fünf (5) Merkmale für organisatorische und technische Komplexität

Dauer

3 Minuten

Abschluss

Gemeinsame Auswertung an der Tafel



MERKMALE VON KOMPLEXITÄT

Organisatorische Komplexität

- > Anzahl Teammitglieder
- > Beziehungen zwischen Teammitgliedern
- > Anzahl der Teams
- > Interne vs. externe Teammitglieder
- > Geographische Verteilung der Teammitglieder
- > Interkulturelle Unterschiede
- > Reputation des Projekts in der Organisation
- > Anzahl Zulieferer und externer Firmen
- > Kundenprojekt vs. Standardsoftware
- > Beziehung zum Kunden
- > Anzahl Stakeholder
- > Beziehung zu anderen Projekten

Technische Komplexität

- > Art, Anzahl und Eindeutigkeit der Anforderungen
- > Anzahl Komponenten und Schnittstellen
- > Benutzer Interface
- > Menge und Art der Daten
- > Code Basis (Lines of Code)
- > Brownfield vs. Greenfield Projekt
- > Legacy Code
- > Bekannte vs. neue Technologien
- > Anzahl Programmiersprachen
- > Verteilte Systeme
- > Parallelität im Code
- > Security

ANMERKUNGEN ZU ORGANISATORISCHEN SYSTEMEN

> In organisatorischen Systemen existieren **offizielle** Beziehungen

Projektmanager gibt Inhalte dem Team vor

Teamleiter ist disziplinarisch Vorgesetzter

Entwicklungsleiter ist fachlicher Entscheidungsträger

Projektleiter gibt Termine vor

Entwickler entscheiden über Code

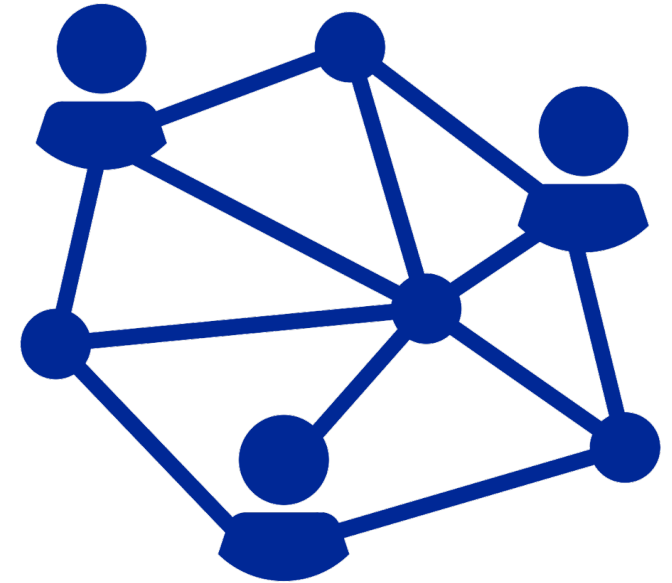
> Und es gibt **inoffizielle** Beziehungen

Teamleiter ist mit einem Teammitglied im gleichen Fußballverein

Entwicklungsleiter „kann nicht“ mit Entwickler A

Entwickler B ist bester Freund von Entwickler C

Externer Berater ist Bruder des Schwagers von Entwickler D



KOMPLEXITÄT MANAGEN

Wie also Komplexität in den Griff bekommen?

> Organisatorische Komplexität

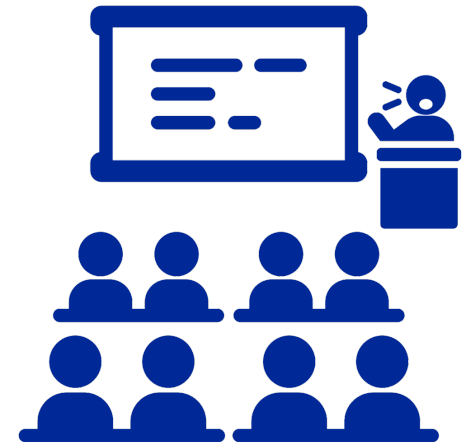
- > Projektmanagement Vorgehensmodelle
- > Schätzen und Schätztechniken
- > Anforderungsanalyse

→ um die organisatorische Komplexität zu reduzieren

> Technische Komplexität

- > Software Architekturen
- > Code Metriken
- > Testen

→ um die technische Komplexität zu reduzieren



ANREGUNGEN ODER FRAGEN BIS HIER HER?

Bei Fragen kontaktieren Sie bitte:



Prof. Dr.-Ing. Andreas Heil

Fakultät für Informatik | Software Engineering

andreas.heil@hs-heilbronn.de