Wissenschaftliche Softwareentwicklung

27. Datenformate: beständig und plattformunabhängig

Till Biskup
Physikalisch-Technische Bundesanstalt
19.02.2024

Zentrale Aspekte

- Formate betreffen nicht nur Roh- und verarbeitete Daten, sondern auch Metadaten, Dokumentation, Abbildungen.
- Datenformate sollten über Jahrzehnte lesbar, plattformunabhängig, quelloffen und dokumentiert sein.
- Daten über Jahrzehnte lesbar zu archivieren, ist nicht nur eine Frage der Formate, sondern auch der Organisation.
- Rohdaten sollten immer (im Originalformat) archiviert und vor (ungewollter) Veränderung geschützt werden.
- Das konkrete Datenformat oder die Art der Datenlagerung ist für ein System zur Datenverarbeitung irrelevant.

Übersicht

Kriterien für Datenformate in der Wissenschaft

Beispiele plattform- und sprachunabhängiger Formate

Zum Umgang mit Daten und Metadaten

Bedeutung im Gesamtkontext einer Auswertungssoftware

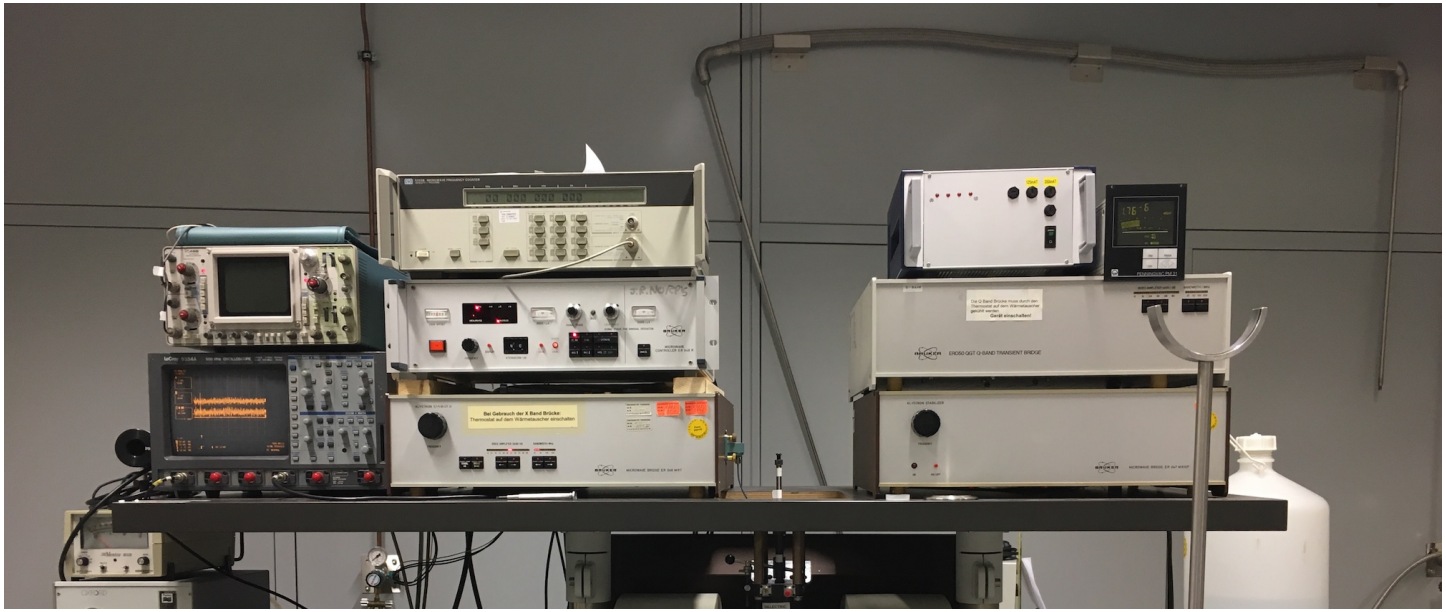
Warum sich mit Datenformaten befassen?

Ein paar Gründe, warum sie jeden Wissenschaftler etwas angehen

- Wir sind alle Nutzer von Datenformaten.
  - Datenformate haben verschiedene Formen/Funktionen.
  - Der *bewusste* Umgang ist auch hier entscheidend.
- Wissenschaftler tragen Verantwortung für ihre Daten.
  - Daten sind die Grundlage aller empirischen Wissenschaften.
  - Nachvollziehbarkeit erfordert Zugriff auf die Daten.
  - Zugriff erfordert offene und plattformunabhängige Formate.
- Datenformate betreffen nicht nur (Roh-)Daten.
  - Metadaten, Dokumentation der Auswertung, Abbildungen
  - Beständigkeit und offener Zugang sind auch hier wichtig.
- Wir alle erzeugen (oft unbewusst) Datenformate.
  - Zwischenergebnisse werden in Dateien abgelegt.
  - Auch ein handgeschriebenes Laborbuch zählt dazu.

Warum sich mit Datenformaten befassen?

Aufbauten aus Einzelkomponenten mit handgeschriebener Steuerung



- Nicht alle Gerätschaften sind aus einem Guss . . .
  - Steuersoftware ist oft selbst geschrieben.
  - gibt freie Hand bei der Wahl der Datenformate

☛ Beispiel: Die Steuersoftware für ein EPR-Spektrometer wurde komplett von einem Doktoranden entwickelt.

Kriterien für Datenformate

Was man bei Nutzung und Entwicklung von Formaten beachten sollte

- zukunftssicher
  - Das Format sollte Jahrzehnte bestehen können.
  - weitgehend ungelöstes Problem (auch für Archivare)
- plattform- und sprachunabhängig
  - mindestens Windows, Linux/Unix, macOS unterstützen
  - unabhängig von der eingesetzten Programmiersprache
- quelloffen, vollständig dokumentiert, standardisiert
  - ermöglicht unabhängigen Zugriff
- geeignet zur Ablage von Metadaten
  - Daten ohne Metadaten sind wertlos.
  - Metadaten so nah wie möglich bei den Daten ablegen
- versioniert
  - Formate entwickeln sich – wenn auch meist eher langsam.
  - Versionen sollten automatisiert erkennbar/auslesbar sein.

Umgang mit proprietären Rohdatenformaten

Eine leider viel zu oft auftretende Situation

proprietäres Datenformat

auf herstellerspezifischen, nicht veröffentlichten Standards basierend und damit in der Regel nicht von Dritten lesbar

- Probleme
  - nicht zukunftssicher
  - selten plattform- und sprachunabhängig
  - Exportformate meist ohne Metadaten
- Umgang
  - Rohdaten in diesem Format trotzdem archivieren
  - Metadaten soweit verfügbar separat (manuell) ablegen
  - immer Daten exportieren und ebenfalls archivieren

Übersicht

Zwei grundsätzliche Möglichkeiten

Reintext- und Binärformate

- reine Textformate
  - ✔ universell lesbar (zumindest ASCII 7-bit)
  - ✔ auch vom Menschen lesbar
  - ✔ ohne zusätzliche Software unmittelbar zugänglich
  - ✘ ggf. langsam im Zugriff
  - ✘ mitunter mit erheblichem Speicherbedarf
- Binärformate
  - ✔ oft sehr performant
  - ✔ meist geringer Speicherbedarf
  - ✘ nie ohne spezielle Software lesbar
  - ✘ nicht vom Menschen lesbar

☛ Beispiel für die Universalität reiner Textformate: Unix-Kommandozeile

Beispiele für Reintextformate

Eine erste Übersicht ohne Anspruch auf Vollständigkeit

Format	**Struktur**
unstrukturierter Text	keine
DSV/CSV	zeilenweise
Windows-INI	blockweise
XML/JSON/YAML	hierarchisch

- Unterschied bzgl. (wiederkehrenden) Strukturen
  - (Mess-)Daten sind meist strukturell hochrepetitiv
  - Metadaten sind i.d.R. hierarchisch
  - unstrukturierter Text ggf. in Einzelfeldern erlaubt (Kommentare)
- Jedes Format hat seine Berechtigung und Anwendungen.
- Die Liste erhebt keinen Anspruch auf Vollständigkeit.

Beispiel für unstrukturierten Text

Die einfachste Variante, Text in einer Datei abzulegen

Listing 1: Beispiel für unstrukturierten Text

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Vestibulum varius gravida arcu, rutrum finibus
ligula mollis sed.
Fusce blandit suscipit ultricies. Curabitur lacus libero,
accumsan sed velit id, semper ultrices ante.

Vestibulum hendrerit finibus ex sit amet tincidunt. Morbi
nec quam id arcu convallis porta.
```

- keinerlei Struktur
- Zeilenumbrüche und Trennungen von Absätzen willkürlich
- gut geeignet für Text ohne strikt wiederkehrende Struktur (z.B. Kommentare, Berichte mit viel Fließtext)
- logische Textauszeichnung durch Steuerelemente möglich

Beispiel für zeichengetrennte Werte (DSV)

Ein Standardformat aus der Unix-Welt

Listing 2: Beispiel für zeichengetrennte Werte (DSV)

```
root:x:0:0:root:/root:/bin/zsh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
```

- zeilenweise Struktur
- eindeutiges Trennzeichen (hier: Doppelpunkt)
- Soll das Trennzeichen in einem Feld verwendet werden, wird ein Backslash (\) vorangestellt.

Beispiel für kommagetrennte Werte (CSV)

Ein wenig standardisiertes, uneinheitlich implementiertes Format

Listing 3: Beispiel für „kommagetrennte“ Werte (CSV)

```
Year,Make,Model,Description,Price
1997,Ford,E350,"ac, abs, moon",3000.00
1999,Chevy,"Venture ""Extended Edition""","",4900.00
1999,Chevy,"Venture ""Extended Edition, Very Large""",,5000.00
1996,Jeep,Grand Cherokee,"MUST SELL!
air, moon roof, loaded",4799.00
```

- zeilenweise Struktur
- oft Spaltenüberschriften in erster Zeile
- diverse Trennzeichen möglich, meist Komma
- jeder Eintrag, der (Leer- oder) Sonderzeichen enthält, von Anführungszeichen eingeschlossen
- Format sehr uneinheitlich implementiert (vgl. RFC 4180)

RFC 4180: `https://tools.ietf.org/html/rfc4180` **(Zugriff am 03.08.2020)**

Beispiel für kommagetrennte Werte (CSV)

Ein wenig standardisiertes, uneinheitlich implementiertes Format

> *While there are various specifications and implementations for the CSV format [. . .], there is no formal specification in existence, which allows for a wide variety of interpretations of CSV files. This section documents the format that seems to be followed by most implementations . . .*
>
> – Y. Shafranovich, RFC 4180

- Beispiel dafür, wie ein Format *nicht* aussehen sollte
- Aufgrund der Uneinheitlichkeit
  keine Kompatibilität untereinander zu gewährleisten
- Sollte *nicht* verwendet werden – Alternative: DSV

https://tools.ietf.org/html/rfc4180 **(Zugriff am 03.08.2020)**

Beispiel für das Windows-INI-Format

Weit verbreitet zur Speicherung von Konfiguration

Listing 4: Beispiel für das Windows-INI-Format

```
; last modified 1 April 2001 by John Doe
[owner]
name=John Doe
organization=Acme Widgets Inc.

[database]
server=192.0.2.62
port=143
file="payroll.dat"
```

- blockweise Struktur
- zwei Hierarchieebenen: Blöcke und Schlüssel-Wert-Paare
- ursprünglich für Konfigurationen entwickelt
- Grundidee lässt sich auch anderweitig einsetzen
  - Beispiel folgt in der nächsten Vorlesung

Beispiel für XML

eXtensible Markup Language – hierarchisch und validierbar

Listing 5: Beispiel für XML

```
<?xml version="1.0"?>
<verzeichnis>
     <titel>Wikipedia Ortsverzeichnis</titel>
     <eintrag>
          <stichwort>Genf</stichwort>
          <eintragstext>Genf ist der Sitz von ...</eintragstext>
     </eintrag>
     <eintrag>
          <stichwort>Mainz</stichwort>
          <eintragstext>Mainz ist eine Stadt, die ...</eintragstext>
     </eintrag>
</verzeichnis>
```

- beliebig verschachtelte, hierarchische Struktur
- syntaktisch validierbar gegen ein definiertes Schema (ohne Notwendigkeit, die Semantik zu verstehen)
- mühsam für Menschen les- und insbesondere schreibbar

Beispiel für JSON

JavaScript Object Notation – hierarchisch und sparsam

Listing 6: Beispiel für JSON

```
{
  "id": 1,
  "name": "Foo",
  "price": 123,
  "tags": [
    "Bar",
    "Eek"
  ],
  "stock": {
    "warehouse": 300,
    "retail": 20
  }
}
```

- beliebig verschachtelte, hierarchische Struktur
- gut geeignet für die Persistenz von Datenstrukturen
- (deutlich) sparsamer als XML

Beispiel für YAML

„YAML Ain't Markup Language“ – hierarchisch und noch sparsamer

Listing 7: Beispiel für YAML

```
id: 1
name: Foo
price: 123
tags:
 - Bar
 - Eek
stock:
 warehouse: 300
 retail: 20
```

- beliebig verschachtelte, hierarchische Struktur
- einfacher menschenles- und schreibbar als JSON
  - keine Klammern oder Anführungszeichen notwendig
- Übermenge zu JSON (seit YAML 1.2)
- Unterstützung durch viele Programmiersprachen

Hinweise zum Umgang mit Reintextformaten

Ggf. Beschränkung auf druckbare Zeichen aus dem ASCII-Zeichensatz

Hex	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00	NUL ^@ 0	SOH ^A 1	STX ^B 2	ETX ^C 3	EOT ^D 4	ENQ ^E 5	ACK ^F 6	BEL ^G 7	BS ^H 8	TAB ^I 9	LF ^J 10	VT ^K 11	FF ^L 12	CR ^M 13	SO ^N 14	SI ^O 15
10	DLE ^P 16	DC1 ^Q 17	DC2 ^R 18	DC3 ^S 19	DC4 ^T 20	NAK ^U 21	SYN ^V 22	ETB ^W 23	CAN ^X 24	EM ^Y 25	SUB ^Z 26	ESC ^[27	FS ^\ 28	GS ^] 29	RS ^^ 30	US ^? 31
20	32	! 33	" 34	# 35	$ 36	% 37	& 38	' 39	(40	) 41	* 42	+ 43	, 44	- 45	. 46	/ 47
30	0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63
40	@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
50	P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	[91	\ 92	] 93	^ 94	_ 95
60	` 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
70	p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119	x 120	y 121	z 122	{ 123	| 124	} 125	~ 126	DEL 127

- Zeichen 20_{Hex} bis $7e_{\text{Hex}}$ (32 bis 126) sind „druckbar“.

ASCII: *American Standard Code for Information Interchange*

- Grund für Steuer- und Sonderzeichen
  - Auch ASCII enthält nicht druckbare Zeichen.
  - Notwendigkeit zur Darstellung weiterer Zeichen
- Strategie der geringsten Überraschung (*least surprise*)
  - Unterstützung der „Backslash“-Konvention

Zeichen	**Bedeutung**
`\n`	neue Zeile (new line)
`\t`	Tabulator
`\r`	Wagenrücklauf (carriage return)
`\xnn`	Zeichen mit Hexadezimalwert `nn`
`\unnnn`	Unicodezeichen mit Hexadezimalwert `nnnn`
`\\`	`\`

Sinnvolle Strategien der Komprimierung:

- Vermeiden redundanter Information
  - Beispiel: äquidistante Achse
    Start-, Endwert und Schrittweite reichen aus
- Kompression der gesamten Textdatei
  - Kompression unabhängig vom Dateiformat
  - Platzersparnis mitunter erheblich
  - Beispiel: Open Document Format (OpenOffice etc.)

Was man vermeiden sollte:

- Kompression von Teilen (Feldern) einer Textdatei
- Binärkodierung von Teilen (Feldern) einer Textdatei

Beispiele für Binärformate

Eine kleine Auswahl ohne Anspruch auf Vollständigkeit

- ANSI/IEEE 754-1985
  - Standard für die Repräsentation von Gleitkommazahlen
  - Möglichkeit, *reine* Zahlenkolonnen abzulegen
  - Achtung: mehr als ein Standard, nur für 64 Bit eindeutig
  - benötigt zusätzliche Informationen (Metadaten) zum konkreten Format und der Dimension der Daten
- HDF5
  - hierarchisches Binärformat
  - selbstbeschreibend mit Unterstützung für Metadaten
  - extrem performant
  - Erlaubt die Arbeit mit (Teilen von) Datensätzen, die viel zu groß für den Arbeitsspeicher wären.

☛ Es gibt auch hier noch (viele) weitere Formate . . .

Übersicht

Grundregeln zum Umgang mit Daten

Unabhängig von der Art und Herkunft der Daten

- Rohdaten sollten immer archiviert werden.
  - Was als Rohdaten gilt, ist nicht immer eindeutig . . .
  - Unversehrtheit sollte sichergestellt und überprüfbar sein
- Daten(sätze) sollten eineindeutig identifizierbar sein.
  - konsistentes Schema für Benennung und Ablage
  - Zugriff weitgehend unabhängig von der Art der Speicherung
- Metadaten sollten mit den Daten verbunden sein.
  - Daten ohne Metadaten sind wertlos.
  - in der gleichen Datei oder direkt daneben ablegen
- Funktionierende Backups sind essentiell.
  - Daten sind wertvoll und von grundlegender Bedeutung.
  - Backups regelmäßig auf korrekte Funktion überprüfen

Umgang mit Rohdaten

Ein paar Anmerkungen

- Was sind Rohdaten?
  - kontextabhängig – meist relativ klare Vorstellungen
  - Widerstandswerte eines Thermometers wohl eher nicht . . .
  - Rohdaten eines Bildsensors dagegen wohl eher schon . . .
  - Interne Verarbeitungsschritte sollten ggf. bekannt sein.
- Schutz vor ungewollter Veränderung
  - Insbesondere für Rohdaten von essentieller Bedeutung.
  - Bei der Langzeitarchivierung können sich Datenfehler einschleichen bzw. sind oft unvermeidbar.
- Sicherstellung der Unversehrtheit
  - kryptographische Hashes: Überprüfung auf Veränderungen
  - regelmäßig überprüfte Backups zur Datensicherung

kryptographische Hash-Funktion

Funktion, die eine Zeichenfolge beliebiger Länge auf eine solche fester Länge abbildet, kollisionsresistent sein sollte
und immer eine Einwegfunktion (unumkehrbar) ist.

- Prüfsummen (*hashes*) viel kleiner als eigentliche Daten
  - lassen sich viel einfacher und schneller vergleichen
  - lassen sich schnell eindeutig aus den Daten erzeugen
- Tipps aus der Praxis
  - Prüfsummen über Daten und Metadaten ggf. trennen
  - ggf. Prüfsumme über Prüfsummen von Einzeldateien bilden

☛ Standard-Hash-Algorithmen weit verbreitet und verfügbar

Identifizierung und Zugriff auf Datensätze

Einfache Lösungen unabhängig von der Art der Datenablage

- Zielstellung
  - einfacher Zugriff auf einen Datensatz
  - eindeutiges Etikett, das die Zuordnung ermöglicht
  - Nachvollziehbarkeit von Auswertungen
  - Rohdaten und verarbeitete Daten ansprechbar
- mögliche Lösung
  - eindeutiger Bezeichner für jeden Datensatz
  - Zugriff erfolgt über Bezeichner und Zuordnungstabelle, die den Bezeichner mit dem Speicherort verknüpft
  - Vorteil: unabhängig vom eingesetzten Ablagesystem (Verzeichnishierarchie, Datenbank, Netzwerkspeicher, . . .)

☛ Die Zuordnungstabelle sollte gut gesichert werden.
Alternative: Zuordnungstabelle automatisch generierbar

Arten der Datenspeicherung

Der Kontext bestimmt die Strategie

- im Speicher
  - flüchtig, nur für Zwischenschritte sinnvoll
- Dateien im Dateisystem
  - Standardsituation in den meisten Fällen in der Wissenschaft
  - Die Wahl eines geeigneten Formates ist entscheidend.
- Datenbank
  - große Vorteile beim gezielten Zugriff
- in zentralem Repository über Netzwerk
  - hauptsächlich bei sehr großen Datenmengen
    oder bei der Notwendigkeit eines verteilten Zugriffs

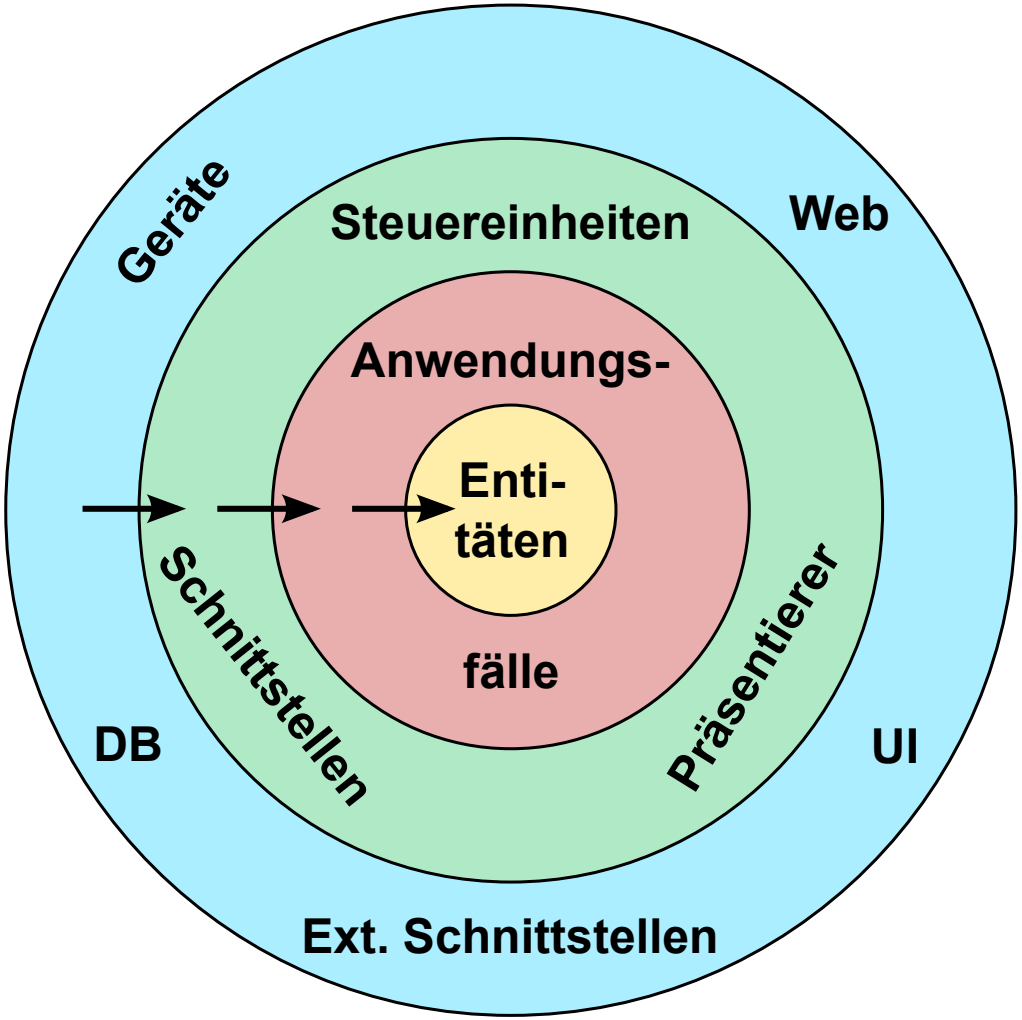
☛ Jede Art hat eigene Ansprüche an das gewählte Format.

☛ sollte ein austauschbares Implementationsdetail sein (DIP)

Übersicht

Datenformate sind peripher

Auswertungsroutinen hängen nicht von Datenformaten ab.



Unternehmens-Geschäftsregeln

Anwendungs-geschäftsregeln

Schnittstellen-Adapter

Frameworks & Treiber

Richtung der Abhängigkeiten

"saubere Architektur"

verändert nach Robert C. Martin: Clean Architecture. Prentice Hall, Boston 2018, S. 203

Dependency-Inversion-Prinzip

Abstraktionen sollten nicht von Details abhängen.

- Speicherung von (Roh-)Daten ist ein Detail.
  - Das Datenformat ist für die Auswertungsroutine egal.
  - Die Organisation der Datenablage ist ebenfalls egal.
- Datenformate sind austauschbar.
  - Alles, was benötigt wird, ist eine Importroutine.
- Die Organisation der Datenablage ist austauschbar.
  - Zugriff auf Datensätze über einen abstrakten Schlüssel (ID) und eine (beliebig implementierbare) Zuordnungstabelle

Auswirkung umgekehrter Abhängigkeiten

Flexibilität und die im jeweiligen Kontext bestmögliche Lösung

- Jede Abstraktionsebene hat eigene Ansprüche.
  - Die Repräsentation der Daten in Auswertungsroutinen kann komplett anders sein als bei der Archivierung.
  - Jede Abstraktionsebene sollte die ihr entsprechende und bestmögliche Repräsentation wählen.
- Abhängigkeiten zeigen immer nach innen.
  - Datenformate sollten die durch sie implizierten Strukturen *nicht* nach innen durchreichen.
  - Auswertungsroutinen wissen nichts von der Datenablage.
  - Der Austausch einer Verzeichnishierarchie durch eine Datenbank ist problemlos möglich.

☛ Flexibilität, Modularität, Wiederverwendbarkeit

Zentrale Aspekte

- Formate betreffen nicht nur Roh- und verarbeitete Daten, sondern auch Metadaten, Dokumentation, Abbildungen.
- Datenformate sollten über Jahrzehnte lesbar, plattformunabhängig, quelloffen und dokumentiert sein.
- Daten über Jahrzehnte lesbar zu archivieren, ist nicht nur eine Frage der Formate, sondern auch der Organisation.
- Rohdaten sollten immer (im Originalformat) archiviert und vor (ungewollter) Veränderung geschützt werden.
- Das konkrete Datenformat oder die Art der Datenlagerung ist für ein System zur Datenverarbeitung irrelevant.