Foto: Pixabay auf Pexels

Software-Design und -Architektur

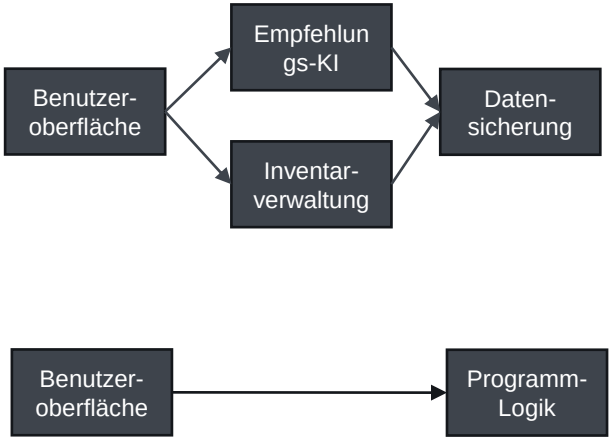
Teil 3 – Software-Komposition

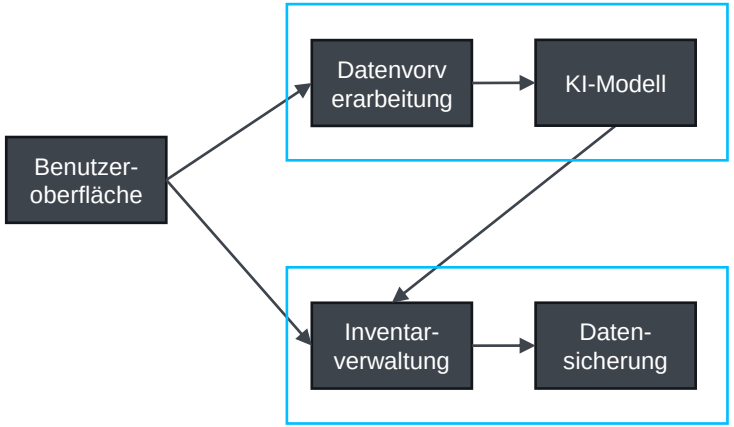
INNOVET
KI B³
Künstliche Intelligenz in die berufliche Bildung bringen

INNOVET
GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung
bibb Bundesinstitut für Berufsbildung

Gefördert als InnoVET-Projekt aus Mitteln des Bundesministeriums für Bildung und Forschung.

Wie baue ich ein Software-System auf?



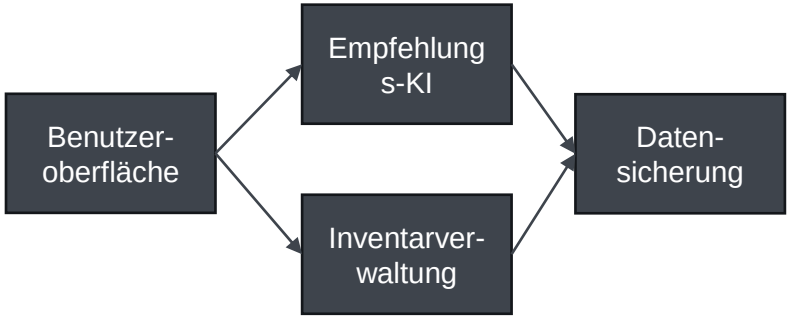


- Ein Software-System kann auf viele verschiedene Weisen aufgebaut werden

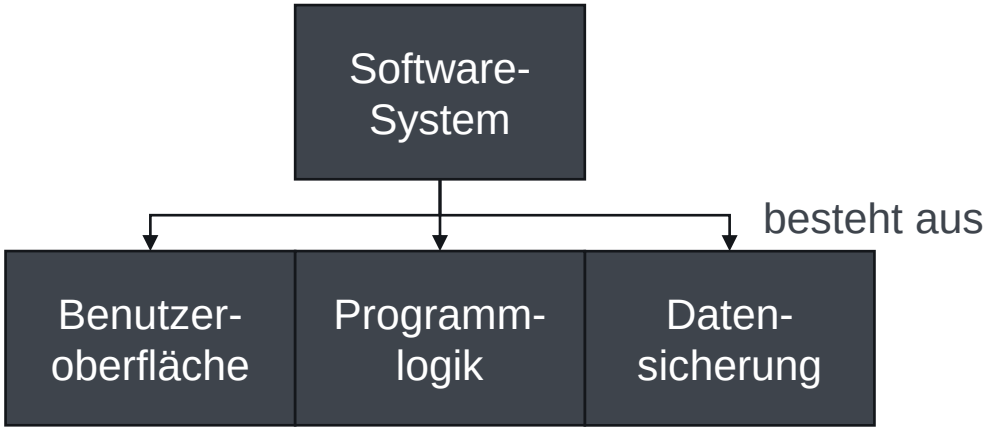
Wie baue ich ein Software-System auf?

Beim Entwurf der Architektur stellen sich folgende Fragen:

- Wie teile ich ein System auf?
- Welche Teile sollten zusammengehören?
- In welcher Beziehung stehen die Teile des Systems?



Komponenten



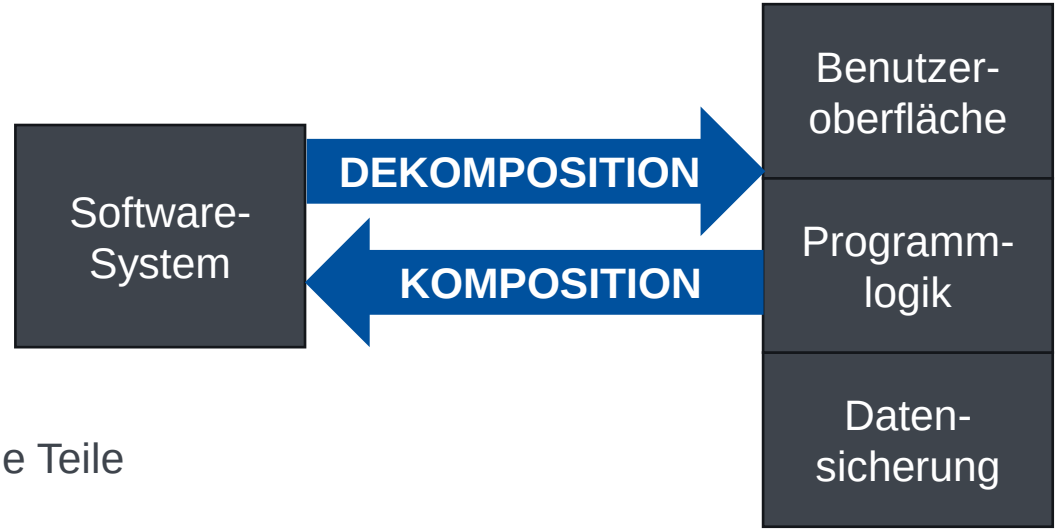
- Wir teilen ein Software-System in kleinere Teile auf, **Komponenten**
- Dieser Ansatz vereinfacht Entwicklung und Wartung, indem er komplexe Software in kleinere, handhabbare Teile gliedert.

Definition: Komponente

Eine "Komponente" ist ein modularer, wiederverwendbarer Teil eines Softwaresystems, der eine spezifische Funktion ausführt und mit anderen Komponenten interagieren kann.

Synonym: **Modul**

Komposition und Dekomposition



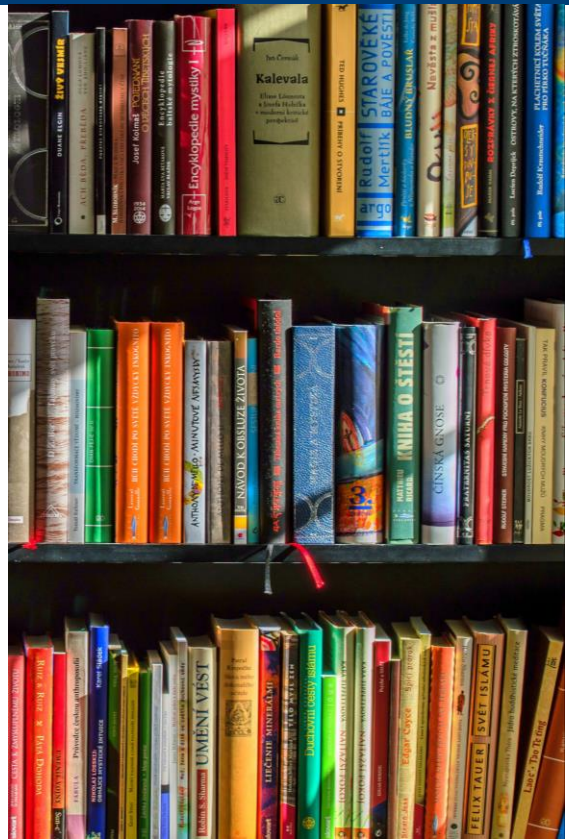
- **Dekomposition**
  - Aufteilung eines Ganzen in seine Teile
- **Komposition**
  - Zusammenführung von Teilen zu einem Ganzen

Definition: Schnittstelle (Interface)

Schnittstellen beschreiben die Kontaktstellen zwischen Komponenten. An Schnittstellen wird definiert, wie Komponenten miteinander kommunizieren.

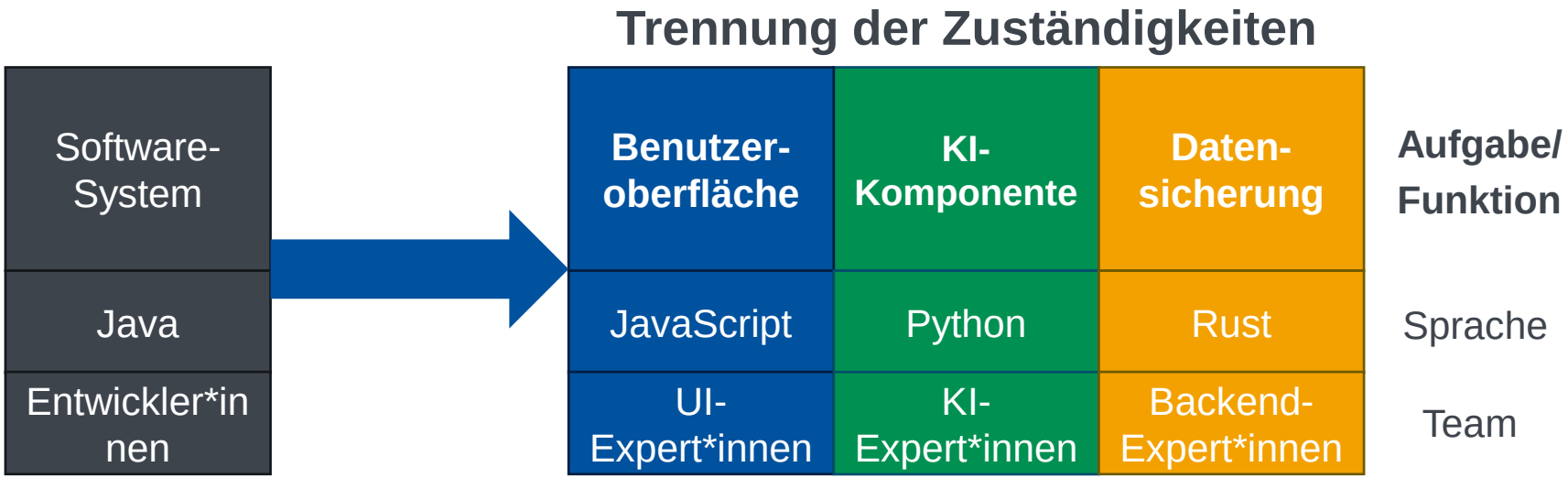
Design-Prinzipien in der Software-Architektur

- Beim Planen der Software-Architektur entscheiden wir, wie wir unser System strukturieren und aufteilen
  - Dabei können wir bewährte **Design-Prinzipien** anwenden.
- **Ziel**: Effiziente, wartbare und skalierbare Software-Architektur
- Berücksichtigung von Aspekten wie Modularität, Wiederverwendbarkeit und Wartbarkeit



Trennung der Zuständigkeiten (Separation of Concerns)
Das Prinzip, ein Programm in verschiedene Abschnitte zu organisieren, wobei jeder Abschnitt einen spezifischen Aspekt oder ein spezifisches Anliegen der Anwendung behandelt.

Trennung der Zuständigkeiten: Beispiel



- Teile der Software, die funktional oder thematisch eng zusammengehören, sollten in Komponenten oder Modulen gebündelt werden

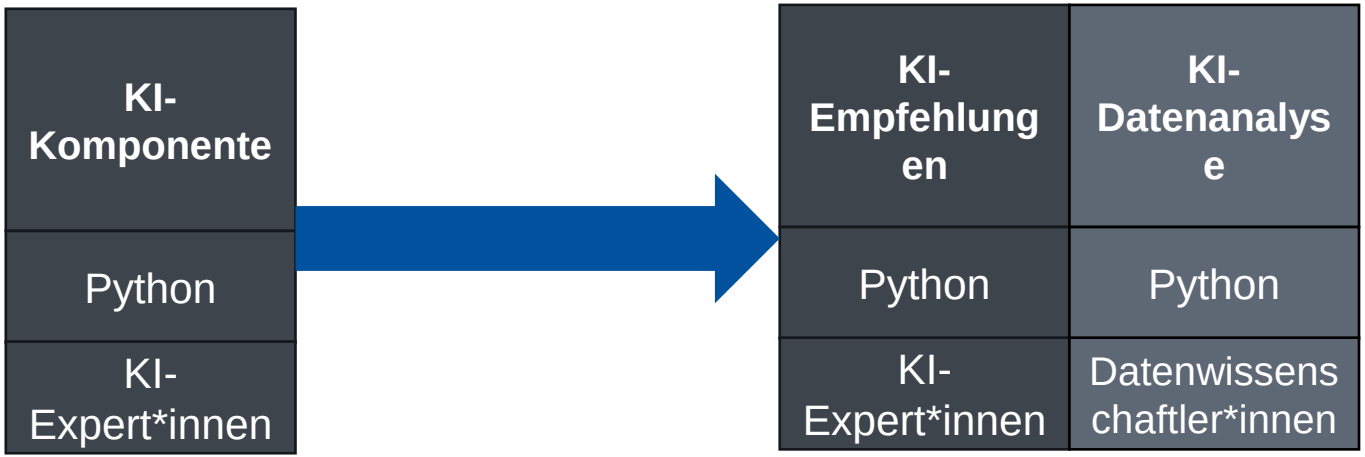
Trennung der Zuständigkeiten: Vorteile

- Das große Gesamtsystem wird in kleinere, **übersichtlichere** Teile aufgeteilt
- Komponenten können **unabhängig** voneinander entwickelt werden
- Komponenten-spezifische Eigenheiten können leichter **ersetzt** werden
  - Teams
  - Technologien
  - Code

→ Dieser Ansatz vereinfacht Entwicklung, Wartung und Verständnis des Systems, indem er unterschiedliche Teile des Programms isoliert, die unabhängig oder lose gekoppelt sind.

(Prinzip der Einzelverantwortung)
Single Responsibility Principle
Ein Konzept, das besagt, dass eine Komponente nur einen einzigen Aspekt der durch die Software bereitgestellten Funktionalität verantwortlich sein sollte, um sicherzustellen, dass jeder Teil des Programms fokussiert ist und einen klaren Zweck hat.

Prinzip der Einzelverantwortung: Beispiel



- Jede Komponente fokussiert sich auf eine spezifische Funktion oder Verantwortlichkeit, um Überladung und Vermischung von Aufgaben zu vermeiden.

Prinzip der Einzelverantwortung: Vorteile

- Verhindert, dass zu viele Funktionen der Software von einer Komponente abhängig sind
  - Keine **Überladung**
- Reduziert **Abhängigkeiten** zwischen verschiedenen Teilen der Software
- Vereinfacht Wartung, Fehlerbehebung und Erweiterung des Codes
- Unterstützt **klar definierte**, **wiederverwendbare** und **testbare** Code-Einheiten

Prinzipien sind nicht alles: KISS

- **KISS**: **K**eep **i**t **s**imple, **s**tupid!
  - Sinngemäße Übersetzung: **Einfachheit ist der Schlüssel!**

- Die intuitive Lösung ist in vielen Fällen ausreichend!

- Design-Prinzipien sollten als grobe Leitgedanken gesehen werden, nicht dogmatisch auf das Wort befolgt werden.

INNOVET
KI B³
Künstliche Intelligenz in die berufliche Bildung bringen

Universität Stuttgart
Institut für Maschinelle Sprachverarbeitung
Institut für Software Engineering

Erstellt durch:

Marvin Muñoz Barón

Umm-e-Habiba

Tobias Eisenreich

Universität Stuttgart
Institut für Software Engineering
Empirisches Software Engineering

IHK Industrie- und Handelskammer Reutlingen
Reutlingen | Tübingen | Zollernalb

IHK Region Stuttgart

IHK Industrie- und Handelskammer Karlsruhe

LMU Ludwig-Maximilians-Universität München

GEFÖRDERT VOM
INNOVET
Bundesministerium für Bildung und Forschung
bibb Bundesinstitut für Berufsbildung

Gefördert als InnoVET-Projekt aus Mitteln des Bundesministeriums für Bildung und Forschung.

Lizenzbestimmungen

Quellenverzeichnis

- Foto von Pixabay: https://www.pexels.com/de-de/foto/notenblatt-auf-orgel-210764/ unter Pexels Lizenz (https://www.pexels.com/license/). Bildausschnitt verändert.
- Foto von Pixabay: https://www.pexels.com/de-de/foto/bucher-im-schwarzen-holzernen-bucherregal-159711/ unter Pexels Lizenz (https://www.pexels.com/license/). Bildausschnitt verändert.