Software Engineering

SS 2015

Hans-Georg Eßer
Dipl.-Math., Dipl.-Inform.

Foliensatz C2: v1.0, 2015/07/10

- Software-Dokumentation (Fortsetzung)

Dieser Foliensatz

A	Vorlesungsübersicht Einführung u. Motivation Software Engineering Seminar / Wiss. Arbeiten / LaTeX
B	Prinzipien und Methoden Klassische Vorgehensmodelle Agile Modelle
Folien C	Software-Dokumentation (Fortsetzung, C2)
D	Secure Software Engineering …

Software-Dokumentation (Teil 2)

Gliederung

- Motivation für Dokumentation
- Code Comprehension und Bloom-Taxonomie
- Code-Kommentare
- Dokumentationsgeneratoren
- UML

→ Teil 1 (Foliensatz C1)

- Veränderungsdokumentation / Versionierung
- Literate Programming

Versionierung und Veränderungsdokumentation

Versionsverwaltung

- Aufgabe: Änderungen an Dokumenten (oft: an Quellcode-Dokumenten) verwalten
- Eine Auswahl von Tools
  - 1972: Source Code Control System (SCCS)
  - 1982: Revision Control System (rcs)
  - 1990: Concurrent Versions System (cvs)
  - 2000: Subversion (svn)
  - 2005: Git
  - 2005: Mercurial (hg)

Versionsverwaltung: Naiver Ansatz (1)

- Naiver Ansatz: Versionen in separaten Ordnern speichern
- Vergleich der Versionen mit `diff`

```
$ find . -name '*.tex' -exec ls -l {} \;
-rw-r--r--@ 1 ...   823127 31 Okt  2013 ./2013-10-31/diss-hgesser-ulix.tex
-rw-r--r--@ 1 ...   896574  7 Jan  2014 ./2014-01-07/diss-hgesser-ulix.tex
-rw-r--r--@ 1 ...  1005074  2 Mai  2014 ./2014-05-02/diss-hgesser-ulix.tex
-rw-r--r--@ 1 ...  1005063  3 Mai  2014 ./2014-05-03/diss-hgesser-ulix.tex
-rw-r--r--@ 1 ...  1594695 25 Aug 22:26 ./2014-08-25/diss-hgesser-ulix.tex
$ diff 2014-05-02/diss-hgesser-ulix.tex 2014-05-03/diss-hgesser-ulix.tex
12312d12311
< char *buf;
$ _
```


Versionsverwaltung: Naiver Ansatz (2)

- Nachteile des naiven Ansatzes
  - alle Dateien mehrfach vollständig gespeichert
  - keine Zusammenfassung der Änderungen von einer Version zur nächsten
  - keine automatische Zuteilung von Versions- bzw. Revisionsnummern
- Alternative: spezialisierte Versionsverwaltungen
  - speichern jeweils nur die Änderungen (Delta)
  - legen Versionsnummern und Kommentare an
  - multi-user-tauglich

Versionsverwaltungssoftware

- Alle Versionsverwaltungen unterstützen (mindestens) diese drei Operationen:
  - **ADD:** Eine Datei (oder einen ganzen Ordner) hinzufügen (= unter die Verwaltung stellen)
  - **CHECK-IN / COMMIT:** Die aktuelle Arbeitsfassung „einchecken“ (sichert die Datei unter einer neuen Revisionsnummer ins Archiv)
  - **CHECK-OUT:** Die letzte oder eine ältere Revision „aus-checken“, also aus dem Archiv wiederherstellen

Beispiel: Mercurial (1)

- hier nur lokaler Einsatz (kein Repository im Netzwerk)
- Verzeichnis für die Nutzung von Mercurial vorbereiten:

  ```
  hg init
  ```

- Datei hinzufügen:

  ```
  hg add Dateiname
  ```


Beispiel: Mercurial (2)

- alle aktualisierten Dateien einchecken:

  ```
  hg commit
  ```

  öffnet Editor-Fenster, Eingabe einer Zusammenfassung

```
Erste Revision; nur Datei test.tex

HG: Bitte gib eine Versions-Meldung ein. Zeilen beginnend mit
  'HG:' werden
HG: entfernt. Leere Versionsmeldung wird das Übernehmen
  abbrechen.
HG: --
HG: Benutzer: Hans-Georg Esser <h.g.esser@gmx.de>
HG: Zweig 'default'
HG: Hinzugefügt test.tex
```


Beispiel: Mercurial (3)

- Neue Dateien im Ordner entdecken:

```
$ hg status
? literatur.bib
```

- Andere Revision aus-checken:

```
$ hg co 0
1 Dateien aktualisiert, 0 Dateien
zusammengeführt, 0 Dateien entfernt,
0 Dateien ungelöst
```

(hat im Arbeitsverzeichnis Version 1 durch die ältere Version 0 ersetzt)

Beispiel: Mercurial (4)

- Verlauf der Änderungen anzeigen:

```
$ hg annotate -r0 test.tex
0: \documentclass{article}
0: \begin{document}
0: Hello World
0: \end{document}
$ hg annotate -r1 test.tex
0: \documentclass{article}
0: \begin{document}
0: Hello World
1: One more line
0: \end{document}
$ hg annotate -r2 test.tex
0: \documentclass{article}
0: \begin{document}
2: Old two lines deleted, one new line
0: \end{document}
```


Beispiel: Mercurial (5)

- Vergleich zweier Versionen:

```
$ hg diff -r0:2 test.tex
diff -r 34972b9741f6 -r 3e988791a22a test.tex
--- a/test.tex Mon Dec 01 11:37:11 2014 +0100
+++ b/test.tex Mon Dec 01 11:54:00 2014 +0100
@@ -1,5 +1,5 @@
 \documentclass{article}
 \begin{document}
-Hello World
+Old two lines deleted, one new line
 \end{document}
```

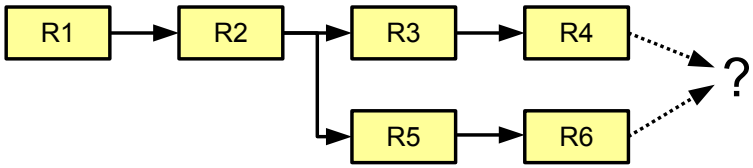

Beispiel: Mercurial (6)

- Protokoll:

```
$ hg log
Änderung:        2:3e988791a22a
Marke:           tip
Nutzer:          Hans-Georg Esser <h.g.esser@gmx.de>
Datum:           Mon Dec 01 11:54:00 2014 +0100
Zusammenfassung: Zwei Zeilen geloescht, eine neu

Änderung:        1:8f2ae3855183
Nutzer:          Hans-Georg Esser <h.g.esser@gmx.de>
Datum:           Mon Dec 01 11:49:28 2014 +0100
Zusammenfassung: Aenderung, eine neue Zeile in test.tex

Änderung:        0:34972b9741f6
Nutzer:          Hans-Georg Esser <h.g.esser@gmx.de>
Datum:           Mon Dec 01 11:37:11 2014 +0100
Zusammenfassung: Erste Revision; nur Datei test.tex
```


Mercurial: Revisions-Bäume

- Durch Ändern einer älteren Version und Wieder-Ein-Checken entsteht ein paralleler Zweig – und damit ein Baum



- Entwicklung kann nun parallel weiter laufen
- Zweige können auch wieder vereinigt werden

Mercurial: Merge

- Weiterentwicklung von Code in mehreren Zweigen
- Zweige sollen zusammengeführt werden

  → **Merge**-Operation

  → Änderungen eines Zweigs in anderen Zweig übernehmen

Mercurial: Merge

Rev. 0:

```
int main () {
  // Block 1
  int i = 0;
  for (; i < 10; i++) {
    printf (i);
  }

  // Block 2
  char c = 'a';
  for (; c <= 'z'; c++) {
    printf (c);
  }
}
```

Rev. 1:

```
int main () {
  // Block 1
  int i = 0;
  for (; i < 10; i++) {
    printf ("%d\n", i);   // format code fehlt!
  }

  // Block 2
  char c = 'a';
  for (; c <= 'z'; c++) {
    printf (c);
  }
}
```

Rev. 2:

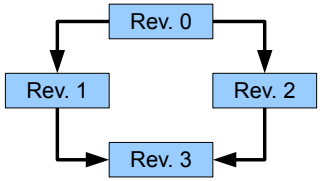
```
int main () {
  // Block 1
  int i = 0;
  for (; i < 10; i++) {
    printf (i);
  }

  // Block 2
  char c = 'a';
  for (; c <= 'z'; c++) {
    printf ("%c \n", c);  // %c!
  }
}
```


Mercurial: Merge

```
$ hg merge -r1            # arbeite gerade mit rev. 2
Führe code.c zusammen
0 Dateien aktualisiert, 1 Dateien zusammengeführt, 0
Dateien entfernt, 0 Dateien ungelöst
(Zusammenführen von Zweigen, vergiss nicht 'hg commit'
auszuführen)
$ hg commit
```

Mercurial: Merge

- Ergebnis:



Rev. 3:

```
int main () {
  // Block 1
  int i = 0;
  for (; i < 10; i++) {
    printf ("%d\n", i);   // format code fehlt!
  }

  // Block 2
  char c = 'a';
  for (; c <= 'z'; c++) {
    printf ("%c \n", c);  // %c!
  }
}
```

Änderung aus Rev. 1

Änderung aus Rev. 2

Mercurial: Merge-Probleme

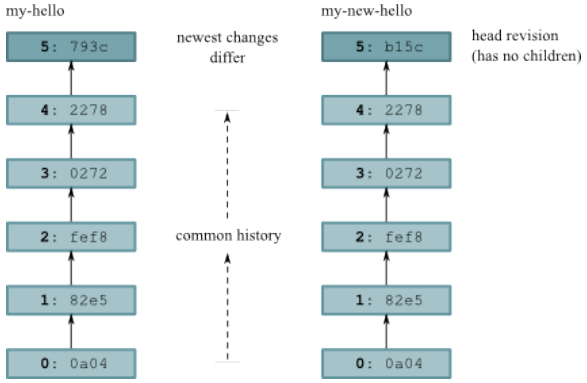
- Merge-Operation erfolgreich, wenn Änderungen in Zweigen verschiedene Bereiche der Datei (oder versch. Dateien) betreffen
- Bei Änderungen derselben Bereiche Konflikt
  → kein automatischer Merge möglich
- Lösen der Konflikte über Tools wie KDiff3

  → KDiff3-Demo

Mercurial: geklonte Repositorys

- Klonen (hg clone) erzeugt unabhängige Kopie eines Repositorys
- Bsp.:

```
$ hg clone \
  my-hello \
  my-new-hello
```

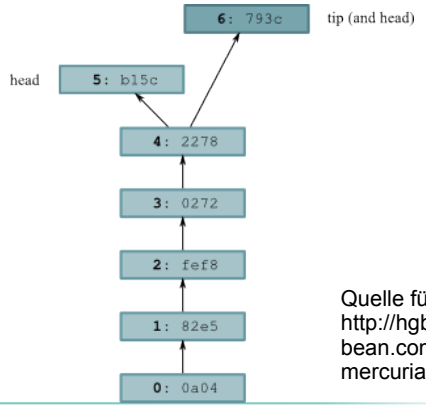


Quelle für Beispiel und Bilder: http://hgbook.red-bean.com/read/a-tour-of-mercurial-merging-work.html

Mercurial: geklonte Repositorys

- Pull-Operation (hg pull) zieht aktuelle Fassung aus anderem Repository
- Bsp.:

```
$ hg pull \
  ../my-hello
```



Quelle für Beispiel und Bilder: http://hgbook.red-bean.com/read/a-tour-of-mercurial-merging-work.html

Mercurial: geklonte Repositorys

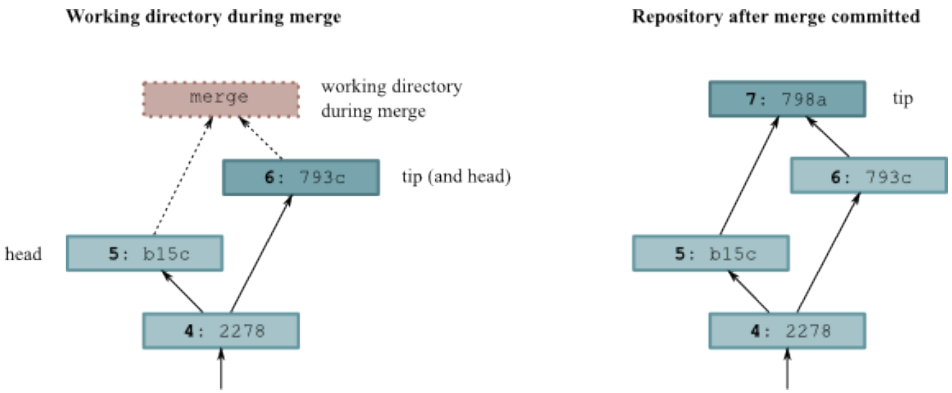
- Zwei Heads (Revisionen ohne Kinder)

```
$ hg heads
changeset:   6:b6fed4f21233
tag:         tip
parent:      4:2278160e78d4
user:        Bryan O'Sullivan <bos@serpentine.com>
date:        Tue May 05 06:55:53 2009 +0000
summary:     Added an extra line of output
changeset:   5:5218ee8aecf3
user:        Bryan O'Sullivan <bos@serpentine.com>
date:        Tue May 05 06:55:55 2009 +0000
summary:     A new hello for a new day.
```

- `hg merge` vereinigt die beiden heads

Mercurial: geklonte Repositorys

- `hg merge` und `hg commit`



Commit Messages (1)

- Commit Messages dokumentieren die *Änderungen* am Code
- Viele Projekte haben eigene Guidelines für Aufbau und Inhalt dieser Doku-Schnipsel
- Immer: Kurz- und Langfassung (vgl. Mail-Subject und Mail-Body)

  → Übersicht der Kurzfassungen

Commit Messages (2)

- Regeln für Commit Messages (exemplarisch)
  1. Separate subject from body with a blank line
  2. Limit the subject line to 50 characters
  3. Capitalize the subject line
  4. Do not end the subject line with a period
  5. Use the imperative mood in the subject line
  6. Wrap the body at 72 characters
  7. Use the body to explain *what and why* vs. *how*

Quelle: Chris Beams, „The seven rules of a great git commit message“, in: http://chris.beams.io/posts/git-commit/

Commit Messages (3)

```
Summarize changes in around 50 characters or less

More detailed explanatory text, if necessary. Wrap it to about 72
characters or so. In some contexts, the first line is treated as the
subject of the commit and the rest of the text as the body. The
blank line separating the summary from the body is critical (unless
you omit the body entirely); various tools like `log`, `shortlog`
and `rebase` can get confused if you run the two together.

Explain the problem that this commit is solving. Focus on why you
are making this change as opposed to how (the code explains that).
Are there side effects or other unintuitive consequenses of this
change? Here's the place to explain them.

Further paragraphs come after blank lines.

 - Bullet points are okay, too

 - Typically a hyphen or asterisk is used for the bullet, preceded
   by a single space, with blank lines in between, but conventions
   vary here

If you use an issue tracker, put references to them at the bottom,
like this:

Resolves: #123
See also: #456, #789
```

Commit Messages (4)

- Regeln (1) bis (6): reine Form
- Regel (7): Inhalt! *What and why?* statt *How?*
  - Das How? beantwortet der Code
  - Commit Messages geben nur Überblick, welche Änderungen es gibt (und warum sie nötig sind)

Ähnliche (aktuelle) Tools

- Subversion (svn)
- Git (entwickelt von Linus Torvalds u. a.); GitHub
  - → **Video:** „The Basics of Git and GitHub“, 51 min., https://www.youtube.com/watch?v=U8GBXvdmHT4 (Start: 155 sec.)
  - Git-Buch: http://git-scm.com/book/en/v2
  - Tutorials: http://teach.github.com/

Literate Programming

Literate Programming

- Grundidee: Nicht Code kommentieren, sondern „erzählen“, wie das Programm funktioniert
- Code und Dokumentation vertauscht
- Beispiel für den Einstieg: Bubblesort

  (Original-Code von http://de.wikipedia.org/wiki/Bubblesort übernommen; nach C portiert)

Bubblesort (1): ohne Kommentare

```
void bubblesort() {
  int SIZE = 10;
  int i, j, tmp;
  for (j=SIZE; j>1; j--) {
    for (i=0; i<j-1; i++) {
      if (arr[i] > arr[i+1]) {
        tmp = arr[i+1];
        arr[i+1] = arr[i];
        arr[i] = tmp;
      }
    }
  }
}
```


Bubblesort (2): mit Kommentaren

```
// bubblesort: sort elements of an array arr[] of size SIZE
void bubblesort() {
  // declarations
  int SIZE = 10;  // size of the array
  int i, j;       // loop variables
  int tmp;        // temporary variable for swapping elements
  // outer loop
  for (j=SIZE; j>1; j--) {
    // check all neighbors in arr[0..j-1] and swap them if their
    // order is wrong
    for (i=0; i<j-1; i++) {
      if (arr[i] > arr[i+1]) {
        // swap neighbors i, i+1 (using tmp as temporary variable)
        tmp = arr[i+1];
        arr[i+1] = arr[i];
        arr[i] = tmp;
      }
    }
  }
}
```


Bubblesort als Literate Program (1)

Bubblesort: The Literate Program

To sort a field `arr[]`, we first need to initialize a `SIZE` variable and declare some local variables, such as `i` and `j` which are used as loop counters, as well as a temporary variable `tmp`:

⟨ *bubblesort: declarations* 15 ⟩≡

```
int SIZE = 10;
int i, j, tmp;
```


Bubblesort als Literate Program (2)

The main routine of the bubblesort algorithm compares each field element with its direct (right) neighbor and corrects their order if it is wrong. After doing this once the biggest element will be at the end of the list. It then repeats these steps with a smaller field (ignoring the right-most element). Thus, with each step in a loop, the unsorted array becomes one element smaller until there is nothing left to sort:

⟨ *bubblesort program* 16 ⟩≡

```
void bubblesort() {
  ⟨ bubblesort: declarations 15 ⟩
  for (j=SIZE; j>1; j--) {
    ⟨ bubblesort: check neighbors in range 0..j 17 ⟩
  }
}
```

Bubblesort als Literate Program (3)

In order to do the neighbor checks, a second loop inside the outer loop is necessary:

```
⟨ bubblesort: check neighbors in range 0..j 17 ⟩≡
 for (i=0; i<j-1; i++) {
   if (arr[i] > arr[i+1]) {
     ⟨ bubblesort: swap elements 17 ⟩
   }
 }
```

For swapping, three commands and usage of a temporary variable are necessary in a C program:

```
⟨ bubblesort: swap elements 17 ⟩≡
 tmp = arr[i+1];
 arr[i+1] = arr[i];
 arr[i] = tmp;
```


Code Chunks (1)

- Elemente der Form ⟨ *name* ⟩ heißen **Code Chunks**.
- Code Chunks werden, wie Makros, an den Stellen ersetzt, wo sie auftreten, z. B.

```
void bubblesort() {
  ⟨ bubblesort: declarations 15 ⟩

  for (j=SIZE; j>1; j--) {
    ...
```

→

```
void bubblesort() {
  int SIZE = 10;
  int i, j, tmp;
  for (j=SIZE; j>1; j--) {
    ...
```


Code Chunks (2)

- Code Chunks dürfen nicht rekursiv sein:

```
⟨ unfug 19 ⟩≡
 for (i=0; i<10; i++) {
   ⟨ unfug 19 ⟩
 }
```

- Chunk kann vor oder nach der Benutzung definiert werden
- Chunk darf auch komplett fehlen
  → verursacht nur eine Warnung

Code Chunks (3)

- Code Chunks können an mehreren Stellen definiert werden
- spätere Auftreten setzen die Definition fort

```
⟨ lange definition 20 ⟩≡
 for (i=0; i<10; i++) {
   // erste Schleife
 }

Text, andere Chunks, …

⟨ lange definition 20 ⟩+≡
 for (i=0; i<10; i++) {
   // zweite Schleife
 }
```

→ wird später zu:

```
for (i=0; i<10; i++) {
  // erste Schleife
}
for (i=0; i<10; i++) {
  // zweite Schleife
}
```

≡ vs. +≡ !

LaTeX und NoWeb (1)

Syntax in LaTeX-Dokumenten:

- `<<chunk name>>` für ⟨*chunk name*⟩
- Definition: `<<name>>= ... @`
- `[[variable]]` für variable

```
<<bubblesort program>>=
void bubblesort() {
  <<bubblesort: declarations>>
  for (j=SIZE; j>1; j--) {
    <<bubblesort: check neighbors in range 0..[[j]]>>
  }
}
@
```


LaTeX und NoWeb (2)

Komplettes Literate Program in LaTeX/NoWeb:

```
\subsubsection{Bubblesort: The Literate Program}

To sort a field, we first need to initialize a [[SIZE]] variable
and declare some local variables, such as [[i]] and [[j]] which are
used as loop counters, as well as a temporary variable [[tmp]]:

<<bubblesort: declarations>>=
int SIZE = 10;
int i, j, tmp;
@

The main routine of the bubblesort algorithm compares each field
element with its direct (right) neighbor and corrects their order
if it is wrong. After doing this once the biggest element will be
at the end of the list. It then repeats these steps with
a smaller field (ignoring the right-most element). Thus, with each
step in a loop, the unsorted array becomes one element smaller
until there is nothing left to sort:
```


LaTeX und NoWeb (3)

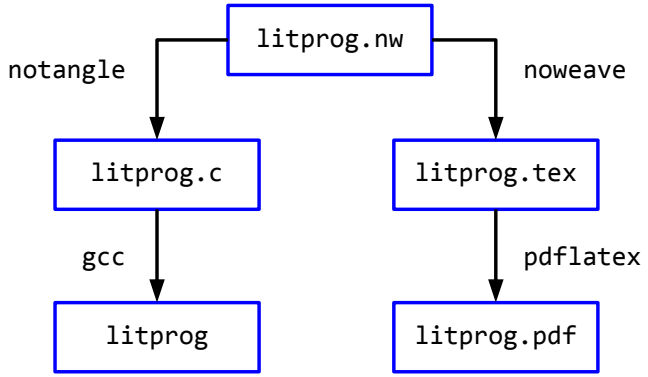
```
<<bubblesort program>>=
void bubblesort() {
  <<bubblesort: declarations>>
  for (j=SIZE; j>1; j--) {
    <<bubblesort: check neighbors in range 0..[[j]]>>
  }
}
@

In order to do the neighbor checks, a second loop inside the
outer loop is necessary:

<<bubblesort: check neighbors in range 0..[[j]]>>=
for (i=0; i<j-1; i++) {
  if (arr[i] > arr[i+1]) {
    <<bubblesort: swap elements>>
  }
}
@
```


LaTeX und NoWeb (4)

```
For swapping, three commands and usage of a temporary variable
are necessary in a C program:

<<bubblesort: swap elements>>=
tmp = arr[i+1];
arr[i+1] = arr[i];
arr[i] = tmp;
@
```

LaTeX und NoWeb (5)

- NoWeb-Tools: `noweave`, `notangle`



noweave

- Syntax von `noweave`:

  ```
  noweave datei.nw > datei.tex
  ```

- `nw`-Datei enthält immer ein vollständiges Literate Program

notangle

- Syntax von `notangle`:

  ```
  notangle -Rchunkname datei.nw > code.c
  ```

- `nw`-Datei enthält viele Code Chunks – welchen Sie extrahieren wollen, legen Sie mit der Option `-Rchunkname` fest.
- Zusatzoption `-L`: gibt Hinweise zu Zeilennummern in `nw`-Datei aus (für den Compiler)

Beispiel aus Ulix-Makefile *)

- Auszug aus `ulix/bin-build/Makefile`:

```
TEXSRC_FILE=../ulix-book.nw
TEXSRC_MODULE_FILE=../student.nw


extract:
	notangle -L -Rulix.c $(TEXSRC_FILE) > ulix.c; true
	notangle -L -Rprintf.c $(TEXSRC_FILE) > printf.c
	notangle -Rstart.asm $(TEXSRC_FILE) > start.asm
	notangle -Rulix.ld $(TEXSRC_FILE) > ulix.ld
	notangle -L -Rmodule.c $(TEXSRC_MODULE_FILE) > module.c
	notangle -L -Rmodule.h $(TEXSRC_MODULE_FILE) > module.h
```

(Zeilennummern mit -L; nur für C-Dateien)

*) Ulix ist ein mit LP entwickeltes Betriebssystem. Verfügbar ab ca. 08/2015 auf http://www.ulixos.org/

Top-down vs. bottom-up (1)

*„In der Informatik bezeichnet man einen Entwicklungsprozess für Software als **Top-down**, wenn der Entwurf mit abstrahierten Objekten beginnt, die dann konkretisiert werden; der Prozess ist **Bottom-up**, wenn von einzelnen Detail-Aufgaben ausgegangen wird, die zur Erledigung übergeordneter Prozesse benötigt werden.“*

(Quelle: http://de.wikipedia.org/wiki/Top-down_und_Bottom-up)

Top-down vs. bottom-up (2)

- Literate Programming unterstützt beide Ansätze und auch Misch-Varianten
- Reihenfolge der Präsentation (der Code Chunks) legt fest, ob der Code top-down oder bottom-up entwickelt wird
- Beispiel: Betriebssystem

Top-down: Betriebssystem (Beispiel)

- Kernel muss Speicher und Festplatte initialisieren und dann die Shell von Platte laden und starten:

```
<<kernel>>=
  <<initialize memory>>
  <<initialize harddisk>>
  <<load shell program from disk>>
  <<run shell>>
@
```

- Wie kann man nun den Speicher initialisieren?

```
<<initialize memory>>=
  <<check available memory>>
  <<create initial page table>>
@
```


Bottom-up: Betriebssystem (Beispiel)

- Kernel muss zunächst den Speicher initialisieren; dafür braucht es eine Seitentabelle:

```
<<page table declaration>>=
  typedef struct {
    unsigned int present         :  1;  //  0
    unsigned int writeable       :  1;  //  1
    ...
    unsigned int frame_addr      : 20;  // 31..12
  } page_desc;

  typedef struct {
    page_desc pds[1024];
  } page_table;
@
```