

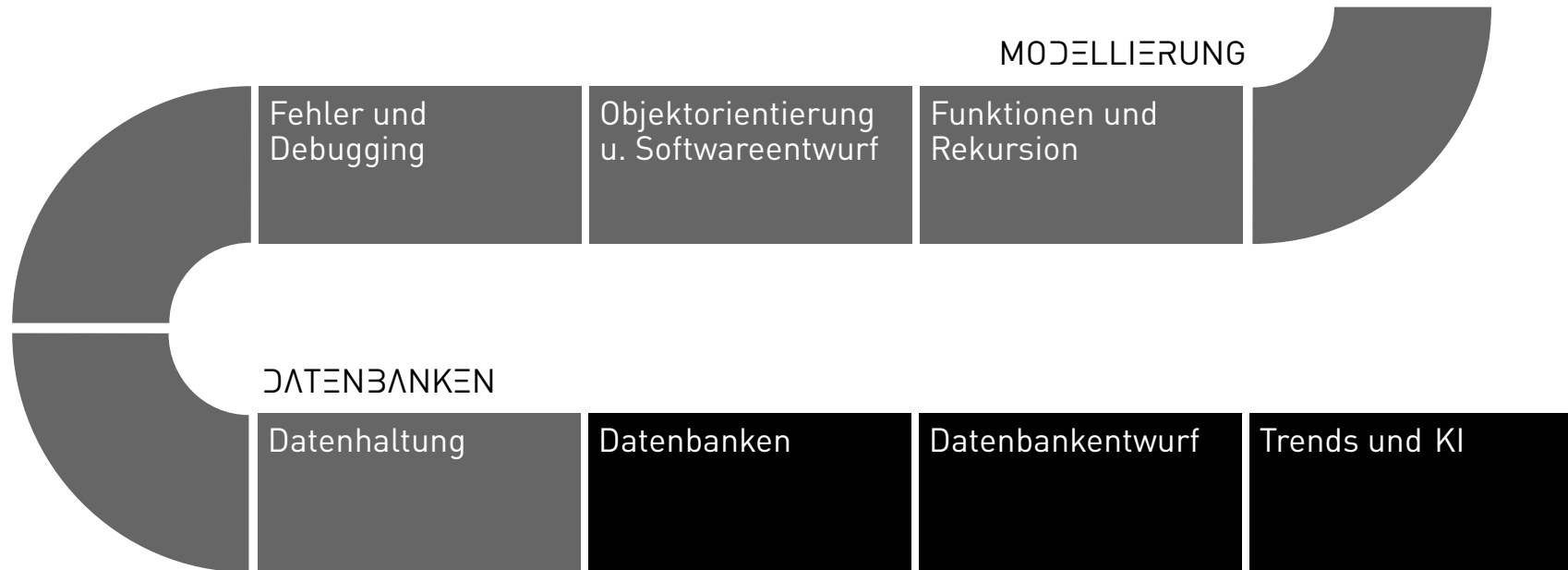
programmierung und datenbanken

Joern Ploennigs

Datenhaltung

MIDJOURNEY: LIBRARIAN, REF. GIUSEPPE ARCIMBOLDO

ABLAUF



WO LIEGEN UNSERE DATEN?

Smartphones und Tablets

- Nutzung: Möglichst simpel bedienbare, fokussierte Anwendungen
- Daten: Daten werden App-bezogen auf dem mitgelieferten Speicher gelagert

Desktop-PCs

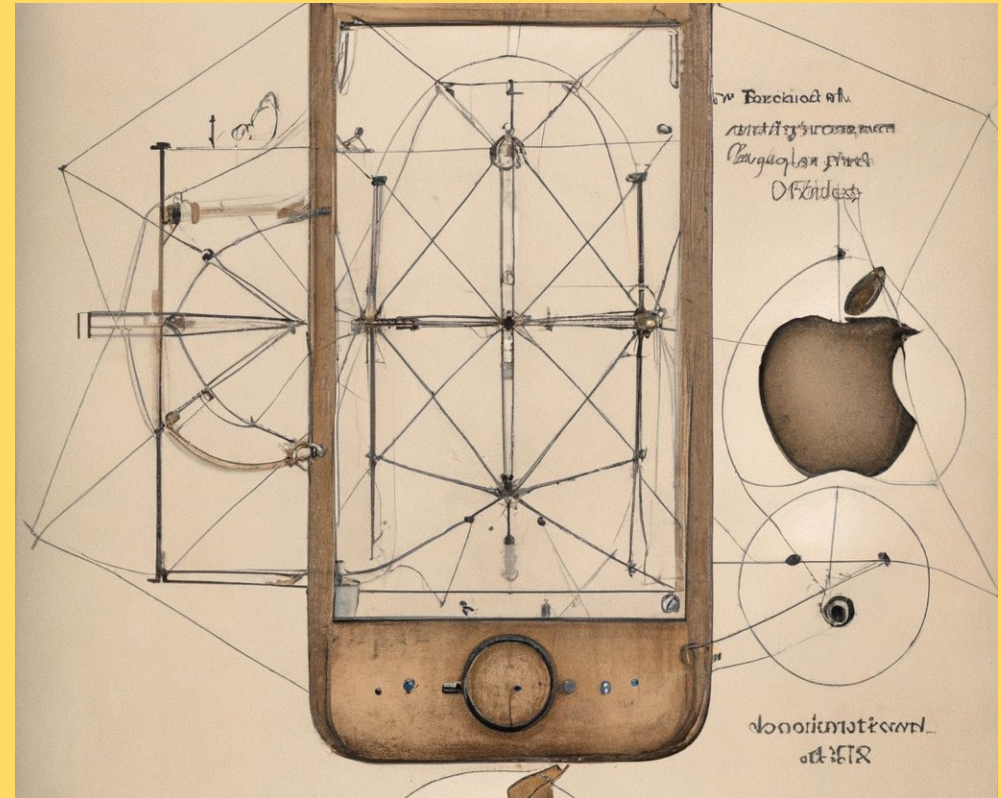
- Nutzung: Heutzutage meist als Arbeitsplatzrechner oder Hobby-Maschine
- Daten: Daten liegen in Ordnerstrukturen, gespeichert auf lokalen Laufwerken

Web- und Cloudanwendungen

- Nutzung: Überall von Apps bis zu Hochleistungsrechnen
- Daten: Liegen in weltweit verteilten Serverfarmen, meist von großen Konzernen verwaltet

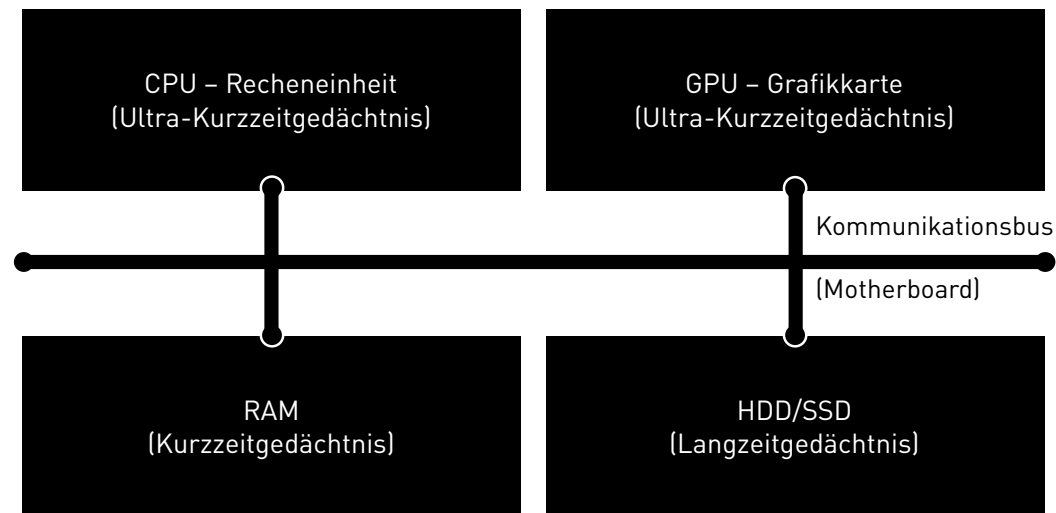
HÖRSAALFRAGE

- Aus welcher Hardware besteht ein Computer?
- Wie vergleichen diese sich zum Gedächtnis des Menschen?



DALL-E 2: Early designs of the iPhone by Leonardo da Vinci

EINFÜHRUNG - Wo werden Daten im Computer ABGELEGT?



Der Computer hat ähnliche Gedächtnisarten wie der Mensch:

- CPU und GPU haben kleine Register und Cache Speicher (Ultra-Kurzzeitgedächtnis)
- Der RAM ist ein volatiler Speicher, d.h. der Inhalt geht beim ausschalten verloren (Kurzzeitgedächtnis)
- Die HDD/SSD ist ein permanenter Speicher, d.h. der Inhalt bleibt erhalten (Langzeitgedächtnis)

CPU REGISTER & CACHE - SPEICHERUNG AUF PROZESSOREBENE

- *Register* – Der von der CPU zum Rechnen benutzte Speicher (sehr klein)
- *Cache* – Hier werden Code und Daten vorausschauend geladen (Caching), die vmtl. gleich gebraucht werden oder welche woanders hin geschrieben werden sollen
- Eigenschaften:
 - Sehr schneller, prozessorinterner Speicher
 - Teuer, extrem kleine Kapazität
 - Muss möglichst in der selben Geschwindigkeit arbeiten wie das Rechenwerk um keinen Flaschenhals zu erzeugen

RAM - RANDOM ACCESS MEMORY

- Auch „Direktzugriffsspeicher“ oder „Arbeitsspeicher“
- Schreib-Lese-Speicher, der nicht sequentiell gelesen werden muss, sondern in dem Daten direkt über ihre Adresse angesprochen werden können
- Diese Zugriffe sind schnell, Blöcke können effizient angesprochen werden
- Heutzutage meistens im Kontext von CPU-/GPU-nahem Arbeitsspeicher verwendet, d.h. es werden aktuell benötigte Daten verwaltet, welche verloren gehen wenn der Strom verloren geht

WIE WERDEN DATEN IM RAM ORGANISIERT?

- Die Speicherzellen im RAM sind in Blöcke mit Adressen eingeteilt
- Das Betriebssystem weist jedem laufenden Programme so viele Blöcke wie dieses Programm braucht zu
- Programme organisieren sich diese Blöcke in *Stack* und *Heap*:
 - *Stack* enthält die Funktionsaufrufe und wichtige (einfache) Variablen
 - *Heap* enthält alle anderen Variablen
- Jede Variable im Code besteht aus:
 - ein Verweis (Pointer) auf diese Adresse
 - die Länge der Variable im Speicher (gegeben durch den Datentyp)
 - bei Programmiersprachen mit Garbage Collection (Python, Java, JavaScript, etc.) gibt es für jede Variable noch einen Zähler wie oft die Variable verwendet wird

HDD/SSD – HARD DISK DRIVE / SOLID STATE DRIVE

Hard Disk Drive (HDD)

- Speicher auf dem Daten magnetisch abgespeichert werden
- Können viele Jahre halten
- Erschütterungsempfindlich

Solid State Drive (SSD)

- Speicher auf dem Daten in elektrischen Ladungen abgespeichert werden
- Können mehrere Jahre halten, entladen sich aber irgendwann
- Zum Löschen wird ein Spannungssprung (Flash) benutzt → Flash-Speicher

Magnetbänder

- Daten werden magnetisch auf Plastikbändern abgespeichert
- Die Daten halten viele Jahrzehnte
- Werden heutzutage noch für Backups benutzt
- Preiswert, allerdings sehr langsam

WIE WERDEN DATEN IN LANGZEITSPEICHERN ORGANISIERT?

- Die Speicherzellen in Langzeitspeichern sind auch in Blöcke mit Adressen eingeteilt
- Das Betriebssystem organisiert diese Blöcke und merkt sich welche Daten wo gespeichert werden (z.B. welche Blöcke zu welcher Datei gehören)
- Diese Organisationsstruktur nennt man *Dateisystem*

DATEISYSTEM - DESKTOP-PCs: DATEIEN IN ORDNERSTRUKTUREN

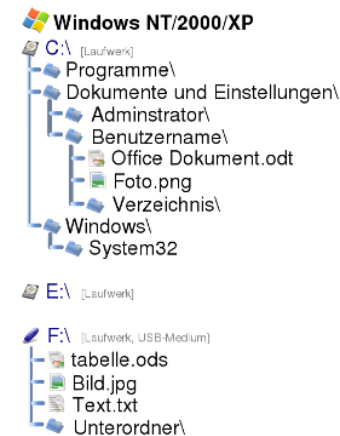
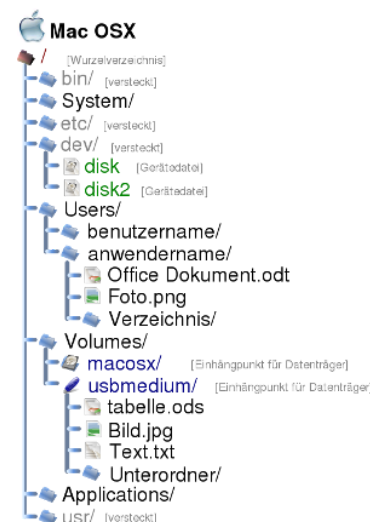
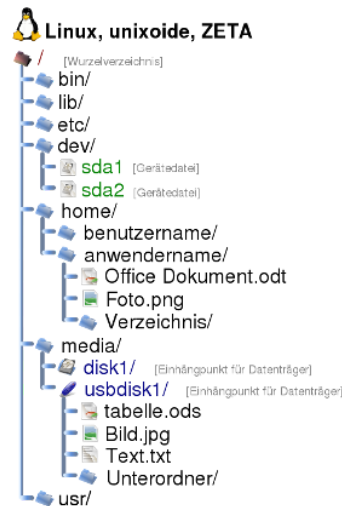
- Hierarchisch (wie bei der Büroablage) in Laufwerke (Register), Ordner & Dateien organisiert
- Daten werden in Dateien abgespeichert
- Diese werden in Ordnerhierarchien (Baumstruktur!) abgelegt
- Identifiziert werden sie durch Ordnerpfade und Namen

Vorteile:

- So können sehr viele Dateien organisiert werden

Nachteile:

- Wird sehr schnell unübersichtlich
- Programme können fast alles lesen



DATEISYSTEM - SMARTPHONES & TABLETS: APP-SPEZIFISCHE SPEICHERUNG

- Daten werden meist Apps zugeordnet (gekapselt)
- So sieht die App nicht das volle Dateisystem (und der Nutzer oft auch nicht)
- Dadurch kann die App weniger Unfug treiben und die Sicherheit wird verbessert

Vorteile:

- Intuitivere Benutzung mit weniger Dateien pro App
- Erhöhte Sicherheit

Nachteil:

- Es ist schwierig Dateien zwischen Apps zu synchronisieren

DATEISYSTEM - CLOUD: DATENBANKEN

- Daten können nicht lokal gespeichert werden
- Werden meist nur in Datenbanken gespeichert → siehe nächste Vorlesung

Vorteile:

- Programme und Daten sind physikalisch getrennt
- Viele Programminstanzen können auf die gleichen Daten zugreifen
- Beliebig hinzugefügt oder entfernt werden

Nachteil:

- Schwerer zu konfigurieren und zu debuggen
- Eine gewisse „Entmündigung“

DATEISYSTEM - IN PYTHON

- Python verallgemeinert das Arbeiten mit Dateisystemen so weit wie möglich
- Viele verschiedene Funktionen und Bibliotheken! `os`, `io`, `open()`, `fileinput` ...
- Verschiedene Abstraktionsgrade – von direkten String-basierten Leseoperationen bis hin zur hierarchischen, objektorientierten Repräsentationen ganzer Ordnerstrukturen
- Dateien werden selten „im Ganzen“ ausgelesen (ineffizient bei großen Dateien), sondern Zeile für Zeile, Zeichen für Zeichen, oder auch selektiv z.B. durch ein Inhaltsverzeichnis

DATEIFORMATE – GROBE EINTEILUNG

Text

- Die Datei ist ein großer String
- Kann von Menschen und Software gelesen werden
- Resultiert meist in größeren Dateien
- Einfacher zu Debuggen da lesbar
- Gut für strukturierte Inhalte (z.B. Text, Attribute, Statistiken)
- Zusätzliche Struktur wird mittels Syntax-Regeln hinzugefügt (.csv, .json, ...)

Binär

- Die Datei ist ein Bytearray
- Nur durch Software lesbar, nicht durch den Menschen
- Resultiert in meist kleineren Dateien
- Schwerer zu Debuggen da nicht lesbar
- Gut für unstrukturierte und große Inhalte (z.B. um Bilder und Videos darzustellen)
- Dateiendungen signalisieren mit welchem Programm Dateien verlinkt sind

DATENFORMATE FÜR BU-INGENIEURE - TYP 1: GEOMETRISCHE MODELL FORMATE

- Vektor- und Rasterdaten definiert mit einem Koordinatensystem
- Verschiedene Dimensionalitäten (2D, 2.5D, 3D, ...)
- Häufig genutzte Formate:
 - *2D-Planungs-Formate*: DWG, DXF, SVG, PDF, PNG, TIFF
 - *3D-Planungs-Formate*: IFC (STEP), IFC (XML), IFC (JSON), DWG, DXF, OBJ, 3DS
 - *Geodaten-Formate*: Shapefile, GML (XML), KML (XML), GeoTIFF, GeoJSON (JSON)

DATENFORMATE FÜR BU-INGENIEURE - TYP 2: ATTRIBUTFORMATE

- Deskriptive, nicht-geometrische Daten für spezifischen Kontext
- Oft durch Tabellen oder Listen von Datenobjekten organisiert
- Häufig genutzte Formate:
 - CSV, ODF (XML), XLSX (XML), XLS, JSON
 - Unterschiedliche Ebenen an Komplexität

DATENFORMATE FÜR BU-INGENIEURE - TYP 3: GEOMETRIEFORMATE

- Geometrische Daten sind meist mathematisch und haben keinen klar definierten Weg
- Grafikformate definieren solche „Stile“, z.B. für Punkte, Linien, Polygone etc.
- Häufig genutzte Formate:
 - CSS, SLD (XML), ArcGIS Styles (*.lyr)

DATENFORMATE FÜR BU-INGENIEURE - TYP 4: TOPOLOGIEFORMATE

- Geometrie durch Nachbarschaftsbeziehungen statt Koordinatensystemen
- Knoten, Kanten, Maschen, Gitter
- Häufig genutzte Formate:
 - GML (XML), TopoJSON (JSON)

AUSTAUSCHFORMATE - JSON (JAVASCRIPT OBJECT NOTATION)

- Menschen- und maschinenlesbares strukturiertes Datenformat
- Hauptsächlich als Austauschformat genutzt
- Realisiert durch Key-Value Paare (ähnlich wie Python Wörterbücher)
- Erlaubt die Abbildung aller Datentypen aus Python:
 - Number
 - String
 - Boolean
 - List
 - Dict
 - None (null)

```
{  
  "firstName": "John",  
  "lastName": "Smith",  
  "isAlive": true,  
  "age": 25,  
  "address": {  
    "streetAddress": "21 2nd Street",  
    "city": "New York",  
    "state": "NY",  
    "postalCode": "10021-3100"  
  },  
  "children": [ ],  
  "spouse": null  
}
```

AUSTAUSCHFORMATE - XML (EXTENSIBLE MARKUP LANGUAGE)

- Markup-Sprachen erlauben es Teile eines Textes zu markieren, um weitere (meist maschinenlesbare) Informationen und Semantik hinzuzufügen
- XML ist dabei eine Meta-Sprache, die es erlaubt solche Sprachen zu definieren
- Markup geschieht hier durch öffnende und schließende Tags, die mit `<` und `>` eingeklammert werden
- Beispiele für XML-nahe Sprachen:
 - HTML
 - XHR (XML HTTP Request)
 - GML

```
<person>
  <name> John </name>
  <isAlive> true </isAlive>
  <age> 25 </age>
  <address>
    <cityStreet> New York, 21 2nd
Street </cityStreet>
    <postalCode> 10021-3100
  </postalCode>
  </address>
  <children> </children>
  <spouse> </spouse>
</person>
```

AUSTAUSCHFORMATE - CSV (COMMA SEPARATED VALUES)

- Textuelle Darstellung von strukturierten Daten (Tabellen, Listen usw.)
- Tabellenzeilen sind Zeilen und Spalten, diese sind durch Begrenzungszeichen angeordnet
- Begrenzungszeichen können Semikolon, Komma, Tabulator usw. sein

```
FirstName;LastName;IsAlive;Age  
John;Smith;true;25  
Mary;Sue;true;30
```

LESSON LEARNED



Midjourney: Wilhelm Tell child with an apple on his head that has a arrow in it

LESSON LEARNED - ZIELESETZUNG

- **S**pezifisch: Ziel leicht verständlich und mit nur zwei Sätzen genau beschreiben.
- **M**essbar: Die Zielerreichung ist quantitativ oder qualitativ feststellbar.
- **A**traktiv: Die Erreichung des Zieles ist für EUCH erstrebenswert (ich-Bezug).
- **R**ealistisch: Das Ziel ist ambitioniert und erreichbar.
- **T**erminiert: Konkreten Termin bis wann das Ziel erreicht werden soll.

fragen?

