DUALE HOCHSCHULE SH

in Trägerschaft der Wirtschaftsakademie Schleswig-Holstein

Zielgruppe: **Wirtschaftsinformatik**

Modul: **Software Engineering**

Foliensatz:

Prinzipien des Software Engineering

Prof. Dr. Malte Prieß

DUALE HOCHSCHULE SH

in Trägerschaft der Wirtschaftsakademie Schleswig-Holstein

Software Engineering

Prinzipien des Software Engineering

Prof. Dr. Malte Prieß

A day in the life of a software engineer ...

Jonathan @ Wayfair.com

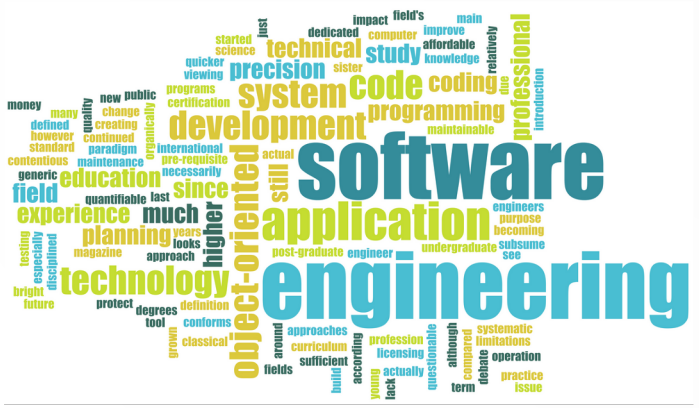
https://www.youtube.com/watch?v=-uMbvR4cq74

Vivien @ Goldman Sachs

https://www.youtube.com/watch?v=Gi1sWYoLQLg

The career of an IT consultant at Team Trade

https://www.youtube.com/watch?v=wHlth8LU3Fo



Softwarekrise

Die Kosten für Software übersteigen die Kosten für Hardware.

Es kommt zu ersten großen scheiternden Software-Projekten.

Wann?

Ende 1960er

> *“The major cause of the software crisis is that the machines have become several orders of magnitude more powerful!*
>
> *[…] when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.”*
>
> *Edsger Dijkstra, The Humble Programmer, 1972*

In Reaktion wird der Begriff **Software Engineering** geprägt.

Definition Software Engineering

Software Engineering (SE oder deutsch Softwaretechnik) ist die **Methodenlehre der Softwarekonstruktion**.

Ihre Hauptbeiträge sind **methodische Hilfsmittel** wie Prozesse, Modelle, Werkzeuge und Prinzipien zur Konstruktion und Beherrschung des Einsatzes **hochwertiger Software** zu akzeptablen Kosten.

„Hochwertig“ bedeutet hier: Zuverlässig, sicher, leicht veränderbar, einfach benutzbar und vor allem mit den richtigen, erwünschten und vom Nutzer benötigten Funktionen.

Broy et al. (2006), unterzeichnet von 34 deutschen SE-Hochschullehrern

The establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.

Naur, Randell (1968)

Ingenieurmäßiges Entwickeln von Software

- Komplexe Softwaresysteme werden **im Team** entwickelt
- **Hohe Qualität** der Software ist zentrales Ziel zur Erfüllung
  - funktionaler (fachlicher) Anforderungen
  - nicht-funktionaler (technischer) Anforderungen

Software Engineering beschäftigt sich mit **Beherrschung von Komplexität**.

Die zu konstruierende Software …

- hat viele Komponenten
- wird von vielen Anwendern benutzt
- wird von vielen Entwicklern bearbeitet
- soll auf vielen Plattformen in Varianten laufen

A software engineer must of course be a **good programmer**, be well-versed in data structures and algorithms, and be fluent in one or more programming languages.

The software engineer must be familiar with several **design approaches**, be able to translate vague requirements and desires into **precise specifications**, and be able to **converse with the user** of a system in terms of the application rather than in computerese.

Ghezzi et al. (2003)

Kompetenzen eines Software Engineers

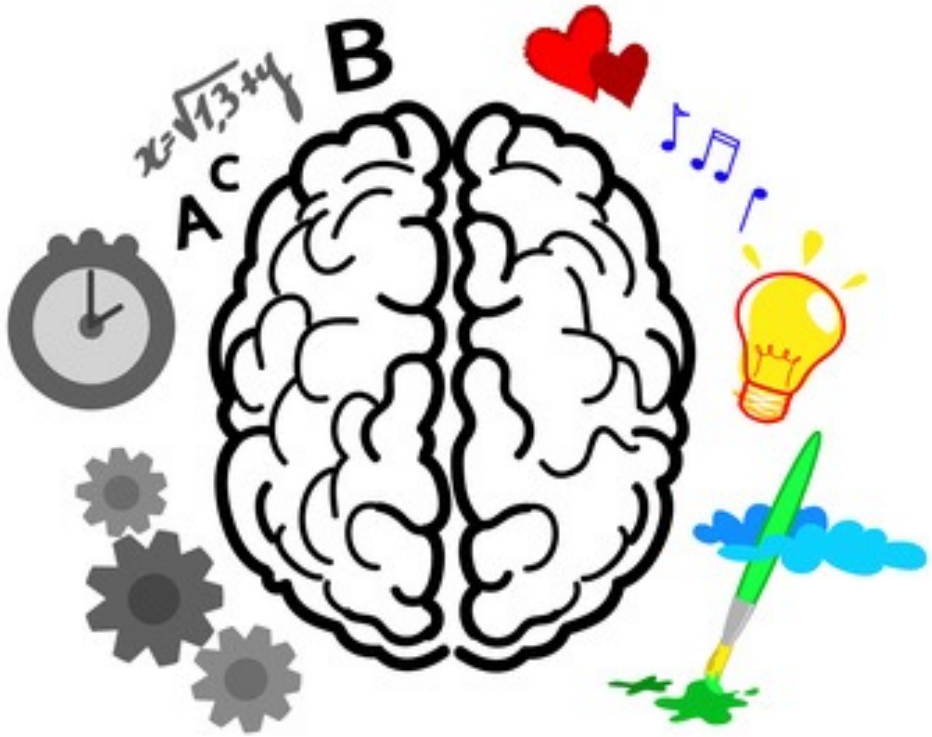
- Programmiertechniken
- Erstellung und Nutzung von Modellen und Spezifikationen zur Abstraktion
- Kommunikation mit Stakeholdern mit unterschiedliche Zielsetzungen, Vorstellungen, Ausbildungen
- Arbeitsplanung und -koordination

Principles of Software Engineering

- Rigor and Formality
- Separation of Concerns
- Modularity
- Abstraction
- Anticipation of Change
- Generality
- Incrementality

Quelle: Ghezzi et al. (2003)

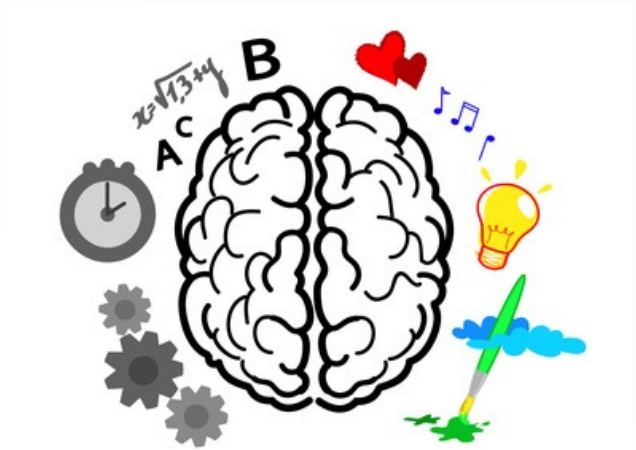
Rigor and Formality



Software Engineering erfordert **Kreativität** in der Entwurfsphase.

Auch in kreativen Phasen vermeidet **systematisches Vorgehen** Chaos.

- **Genauigkeit** in kreativen Phasen ist notwendig, um Vertrauen in den Entwurf zu schaffen
- **Formalität** ist Genauigkeit in höchstem Maße



Anwendung

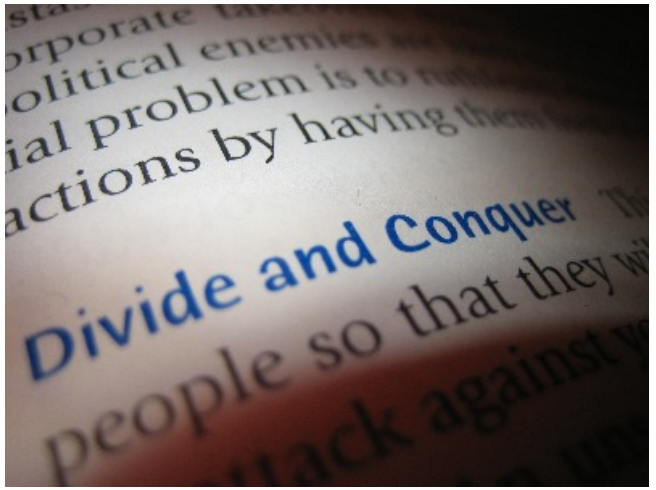
- Mathematisch-formale Verifikation eines Programms
- Modelle gemäß standardisierter Notationen
- Systematische Herleitung von Testdaten
- Gründliche Dokumentation von Aktivitäten unterstützt Projekt-management

Trennung von Belangen ist Grundlage für Komplexitätsreduktion

- Konzentration auf einen Aspekt zur Zeit ist förderlich
- Parallelisierung und Trennung von Verantwortlichkeiten wird möglich

Anwendung

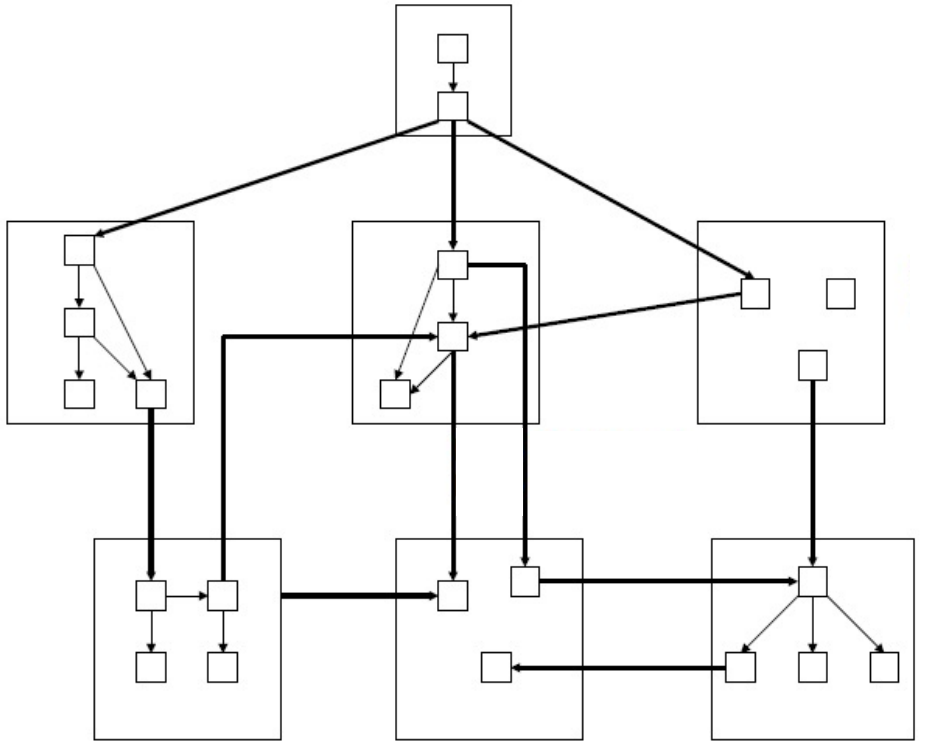
- Trennung nach Aspekten Funktionalität, Performance, Usability, …
- Modelle bieten verschiedene aspektorientierte Sichten



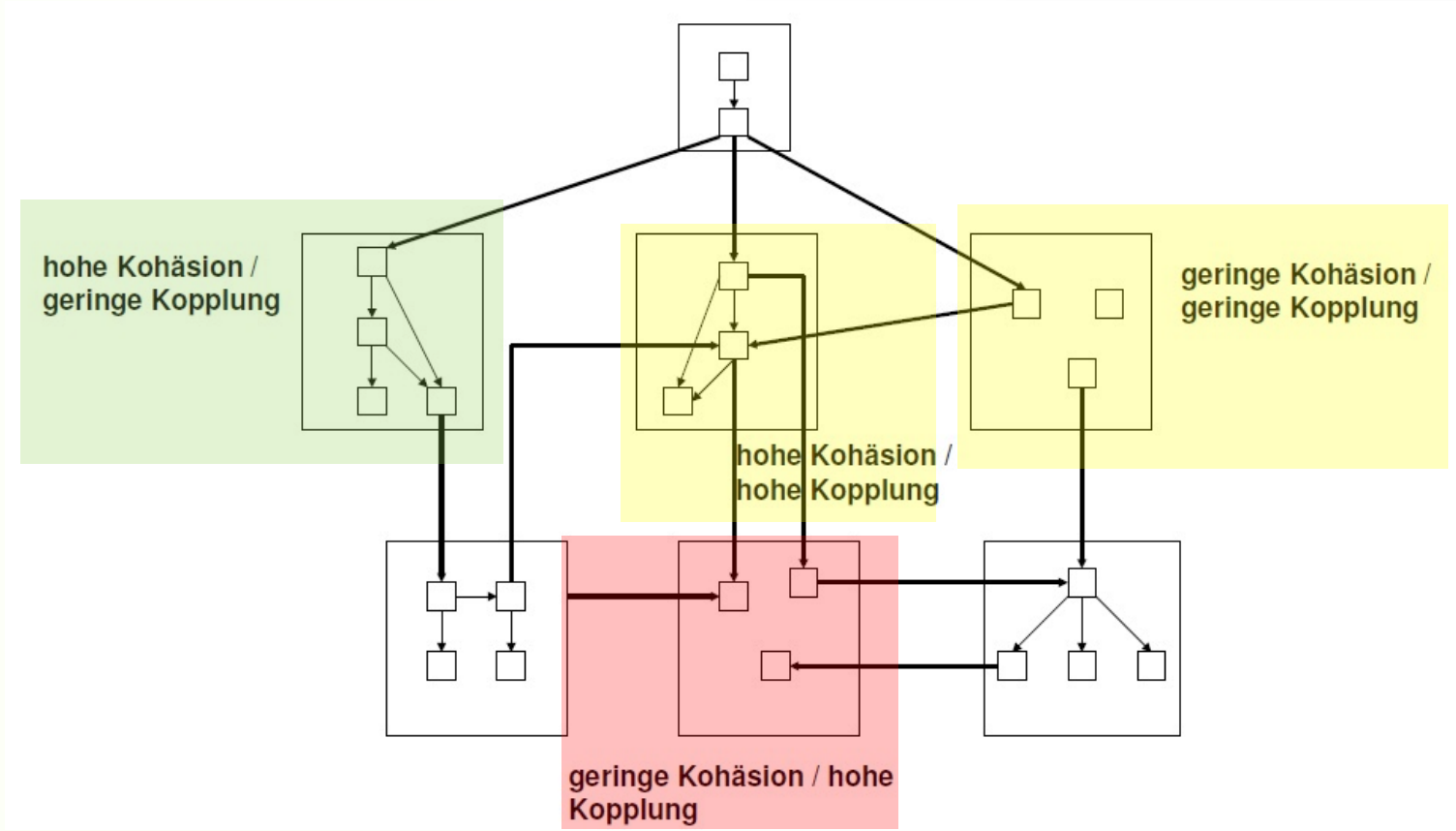
Ein komplexes System kann in **Module (= Komponenten)** geteilt werden

- **Hierarchisierung**
  Module können wiederum in feinere Teilmodule zerlegt werden.
- **Kapselung**
  Details eines Moduls werden in ihrer Sichtbarkeit nach außen bewusst begrenzt (**Information Hiding**).
- **Hohe Kohäsion**
  Innerhalb eines Moduls soll ein enger Zusammenhang zwischen seinen Teilen bestehen (d.h. die Verwandtschaft zwischen den Teilen eines Moduls).
- **Geringe Kopplung**
  Zwischen Modulen sollen möglichst wenig Abhängigkeiten bestehen (d.h. die Anzahl und Komplexität der Schnittstellen).

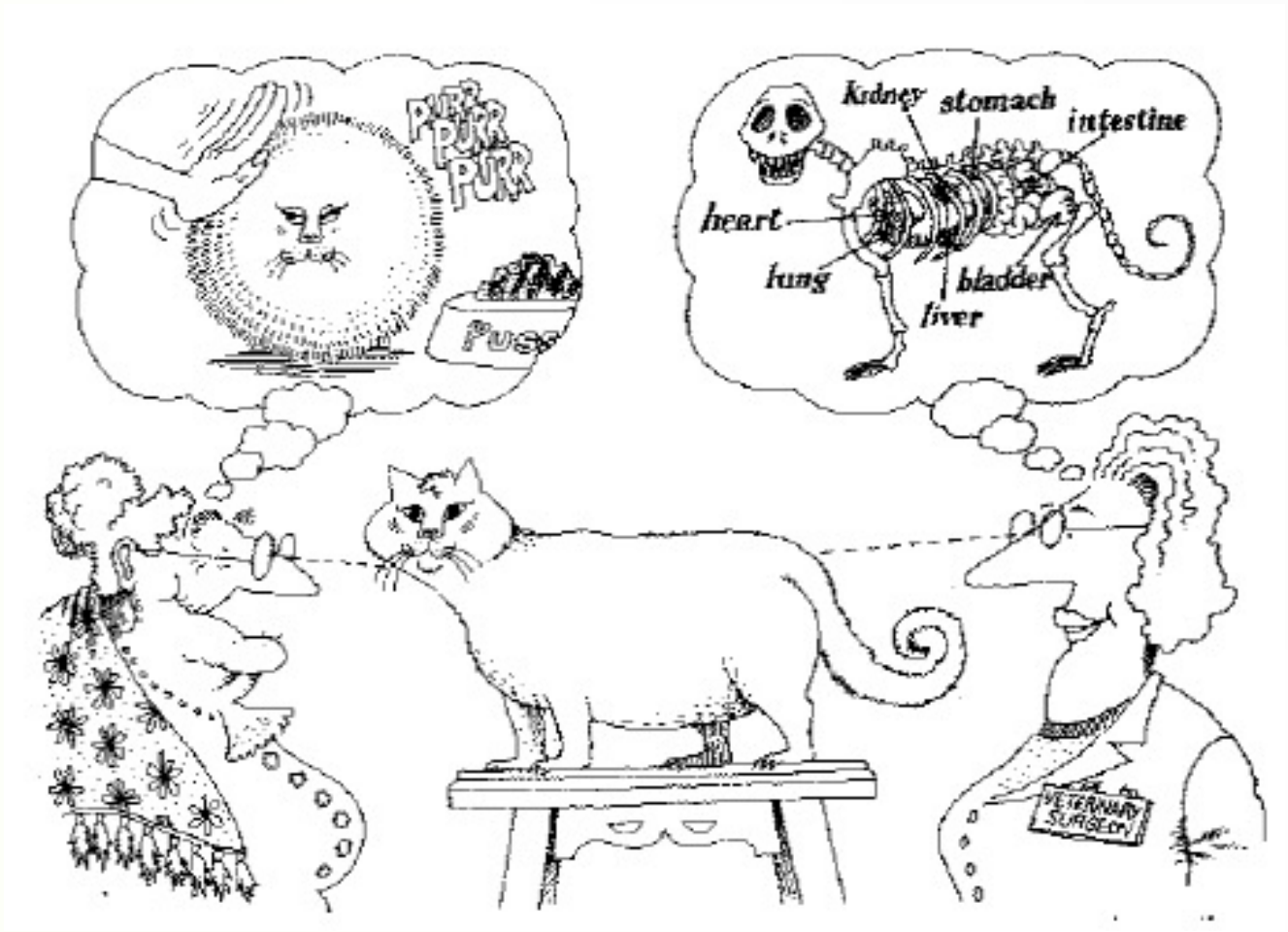
Ein komplexes System kann in **Module (= Komponenten)** geteilt werden.



Ein komplexes System kann in **Module (= Komponenten)** geteilt werden.

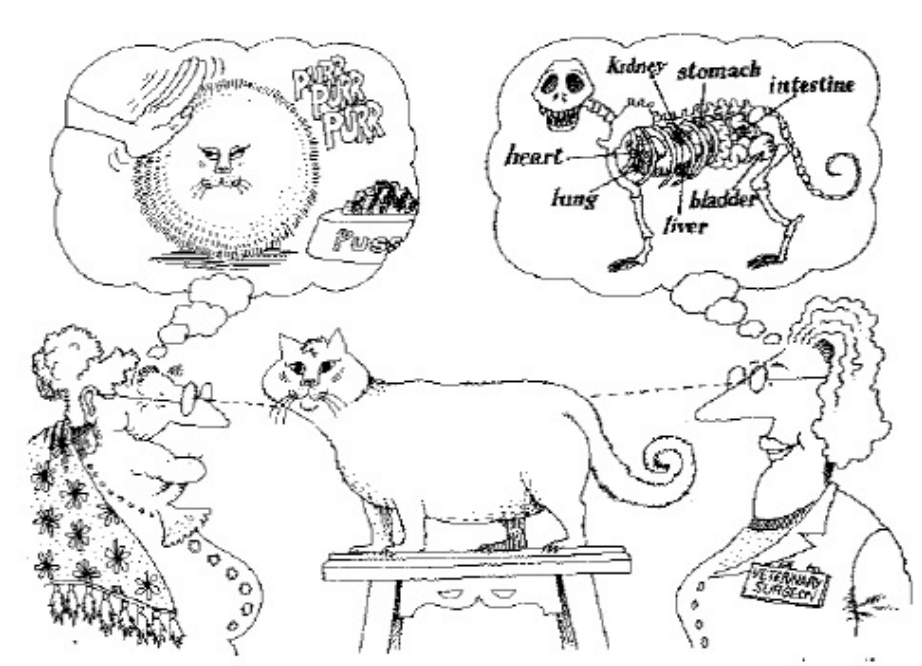


Abstraction

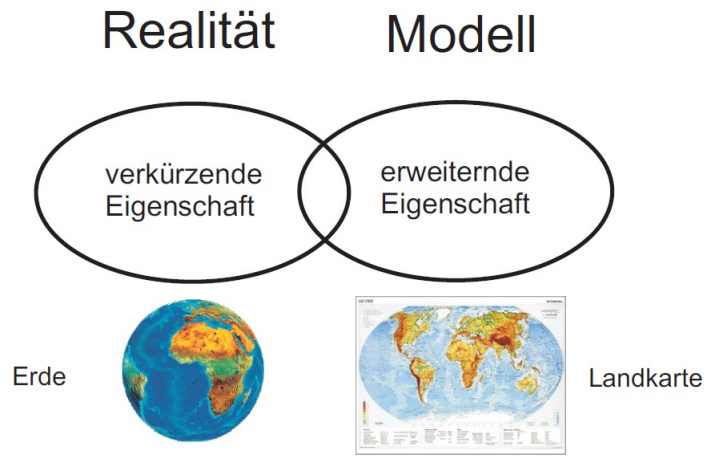


Abstraktion (Verallgemeinerung)

- Identifizierung **wichtiger Aspekte** und **Vernachlässigung** der **Details**
- Pragmatismus: Abstraktionsniveau/Sicht ist **zweckgebunden**
- Abstraktion ist notwendig, um **Komplexität** zu **beherrschen**



Abstraktion (Verallgemeinerung)



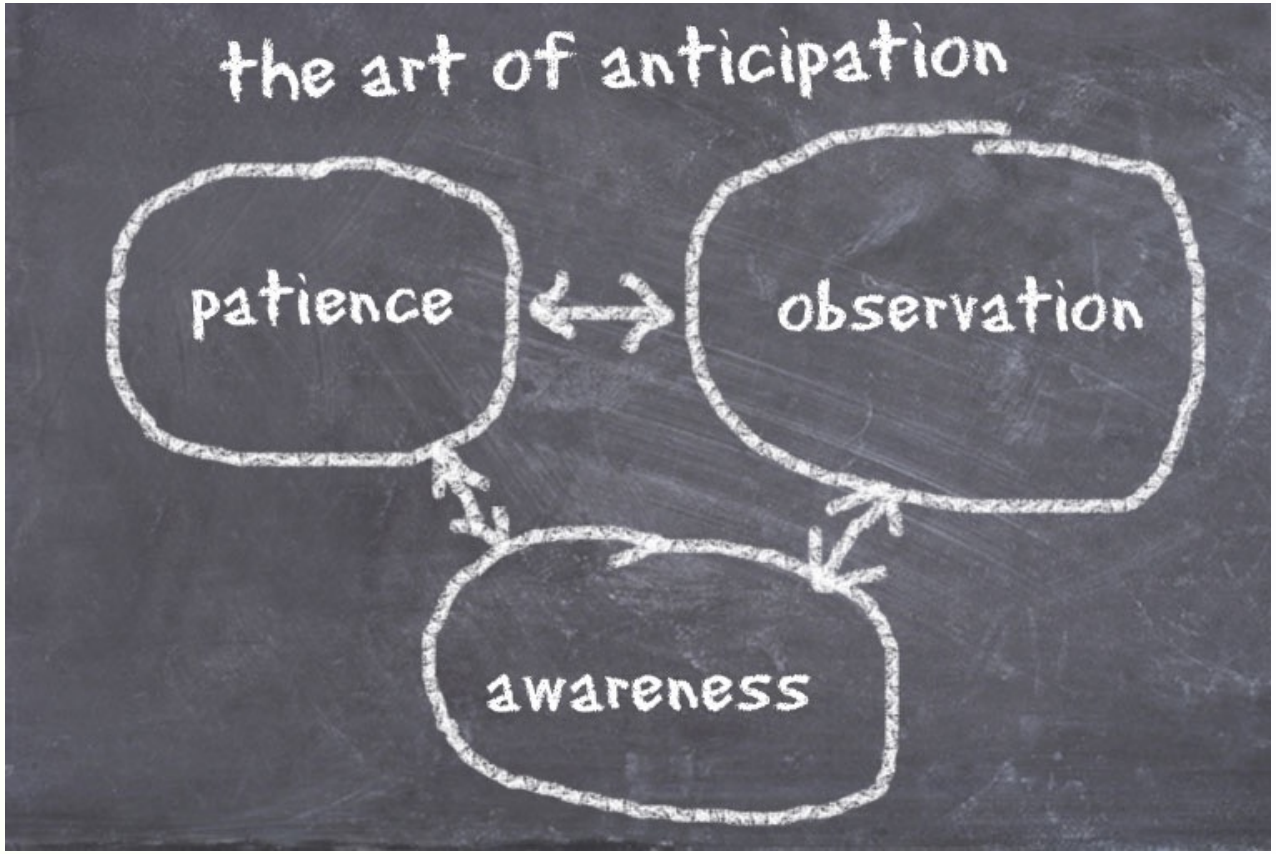
Verkürzende Eigenschaften:
z.B. Temperatur, Luftfeuchtigkeit

Erweiternde Eigenschaften:
z.B. Längen- und Breitengrade, Legende

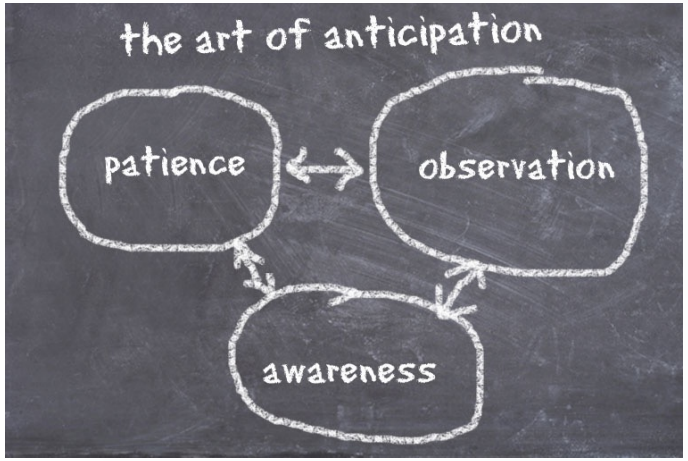
Anwendung

- Software-Modelle
  - In Analyse- und Entwurfsphase entstehen i.d.R. semiformale Modelle des zu konstruierenden Softwaresystems
  - Modell-Simulationen erlauben Schlussfolgerungen über das reale System
- Kostenschätzungen
  - Bei Kostenschätzungen werden Kennzahlen auf Basis von Erfahrungen in anderen Projekten geschätzt
  - Unterschiede in Details werden vernachlässigt

Anticipation of Change



Früherkennung von zukünftig erforderlichen Änderungen
unterstützt die Software-Evolution und verlängert den Lebenszyklus



Mögliche Ursachen für Änderungen

- Beseitigung von Fehlern (**korrektive Wartung**)
- Verbesserung nicht-funktionaler Eigenschaften (**perfektive Wartung**)
- Anpassung der Funktionalität wegen sich ändernder Bedingungen (**adaptive Wartung**)
- Erweiterung der Funktionalität auf Grund von Erkenntnisgewinn über den Software-Lebenszyklus (**inkrementelle Wartung**)

Generality

Ist mein Problem neu?

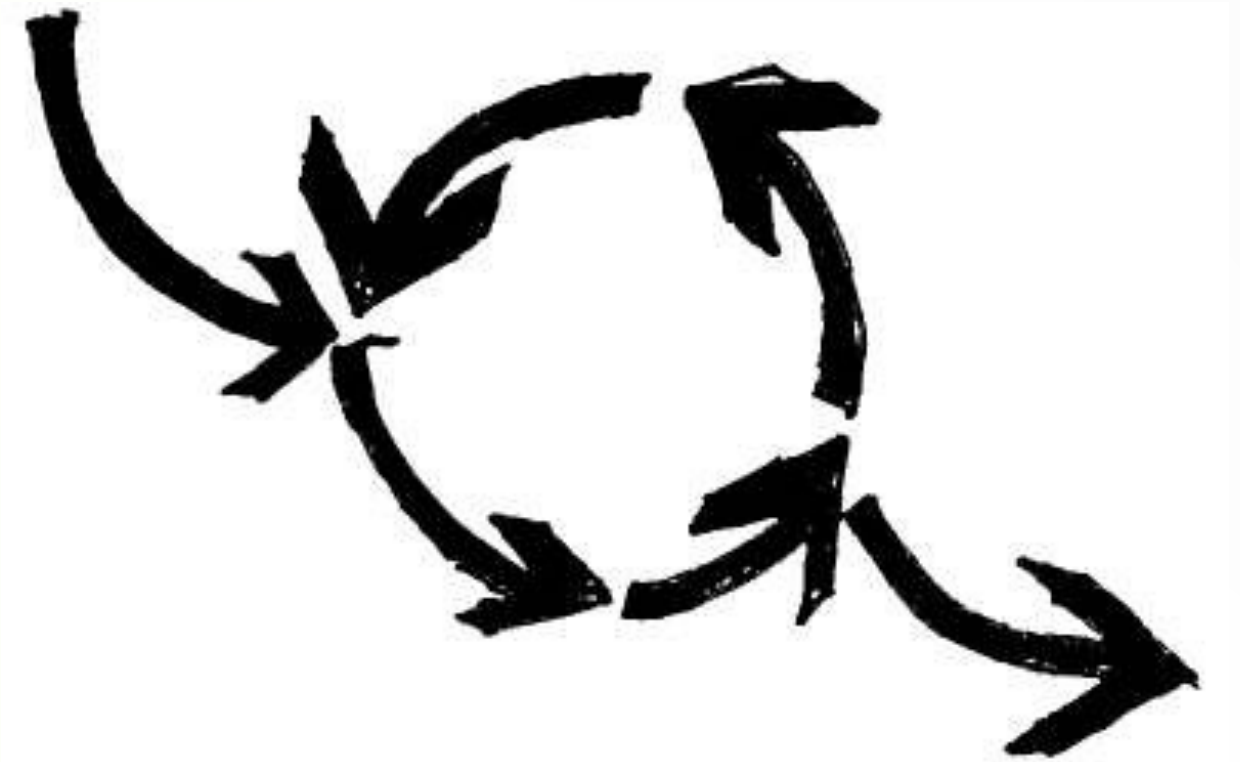
Bei der Lösung eines Problems ist zu **hinterfragen**, ob es ein spezielles Vorkommen eines allgemeinen Problems ist, für das bereits ein **bewährtes Lösungsmuster** vorliegt.

- Möglichkeiten der **Wiederverwendung** erprobter Ansätze prüfen
- Vor- und Nachteile einer **individuellen Lösung** gegenüber einer **adaptierten Standardlösung** abwägen

Erfahrungen aus dem Modul **Programmierung II?**

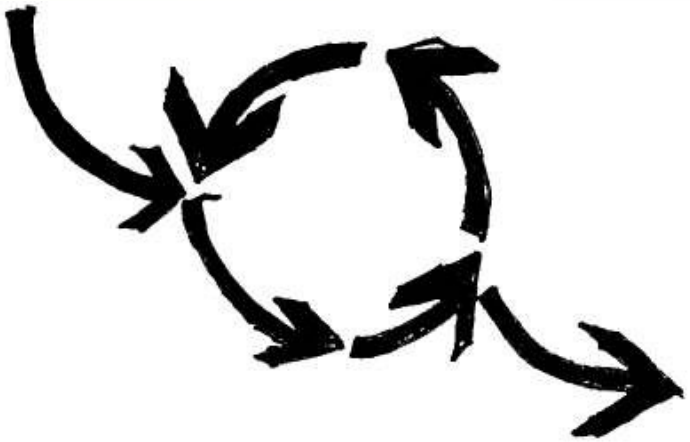
- Persistenz, bspw. Speicherung als CSV
- `javax.swing.SwingWorker`
- ...

Incrementality



Incrementality

- Inkrementalität impliziert **iteratives Vorgehen**
- Nach jeder Iteration erfolgt eine **Rückkopplung**
- Inkrementalität basiert auf der Annahme, dass im Verlauf der Entwicklung **Änderungen nicht zu verhindern** sind



DUALE HOCHSCHULE SH

in Trägerschaft der Wirtschaftsakademie Schleswig-Holstein

★ Vielen Dank für die Aufmerksamkeit ★