Programmierkonzepte in der Physikalischen Chemie

3. Modularität
Dokumentation im Code

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Dr. Till Biskup

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

Übersicht

UNI FREIBURG

Modularität

- Eine Aufgabe, eine Routine
- Klar definierte Schnittstellen
- Trennung von Datenverarbeitung und Nutzerschnittstelle
- Hintergrundwissen: Kontext der Ausführung

Dokumentation

- Motivation: Warum dokumentieren?
- Arten von Dokumentation

Dokumentation im Code

- Kein Code ohne Dokumentation
- Grundlegende Struktur eines Kommentarkopfes
- Dokumentationshilfen
- Probleme von Dokumentation

Modularität

Der Schlüssel zur Wiederverwertbarkeit

Alan Chi, Flickr

Modularität

Der Schlüssel zur Wiederverwertbarkeit

UNI FREIBURG

Aspekte

- Das Unix-Prinzip: eine Aufgabe, eine Routine
- Klar definierte Schnittstellen
- Strikte Trennung zwischen Datenverarbeitung und Nutzerschnittstelle
- Code wird nur einmal geschrieben.
  DRY — *Don't Repeat Yourself*

Hintergrundwissen

- Lokale vs. globale Variablen, Kontext der Ausführung
- Objektorientierte Programmierung

Grundlegend zwei Konzepte:

1. ein Skript für jeden Datensatz
2. eine Toolbox aus Funktionen,
   die generisch jeden Datensatz verarbeiten kann

☛ Beide Konzepte haben Vor- und Nachteile.

☛ Wir werden uns im Folgenden auf das
Konzept der Toolbox beschränken.

Modularität

Kurze Wiederholung: Toolbox

- Ein Werkzeug, eine Funktion
- Es gibt unterschiedliche Werkzeugkästen/Toolboxen für unterschiedliche Aufgaben.
- Ein gut sortierter Werkzeugkasten ist essentiell.

Das „Lego-Prinzip“: Alles ist möglich

- Erfolg durch unendliche Kombinierbarkeit
- Einfach(st)e selbsterklärende Bausteine

Die fehlende Kristallkugel: unvorhersehbare Anforderungen

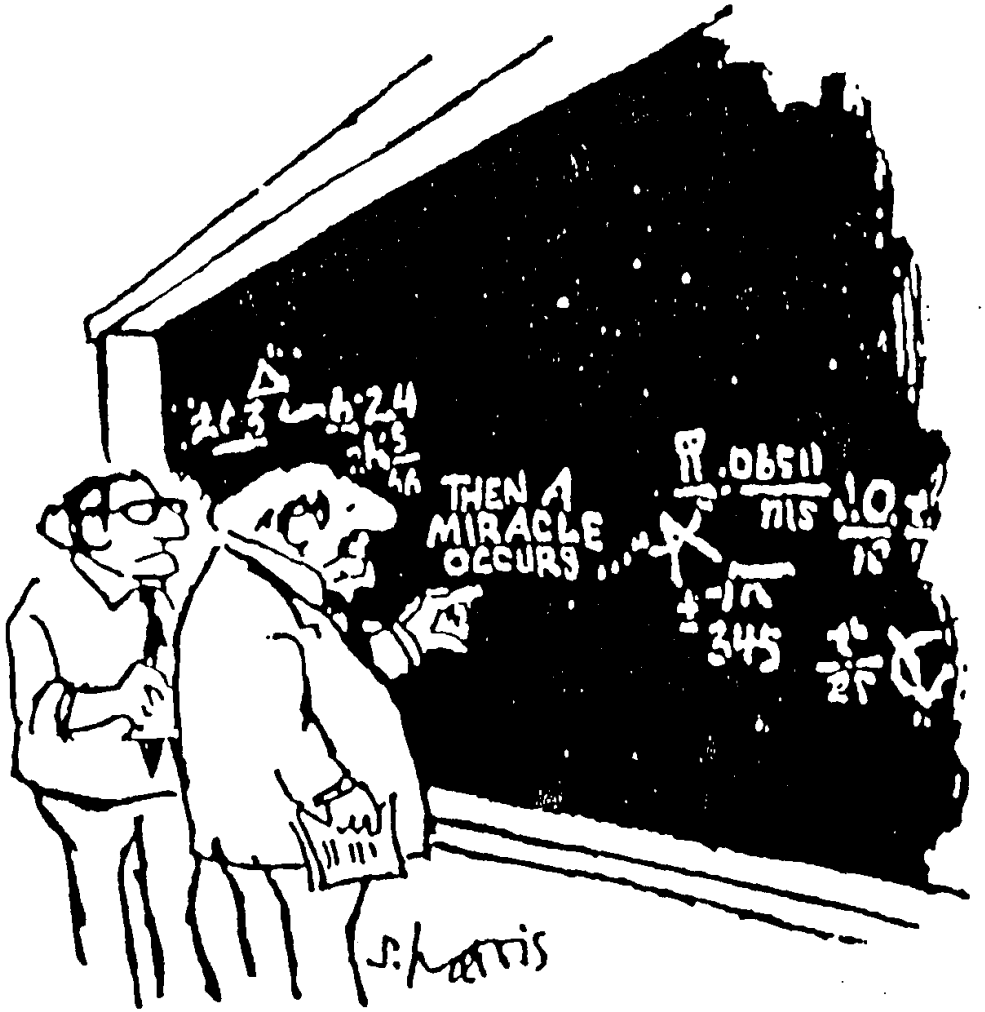
- Wir wissen (und ahnen) meist nicht, was kommt.
- Modularität ist zukunftsfähig.

Eigenständigkeit und Eigenverantwortung

- Alles ist erlaubt, alles ist möglich – auch das „Sinnlose“.
- Grundrecht des Wissenschaftlers: Scheitern

Modularität

Eine Aufgabe, eine Routine



„I think you should be more explicit here in step two.“

Sydney Harris

Typischer Ablauf einer Datenverarbeitung:

1. (Roh-)Daten einlesen
2. Vorverarbeitung der Daten
   (z.B. Hintergrundkorrektur, Glättung)
3. Simulation anpassen
4. Daten und Simulation darstellen
5. Abbildung speichern

- ☛ Jeder einzelne dieser Schritte sollte in eine eigene Funktion ausgelagert werden.
- ☛ Mancher Schritt erfordert mehr als eine Funktion.

Skript

- Aufeinanderfolge einzelner Befehle
- Keine definierte Schnittstelle nach außen
- Alle definierten Variablen sind global
- Gut geeignet für „Rapid Prototyping“

Funktion (Routine)

- Funktion verhält sich wie ein „normaler“ Befehl
- Klar definierte Schnittstelle nach außen
- Alle definierten Variablen sind lokal
- Gut geeignet für immer wiederkehrende Abläufe

Listing 1: Skript zur Datenverarbeitung (schlecht formatiert)

```
1  fid=fopen('uvvis.txt');
2  k=1; while ~feof(fid) line{k}=fgetl(fid); k=k+1; end
3  fclose(fid);
4  line(1:2)=[]; data=[];
5  for k=1:length(line)
6  data(k,:)=cell2mat(textscan(strrep(line{k},',','.'),'%f %f'));
7  end
8  bg=load('bg.txt'); bg=bg(1:length(data),:); bg(:,2)=bg(:,2)-bg(end,2);
9  bg(:,2)=bg(:,2)*(data(data(:,1)==520,2)/bg(bg(:,1)==520,2));
10 data(:,2)=data(:,2)-bg(:,2);
11 plot(data(:,1),data(:,2),'k-');
12 hold on; plot(data(:,1),zeros(length(data(:,2)),1),'k--'); hold off;
13 xlabel('{\it wavelength} / nm'); ylabel('{\it intensity} / OD');
14 set(get(gca,'xlabel'),'fontsize',12);set(get(gca,'ylabel'),'fontsize',12);
15 set(get(gca,'xlabel'),'fontname','Arial');
16 set(get(gca,'ylabel'),'fontname','Arial');
17 set(gca,'fontsize',12); set(gca,'fontname','Arial');
18 set(gcf,'paperunits','centimeters');set(gcf,'papersize',[16 10]);
19 set(gcf,'paperpositionmode','auto');
20 set(gca,'Units','centimeters');set(gca,'OuterPosition',[0 0 16 10]);
21 set(gcf,'Units','centimeters');oldpos = get(gcf,'Position');
22 set(gcf,'Position',[oldpos([1 2]) 16 10]);
23 print(gcf,'data.pdf','-dpdf');
```

Listing 2: Reihe von Funktionsaufrufen zur Datenverarbeitung

```
1 data = loadUVVisData('uvvis.txt');
2 bg   = loadUVVisData('bg.txt');
3 data = subtractBackgroundFromData(bg,data);
4 plotUVVisData(data,'zeroLine',true);
5 figure2file(gcf,'data.pdf');
```

Anmerkungen

- Sprechende Funktionsnamen
  - Name impliziert Reihenfolge der Parameter
  - Code kommt quasi ohne Dokumentation aus
- Eine Aufgabe, eine Routine
  - Funktionen müssen implementiert werden
  - Routinen ggf. in der Realität komplexer als hier gezeigt

UNI FREIBURG

Das „Unix-Prinzip“

Eine Aufgabe, eine Routine

Wie erkennt man, was „eine Aufgabe“ ist?

- DRY Code — *Don’t Repeat Yourself*
- KISS-Prinzip — *Keep it simple and stupid.*

☛ Wenn man den Code in Blöcke aufteilen kann, kann man ihn in Funktionen aufteilen.

Funktionen in Matlab

- Funktionsname und Dateiname müssen übereinstimmen.
- Nur eine Funktion pro Datei (Ausnahme: Unterfunktionen)

Benennung von Funktionen in Matlab

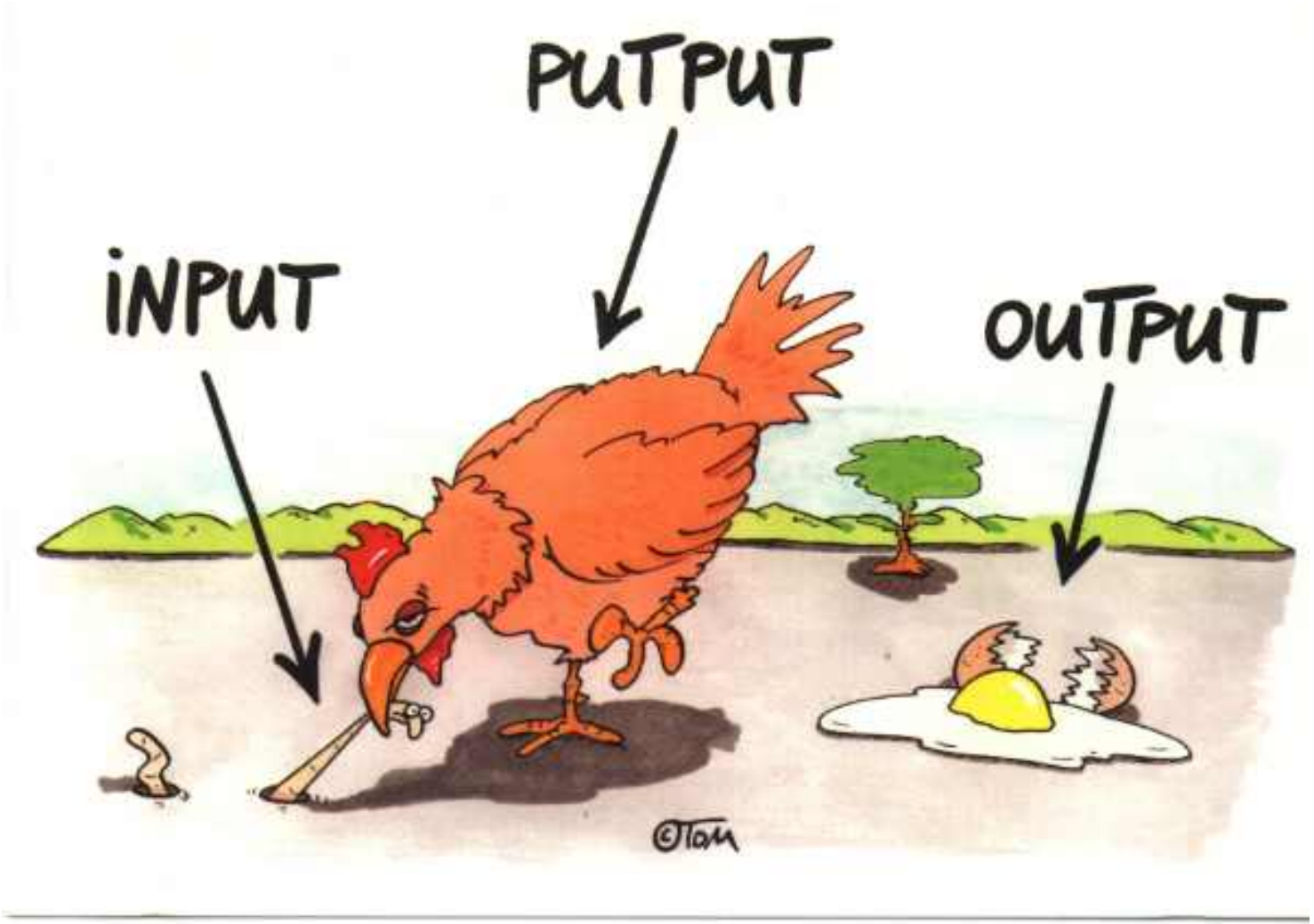
- Matlab unterscheidet zwischen Groß- und Kleinschreibung.
- Funktionsnamen müssen mit einem Buchstaben beginnen.
- Sonderzeichen sind nicht erlaubt. (Ausnahme: „_“)

☛ Tipp: Sprechende Namen erhöhen die Lesbarkeit

☛ Tipp: Präfix zur Vermeidung von Doppelungen

Modularität

Klar definierte Schnittstellen



Thomas Körner, alias ©TOM

Listing 3: Funktionsdeklaration in Matlab

```
function [out1,out2] = myFunction(in1,in2)
```

Funktionsdeklaration in Matlab

1. Schlüsselwort „`function`“
2. Liste der Rückgabeparameter (output)
3. Funktionsname
4. Liste der Übergabeparameter (input)

☛ Es gibt Funktionen *ohne* Parameter

Listing 4: Funktionsdeklaration mit optionalen Parametern

```
function [varargout] = myFunction(varargin)
```

Erhöhung der Flexibilität: optionale Parameter

- optionale Übergabeparameter: `varargin`
- optionale Rückgabeparameter: `varargout`
- Lassen sich mit obligatorischen Parametern kombinieren

☞ „`varargin`“ immer als letzter Übergabeparameter bei der Funktionsdeklaration erhöht die Flexibilität.

Vorgriff: Robuster Code

- Anzahl der Parameter überprüfen
  - `nargin`, `nargout`
- Typ der Übergabeparameter überprüfen
  - `isnumeric`, `ischar`, `iscell`, `isstruct`...
- Sinnvolles Verhalten bei falschen Parametern
  - „graceful exit“
  - `error` ist die schlechteste Lösung
- Rückgabeparameter zu Beginn der Funktion auf Standardwerte setzen
  - Führt sonst zu Fehlern je nach Funktionsaufruf

☛ Robustheit von Code wird separat behandelt

Stabilität von Schnittstellen

- Schnittstellen sind das, was der Anwender von einer Funktion „sieht“ und was er verwendet.
  - Innerhalb der Funktion kann man (fast) alles ändern.
  - Schnittstellen sollten möglichst früh definiert werden.
  - Nachdenken zahlt sich aus: vorausschauend planen.
- Änderungen der Schnittstellen nur in begründeten Fällen
  - Inkompatible Änderungen führen zu Fehlern in Code, der auf älteren Versionen der Funktion basiert.
  - Wann immer möglich abwärtskompatibel programmieren
  - Wichtig: Dokumentation inkompatibler Änderungen

☛ Klares Konzept für Versionsnummern

Ein Wort zu Versionsnummern

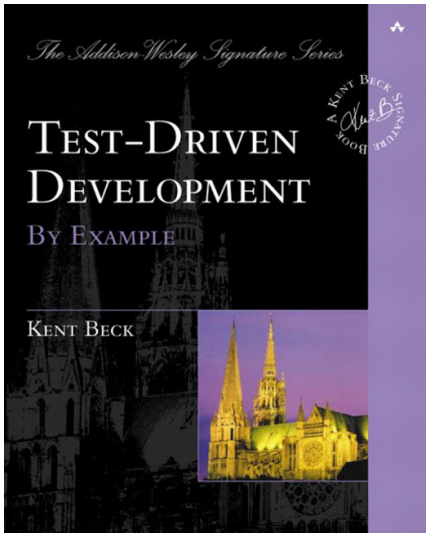
- Stichwort: „Semantic Versioning“
  - Dreistufige Versionsnummern: a.b.c
  - Hauptversion (a), Nebenversion (b), Revision (c)
  - Jede Veränderung in einer veröffentlichten Version führt zu einer Änderung der Versionsnummer.
- Ab Version 1.0.0 muß eine Schnittstelle innerhalb einer Hauptversion (n.x.x) abwärtskompatibel gestaltet werden.
  - Saubere Dokumentation der Schnittstelle
  - Dokumentation aller Änderungen
  - Ggf. Hilfsroutinen zur Gewährleistung der Kompatibilität

☛ `http://semver.org/`

Testgetriebene Entwicklung

engl. *test-driven development*, konsequente Entwicklung von Tests *vor* den zu testenden Komponenten

- Zwei Prinzipien:
  1. Never write a single line of code unless you have a failing automated test.
  2. Eliminate duplication.
- Voraussetzungen
  - Tests mit ausreichender Abdeckung
  - Versionsverwaltung



Nutzerschnittstelle

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

Beispiele für Nutzerschnittstellen

- Textschnittstelle (CLI)
- graphische Schnittstelle (GUI)

☞ Nutzerschnittstellen verarbeiten Daten nie selbst, sondern rufen dazu die passenden Routinen auf!

Gründe für die Trennung

- Saubere und fehlerfreie Datenverarbeitung ist in der Wissenschaft von allergrößter Bedeutung.
  - Nachvollziehbarkeit der Prozessierung der Daten
- 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 GUI 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.

☛ Nutzerschnittstellen werden im Detail später an einem gesonderten Termin besprochen.

Möglicher Kontext einer Variablen

lokal nur für die jeweilige Funktion „sichtbar“

global für alle Funktionen „sichtbar“

Konsequenzen

- Unterschiedliche Funktionen können Variablen mit dem gleichen Namen verwenden.
- Eine Funktion kennt nur globale, ihr übergebene oder in ihr definierte Variablen.
- Die Trennung nach Kontext sorgt für Übersichtlichkeit.

Kontext einer Variablen bei Matlab

- Variablen sind *per se* lokal.
- Die Verwendung globaler Variablen ist möglich, aber nicht zu empfehlen.
  - Alternative: Funktion mit sauberer Schnittstelle

Besonderheiten bei Unterfunktionen in Matlab

- Unterfunktion innerhalb der Hauptroutine definiert
  - Variablen der Hauptroutine in der Unterfunktion sichtbar
- Unterfunktion außerhalb der Hauptroutine definiert
  - Variablen der Hauptroutine in der Unterfunktion nicht sichtbar

Listing 5: Unterfunktion innerhalb der Hauptroutine

```
1  function [out1,out2] = myFunctionWithSubfunction(in1,in2)
2  % MYFUNCTIONWITHSUBFUNCTION Demonstrating subfunctions and variable context.
3
4      % Some nice code...
5
6      % Call to subfunction
7      mySubfunction;
8
9      function mySubfunction
10     % MYSUBFUNCTION Demonstrating a rather silly subfunction.
11
12         out1 = in2;
13         out2 = in1;
14     end % subfunction
15
16 end % function
```

☛ Unterfunktion kann direkt auf die Variablen der Hauptroutine zugreifen.

Listing 6: Unterfunktion außerhalb der Hauptroutine

```
1  function [out1,out2] = myFunctionWithSubfunction(in1,in2)
2  % MYFUNCTIONWITHSUBFUNCTION Demonstrating subfunctions and variable context.
3
4      % Some nice code...
5
6      % Call to subfunction
7      [out1,out2] = mySubfunction(in1,in2);
8
9  end % function
10
11 function [out1,out2] = mySubfunction(in1,in2)
12 % MYSUBFUNCTION Demonstrating a rather silly subfunction.
13
14     out1 = in2;
15     out2 = in1;
16 end % subfunction
```

☛ Unterfunktion muss Variablen der Hauptroutine über ihre Schnittstelle übergeben bekommen.

Unterfunktionen in Matlab

- Unterfunktion kann nur von der Hauptfunktion in der gleichen Datei aufgerufen werden.
- Vorteil: Unterfunktion „versteckt“
- Nachteil: Codeduplizierung wahrscheinlich

Alternative: `private`-Verzeichnis

- Alle Funktionen in diesem Verzeichnis sind nur von Funktionen im Verzeichnis direkt darüber aufrufbar.
- Funktionen in diesem Verzeichnis erscheinen nicht im Matlab-Suchpfad.

Objektorientierte Programmierung (OOP)

- Daten und verarbeitende Routinen in einem Objekt
- (Eigentlich) intuitiver als prozedurale Programmierung
- Erzwingt und erleichtert Modularisierung

Grundbegriffe

Klasse	Definition der Datenstruktur eines Objektes
Objekt	Instanz einer Klasse
Methoden	Algorithmen (Funktionen) eines Objekts
Attribute	Eigenschaften eines Objekts

Grundbegriffe (Fortsetzung)

Abstraktion	Beschreibung der Fähigkeiten eines Objektes ohne Rücksicht auf Details der Implementation
Kapselung	Kein (direkter) Zugriff auf die interne Datenstruktur eines Objektes
Feedback	Objekte kommunizieren über einen Nachricht-Antwort-Mechanismus
Persistenz	Objektvariablen existieren, solange die Objekte vorhanden sind.
Vererbung	Abgeleitete Klassen „erben“ die Methoden und Attribute der Basisklasse.

Modularität

Objektorientierte Programmierung einfach erklärt: das Auto

☛ Was macht ein Auto aus? Was hat es? Was kann es?

Die abstrakte Klasse „Auto“

- Was *hat* ein Auto? (Attribute)
  - Räder (Wie viele?)
  - Türen (Wie viele?)
  - Sitze
  - Motor (Was für einer? Diesel? Benziner?)
  - Lenkrad
  - Gangschaltung (manuell? Automatik? Wie viele Gänge?)
- Was *kann* ein Auto? (Methoden)
  - Motor starten und stoppen
  - Fahren
  - Lenken
  - Gang wechseln
  - Blinken

Vererbung: Die abgeleitete Klasse „VW Käfer“

- 4 Räder
- 3 Türen
- 4-Zylinder-Boxermotor
- 5-Gang-Schaltgetriebe
- ...

- „VW Käfer“ implementiert die abstrakte Klasse Auto.
- Das Objekt ist das einzelne Auto der Klasse VW Käfer.
- Autofahren: Wir kennen die Methoden der abstrakten Klasse Auto, die Implementierung ist Detail.

Typische objektorientierte Programmiersprachen

- Smalltalk (damit fing alles an)
- C++
- Java

Matlab und Objektorientierung

- Auf den ersten Blick prozedural
- Intern stark objektorientiert, teils Java-basiert
- Unterstützt (in neueren Versionen) Objekte und OOP

☛ Interessierte finden Details in den Matlab-Handbüchern.

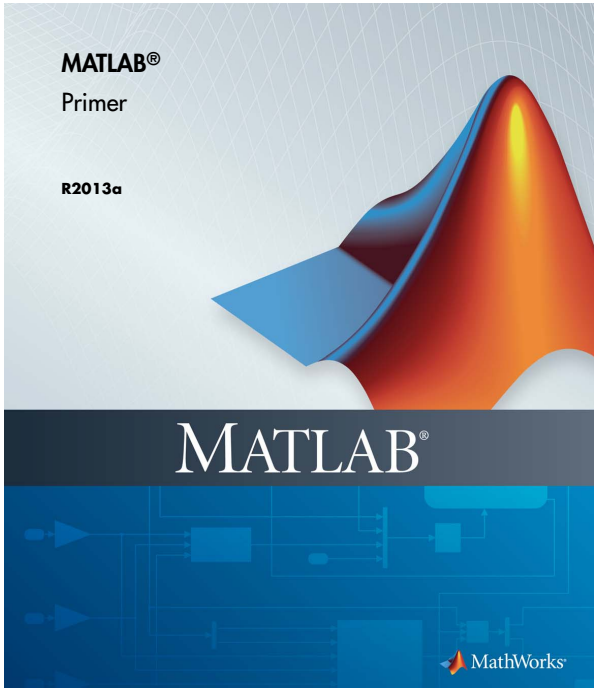
Kurze Pause

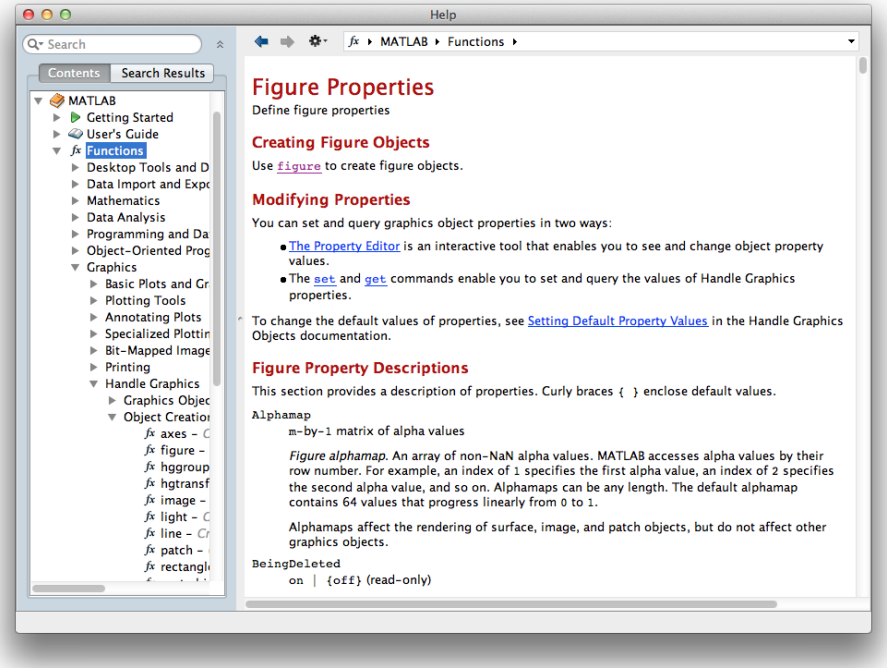
Gleich geht's weiter...

„Lunch atop a Skyscraper“, Charles C. Ebbets, 1932

Dokumentation

Entscheidend für gute Code-Qualität





Listing 7: linux-2.2.16/fs/buffer.c

```
1 /*
2  * We used to try various strange things. Let's not.
3  */
```

Real programmers don't comment their code.
If it was hard to write, it should be hard to read.

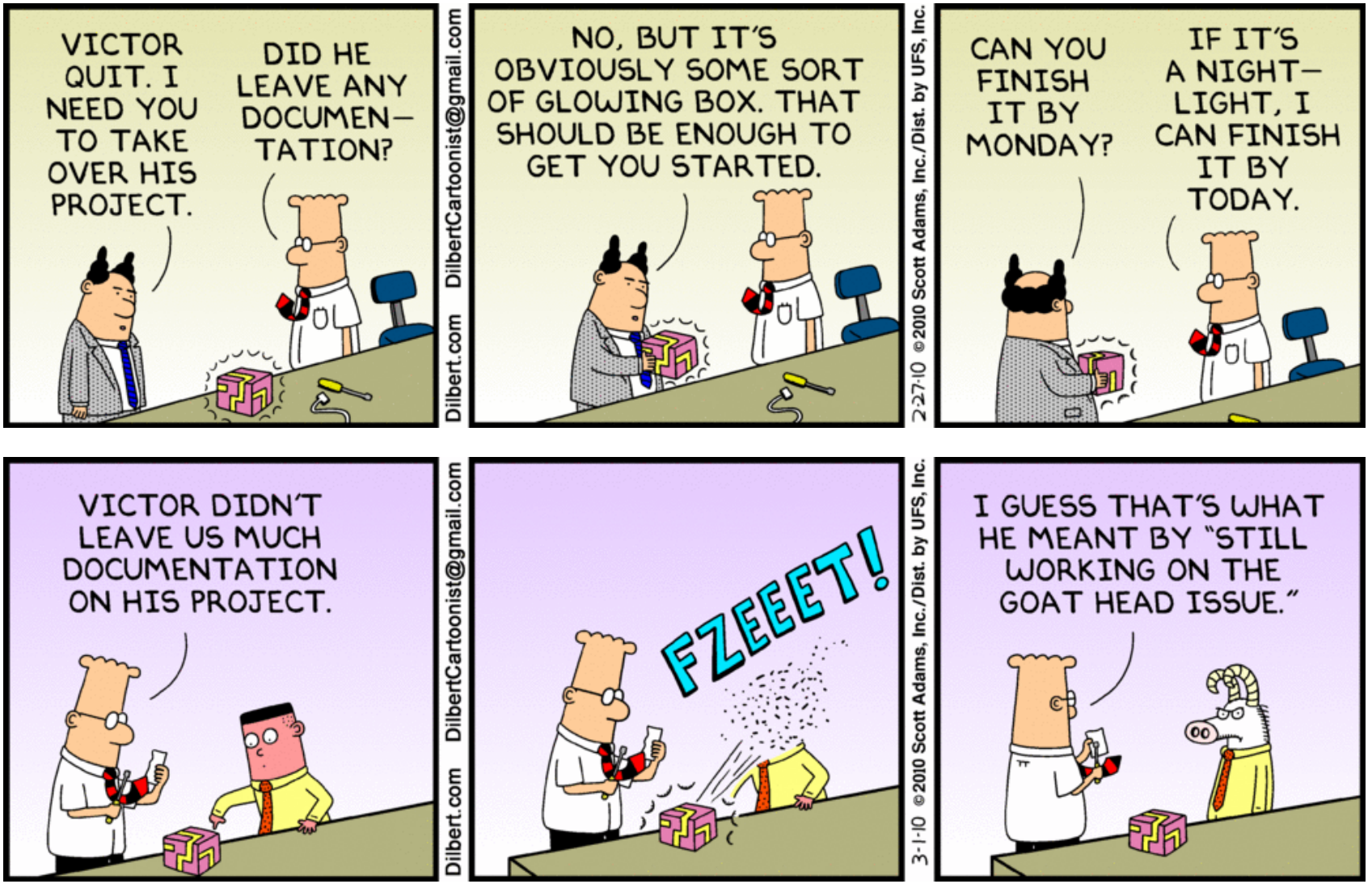
Warum dokumentieren?

- Weil andere das Programm nutzen/verstehen wollen.
- Weil man sich selbst nach zwei Monaten nicht mehr daran erinnern kann, was man da programmiert hat.
- Weil wir in aller Regel zu schlecht programmieren.
- Weil Dokumentation (gerade von Konzepten) für die weitere Entwicklung sehr hilfreich ist.

☛ Weil nur sauberer (dokumentierter) Code Zukunft hat.

Dokumentation

Motivation: Warum dokumentieren?



Viele verschiedene Arten

- Im Quellcode
  - Schnittstellen-Dokumentation
  - Quellcode-Dokumentation (einzelne Zeilen)
- Im gleichen Verzeichnis wie die Funktionen/Toolbox
  - `README`
  - `INSTALL`
- Nutzerhandbuch
  - Beschreibung der Verwendung jeder Funktion
- Konzepte
- Beispiele

Unterschiedliche Einteilung

- nach Zielgruppe
  - Programmierer
  - Anwender
- nach Inhalt
  - Quellcode-Dokumentation
  - Schnittstellen-Dokumentation
  - Dokumentation der Konzepte
  - Installation, Bedienung, ... (Anwenderdokumentation)
- nach Medium
  - im Quellcode
  - in Dateien neben dem Quellcode
  - getrennt vom Quellcode (Webseite, ...)

Dokumentation im Code

Kein Code ohne Dokumentation

UNI FREIBURG

Listing 8: linux-2.2.16/lib/vsprintf.c

```
1 /* vsprintf.c -- Lars Wirzenius & Linus Torvalds.
2  *
3  * Wirzenius wrote this portably, Torvalds fucked it up :-)
4  */
```

Listing 9: linux-2.2.16/fs/buffer.c

```
1 /*
2  * After several hours of tedious analysis, the following
3  * hash function won.   Do not mess with it... -DaveM
4  */
```

Listing 10: linux-2.0.38/arch/m68k/atari/atafb.c

```
1 /* Nobody will ever see this message :-) */
2 panic("Cannot initialize video hardware\n");
```

Dokumentation im Code

Kein Code ohne Dokumentation

UNI FREIBURG

Real Programmers don't need comments–
the code is obvious.

— Ed Post, 1983

- Nicht dokumentierter Code ist (oft) wertlos.
- Fehlende Kommentare erschweren das Lesen von Code.
- Dokumentation im Code sollte kurz, prägnant und informativ sein.

☛ Beispiel gefällig?

Ed Post, „Real Programmers Don't Use Pascal", Datamation 29(7), 1983

Dokumentation im Code

Kein Code ohne Dokumentation: Ein Beispiel

UNI FREIBURG

Listing 11: Undokumentierter Code

```
1 fun = @(x,s,t)s^2/4./(s^2/4+(x-t).^2);
2 s = 2;
3 t = 5;
4 x = 0:0.1:10;
5 y = fun(x,s,t);
6 plot(x,y,'k-');
```

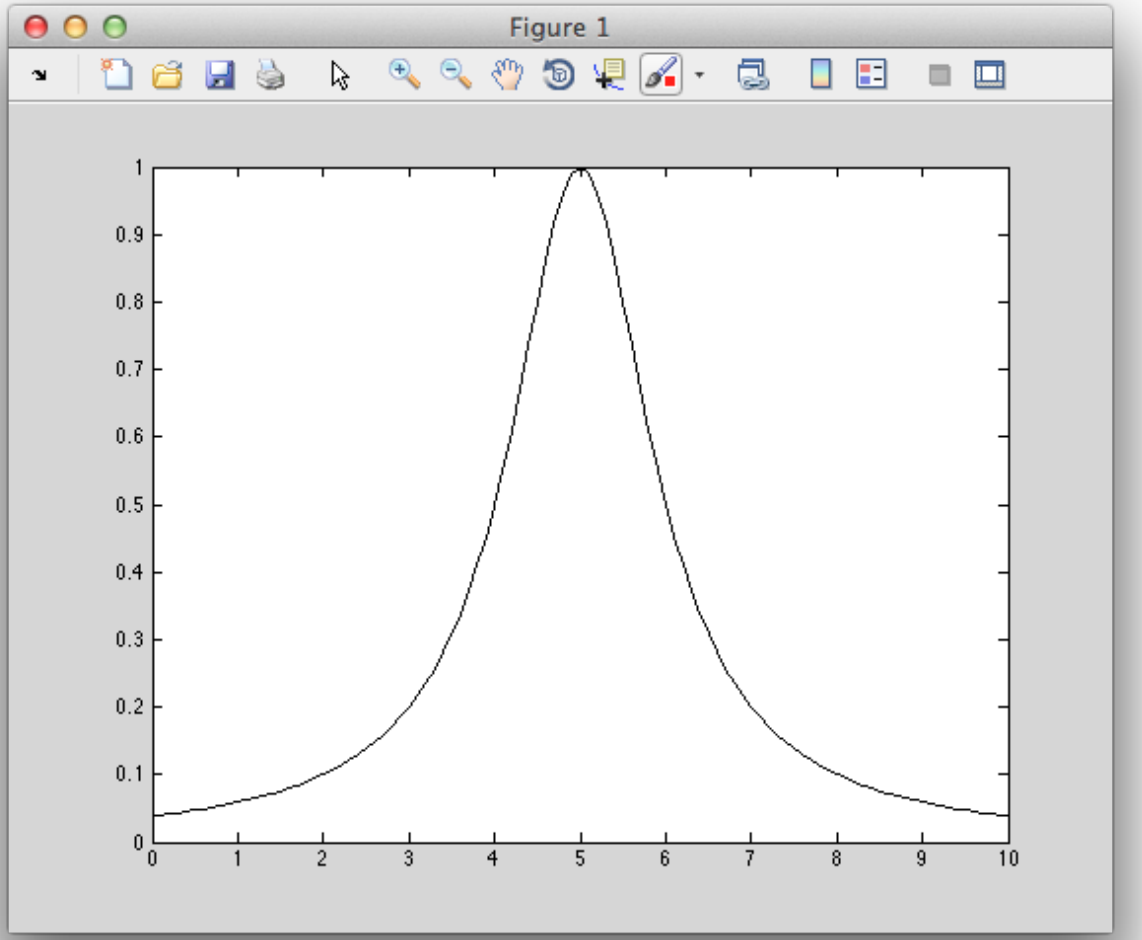
- Keinerlei Kommentare
- Keine Gliederung des Codes durch Leerzeilen
- Keine oder wenige sprechende Variablennamen

☛ Irgendeine Idee, was der Code macht?

☛ Wie könnte der Code vernünftig dokumentiert aussehen?

Dokumentation im Code

Kein Code ohne Dokumentation: Ein Beispiel



Dokumentation im Code

Kein Code ohne Dokumentation: Ein Beispiel

UNI FREIBURG

Listing 12: Dokumentierter Code (lorentzian.m)

```
1  % LORENTZIAN Plot Lorentzian curve with fixed height.
2
3  % (c) 2013-14, Till Biskup <till.biskup@physchem.uni-freiburg.de>
4  % 2014-01-14
5
6  % Define Lorentzian with fixed height
7  %   s - width of the curve (FWHM)
8  %   t - position of the maximum
9  Lorentzian = @(x,s,t)s^2/4./(s^2/4+(x-t).^2);
10
11 % Define values for Lorentzian curve
12 width  = 2;
13 maxpos = 5;
14
15 % Define x,y vectors
16 x = 0:0.1:10;
17 y = Lorentzian(x,width,maxpos);
18
19 % Plot
20 plot(x,y,'k-');
```

Dokumentation im Code

Kein Code ohne Dokumentation

UNI FREIBURG

Regel

Keine Datei (Routine, Skript) ohne Dokumentation am Anfang.

- Dokumentation ist eine Frage der Disziplin.
- Zum „Rapid Prototyping“ eignet sich die Kommandozeile.
  Skripte werden dokumentiert!
- Dokumentation muss zur Routine werden.
  Im Nachhinein zu dokumentieren ist keine Option,
  weil es dann nie gemacht wird.

Zwei Arten von Dokumentation im Code

1. Kommentarköpfe von Funktionen
   - kurze Zusammenfassung der Aufgaben der Routine
   - präzise Beschreibung der Schnittstelle
   - Autor, Copyright und Datum der letzten Änderung
   - ggf. Hinweise auf Besonderheiten

2. (Meist) einzeilige Kommentare im Code
   - kurze Erklärungen zum folgenden Code-Abschnitt
   - wichtig: nicht das Offensichtliche dokumentieren

☛ Bis auf die Beschreibung der Schnittstelle gilt der Kommentarkopf für Skripte ebenso.

Dokumentation im Code

Grundlegende Struktur eines Kommentarkopfes

UNI FREIBURG

Ein Kommentarkopf besteht mindestens aus drei Teilen

1. Kurze Zusammenfassung der Aufgaben der Routine
   - Mindestens ein Satz, besser ein kurzer Absatz, der die Aufgabe der Routine präzise beschreibt.
2. Präzise Beschreibung der Schnittstelle
   - Funktionsaufruf
   - Dokumentation der Ein- und Ausgabeparameter (Typ, Bedeutung)
3. Autor, Copyright und Datum der letzten Änderung
   - Wichtig: Datum der letzten Änderung immer anpassen!

☛ Der Kommentarkopf liefert dem Nutzer ein Maximum an Information auf minimalem Raum.

Dokumentation im Code

Grundlegende Struktur eines Kommentarkopfes

UNI FREIBURG

Listing 13: Dokumentationsblock zu Beginn einer Routine

```
1  function spectrum = simCO2spectrum(x,fwhm,pos,height)
2  % SIMCO2SPECTRUM Function for simulating CO2 spectra using Lorentzians.
3  %
4  % Usage
5  %   spectrum = simCO2spectrum(x,fwhm,pos,height)
6  %
7  %   x        - vector
8  %              wavelength axis for spectrum
9  %
10 %   fwhm     - vector
11 %              spectral width of each Lorentzian
12 %
13 %   pos      - vector
14 %              position of each Lorentzian
15 %
16 %   height   - vector
17 %              height of each Lorentzian
18 %
19 %   spectrum - vector
20 %              calculated spectrum
21 %              same length as x
22 %
23 % The vectors fwhm, pos, and height have to be of the same size.
```

Dokumentation im Code

Grundlegende Struktur eines Kommentarkopfes

UNI FREIBURG

Listing 14: Copyrighthinweis mit einem Autor

```
1 % (c) 2011-13, Till Biskup <till.biskup@physchem.uni-freiburg.de>
2 % 2013-11-02
```

Listing 15: Copyrighthinweis mit mehreren Autoren

```
1 % (c) 1997-2005, A. Kabelschacht
2 % (c) 2006-2013, Till Biskup <till.biskup@physchem.uni-freiburg.de>
3 % 2013-11-02
```

- Die Angabe einer Email-Adresse ist optional.
- Der Copyrighthinweis wird durch eine Leerzeile vom Kommentarkopf getrennt.

Dokumentation im Code

Konkrete Hinweise für Matlab

UNI FREIBURG

- Der erste Kommentarblock einer Datei wird beim Aufruf des Befehls `help` ausgegeben.
  - Ähnliches gilt für andere Sprachen und im Zusammenhang mit automatischer Generierung der Dokumentation (Doxygen, Javadoc etc.)
- Da der Copyright-Hinweis nicht ausgegeben werden soll, wird dieser durch eine Leerzeile vom ersten Kommentarblock abgetrennt.
- Symbolworte, die vom `help`-Befehl erkannt werden
  - `see also`

Dokumentation im Code

Ein Wort zur Sprache

UNI FREIBURG

- Kommentare im Code idealerweise auf Englisch
  - Wissenschaft ist international.
  - Japanische oder russische Kommentare im Qellcode sind leider wenig hilfreich...
- Aber: Lieber deutsche als keine Kommentare!
  - Es ist (meist) einfacher, einen Übersetzer zu finden, als einen undokumentierten Code zu verstehen.
- Grundsätzlich nie Sonderzeichen (z.B. Umlaute)
  - Unicode hat sich (leider) immer noch nicht durchgesetzt.
  - Beschränkung auf den ASCII-7-bit-Zeichensatz

Dokumentation im Code

Der ASCII-7-bit-Zeichensatz

UNI FREIBURG

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 $7e_{Hex}$ (32 bis 126) sind „druckbar“.

Dokumentation im Code

Dokumentationshilfen

UNI FREIBURG

- Saubere Dokumentation im Code zahlt sich aus:
  - Automatische Generierung von Dokumentation aus den Kommentaren im Quellcode.
- Matlab
  - `help`-Befehl gibt ersten Kommentarblock aus.
  - Interne Funktionen sind gute Vorlagen/Beispiele.
  - Unterstützt das „Publizieren“ von Funktionen
- Dokumentationshilfen für andere Sprachen als Matlab
  - Doxygen, Javadoc, ROBODoc
  - Liefern Ideen für eine sinnvoll strukturierte Dokumentation
  - Erlauben die automatische Erstellung in diversen Formaten (LaTeX, DocBook, RTF, ...)

Dokumentation im Code

Häufige Probleme von Dokumentation

UNI FREIBURG

Problem

- Dokumentation hinkt der Entwicklung des Codes hinterher
  - Problem gerade bei Kommentarzeilen im Code:
    im schlimmsten Fall falsch und irreführend!

Lösungsansatz

- Keine unnötigen Kommentare im Code
  - Offensichtliches nicht dokumentieren
  - Offensichtliche Programmierung der Dokumentation
    einer weniger offensichtlichen Lösung vorziehen
- Beim Ändern von Code immer auf Kommentare achten

☛ Letztlich eine Frage der (persönlichen) Disziplin.

Dokumentation im Code

Häufige Probleme von Dokumentation

UNI FREIBURG

Problem

- Fehlende Dokumentation der Installation und Bedienung
  - Nichts ist schlimmer als ein Verzeichnis voller Skripte/Funktionen ohne jegliche Beschreibung.

Lösungsansatz

- Externe Dokumentation
  - Extern zu den Funktionen
  - ggf. in eigenem Unterverzeichnis (`doc`)
  - reine Textdateien, kurz und prägnant
  - mindestens `README`, ggf. `INSTALL`

Dokumentation im Code

Häufige Probleme von Dokumentation

UNI FREIBURG

Problem

- Fehlende Dokumentation der Konzepte und Ideen
  - Schnittstellen-Dokumentation meist nicht ausreichend
  - Grundlegende Konzepte im Kontext dokumentieren
  - Statisches Dokument oft zu unflexibel

Lösungsansatz

- Konzeptionelle Dokumentation in einem Wiki
  - Flexibel
  - Erlaubt einfache Aktualisierungen
  - Geeignet als primäre Informationsquelle für Anwender.

☛ Wird in der nächsten Lektion detaillierter besprochen.

Ausblick

Das war's. Wie geht's weiter?

UNI FREIBURG

So long, and thanks for all the fish.

Vorschau: Infrastruktur

- Versionsverwaltung
- Bug-Verwaltung
- Dokumentation (Wiki)

Douglas Adams

- ☛ Saubere Dokumentation erspart anderen böse Überraschungen und schmerzhafte Erfahrungen.