Programmierkonzepte in der Physikalischen Chemie

6. Datenverarbeitung und Nutzerschnittstellen

UNI FREIBURG

Albert-Ludwigs-Universität Freiburg

Dr. Till Biskup

Institut für Physikalische Chemie
Albert-Ludwigs-Universität Freiburg
Wintersemester 2013/14

Übersicht

UNI FREIBURG

Datenverarbeitung

- Metadaten – Informationen zu Daten
- Hierarchische Datenstrukturen
- Datensatz – Einheit von Daten und Metadaten
- Parser – Metadaten einlesen und verarbeiten
- Ausblick

Nutzerschnittstellen

- Arten von Nutzerschnittstellen
- Ein Plädoyer für gutes Design
- Trennung von Datenverarbeitung und Nutzerschnittstelle
- Bausteine textbasierter Nutzerschnittstellen
- Textbasierte vs. grafische Schnittstellen

Daten: Währung der empirischen Wissenschaften

- Grundlage und Ausgangspunkt empirischer Wissenschaft
  - Daten sind nicht notwendigerweise „offensichtlich“
  - Messung zur „Aufnahme“ von Daten
- Daten allein sind wertlos
  - Hintergrundinformationen (Metadaten) gehören immer zwingend dazu.
  - Ein gut gepflegtes Laborbuch ist essentiell
  - Zur Datenverarbeitung ist ein Zugriff auf die Metadaten oft notwendig/hilfreich ($\Rightarrow$ maschinenlesbar ablegen)
- Daten überdauern, Interpretationen ändern sich
  - Die Verantwortung des Wissenschaftlers: saubere Datenaufnahme und -dokumentation

In einer idealen Welt... (*Strelitzia reginae*)

In einer idealen Welt...

- ...gibt es nur rauschfreie Daten.
- ...weiß man immer, wer was wann wie gemessen hat.
  - Experimentator, Probe, Datum, Experiment
  - Zweck des Experiments
- ...sind sämtliche Informationen zum Aufbau bekannt.
  - Bei Handaufbauten essentiell
  - Auch bei kommerziellen Geräten wichtig
  - Dokumentation austauschbarer Komponenten
- ...lassen sich alle Verarbeitungsschritte nachvollziehen.
  - Historie der Probenvorbereitung
  - Historie der Datenaufbereitung

Metadaten

Informationen über Merkmale anderer Daten,
aber nicht diese Daten selbst.

Wie lassen sich Metadaten (sinnvoll) ablegen?

- Sofort während der Messung
- Möglichst akkurat und vollständig
- In maschinenlesbarer Form
- Möglichst bei und nicht getrennt von den Daten

☛ Beispiele für Metadaten und ihre Ablage folgen

Beispiele für Metadaten

- Wann? – Datum der Messung
- Wer? – Experimentator
- Was? – Vermessene Probe
  - Probenname
  - Details zur Präparation
  - Herkunft (Kooperationspartner, ...)
- Wie? – Aufbau
  - Gerätebezeichnung und Hersteller
  - Software inkl. Versionsnummer
  - Handaufbauten: detaillierte Liste der Komponenten
- Weitere Bedingungen
  - Temperatur

Infodatei – eine Möglichkeit der Ablage von Metadaten

- Menschen- und maschinenlesbar
  - Wird vom Experimentator erzeugt
  - Wird vom verarbeitenden Computer eingelesen
- Intuitiv aufgebaut
  - Nur einfache Dinge werden in der Praxis genutzt.
  - Der Anwender muss den Mehrwert erkennen können.
- Modular
  - Aufbauten und Anforderungen ändern sich.
  - Zukunftsfähigkeit sorgt für Akzeptanz

☛ Fokus: Einfache Benutzbarkeit
(Erfahrung: Nur was einfach und intuitiv ist, wird benutzt.)

Listing 1: Beispiel für eine Infodatei

```
1  general Info file - v. 0.0.1 (2014-01-20)
2
3  GENERAL
4  Filename:                somefile
5  Date:                    2014-01-21
6  Time start:              11:05:00
7  Time end:                15:50:00
8  Operator:                Alois Kabelschacht
9  Label:                   Short and comprehensive label
10 Purpose:                 Kill time
11
12 SAMPLE
13 Name:                    Sample
14 Description:             Cool sample that doesn't show any signal
15 Preparation:             No clue, just found it lying 'round
16
17 COMMENT
18 I had a dream: Measuring this sample would solve all my problems,
19 answer all my questions, help me finishing my thesis.
20
21 Alas, it seems not to work out...
```

Infodatei – grundsätzliches Konzept

- Reine Textdatei (ASCII 7-bit)
- Ähnlich herkömmlicher Konfigurationsdateien
  - Blöcke
  - Schlüssel-Wert-Paare
- „Formular“ zur Erfassung aller Informationen

Infodatei – Aufbau und Bestandteile

- Erste Zeile: Zeichenkette zur Identifizierung
- Blocknamen (in Großbuchstaben)
- Schlüssel-Wert-Paare innerhalb eines Blocks
- Kommentarblock (letzter Block, Freitext)

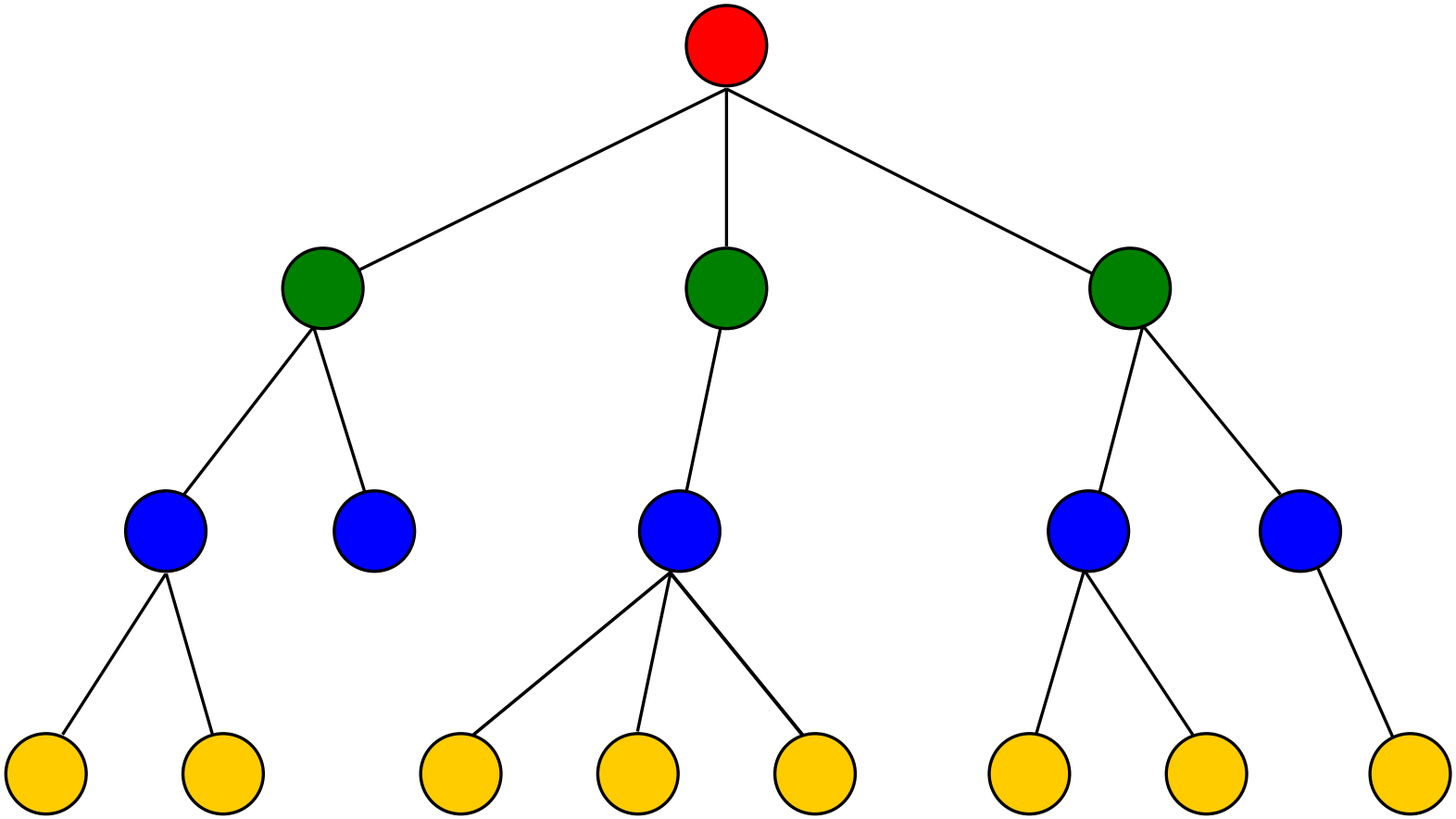
Infodateien – aus der Praxis für die Praxis

- Vorgänger seit vielen Jahren im produktiven Einsatz
- Generalisierung und Maschinenlesbarkeit seit 2011
- Modular, beliebig erweiterbar
- Erweitert (und ersetzt) das Laborjournal
- Hilfestellung zur Erfassung aller relevanten Parameter

☛ Fokus: Einfache Benutzbarkeit

Weitere Informationen

`http://www.till-biskup.de/de/software/info/`



Hierarchische Datenablage

- Zusammenfassen, was zusammengehört
  - Blöcke zusammengehöriger Parameter
  - Sorgt für Übersichtlichkeit (und kurze Feldnamen)
- Gerichteter Graph (gewurzelter Baum)
  - Baum mit einer Wurzel
  - Knoten: Verzweigungen des Baums
  - Blätter: Enden von Knoten (Verzweigungen)
  - In der Informatik häufig als Datenstruktur verwendet
- Kriterien
  - Ein Baum hat genau eine Wurzel.
  - Blätter gehören zu genau einem Knoten.
  - Knoten können beliebig viele Blätter/Unterknoten haben.

Assoziatives Datenfeld (*associative array*)

Datenstruktur, die Zeichenketten (statt Zahlen) verwendet, um die enthaltenen Elemente zu adressieren.

- Keine festgelegte Reihenfolge der Felder
- Ideal zur Ablage von Schlüssel-Wert-Paaren
- Komplexe hierarchische Datenstrukturen durch Verschachtelung möglich

☛ Tipp: Schlüsselname sollte nachvollziehbare Verbindung zum Datenwert liefern.

Geordnete Listen

#	Wert
1	0.0000
2	0.0025
3	0.0050
⋮	
n-1	0.2475
n	0.2500

#	Wert
1	'Im'
2	'Anfang'
3	'war'
⋮	
n-1	'die'
n	'Tat'

Assoziative Datenfelder

Schlüssel	Wert
Name	'K. Racht'
Alter	42
Adresse	
Hobbies	{'...','...'}

Schlüssel	Wert
Straße	'Talstraße'
Nummer	21

Hierarchische Datenablage in Matlab: structs

Listing 2: Allgemeine Definition eines structs in Matlab

```
myStruct = struct('field1', values1, 'field2', values2, ...);
```

- Assoziatives Datenfeld
  - Name und Schlüssel werden durch Punkt getrennt
- Beliebig verschachtelbar
  - Werte dürfen beliebigen Datentyp annehmen
  - Ermöglicht Abbildung komplexer Hierarchien
- Funktionen zur Arbeit mit structs
  - Felder: `isfield`, `fieldnames`, `rmfield`, ...
  - Strukturen: `isstruct`, `structfun`

Dynamische Feldnamen statt `getfield` und `setfield`

Problem Feldnamen oft in Variablen codiert

Lösung dynamische Feldnamen verwenden

Listing 3: Direktes Ansprechen eines Feldes

```
1 myStruct = struct('field1', values1, 'field2', values2, ...);
2 valueOfDesiredField = myStruct.field1;
```

Listing 4: Verwendung dynamischer Feldnamen

```
1 myStruct = struct('field1', values1, 'field2', values2, ...);
2 nameOfDesiredField = 'field1';
3 valueOfDesiredField = myStruct.(nameOfDesiredField);
```

`structfun` statt Schleifen

Problem Operation auf jedem Feld einer Struktur

Lösung `structfun` (deutlich schneller als Schleife)

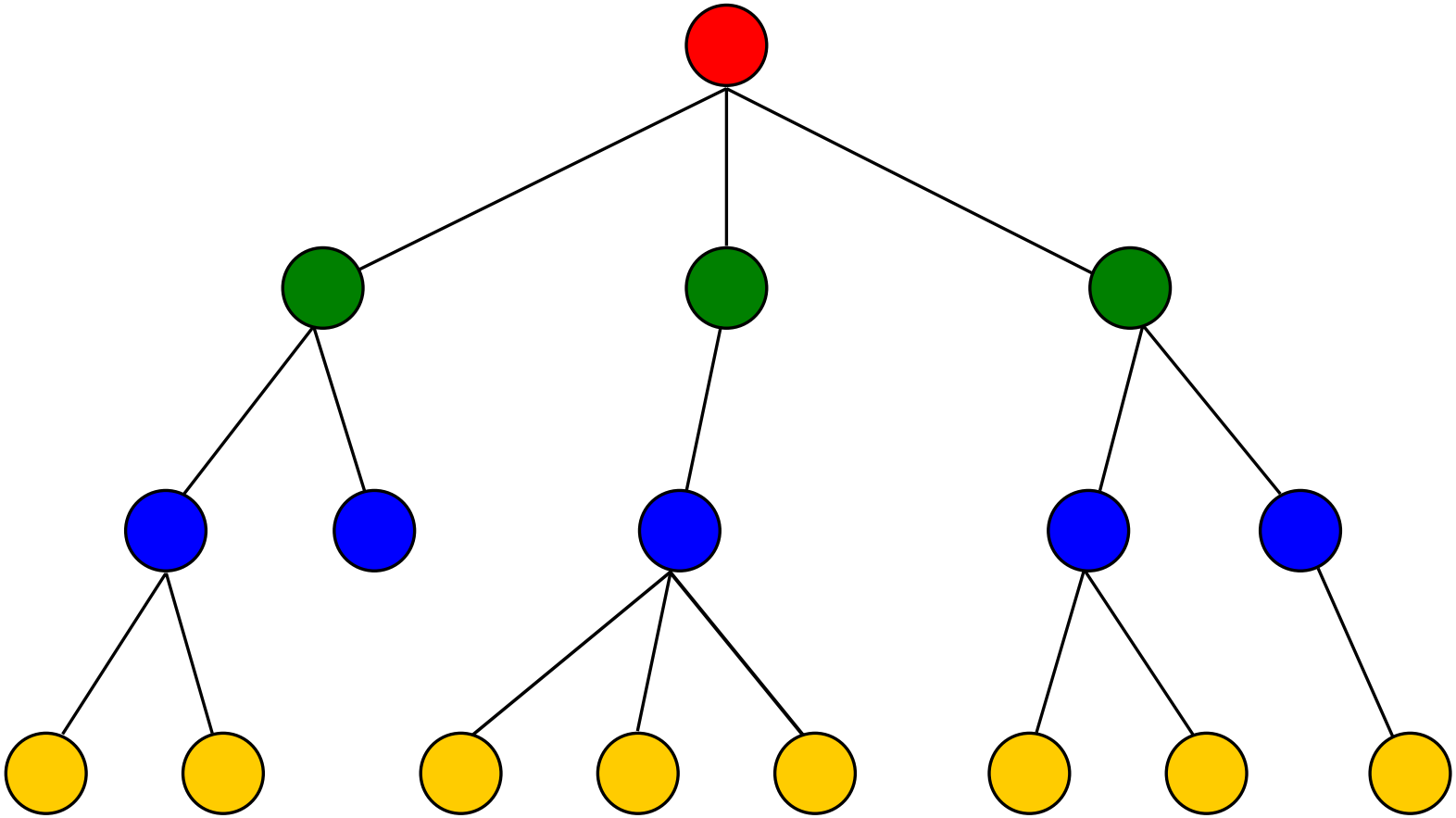
Listing 5: Operation auf jedem Feld über eine Schleife

```
1 myStruct = struct('field1', values1, 'field2', values2, ...);
2 myStructFields = fieldnames(myStruct);
3 for field = 1:length(myStructFields)
4     disp(myStruct.(myStructFields{field}));
5 end
```

Listing 6: Operation auf jedem Feld über structfun

```
1 myStruct = struct('field1', values1, 'field2', values2, ...);
2 structfun(@disp,myStruct);
```

Und was, wenn wir mehr als zwei Ebenen haben?



Rekursive Funktionen: Verarbeitung hierarchischer Daten

- Wichtig für Bäume mit mehr als einem Knoten
- Steigen in jeder Verzweigung bis zum Blatt ab
- Unterschiedliche Operationen auf Knoten und Blättern
  - Knoten: Funktion selbst erneut (rekursiv) aufrufen
  - Blatt: eigentliche Operation durchführen

Wichtige Aspekte

- Elternknoten niemals verlieren
- Entscheidung auf jeder Ebene: Blatt oder Knoten?
- Rekursiver Aufruf für einen Knoten

Listing 7: Beispiel für eine rekursive Funktion in Matlab

```
1  function value = getCascadedField(structure,fieldName)
2      % Get number of "." (field separators) in fieldName
3      nDots = strfind(fieldName,'.');
4      % Check if we're already at a leave
5      if isempty(nDots)
6           value = structure.(fieldName);
7           return;
8      end
9      % Reset parent to current knot
10     structure = structure.(fieldName(1:nDots(1)-1));
11     % Recursive function call
12     value = getCascadedField(structure,fieldName(nDots(1)+1:end));
13 end
```

Anmerkungen

- Dieses Beispiel arbeitet nur auf linearen Strukturen.
- Kein robuster Code: keinerlei Überprüfungen

Standardisierte Speicherformate für hierarchische Daten

Aufgabe Speicherung verschachtelter hierarchischer Daten

Lösung XML als universelles, standardisiertes Format

XML (Extensible Markup Language)

Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdateien

- Vom World Wide Web Consortium (W3C) standardisiert
- reines Textformat, menschenlesbar
- Einsatz: plattform- und implementationsunabhängiger Austausch von Daten zwischen Computersystemen

Listing 8: Beispiel für eine XML-Datei

```
1  <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2  <verzeichnis>
3       <titel>Wikipedia Städteverzeichnis</titel>
4       <eintrag>
5            <stichwort>Genf</stichwort>
6            <eintragstext>Genf ist der Sitz von ...</eintragstext>
7       </eintrag>
8       <eintrag>
9            <stichwort>Köln</stichwort>
10           <eintragstext>Köln ist eine Stadt, die ...</eintragstext>
11      </eintrag>
12 </verzeichnis>
```

- XML besteht aus einzelnen Elementen.
- Jedes Element hat Beginn- und Endauszeichner (Tags).
- Elemente dürfen (fast) beliebig verschachtelt werden.

Warum XML zur Speicherung hierarchischer Daten?

- International standardisiert (W3C)
- Extrem flexibel
  - Elementnamen quasi frei wählbar
  - Nahezu beliebig verschachtelbar
  - Zusätzliche Attribute für Elemente (im Start-Tag)
- Reines Textformat
  - Menschenlesbar
  - Plattformunabhängig
  - Unabhängig von Anzeigesoftware (Texteditor genügt)
- Logischer Aufbau
  - Einfach maschinenlesbar
  - Eineindeutig, validierbar

XML und Matlab

- Matlab unterstützt XML
  - `xmlread`, `xmlwrite`
  - Nutzt intern Java-Bibliotheken
  - Objektorientierte Programmierung
- Matlab-Strukturen lassen sich in XML ablegen
  - Rekursive Funktion für hierarchische Strukturen
- XML-Strukturen lassen sich in Matlab einlesen
  - Rekursive Funktion für hierarchische Strukturen

☞ XML eignet sich zur plattformunabhängigen Speicherung von Metadaten aus Matlab heraus.

XML und Infodateien

- XML
  - Allgemeinste Form einer logischen Auszeichnungssprache
  - Nahezu beliebig schachtelbar
  - Optimiert für Maschinenlesbarkeit
  - Menschenlesbar, aber nicht komfortabel
  - Manuelles Schreiben aufwendig
- Infodatei
  - Einfache Erfassung von Metadaten zu einem Datensatz
  - Eingeschränkte Hierarchie (drei Ebenen)
  - Optimiert für menschliche Interaktion
  - Maschinenlesbar und hinreichend eindeutig
  - Manuelles Schreiben einfach

Datensatz

Einheit von (gemessenen) Daten und zugehörigen Metadaten

- Daten und Metadaten liegen immer zusammen vor
  - Daten ohne Metadaten sind wertlos.
  - Metadaten personenunabhängig speichern
- Daten und Metadaten sind maschinenlesbar
  - Auswertesoftware ist sich der Metadaten „bewusst“
  - Automatische Auswertung abhängig von den Metadaten
- Zentrales Konzept für die Datenverarbeitung
  - Hilft, das Versprechen einzulösen, dass Toolboxen die Nachteile gegenüber Skripten ausgleichen können.

Ein Datensatz besteht aus mindestens drei Teilen

1. Daten
   - Eigentliche (gemessene) Daten
   - (Meist) numerisch
2. Metadaten
   - Zusätzliche Informationen zu den Messdaten
   - Z.B. aus einer Infodatei
3. Historie
   - Dokumentation aller Verarbeitungsschritte der Messdaten
   - Vollständige Nachvollziehbarkeit und Wiederholbarkeit

☛ In der Praxis ggf. noch weitere Teile

Historie – Dokumentation aller Verarbeitungsschritte

- Zielstellung
  - Nachvollziehbarkeit
  - Reproduzierbarkeit
- Felder für einen Eintrag
  - Name des Durchführenden
  - Datum
  - Name und Version der verarbeitenden Routine
  - Version des zugrundeliegenden Programms (z.B. Matlab)
  - Name und Version des Betriebssystems
  - ggf. sämtliche Eingabeparameter für die Routine

☞ Gewährleistet zusammen mit einer Versionsverwaltung (für den Code) die vollständige Reproduzierbarkeit.

Datensatz – Weitere Felder

- Originaldaten
  - Rohdaten, wie sie erstmalig eingelesen wurden
  - Wichtig für Reproduzierbarkeit und Rücknahme von Prozessierungsschritten („Undo“)
- Format
  - Informationen zum Format (Struktur) des Datensatzes
  - Versionsnummer, wird bei jeder Änderung inkrementiert
  - Wichtig für die (Abwärts-)Kompatibilität von Einleseroutinen
- Informationen zur Originaldatei
  - Name und Format der Originaldatei
  - Pfad zur Datei ist wenig aussagekräftig
  - Originale Datei mit Rohdaten *nie* löschen

Datensatz – Implementation in Matlab

- Datensatz als `struct`
  - Hierarchisch, verschachtelt (Beispiel folgt)
  - Einheit und Wert jeweils in getrennte Felder
- Funktion für die Definition des Datenmodells
  - Ein (*einziger*) Ort für die Definition
  - Erzeugung leerer Datensätze mit allen Feldern
- Funktion(en) für das Einlesen der Rohdaten
  - Abhängig vom Dateiformat der Rohdaten
  - Füllt ggf. Teile der Metadaten-Felder mit Inhalt
- Funktion für das Einlesen der Infodateien
  - Füllt Metadaten-Felder mit Inhalt

Datensatz – Beispiel für die Felder auf oberster Ebene

data	`numeric`	(verarbeitete) Daten
origdata	`numeric`	Rohdaten aus der Originaldatei
axes	`struct`	Informationen zu den Achsen
parameters	`struct`	Informationen zur Messung
sample	`struct`	Informationen zur Probe
comment	`cell`	Kommentar zur Messung
history	`cell`	Historie der Datenverarbeitung
version	`string`	Version der Datenstruktur
file	`struct`	Informationen zur Originaldatei
label	`string`	kurzer Bezeichner

Parsen (Syntaxanalyse)

Zerlegung und Umwandlung einer beliebigen Eingabe in ein für die Weiterverarbeitung brauchbares Format

- U.a. erste Aufgabe eines Compilers
  - Zerlegen des Quelltextes in logische Einheiten
- Auch für natürliche Sprachen möglich
  - Wegen der vielen Uneindeutigkeiten sehr schwer
- Parser für strukturierte Textdateien (Infodatei, XML)
  - Recht einfach selbst zu implementieren
  - Wichtiger erster Schritt zur Verarbeitung von Metadaten

Mögliches Vorgehen eines Matlab-Parsers für Infodateien

1. Datei einlesen
   - `fopen`, `fgetl`, `fclose`
   - Ergebnis: `cell array` mit zeilenweisem Dateiinhalt
2. Blöcke identifizieren
   - Zwei Möglichkeiten:
     1. Über Liste der Blocknamen
     2. Über Leerzeilen als Blocktrenner
   - Ergebnis: Vektor mit Zeilenindices der Blockgrenzen
3. Blöcke parsen
   - Innerhalb jedes Blocks befinden sich Schlüssel-Wert-Paare
   - Ergebnis: `struct` mit den Schlüssel-Wert-Paaren

Nächste Schritte nach dem Parsen

- Abbilden der Felder aus der Infodatei auf die Datenstruktur eines Datensatzes
  - Meist über eine Zuordnungstabelle
  - Möglichst zentral an einer Stelle abgelegt
  - Anpassung bei Änderungen der Spezifikationen von Infodatei oder Datenstruktur

Voraussetzungen zum Schreiben eines Parsers

- Kenntnis des Ausgangsformates und seiner Spezifikation
- Klar erkennbare Muster im Ausgangsformat
- Vertrautheit mit regulären Ausdrücken
- Gute Ideen und strukturiertes Denken

Berichte generieren

- Motivation
  - Was sind die Charakteristika eines Datensatzes?
  - Wie wurden die Daten konkret aufgenommen?
  - Was wurde mit dem Datensatz alles gemacht?
- Inhalte
  - Zusammenfassung aller Informationen zu einem Datensatz
  - *Auf einen Blick* – charakteristische Abbildungen
- Vorteile
  - Berichte unabhängig von Matlab
  - Katalog vorhandener Daten einfach erstellbar
  - Vergleichbar: identische Abbildungen für jeden Datensatz

Berichte generieren (Fortsetzung)

- Zielstellung
  - Automatische Berichterstellung aus Matlab heraus
  - Flexible Anpassung der Berichte
  - (Möglichst) publikationsreife Abbildungen
- Voraussetzungen
  - Alle notwendigen Informationen im Datensatz
  - Alle notwendigen Routinen in einer Toolbox
  - Vorlagen (*Templates*) für Berichte
- Templates
  - Trennung von Logik (Erstellung) und Darstellung
  - Enthalten Platzhalter für Inhalte
  - Werden automatisch mit Inhalten gefüllt

Berichte generieren: Templates

- Anforderungen an ein Template-Format
  - Programmatisch mit Inhalt zu füllen
  - Textformat mit vollständig offengelegter Spezifikation
  - Einfach vom Nutzer anpassbar
- Welche existierenden Formate kommen in Frage?
  - LaTeX
  - Open Document Format (ODT)
  - XML, HTML, DocBook, ...
- Template-Systeme
  - Viele verschiedene Implementationen möglich
  - Wenige grundlegende Design-Entscheidungen notwendig
  - Tipp: Bestehende Implementationen als Vorlage verwenden

Laborinformationssystem

- Motivation
  - Übersicht über Proben, Messungen, Daten, Auswertungen
  - Einfache Durchsuchbarkeit nach unterschiedlichen Kriterien
- Voraussetzungen
  - Daten und Metadaten in maschinenlesbarer Form
  - Standardisierte Datenerfassung (Formulare, Infodateien)
- Kernaspekte der Umsetzung (Nutzersicht)
  - Hohe Verfügbarkeit (räumlich und zeitlich)
  - Einfache Bedienbarkeit
  - Plattformunabhängig
  - Einfacher Export der verfügbaren Informationen
  - Offensichtlicher Mehrgewinn bei konsequenter Nutzung

Laborinformationssystem

- Kernaspekte der Umsetzung (Entwicklersicht)
  - Datenbankbasiert (Durchsuchbarkeit)
  - Serverbasiert (Verfügbarkeit)
  - Webbasiert (Plattformunabhängigkeit)
- Kommerzielle Lösungen
  - Qualitätssicherung (u.a.) in Industrie und Medizin
  - Meist sehr teuer und mit enormem Anpassungsaufwand
- Weitere Aspekte
  - Datenschutz von Anfang an beachten
  - Nach Möglichkeit auf Standardkomponenten zurückgreifen
  - Datensicherheit (I): robuste Backup-Lösungen
  - Datensicherheit (II): Schutz vor unerlaubtem Zugriff

Nutzerschnittstellen

Vermittler zur analogen Welt des Nutzers

Nutzerschnittstelle

Abstrakte Schicht zwischen dem Nutzer und den eigentlichen Routinen, die dem Nutzer die Bedienung erleichtert.

Zwei Arten von Nutzerschnittstellen

- Textbasierte Schnittstelle
  command line interface, CLI
- grafische Schnittstelle
  graphical users interface, GUI

☛ Jede dieser Schnittstellen hat ihre Vor- und Nachteile.

Textbasierte Nutzerschnittstelle (CLI)

- Menüs und Nutzereingaben in einer Textkonsole
- Vollständig deterministisch (bis auf Nutzereingaben)
- Linear: immer nur eine Entscheidungsmöglichkeit
- Strukturiert, aber mit wenig Freiheiten

Grafische Nutzerschnittstelle (GUI)

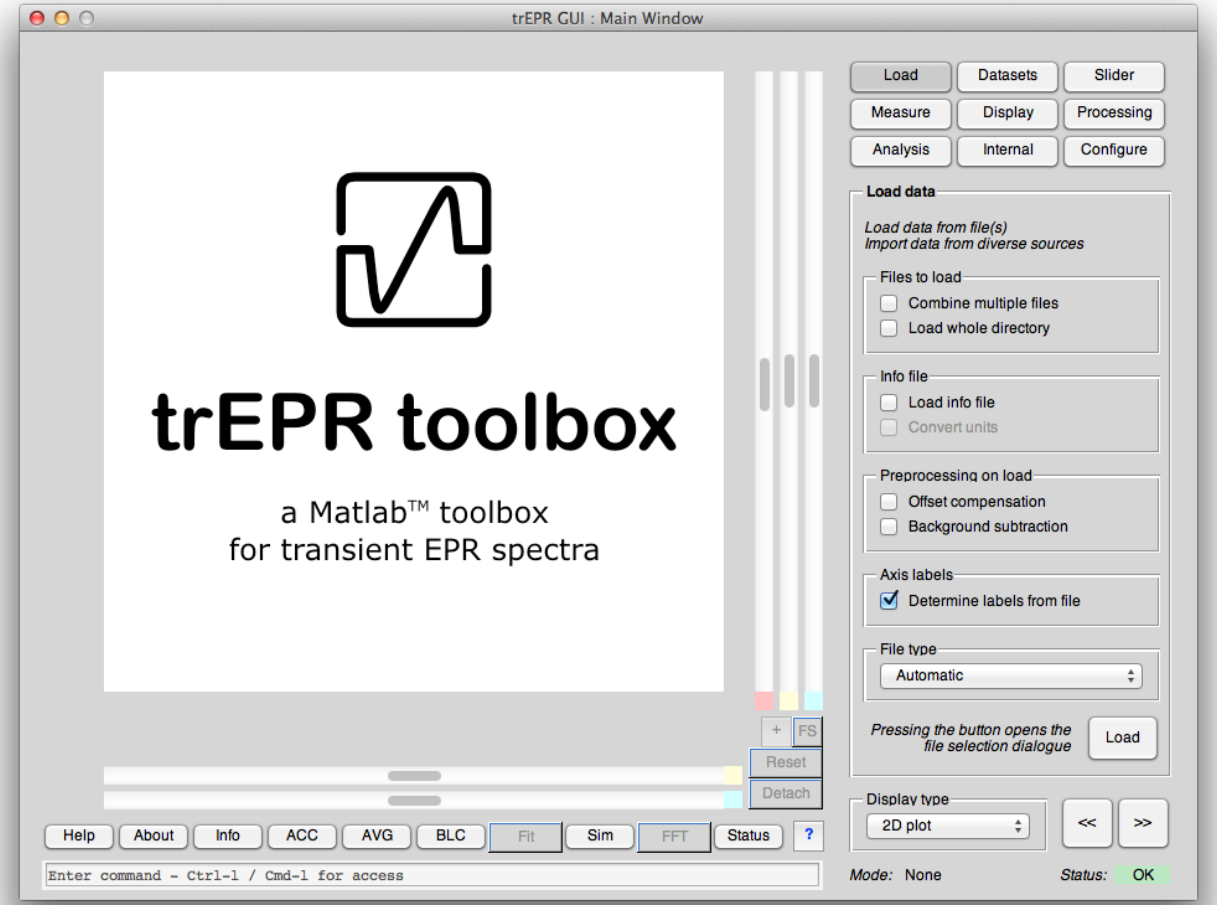
- Grafische Anordnung von Bedienelementen
- Reihenfolge der Ereignisse unvorhersehbar
- Nichtlinear: beliebige Entscheidungsmöglichkeiten
- Große Freiheit: Alles (implementierte) jederzeit möglich.

Beispiel: Textbasierte Nutzerschnittstelle (CLI)

Listing 9: Programm zur Simulation von EPR-Spektren

```
1  Do you wish to simulate or to fit?
2   [f] Fit
3   [s] Simulate
4   [q] Quit
5  Your choice (default: [f]): s
6
7  Do you wish to load experimental data?
8   [y] Yes
9   [n] No
10 Your choice (default: [n]): n
11
12 The simulation parameters currently chosen:
13 g               2.0200     2.0200     2.0200
14 D            3900.0000
15 E             130.0000
16 mwFreq          9.7000
17 nPoints       361.0000
18 Range         260.0000   440.0000
19 Temperature     0.0000     0.4500     0.5500
20 Method      matrix
```

Beispiel: Grafische Nutzerschnittstelle (GUI)



Lounge Chair & Ottoman, Charles & Ray Eames, 1956

Gutes Design ist wichtig – und zahlt sich aus

- Nutzerschnittstellen werden (arbeits-)täglich genutzt.
  - Selbst kleine Verbesserungen zahlen sich aus.
  - Auf den Nutzer und seine Bedürfnisse hören.
- Intuitive Nutzerführung vereinfacht komplexe Abläufe.
  - Freiheit, alles zu tun, was grundsätzlich möglich ist
  - Das Naheliegende nahe liegend anordnen.
  - Reduktion einzelner Ebenen: Erfassbarkeit auf einen Blick
- Gutes Design steht am Ende eines langen Prozesses.
  - Reduktion auf das Wesentliche ist eine Kunst.
  - Dauerhaftigkeit erfordert rigoroses Durchdenken.
  - Prototypen konsequent in der Praxis testen

Was ist „Gutes Design“?

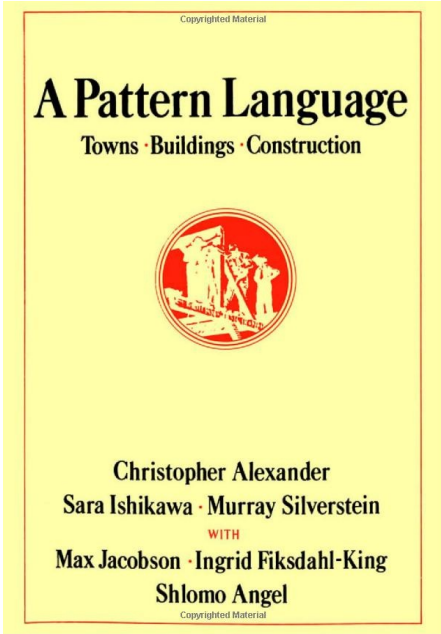
- Die Antwort ist letztlich subjektiv.
- Ein erkennbares Konzept ist vermutlich *ein* Aspekt.
- Was ist das Ziel des Designs?
  - Qualität als Maß: Wie gut wurde das Ziel erreicht?
- Welche Ebene betrachten wir?
  - Anwender und Entwickler haben unterschiedliche Ansprüche.

☛ Pragmatischer Ansatz: Was funktioniert, ist gut (genug).

Piero Gatti, Cesare Paolini, Franco Teodoro, Sacco, 1968

Bewährte Muster aus der Praxis

- Wiederkehrende Muster
  - Funktional oder „kulturell“ bedingt
  - Bewährt, aber nicht immer optimal
  - Sollten Kreativität nicht behindern
- Wiedererkennungseffekte
  - Kann die Bedienung beschleunigen
  - Ersetzt nicht die Einarbeitung
  - Gefahr: vermeintliche Vertrautheit



- ☛ Konzepte und Muster nicht unhinterfragt einsetzen
- ☛ Aber: Es lohnt nicht, das Rad immer neu zu erfinden...

Gründe für die Trennung

- Saubere und fehlerfreie Datenverarbeitung ist in der Wissenschaft von allergrößter Bedeutung.
  - Nachvollziehbarkeit der Prozessierung der Daten
  - Automatisierte Wiederholbarkeit einer Prozessierung
- Eine Routine für einen Schritt der Datenverarbeitung
  - Kann direkt oder von der jeweiligen Nutzerschnittstelle aufgerufen werden
  - Fehler müssen nur an *einer* Stelle behoben werden.
- Modularisierung vereinfacht die Programmierung von Nutzerschnittstellen
  - Konzentration auf die Schnittstelle
  - Die verarbeitenden Routinen sind bereits vorhanden.

Gründe für die Trennung (Fortsetzung)

- Fehler in der Nutzerschnittstelle verhindern nicht die weitere Auswertung.
  - Komplexe Matlab-GUIs sind schwer plattformunabhängig und unabhängig von der Matlab-Version lauffähig zu halten.
  - Immer auch die Möglichkeit geben, die Auswerteroutinen händisch aufzurufen.
- Freiheit und Unvorhersehbarkeit
  - Eine feste Schnittstelle (CLI und besonders GUI) schränkt den Benutzer zu stark ein.
  - Wissenschaft lebt vom frischen Blick auf alte Probleme: „Lego-Prinzip“ als Erfolgsgarantie.

Gründe für die Trennung (Fortsetzung)

- Testbarkeit
  - CLIs und GUIs (fast) nicht automatisiert testbar
  - Tests der Datenverarbeitungsschritte gerade in der Naturwissenschaft von essentieller Bedeutung
  - Tests oft über Parameter für Spezialbedingungen
- Arbeitsteilung
  - Schnittstellendesign und Datenverarbeitungsroutinen erfordern vollkommen unterschiedliche Qualifikationen.
  - Model-View-Control-Ansatz (MVC) in der Praxis bewährt
- Automatisierbarkeit
  - Datenverarbeitung oft nach festem Schema
  - „Skriptbarkeit“ der Verarbeitung zur Arbeitserleichterung

Listing 10: Textbasierte Schnittstelle zur Simulation von EPR-Spektren

```
>> trEPRTSim_cli
Do you wish to simulate or to fit?
 [f] Fit
 [s] Simulate
 [q] Quit
Your choice (default: [f]): s

Do you wish to load experimental data?
 [y] Yes
 [n] No
Your choice (default: [n]): y

Please enter the filename of the experimental data
you wish to fit ('q' to quit): spectrum.dat

Perform pretrigger offset compensation (POC)

Perform simple background correction (BGC)

The simulation parameters currently chosen:
g               2.0200     2.0200     2.0200
D            3900.0000
E             130.0000
```

Bausteine textbasierter Nutzerschnittstellen

- Auswahlmenüs
- Eingabeaufforderung mit freier Texteingabe (*prompt*)
- Hinweise an den Nutzer (ggf. mit Bestätigung)

☛ Beispiele für jeden Baustein folgen

Hinweise zur Umsetzung

- Einheitlich, prägnant, nutzerfreundlich
- Wiedererkennung erhöht die Nutzerfreundlichkeit

☛ Eigene Funktion für jeden Baustein

Auswahlmenüs

- Unterscheidbare Bestandteile
  - Titel
  - Liste der Optionen
  - Eingabeaufforderung (*prompt*) zur Eingabe der Option
- Tipps aus der Praxis
  - Nicht zu viele Optionen (wird schnell unübersichtlich)
  - Vorauswahl, wann immer möglich und sinnvoll
  - Überprüfung der Nutzereingabe auf Sinnhaftigkeit
  - Rückkehr zum Menü im Falle falscher Eingaben
  - Einheitliches Layout (über eigene Funktion realisieren)

Auswahlmenüs

Listing 11: Beispiel für ein Auswahlmenü

```
Please chose one or more fit parameters
  [1]  gx value
  [2]  gy value
  [3]  gz value
  [4]  Zero field splitting parameter D
  [5]  Zero field splitting parameter E
  [6]  Population of level 1
  [7]  Population of level 2
  [8]  Population of level 3
  [9]  Scaling factor between experiment and fit
  [10] Overall inhomogeneous linewidth Gaussian
  [11] Overall homogeneous linewidth Lorentzian
  [12] local inhomogenes linewith, D strain
  [13] local inhomogenes linewith, E strain
  [14] Frequency correction via field offset
  [15] g strain in x direction
  [16] g strain in y direction
  [17] g strain in z direction
Your choice (default: [1,2,3,4,5,6,7,8,9,10,12]):
```

Eingabeaufforderung (*prompt*)

- Unterscheidbare Bestandteile
  - Hinweistext für den Nutzer
  - Eingabeaufforderung (*prompt*)
- Tipps aus der Praxis
  - Keine zu langen Texte (wird schnell unübersichtlich)
  - Überprüfung der Nutzereingabe auf Sinnhaftigkeit
  - Rückkehr zum Prompt im Falle falscher Eingaben
  - Einheitliches Layout (über eigene Funktion realisieren)
  - Bei längeren Eingaben Prompt in neuer Zeile

Eingabeaufforderung (*prompt*)

Listing 12: Beispiel für eine Eingabeaufforderung

```
1 Please enter the filename of the experimental data
2 you wish to fit ('q' to quit):
```

Listing 13: Beispiel für eine robuste Eingabeaufforderung

```
1 Please enter the filename of the experimental data
2 you wish to fit ('q' to quit):
3
4 File "" not found. Please try again
5
6 Please enter the filename of the experimental data
7 you wish to fit ('q' to quit):
```

Hinweise an den Nutzer

- Grundsätzlich zwei Arten
  1. interaktive Hinweise (Bestätigung des Nutzers notwendig)
  2. nicht-interaktive Meldungen
- Tipps aus der Praxis
  - Keine zu langen Texte (wird schnell unübersichtlich)
  - Einheitliches Layout (über eigene Funktion realisieren)
  - Art des Hinweises (durch Kürzel am Anfang) angeben
  - Mögliche Klassifizierung:

    (II) Information
    (WW) Warnung
    (EE) Fehler (*error*)
    (DD) Hinweis für Entwickler (*debugging*)

Hinweise an den Nutzer

Listing 14: Interaktiver Hinweis

```
1 Going to erase your hard drive compleletely.
2 Press any key to continue...
```

Listing 15: Nicht-interaktive Meldung

```
1 Perform simple background correction (BGC)
```

Listing 16: Nicht-interaktive Meldung mit Klassifizierung

```
1 (DD) File format: fsc2
```

Textbasierte Nutzerschnittstellen (CLI)

- Relativ einfach zu implementieren.
- Lineare (festgelegte) Nutzerführung
- Notwendigkeit, alles textlich zu beschreiben

Graphische Nutzerschnittstellen (GUI)

- Oft schnellerer Zugang für den Gelegenheitsnutzer
- Wesentlich aufwendiger in der Programmierung
- Matlab: Festlegung auf ein kommerzielles Programm
- Plattformunabhängigkeit schwer zu gewährleisten

☛ Klare Abwägung der jeweiligen Kosten und Nutzen

Beispiel einer ausführlicheren textbasierten Schnittstelle

- Aufgabe: Simulation spinpolarisierter EPR-Spektren
- Einige der notwendigen Nutzerinteraktionen
  - Simulation oder Anpassung an gemessene Daten (Fit)?
  - Experimentellen Datensatz laden?
  - Auswahl der Simulationsparameter
  - Werte für die Parameter (und Grenzen für die Anpassung)
- Vorteile
  - Schnelle Implementation (Kern in zwei Tagen)
  - Nutzer wird „geführt“ (keine weitere Hilfe notwendig)
- Grenzen
  - Steigende Komplexität führt zu langen Wegen
  - Trotz Voreinstellungen „Orgie des Return-Taste Hauens“

Ausblick

Das war's. Wie geht's weiter?

UNI FREIBURG

So long, and thanks for all the fish.

Vorschau: Graphische Nutzerschnittstellen (GUIs)

- Besonderheiten von GUIs
- Bausteine von GUIs
- Allgemeines zur GUI-Entwicklung
- GUI-Entwicklung in Matlab

Douglas Adams