

# Software Engineering 1



Anna Prenzel

# Einführung

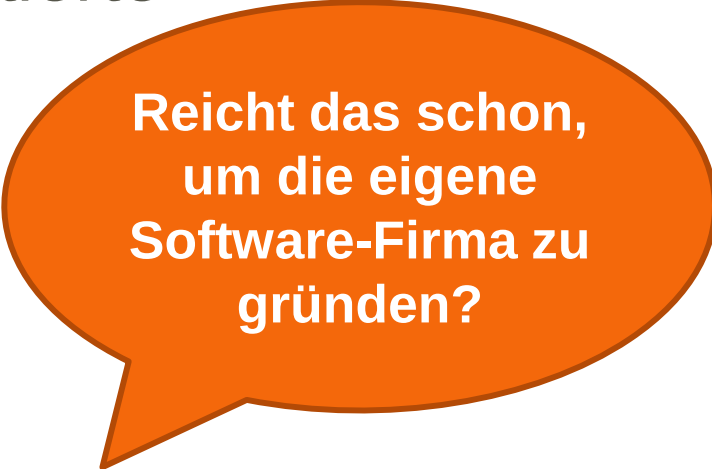
## Ziele der heutigen Veranstaltung:

- ☞ Software-Entwicklung als System-Entwicklung verstehen
- ☞ Software-Entwicklung als Ingenieursaufgabe verstehen
- ☞ Ablauf dieser Lehrveranstaltung klären

# Einführung

## Wo stehen Sie jetzt?

- ☞ Sie haben bereits eigene Algorithmen geschrieben
- ☞ Sie beherrschen objektorientierte Programmierung



Reicht das schon,  
um die eigene  
Software-Firma zu  
gründen?

# Einführung

## Nein, denn...

- ☞ Sie beherrschen noch nicht die Komplexität industrieller Software
- ☞ Die Komplexität haben Sie bereits beim BookStore und beim Zeiterfasser bemerkt.

**Zeiterfasser:** < 500 Zeilen Code

**Microsoft Word:** > 2 Millionen Zeilen Code<sup>1</sup>

1) <http://www.wired.com/wired/archive/3.09/myhrvold.html>

# Was ist Software?

## Definition (Software)

**Sammelbezeichnung für Programme, die für den Betrieb von Rechensystemen zur Verfügung stehen, einschließlich der zugehörigen Dokumentation.**

# Eine Software ist ein System

## Diese Sichtweise ist nützlich, um...

- ∞ die Komplexität besser zu verstehen
- ∞ mit der Komplexität umgehen zu können

## Definition (System)

**Ein System ist eine zweckorientierte Zusammenstellung von interagierenden Objekten oder Systemen, die zusammenarbeiten um ein Ziel zu erreichen.**

# Eine Software ist ein System

- ☞ Systeme bestehen aus Hardware oder Software oder einer Kombination von beiden
- ☞ Software kann als System oder Sub-System aufgefasst werden



# Eigenschaften von Systemen

## 1. Systeme sind in ihre Umgebung eingebettet

- ✎ Sie nehmen Eingaben (Daten, Signale) aus der Umgebung entgegen
- ✎ Sie liefern Ausgaben bzw. beeinflussen die Umgebung
- ✎ Die Ausführung wird als **Black Box** betrachtet
- ✎ Die Funktionstüchtigkeit wird nur anhand der Ausgaben bewertet

# Eigenschaften von Systemen

## Rückschlüsse

### Vor der SW-Entwicklung

 Die Grenze zwischen System und Umgebung muss klar definiert sein. **Eine Software benötigt eine klare Schnittstellendefinition (Anforderungen).**

### Während/nach der SW-Entwicklung

 **Es muss verifiziert und/oder getestet werden, dass die Software aus Eingabedaten die korrekten Ausgabedaten berechnet.**

# Eigenschaften von Systemen

## 2. Systeme sind modular aufgebaut

- ∞ Mehrere Teilsysteme (Komponenten) interagieren miteinander, sind voneinander abhängig
- ∞ Teilsysteme sind wiederverwendbare Bausteine, aus denen sich größere Systeme zusammensetzen lassen

# Eigenschaften von Systemen

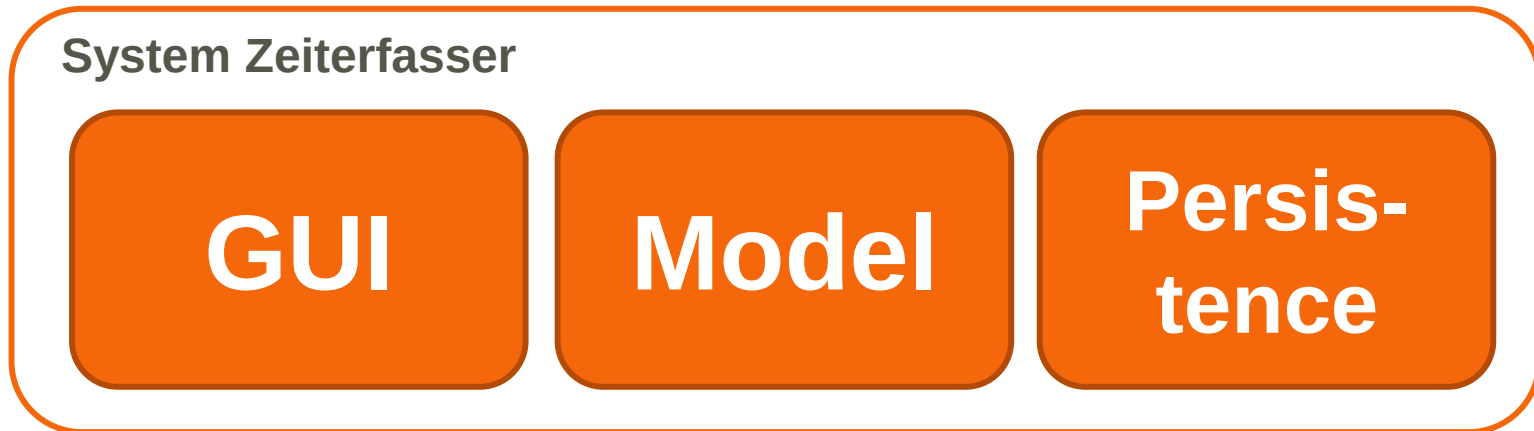
## Rückschlüsse

### Während der SW-Entwicklung

-  **Teile und Herrsche:** Der SW-Entwickler muss festlegen, welche Teilsysteme
  -  wiederverwendet,
  -  hinzugekauft oder
  -  neu entwickelt werden
  
-  Er muss festlegen, wie die Teilsysteme zum Gesamtsystem zusammengesetzt werden sollen.

# Eigenschaften von Systemen

## Beispiel Zeiterfasser



- ∞ Jede Komponente lässt sich weiter zerlegen in Subkomponenten, Klassen und Methoden

# Eigenschaften von Systemen

## Beispiel Zeiterfasser

### Ihre Vorgehensweise bisher

- ✎ Sie arbeiten auf der untersten Abstraktionsebene
- ✎ Sie arbeiten abwechselnd an der Kommunikation zwischen Komponenten und innerhalb von Komponenten

# Eigenschaften von Systemen

## Beispiel Zeiterfasser

### Neue Vorgehensweise

1. Komponenten identifizieren
2. Kommunikation zwischen den Komponenten entwerfen. Von der Implementierung der einzelnen Komponenten wird abstrahiert.
3. Komponenten ausprogrammieren.

 **Idealerweise erfolgt die Kommunikation über Schnittstellen an der Komponentengrenze.**

# Eigenschaften von Systemen

## Übung: Aneignung von Abstraktionsvermögen

**Geben Sie 3 unterschiedlich abstrakte Beschreibungen für das System „Studentenausweis“ an.**

# SW-Entwicklung ist Problemlösung

**Vor dem „Wie“ stellt sich in der Praxis erst einmal die Frage:**

**„Was soll überhaupt entwickelt werden?“**

# SW-Entwicklung ist Problemlösung

**Die Anforderungen des Kunden sind von Natur her**

- ∞ vage
- ∞ unzusammenhängend
- ∞ unvollständig
- ∞ widersprüchlich

# SW-Entwicklung ist Problemlösung

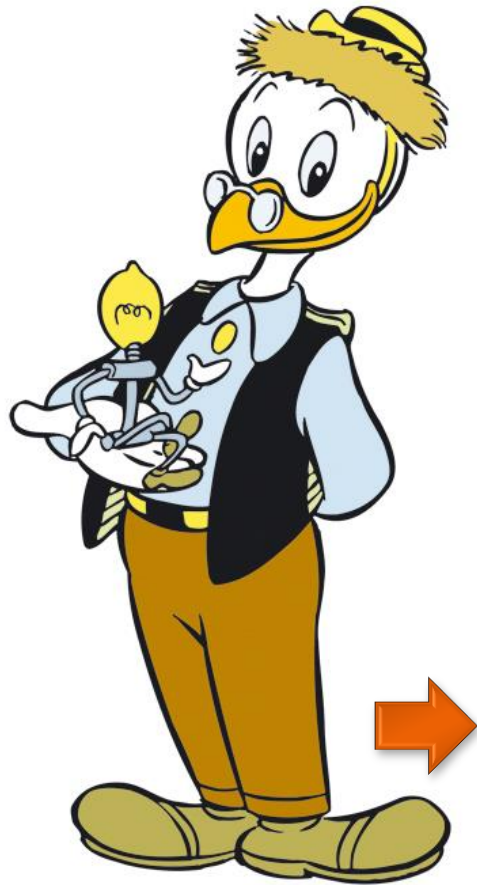
## Analyse des Problems

- ∞ Problem des Kunden verstehen
- ∞ Systemgrenze (Schnittstelle) verhandeln
- ∞ Problem in Teilprobleme zerlegen

## Synthese der Lösung

- ∞ Teilprobleme lösen
- ∞ Teilprobleme zur Lösung zusammensetzen

# SW-Entwicklung ist Problemlösung



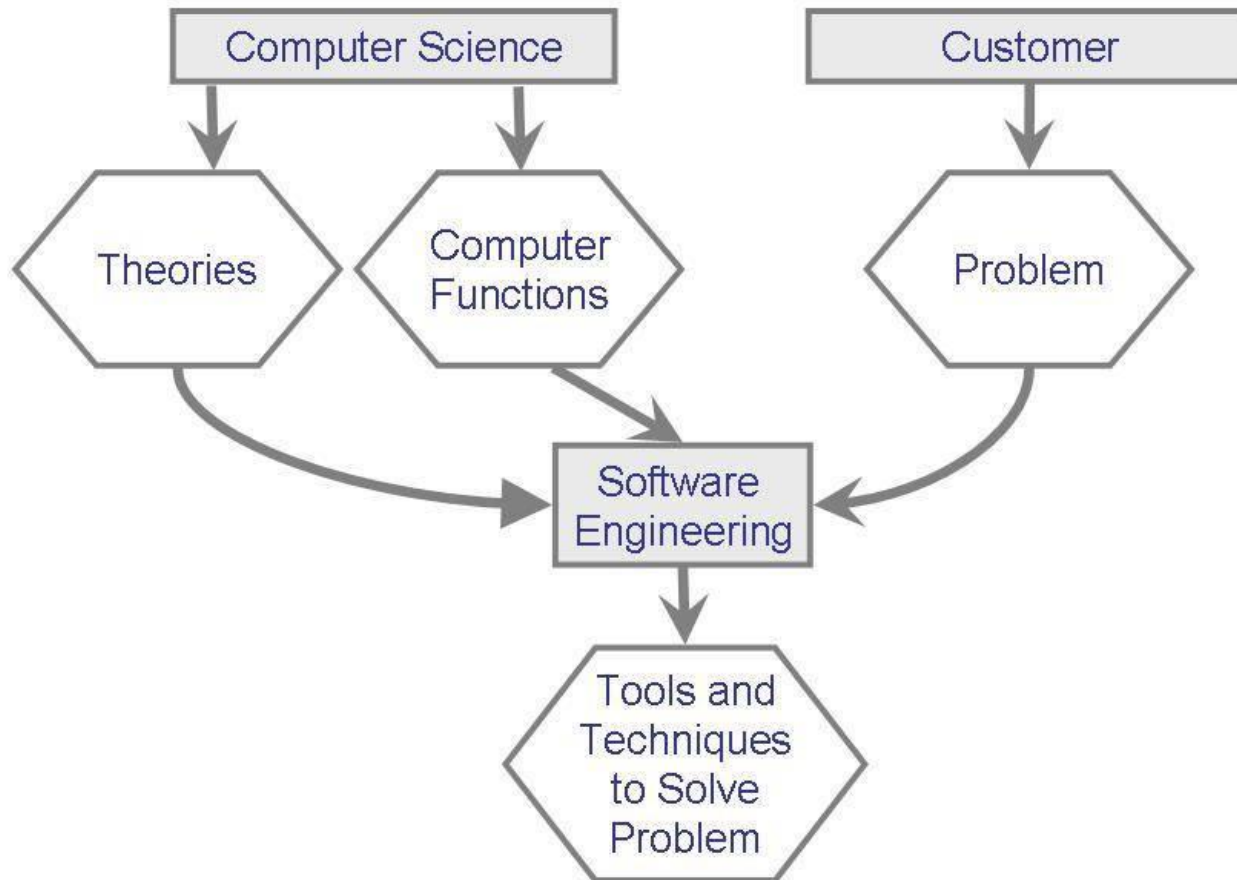
## „Dem Ingenieur ist nichts zu schwör“

- ☞ Der SW-Entwickler benutzt *Technologien, Methoden, Paradigmen* und *Werkzeuge*, um das Problem des Kunden zu lösen
- ☞ Er benutzt die wissenschaftlichen Erkenntnisse der Informatik

Software-Entwicklung ist eine Ingenieursaufgabe.

**Wir sprechen von Software-Engineering!**

# SW-Entwicklung ist Problemlösung



# SW-Entwicklung ist Problemlösung

## Wichtige Fähigkeiten eines (Software)-Ingenieurs

1. Er besitzt das erforderliche Wissen
2. Er kann Wissen zur Lösung eines Problems anwenden
3. Er besitzt das notwendige Abstraktionsvermögen, um Analyse und Synthese durchzuführen

# SW-Entwicklung ist Problemlösung

## Wichtige Fähigkeiten eines (Software)-Ingenieurs

4. Er kann den gesamten Prozess von der Planung bis hin zur Kontrolle leiten.

# SW-Entwicklung ist Problemlösung

## Aneignung von anwendbarem Wissen Beispiel Sortieralgorithmen

### Für die praktische Anwendung wichtig:

- ⌘ Kenntnis von Performanz und Speicherbedarf der einzelnen Verfahren
- ⌘ Kenntnis, für welche Daten welcher Algorithmus geeignet ist (wie groß ist die Datenmenge?, sind die Daten vorsortiert?)
- ⌘ Kenntnis, in welchem Kontext welcher Algorithmus geeignet ist (wie oft wird sortiert?, wie schnell muss sortiert werden?)

# SW-Entwicklung ist Problemlösung

## Aneignung von anwendbarem Wissen Beispiel Sortieralgorithmen

**Für die praktische Anwendung unwichtig:**

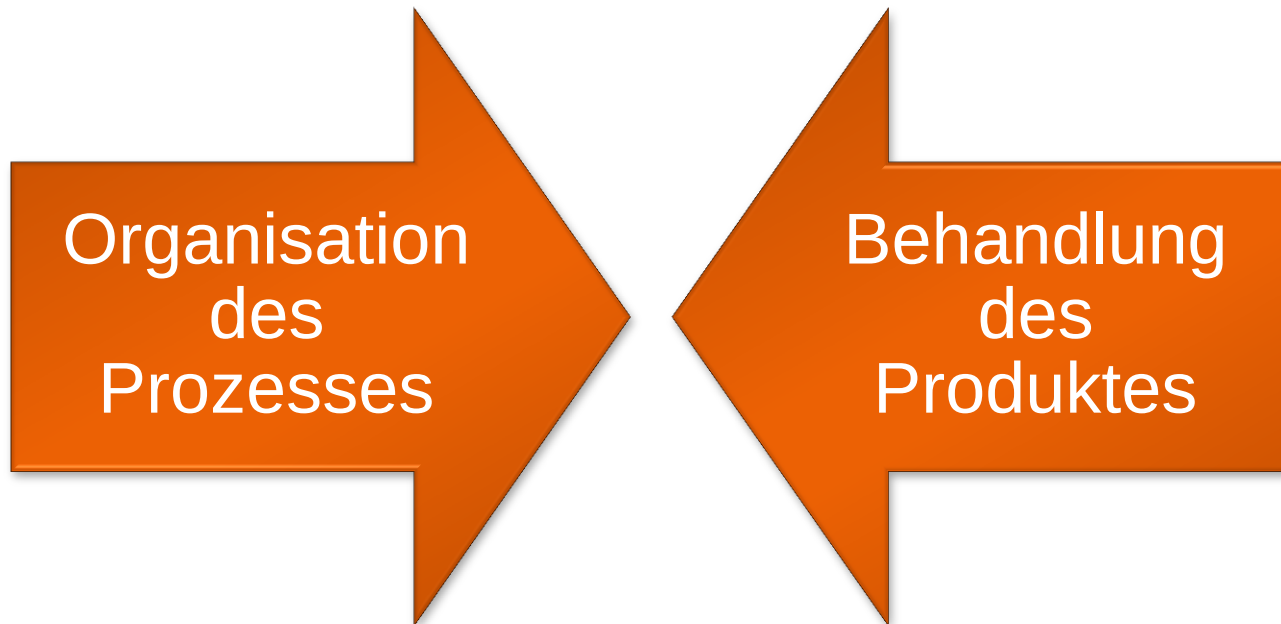
- ✎ Fähigkeit, die Algorithmen aus dem Kopf nachzuprogrammieren

**Warum müssen Sie dann so viele Algorithmen üben?**

- ✎ Fertige Implementierungen müssen für den jeweiligen Anwendungsfall angepasst werden
- ✎ Sie lernen, eigene Implementierungen nach denselben Kriterien wie fertige Lösungen zu bewerten

# Methoden des Software-Engineering

## Die 2 Aspekte des SE:



# Methoden des Software-Engineering

## Definition (Grundsatz des Software-Engineering)

**Zielorientierte Bereitstellung und systematische Verwendung von Paradigmen, Methoden und Werkzeugen für die arbeitsteilige, ingenieurmäßige Entwicklung von umfangreichen Softwaresystemen.**

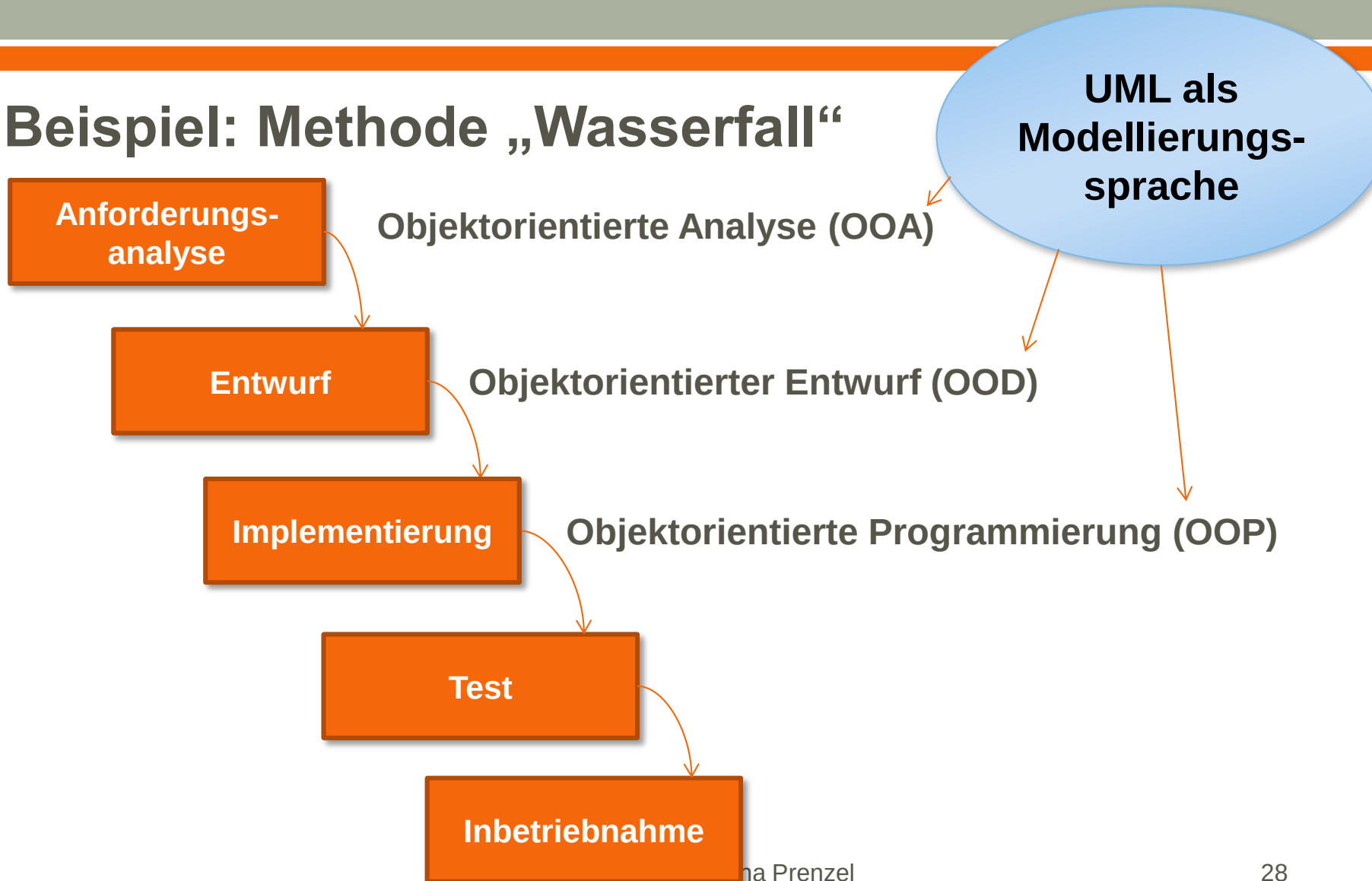
# Methoden des Software-Engineering

## Eine Methode des SE legt fest,

- ✎ was zu tun ist,
- ✎ wann es zu tun ist,
- ✎ welche Werkzeuge benutzt werden,
- ✎ wie der Prozess zu planen ist,
- ✎ wie der Prozess kontrolliert wird

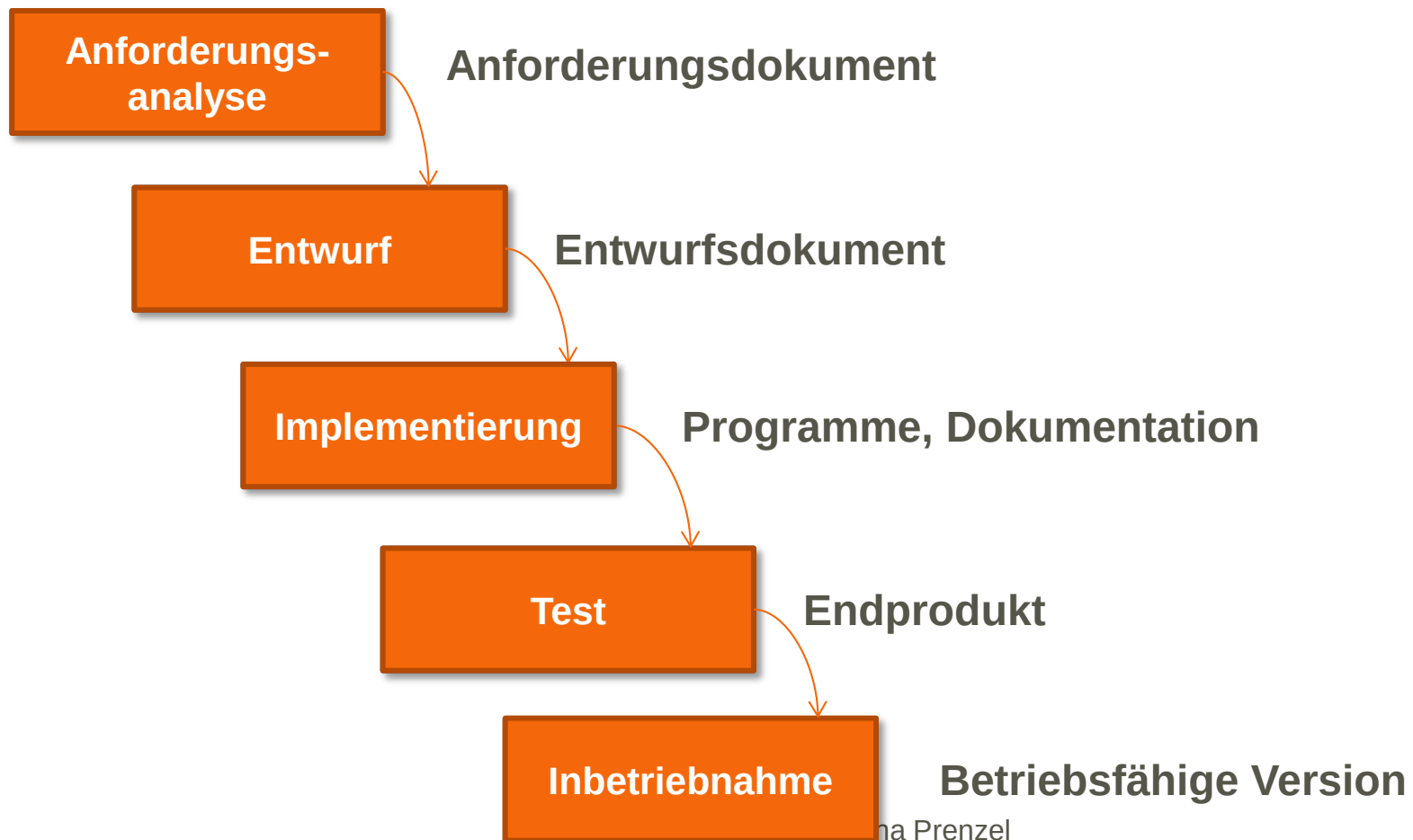
# Methoden des Software-Engineering

## Beispiel: Methode „Wasserfall“



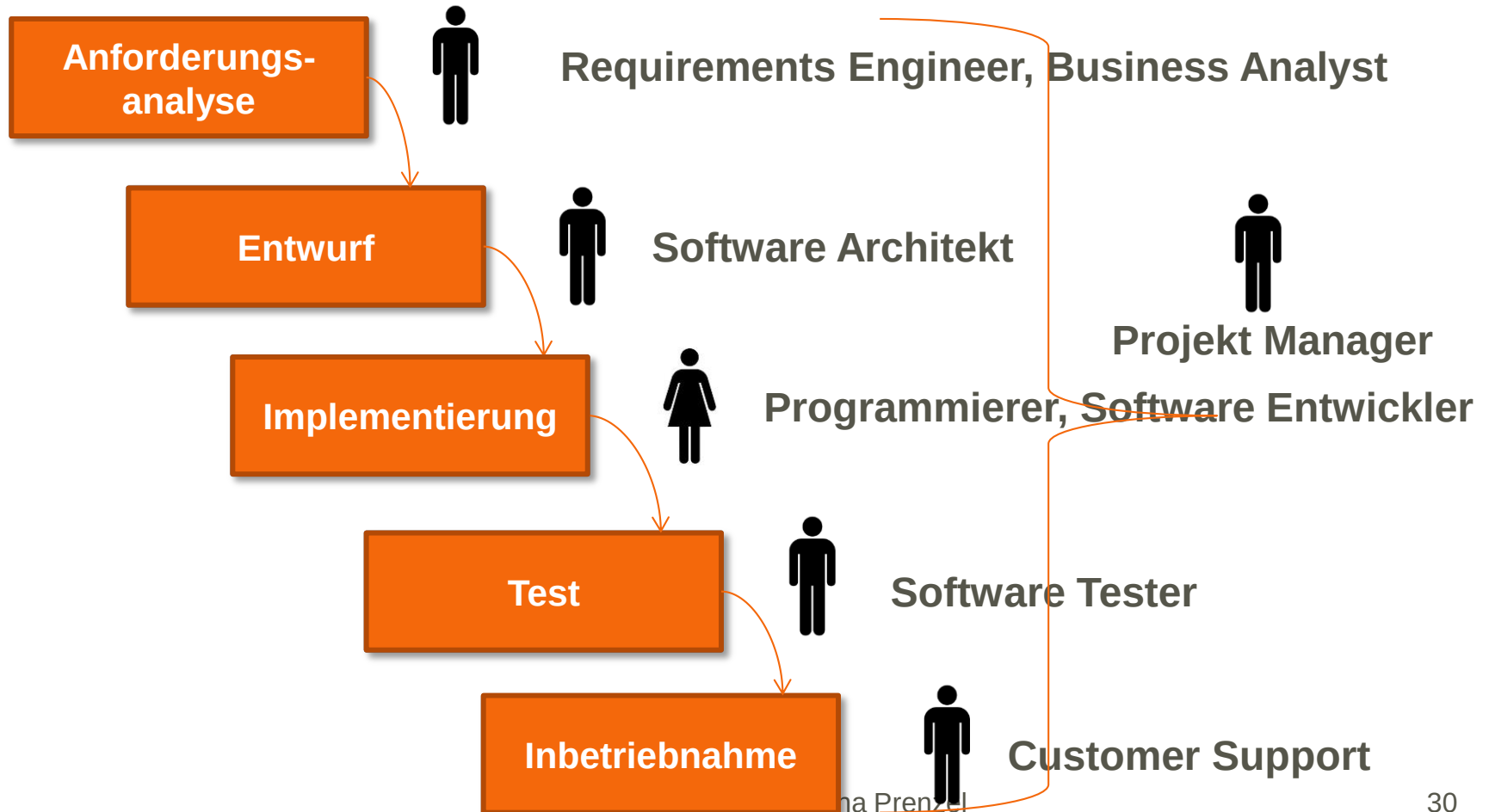
# Methoden des Software-Engineering

## Artefakte



# Methoden des Software-Engineering

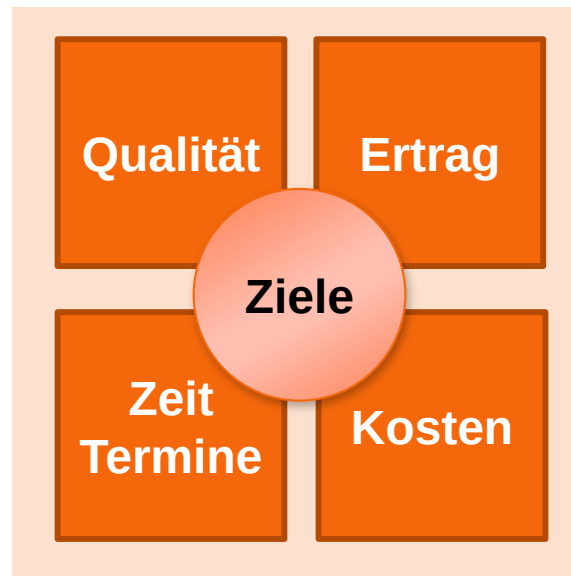
## Berufsbilder



# Methoden des Software-Engineering

## Projektmanagement im SE

- ∞ Vorrangiger Zweck von SE ist die Erfüllung wirtschaftlicher Ziele



# Methoden des Software-Engineering

## Projektmanagement im SE

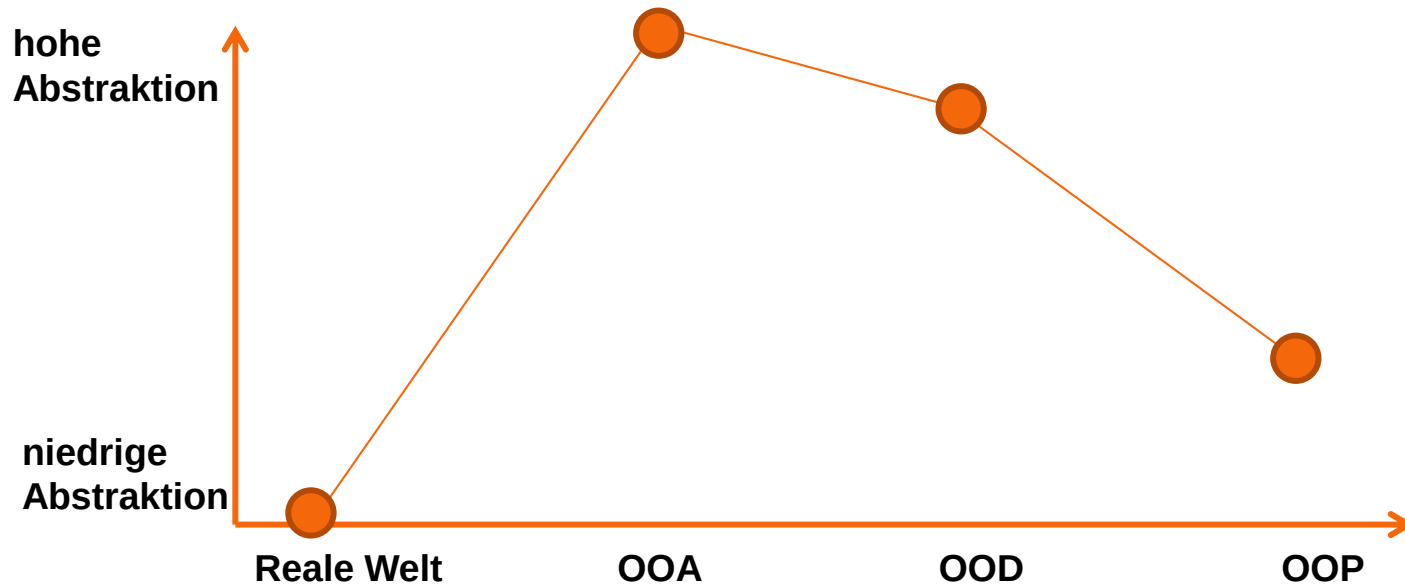
### Kostenverteilung für die Entwicklung

- ∞ Analyse und Entwurf: 40 %
- ∞ Programmierung: 20 %
- ∞ Testen: 40 %

# Methoden des Software-Engineering

## Kommunikation im SE-Team ist ein Spannungsfeld

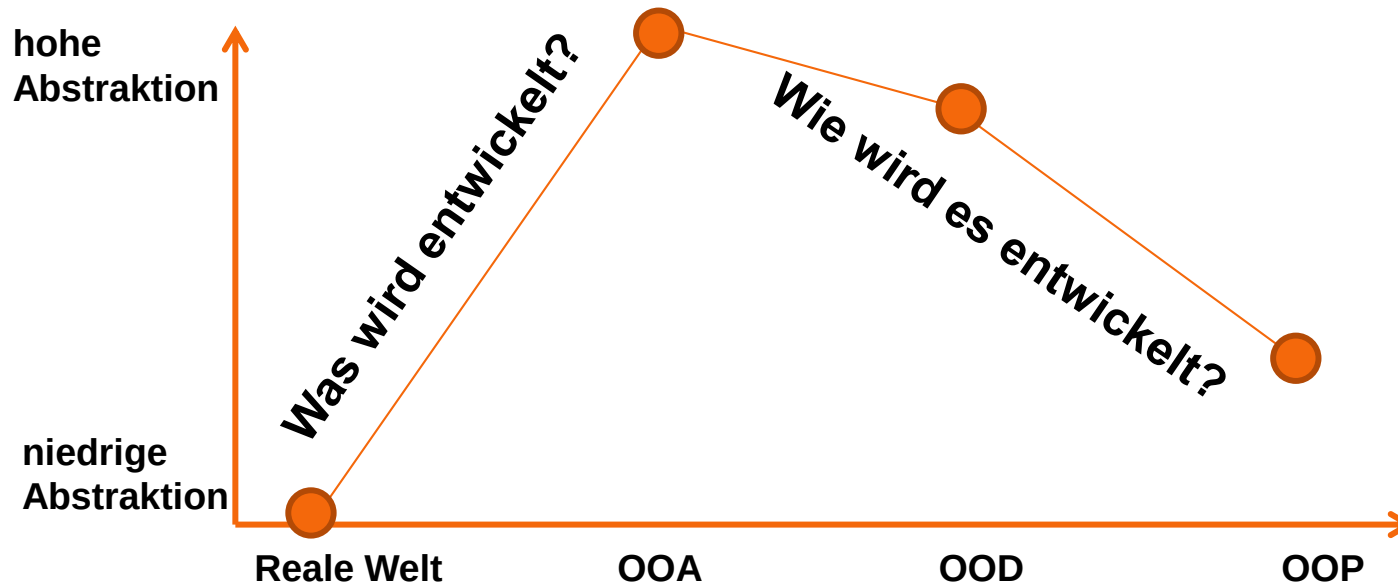
Grund: Unterschiedliche Abstraktionsebenen



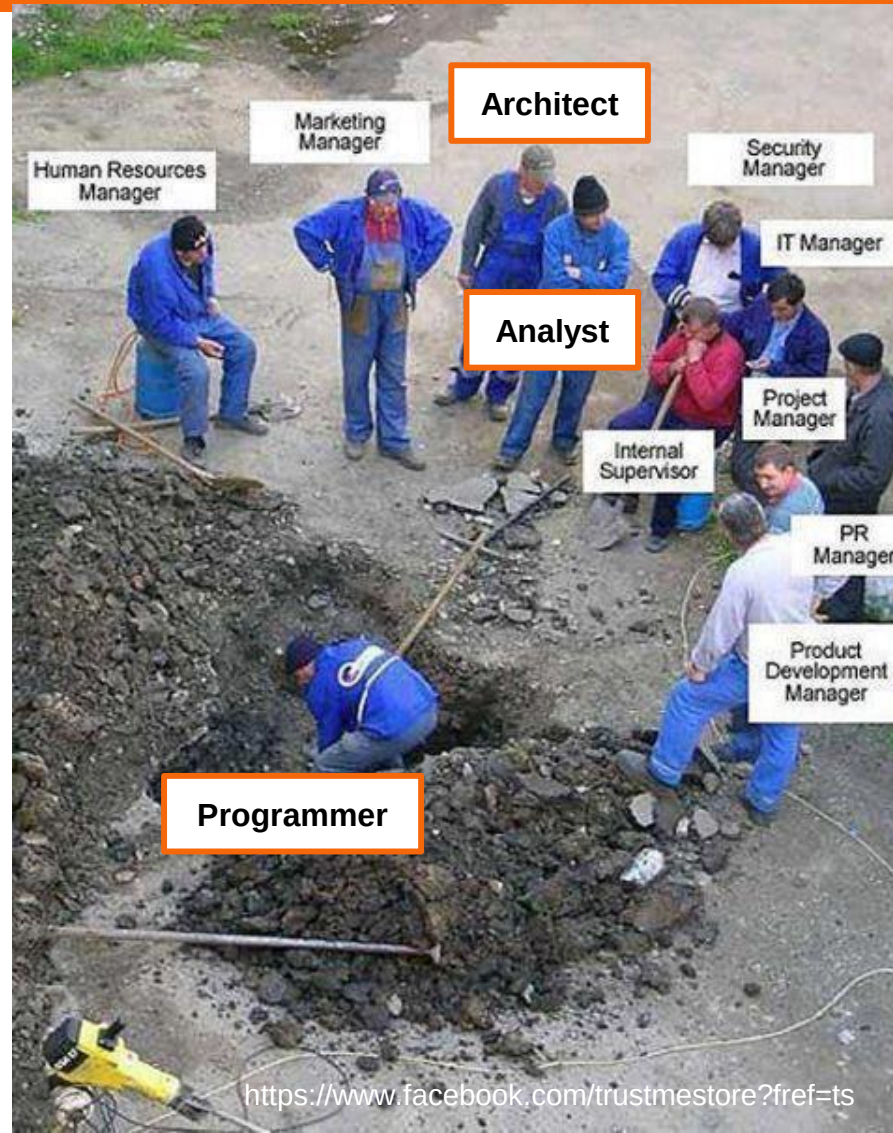
# Methoden des Software-Engineering

## Kommunikation im SE-Team ist ein Spannungsfeld

Grund: Unterschiedliche Abstraktionsebenen



# Methoden des Software-Engineering



# THE WAR BETWEEN DEVELOPERS, DESIGNERS & PROJECT MANAGERS

Original picture at <http://bit.ly/war-developers> by @hteumeuleu

## DEVELOPERS

## DESIGNERS

## PROJECT MANAGERS

SEEN BY  
DEVELOPERS



SEEN BY  
DESIGNERS



SEEN BY  
PROJECT  
MANAGERS



TRANSLATED BY  
@ALEXTOUL

# Ablauf der LV

## ∞ Lernform:

- Vorlesung
- betreute Projektarbeit
- Selbststudium(!) → 75h, das macht ca. 19h im Monat bzw. ca. **5h die Woche**

## ∞ Prüfungsvorleistung:

- Präsentation des Projektes (15-20 min. pro Team)

## ∞ Prüfung:

- schriftlich, Klausur (120 min.)

# Ablauf der LV

## Projektarbeit

- ✎ Bilden Sie 3er Teams! Entwickeln Sie eine Schichtplanungssoftware!
- ✎ Ihre Aufgabenstellung (Details später):
  - Erstellung eines Anforderungsdokumentes (OOA)
  - Erstellung eines Entwurfsdokumentes (OOD)
  - Implementierung der Software und Erstellung einer Softwaredokumentation (OOP)

# Ablauf der LV

## Projektarbeit

- ∞ Sie erhalten wöchentlich Übungsblätter, um das Wissen aus der Vorlesung an Ihrem Projekt anzuwenden und die Dokumente weiterzuentwickeln.
- ∞ Die Teammitglieder bearbeiten in jeder Stufe unterschiedliche Bereiche der Software.

# Ablauf der LV

**Sammeln Sie Ihre ersten positiven und negativen Erfahrungen in Teamarbeit und in der langfristigen Arbeit an komplexer Software!**

- ✎ Es geht nicht darum, am Ende ein perfektes System abzuliefern.
- ✎ Sie müssen mit der Implementierung am Ende noch nicht fertig sein.
- ✎ Bei der Abschlusspräsentation kommt es aber darauf an,
  - dass jedes Teammitglied alle Schritte von der Analyse bis zur Implementierung durchlaufen hat,
  - dass Sie den Entwurf Ihres Teams kritisch diskutieren können, d.h. dass Sie Fehler erkannt und daraus gelernt haben.

# Ablauf der LV

## Was passiert, wenn Sie jetzt nicht mitziehen?

- ☞ Sie machen ihre Anfängerfehler in der Software-Firma (peinlich).
- ☞ Sie trauen sich nicht zu, sich auf die besten Jobs zu bewerben, z.B. als Software Analyst oder Software Entwickler.
- ☞ Sie werden von der Komplexität und dem Umfang industrieller Software-Projekte erschlagen.

 **Jede Minute, die Sie in die Übung „SWE“ investieren, ist unwiederbringlich und bereitet Sie aufs Berufsleben vor.**