„Schöner Wohnen mit MinMax“

Praktische Lebenshilfe für das BSP/PSP und darüber hinaus

Daniel Lüdtke

Technische Universität Berlin
Lehrstuhl für Prozessdatenverarbeitung und Robotik

dluedtke@cs.tu-berlin.de

`http://pdv.cs.tu-berlin.de/MinMax/`

WS 2007/08

Motivation

für die heutige Veranstaltung

- Für die *Werkzeugkiste* findet sich selten Zeit im regulären Curriculum.
- Selbst das Erlernen von Programmiersprachen findet meistens im Selbststudium statt.
- Man kann sich aber nur etwas aneignen, wenn man weiß, dass es existiert.
- Diese Veranstaltung möchte euch ...
  - ... einige nützliche Werkzeuge vorstellen.
  - ... Tipps für die Auswahl der richtigen Werkzeuge geben.
  - ... *kluge* Ratschläge für eine pragmatische Arbeitsweise geben.
  - ... Hinweise für eine effektive Gruppenarbeit geben.
- Alles im Kontext von MinMax.

Übersicht

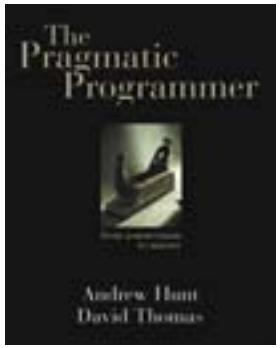
Warum das Ganze?

. . . und Inspiration

Care About Your Craft

Why spend your life developing software unless you care about doing it well?

Andrew Hunt, David Thomas:
The Pragmatic Programmer
from journeyman to master
Addison-Wesley 2000



Der Pragmatische Programmierer
Hanser Fachbuch 2003

Übersicht

Pragmatisches Programmieren

Oder: mit minimalem Einsatz ein maximales Ergebnis erzielen

Ziele dieses Praktikums:

- Verständnis für Betriebssysteme und Systemprogrammierung
- C lernen
- besser Programmieren lernen
- Kennenlernen von Werkzeugen
- den Schein bekommen!

Maximaler Ertrag durch Eigeninitiative:

Think! About Your Work

Turn off the autopilot and take control. Constantly critique and appraise your work.

Du bist nicht allein

- selten arbeitet man an einem Software-Projekt alleine
- Team-Zusammenstellung oft von außen bestimmt
- unterschiedlichste Wissensstände prallen aufeinander

Hauptproblem in vielen Projekten/Arbeitsgruppen:

- mangelnde Kommunikation
- unterschiedliche Motivation

Wie gestaltet sich die Zusammenarbeit bei MinMax I

Verschiedene Modelle zu beobachten

„Viele Köche verderben den Brei“

Vier Menschen knäulen sich um einen Bildschirm. Fortschritt ist oftmals schleppend.

„Geek And Groupies“

Es sitzt immer derselbe an der Tastatur und die restlichen Gruppenmitglieder schauen mehr oder weniger zu.

Steigerung: „Geek Only“

Aufgaben werden von einem alleine in Heimarbeit gelöst und nur noch präsentiert.

Wie gestaltet sich die Zusammenarbeit bei MinMax II

Verschiedene Modelle zu beobachten

„Divide And Conquer“

Einzelne Unteraufgaben werden auf die Gruppenmitglieder aufgeteilt.

Die beste Methode aus unserer Sicht:

Diversität und Pair-Programming

- die Aufgabe wird zweimal unabhängig gelöst
- Vorteil: am Lösungsweg sind mehrere beteiligt
- Pair-Programming: man sitzt zu zweit am Computer
- einer tippt, der andere behält den Überblick
- Rollen und Teams immer wieder tauschen
- Zusammenführung der Lösungen
  $\Rightarrow$ das Beste aus beiden Welten

Herangehensweise an eine Aufgabenstellung I

Bevor die Tastatur angefasst wird

Estimate to Avoid Surprises

Estimate before you start. You'll spot potential problems up front.

- Gehirn einschalten :-)
- Aufgabenstellung lesen
- überlegt, was ihr wie und wo implementiert
- Blatt Papier kann hilfreich sein

Estimate the Order of Your Algorithms

Get a feel for how long things are likely to take *before* you write code.

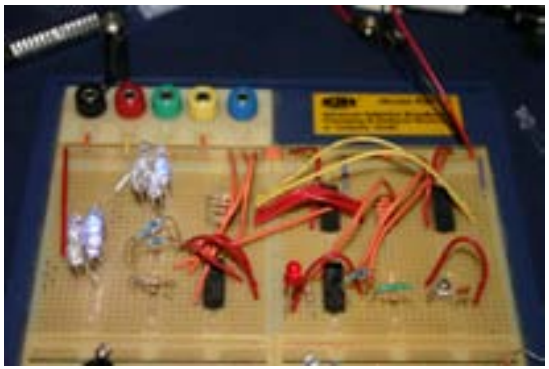
Herangehensweise an eine Aufgabenstellung II

Seht ihr den Wald vor lauter Bäumen nicht?

Prototype to Learn

Prototyping is a learning experience. Its value lies not in the code you produce, but in the lessons you learn.

- scheut euch nicht einen Prototypen zu entwickeln
- nicht zwangsläufig auf der Ziel-Hardware
- Prototyp z.B. in einer Skriptsprache
- oftmals Zeitersparnis, da debugging viel effizienter
- Prototypen sind für die virtuelle Müllhalde



Während der Entwicklung I

Entwicklung ist ein iterativer Prozess

Use Tracer Bullets to Find the Target

Tracer bullets let you home in on your target by trying things and seeing how close they land.

- Versucht immer ein lauffähiges Programm zu haben.
- Nicht den kompletten Code schreiben und dann erst testen.
- Mit jeder Iteration wird die Funktionalität erweitert.

There Are No Final Decisions

No decision is cast in stone. Instead, consider each as being written in the sand at the beach, and plan for change.

Während der Entwicklung II

„Ach, das mach ich später noch schön“

Don’t Live with Broken Windows

Fix bad designs, wrong decisions, and poor code when you see them.

- Fängt man einmal an zu schludern, hört man nicht mehr auf.
- Auch wenn es nervt, lieber die Zeit gleich investieren.
- Es zahlt sich aus!

Aber:

Remember the Big Picture

Don’t get so engrossed in the details that you forget to check what’s happening around you.

Während der Entwicklung III

Noch ein paar Ratschläge

DRY – Don't Repeat Yourself

Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.

Jedes mal, wenn ihr in Versuchung kommt *Copy-Paste* anzuwenden, läuft wahrscheinlich etwas schief. Nutzt Makros und Funktionen.

Don't Program by Coincidence

Rely only on reliable things. Beware of accidental complexity, and don't confuse a happy coincidence with a purposeful plan.

Versteht, warum eurer Programm funktioniert.
Ansonsten bereut ihr es später (wahrscheinlich).

Testen

Design to Test

Start thinking about testing before you write a line of code.

- Überlegt vor der Implementierung, wie getestet werden soll.
- Denkt an Murphy & Co.
- Guter Ansatz: zuerst den Test schreiben, dann das Programm.

Test Your Software, or Your Users Will

Test ruthlessly. Don't make your users find bugs for you.

- Vorgegebene Tests decken nicht alles ab.
- Testet mehr als nur den Trivialfall.

Fehlersuche I

Don't Panic When Debugging

Take a deep breath and THINK! about what could be causing the bug.

- Bevor ihr euren Code komplett zerstört: aktuellen Stand sichern.
- Gibt es Compiler-Warnings?
  $\Rightarrow$ die haben ihren Grund
- Fehler reproduzierbar machen
- Nicht nur die Symptome bekämpfen! Das Problem liegt wahrscheinlich woanders.
- Aufgabenstellung nochmal lesen!
- Blieb der Fehler bisher unbemerkt?

Fehlersuche II

Werkzeuge

Visualisierung der Daten

- `printf`, `kprintf` und `printk` sind eure Freunde
- grenzt die *Absturzstelle* damit ein
- gebt eure Daten aus: haben sie den erwarteten Inhalt?

Problem mit Header-Dateien

- oft liegt der Fehler nicht im angezeigten Modul
- `mx-gcc -E` führt nur den Präprozessor aus

MinMax *screen of death*

- `...` gibt die Adresse aus, bei der der Absturz auftrat
- `mxmake dump > datei.txt` disassembliert MinMax
- auf dem Stack liegt meistens auch noch die Rücksprungadresse

Fehlersuche III

Und, ach ja:

"select" Isn't Broken

It is rare to find a bug in the OS or the compiler, or even a third-party product or library. The bug is most likely in the application.

Zählt man allerdings MinMax als das *OS* . . .

Handwerkszeug

- Programmieren ist zum einen Kunst
  und zum anderen Handwerk.
- Mächtige Werkzeuge nötig, um effizient arbeiten zu können.
- Es gibt ein Basis-Set an Werkzeugen, die man gut
  beherrschen sollte.

Zu den wichtigsten Werkzeugen gehören:

- Editor
- Kommandozeilen-Tools (grep, sed, . . .)
- Debugger
- Versionsverwaltung

Editoren I

Use a Single Editor Well

The editor should be an extension of your hand; make sure your editor is configurable, extensible, and programmable.

- IDEs, wie Eclipse, erleichtern die Entwicklung in bestimmten Bereichen
- die Editor-Komponente in diesen Tools ist jedoch oft eingeschränkt
- mit einem allgemeinen Editor kann man
  - programmieren
  - Webseiten gestalten
  - Konfigurationsdateien ändern
  - Dokumente erstellen
  - ...

Editoren II

Anforderungen

Ziel sollte immer sein, die Anzahl der Tastaturanschläge und Mausbewegungen zu minimieren.

Deshalb sollte der plattformunabhängige Editor

- Syntax-Highlighting, Einrückung
- Auto-Vervollständigung
- Code-Faltung
- erweiterte Navigations- und Editiermöglichkeiten
- Makrofähigkeiten
- Einbindung externer Werkzeuge
- Rechtschreibprüfung
- Erweiterungsmöglichkeiten
- ein Command-Line und graphisches Interface

bieten.

Editoren II

Empfehlungen

Es gibt viele gute Editoren: Kate, jEdit, WinEdt, Notepad++, ...

Meine Empfehlung:

(X)Emacs und Vi(m)

- erfüllen genannte Anforderung
- sind weit verbreitet, d. h. auf vielen Systemen vorhanden (z. B. MacOS X)
- bringen sehr viele Erweiterungen vorhanden
- haben eine hohe Lernkurve, danach sehr effizient
- haben andere Tastenbelegung und Bedienkonzepte
- haben Standard-Einstellungen, die oft nicht benutzbar sind

(X)Emacs

Für die Freunde der abgewetzten Strg-Taste

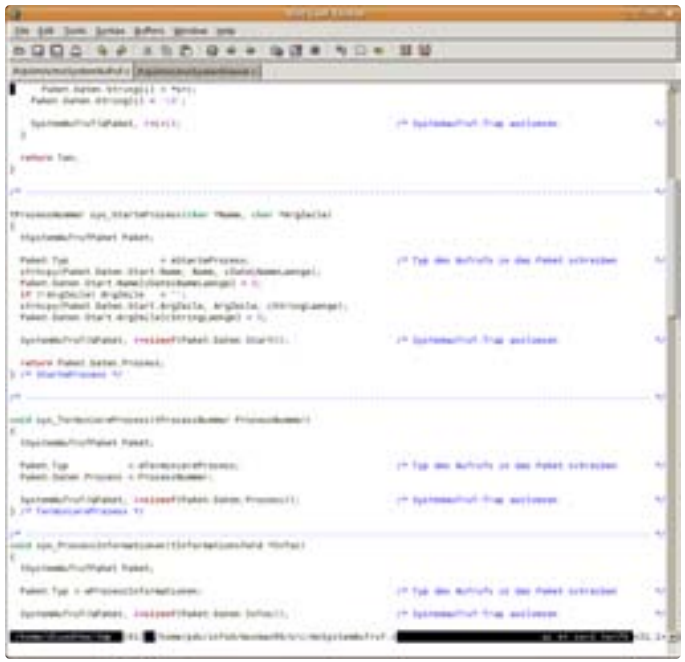
- Einstiegsschwelle nicht so hoch
- Emacs ist relativ schwergewichtig
- enthält von Hause aus sehr viele Erweiterungen (Webbrowser, . . .)
- Emacs arbeitet in verschiedenen kombinierbaren Modi (C-Mode, Syntax-Highlighting)
- in Lisp geschrieben (wer Klammern mag . . .)
- speichern: `Strg-x Strg-s`, beenden: `Strg-x Strg-c`

Emacs is a great operating system – it lacks a good editor, though. [Thomer M. Gil]

Vim

Für die Freunde der abgewetzten Esc-Taste

- Weiterentwicklung des vi
- Einstiegsschwelle höher, da man nicht sofort losschreiben kann
- Vim ist deutlich kompakter und startet schneller als Emacs
- (g)vi(m) arbeitet in verschiedenen Betriebs-Modi (Normal-, Einfüge-, Kommandozeilen-, Visual-Modus)
- dadurch stehen sehr viele Tastaturkürzel zur Navigation und Bearbeitung zur Verfügung
- speichern: `(ESC):w`,
  speichern und beenden: `(ESC):wq`



Keine Angst vor der *bösen* Kommandozeile

Use the Power of Command Shells

Use the shell when graphical user interfaces don't cut it.

GUIs sind nicht böse, aber sie können nicht alles!

- `grep` und `find` sind viel flexibler als jede andere Suche
- durch die Hintereinanderschaltung verschiedener kleiner Tools entsteht die *wahre Macht*
- mal eben in allen .c Dateien eine Funktion umbenennen: fast kein Problem
- Skripte automatisieren wiederkehrende Aufgaben

Versionsverwaltung

Warum überhaupt Versionsverwaltung?

Always Use Source Code Control

Source code control is a time machine for your work – you *can* go back.

Versionsverwaltung . . .

- ist eine Art Zeitmaschine.
- vereinfacht die Zusammenarbeit mehrerer Entwickler (oder mehrerer Computer).
- hilft bei der Organisation von Projekten.
- vereinfacht Backups.
- funktioniert nicht nur bei Source Code.

Welche Versionsverwaltung?

- Der *Markt* bietet eine Vielzahl von Versionsverwaltungssystemen an.
- Wikipedia listet über 30 unterschiedliche Systeme auf.
- Beispiele für bekannte und freie Systeme sind:
  - ··· Bazaar (Ubuntu)
  - ⋆ CVS (SourceForge)
  - ··· Git (Linux kernel)
  - ··· Monotone (Pidgin (formerly Gaim))
  - ⋆ Subversion (Apache)
- Benutzt was ihr wollt, solange ihr überhaupt etwas benutzt!
- **Empfehlung:** Subversion

Warum Subversion?

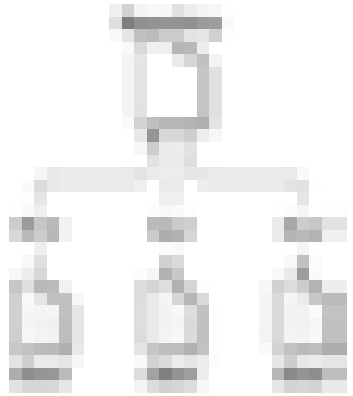
- etabliert, stabil, verbreitet (im cs-Netz installiert)
- plattformübergreifend
- Client-Server Struktur bietet sich im cs-Netz an
- einfach zu bedienen
- nicht nur für Quelltexte geeignet
- ich benutze es auch :-)

Sehr gute, vollständige Subversion Dokumentation

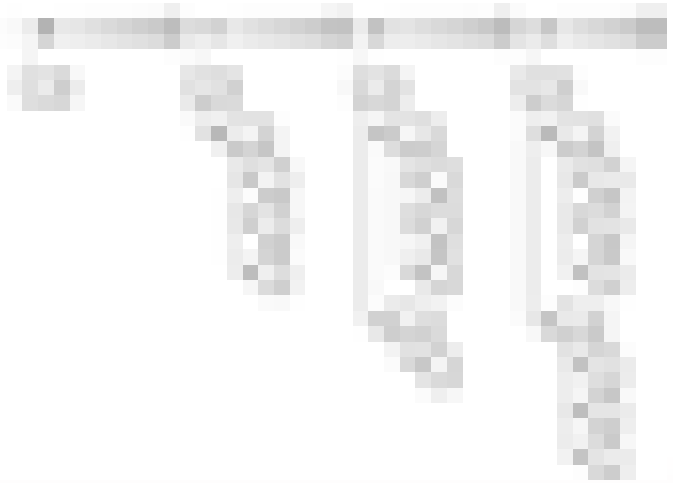
http://svnbook.red-bean.com/

Versionsverwaltung mit Subversion

die Basics



- Unterscheidung: Working Copy $\Leftrightarrow$ Repository
- Synchronisation explizit:
  `svn commit`, `svn update`



- Durch Subversion wird ein Verzeichnisbaum um eine zeitliche Dimension erweitert.
- Zeit wird durch eine Revisionsnummer repräsentiert.
- Jede *commit*-Operation auf dem Verzeichnisbaum inkrementiert die Revisionsnummer.
- Im Repository werden nur die Änderungen gespeichert.

Gemeinsames, zerstörungsfreies Arbeiten

> *All version control systems have to solve the same fundamental problem: how will the system allow users to share information, but prevent them from accidentally stepping on each other's feet? It's all too easy for users to accidentally overwrite each other's changes in the repository.* [Version Control with Subversion]

Zwei Modelle:

Lock-Modify-Unlock

Ein Objekt (Datei) wird zur Bearbeitung gesperrt.

Copy-Modify-Merge

Jeder arbeitet mit seiner eigenen Kopie. Später werden die unterschiedlichen Änderungen zusammen gebracht.

Subversion unterstützt beide Verfahren.

Beispiel für konfliktfreies Copy-Modify-Merge

- Orginalversion
- Änderungen von Alice
- Änderungen von Bob

- Nach Zusammenführung

```
/* mxBenutzer.c */
tProzessAufruf ProzessAufruf[] = {
 {"Interpreter", Interpreter},
 {"A", ProzessAB},
 {"X", ProzessAB},
 {"C", ProzessC},
 {"", NULL}};

void ProzessAB(int argc, char *argv[])
{
 while (TRUE) {
  printf("%s\n", argv[0]);
 }
}/* ProzessAB */
```

Geht nur bei Klartext-Dateien.
Lock-Modify-Unlock für Binärdateien verwenden.

Was ist zu beachten

wenn man Subversion einsetzen möchte

- in das Repository gehören nur die nicht-erzeugbaren Dateien (keine .o usw.)
- einheitliches Quelltext-Layout (Tabs, Leerzeichen, Einrückungstiefe, . . .)
- Regeln, wie: „Nur kompilierbarer Code wird eingecheckt“
- Regeln, was in die Kommentare geschrieben wird

Subversion

im cs-Netz einrichten

- Subversion lässt sich problemlos im Fachbereichs-Netz einsetzen
- Zugriff auch von zu Hause möglich (`svn+ssh://`)
- Repository ist sehr pflegeleicht: *create and forget*

Voraussetzungen:

- Unix-Gruppe ist eingerichtet
- *sauberes* Quellverzeichnis für den initialen Import
- Disk-Quota ist noch nicht ausgeschöpft

Kurzbeschreibung der Freitagsrunde

`https://wiki.freitagsrunde.org/Subversion_im_CS-Netz`

Repository aufsetzen

im cs-Netz

Verzeichnis anlegen

```
dluedtke@fiesta:~$ mkdir minmaxsvnrepos
```

Repository anlegen

```
dluedtke@fiesta:~$ svnadmin create minmaxsvnrepos
```

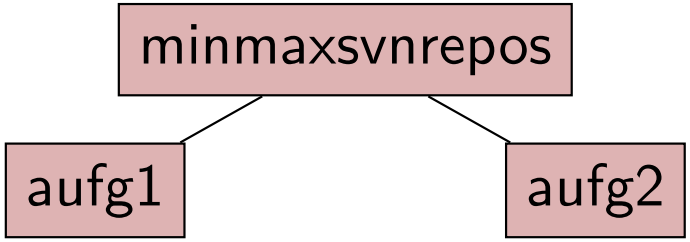
Rechte setzen

```
dluedtke@fiesta:~$ chgrp -R pdv minmaxsvnrepos
dluedtke@fiesta:~$ chmod -R g+rX minmaxsvnrepos
dluedtke@fiesta:~$ chmod -R g+u minmaxsvnrepos/db
```

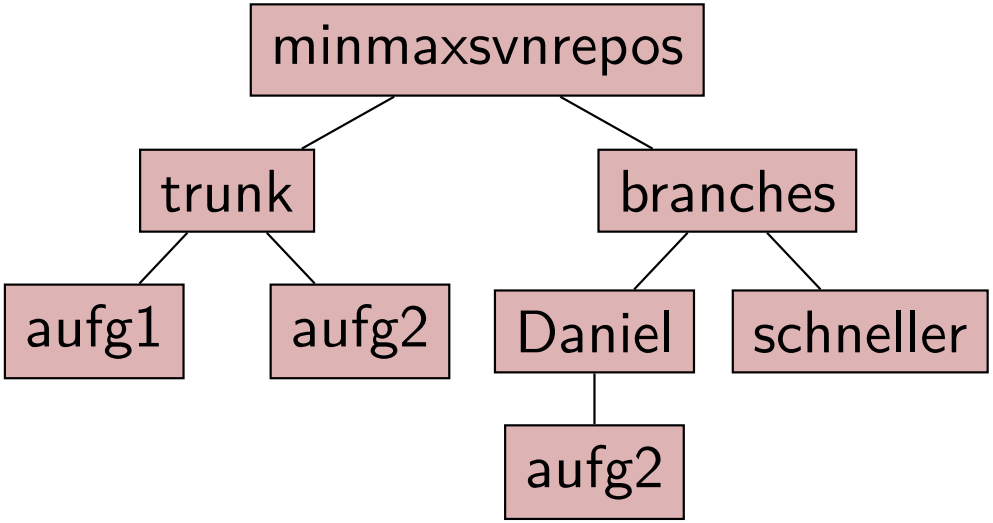
Repository-Struktur

Wie organisiere ich das Repository am Besten?

Einfachste Lösung:



Elegantere Lösung für
verästeltete Entwicklung



Initialer Quelltext-Import I

Verzeichnisstruktur für den Import anlegen

```
dluedtke@fiesta:~$ ls -R import
branches trunk
import/branches:
import/trunk: aufg1 aufg2
import/trunk/aufg1: Makefile mxLED.c mxSeriell.c
import/trunk/aufg2: Makefile mxEingabePuffer.c
mxSystemDienst.c MinMax.c mxEingabePuffer.h
mxUnterbrechung.c mxAbbruch.c mxInterpreter.c
mxUnterbrechung.h mxCPUVerwaltung.c mxSeriell.c
mxUnterbrechungAsm.S mxCPUVerwaltung.h
mxSystemAufruf.c mxDienstStruk.h mxSystemAufruf.h
```

Vorher noch ein `mxmake distclean` spendieren!

Initialer Quelltext-Import II

Importieren

```
dluedtke@fiesta:~/import$
svn import file:///home/dluedtke/minmaxsvnrepos -m
"Initialer Quelltext-Import"
Adding trunk
Adding trunk/aufg1
Adding trunk/aufg1/mxSeriell.c
Adding trunk/aufg1/mxLED.c
Adding trunk/aufg1/Makefile
Adding trunk/aufg2
Adding trunk/aufg2/mxSystemDienst.c
...
Adding branches
Committed revision 1.
```

Jetzt *kann* das `import` Verzeichnis gelöscht werden.

Initialer Quelltext-Import III

Erster checkout

```
dluedtke@fiesta:~$
svn checkout file:///home/dluedtke/minmaxsvnrepos/trunk
minmax

A minmax/aufg1
A minmax/aufg1/mxSeriell.c
A minmax/aufg1/mxLED.c
...
A minmax/aufg2/Makefile
Checked out revision 1.
```

`minmax` ist jetzt eine Arbeitskopie
(erkennbar an `.svn` Verzeichnissen).

Die wichtigsten Befehle I

für das Arbeiten mit Subversion

`svn help`

Selbsterklärend! Eine Hilfe die auch hilft.

`svn update`

Seine Arbeitskopie auf den neusten Stand bringen. `svn update datei` erneuert nur die betreffende Datei. Lokale Änderungen bleiben erhalten.

`svn commit`

Überträgt Änderungen zum Repository. Es muss ein Kommentar angegeben werden; bitte auch nutzen!

`svn status`

Zeigt den Status an: lokale Änderungen, Konflikte, nicht-versionierte Dateien. Mit `-u`:, ob sich was im Repository geändert hat.

Die wichtigsten Befehle II

für das Arbeiten mit Subversion

`svn cp`

Kopiert Dateien oder Verzeichnisse. Die Versiongeschichte bleibt dabei erhalten.

`svn move|rename`

Verschiebt bzw. benennt Dateien oder Verzeichnisse um.

`svn delete`

...

`svn diff`

Zeigt Änderungen (seit dem letzten commit) an.

`svn log`

Zeigt die Logmeldungen an. Deshalb commit-Kommentare!

Die (mehr oder weniger) wichtigsten Befehle III

für das Arbeiten mit Subversion

`svn info`

Mehr oder weniger sinnvolle Informationen.

`svn blame`

Wer hat was, wann und wo *verbrochen*.

Fix the Problem, Not the Blame.

It doesn't really matter whether the bug is your fault or someone else's – it is still your problem, and it still needs to be fixed.

`svn resolved`

Sollte svn nicht in der Lage sein, einen Konflikt selbst zu lösen, muss man selbst Hand anlegen. Mit diesem Befehl teilt man svn mit, dass der Konflikt gelöst wurde.

Die (mehr oder weniger) wichtigsten Befehle IV

für das Arbeiten mit Subversion

`svn revert`

Stellt ursprüngliche Version wieder her.

`svn lock|unlock`

Damit können Dateien exklusiv bearbeitet werden (sparsam einsetzen). Ein Lock wird durch einen commit automatisch wieder freigegeben.

`svn prop*`

- Zu jeder Datei oder jedem Verzeichnis können Metadaten gespeichert werden: *Properties*.
- Schlüssel-Wert Paare
- Wert kann Text, aber auch binären Inhalt haben (z.B. Thumbnails von Bildern)
- Hilfreich: `svn:ignore` für Verzeichnisse (*.o *.mxe . . .)

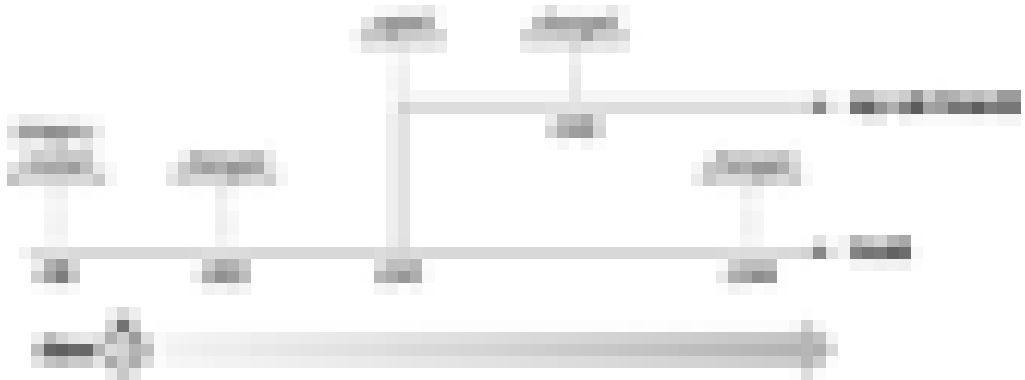
Verästeltete Entwicklung I

Branching And Merging

Folgendes Szenario:

- Vierer-Gruppe verfährt so: Aufgaben werden jeweils als Zweier-Teams parallel gelöst; am Ende werden die Lösungen zu einer *Über*-Lösung zusammengefügt.
- Im `trunk` wird die *Über*-Lösung gehalten.
- Jedes Team legt sich einen eigenen Branch an, um dort zu entwickeln.

Ein Branch ist nichts anderes als eine Kopie.



Verästeltete Entwicklung II

Start Aufgabe 3 von Team A

In der Arbeitskopie (zur Zeit `trunk`) wird A3 vorbereitet

```
dluedtke@fiesta:~/minmax$ svn mkdir aufg3
dluedtke@fiesta:~/minmax$ svn copy aufg2/Makefile aufg3
Makefile anpassen
dluedtke@fiesta:~/minmax$ svn commit -m "Start Aufgabe 3"
```

Branch (im Repository) anlegen

```
dluedtke@fiesta:~/minmax$ svn copy aufg3
file:///home/dluedtke/minmaxsvnrepos/branches/TeamA_aufg3
-m "Branch für Team A, A3 angelegt"
Committed revision 23.
```

Umschalten auf den Branch

```
dluedtke@fiesta:~/minmax$ cd aufg3
dluedtke@fiesta:~/minmax/aufg3$ svn switch
file:///home/dluedtke/minmaxsvnrepos/branches/TeamA_aufg3
```

Verästelte Entwicklung III

Zurück in den Trunk

Team A möchte seine Lösung in den Trunk übertragen:

Arbeitskopie zurück auf den Trunk umstellen

```
dluedtke@fiesta:~/minmax/aufg3$ svn switch
file:///home/dluedtke/minmaxsvnrepos/trunk/aufg3
```

Nun alle Änderungen aus dem Branch lokal übertragen

```
dluedtke@fiesta:~/minmax/aufg3$ svn merge -r 23:45
file:///home/dluedtke/minmaxsvnrepos/branches/TeamA_aufg3
```

Jetzt können auch die Ergebnisse von Team B eingepflegt werden.

Nach Lösung evtl. Konflikte: Ergebnis übertragen

```
dluedtke@fiesta:~/minmax/aufg3$ svn commit -m
"merged: r 23:45 aus Branch TeamA_aufg3 in den Trunk"
```

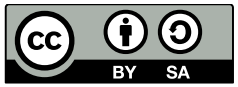
Übersicht

Zusammenfassung

- Nutzt das Praktikum, um eure Fähigkeiten zu verbessern!
- Überlasst beim Programmieren nichts dem Zufall!
- Macht euch mit einem mächtigen Editor vertraut!
- Keine Angst vor der Kommandozeile!
- Nutzt Versionsverwaltung!

Bedenkt: alles Gesagte war rein subjektiv.
Bildet euch eure eigene Meinung!

Lizenz



Bildnachweis I

 by Bogdan Costea CC PD

 © Addison-Wesley

 © Carl Hanser Verlag

 by Gaetan Lee CC BY

 by Unhindered by Talent CC BY-SA

 by barnoid CC BY-SA

 by Hiddenloop CC BY

 by C.P.Storm CC BY

 by jurvetson CC BY

 by Meta Mago CC BY

 by johnmarkos CC BY-SA

 Emacs screenshot

 Vim screenshot

Bildnachweis II


GNOME Terminal screenshot 


© CollabNet


from *Version Control with Subversion* 


from *Version Control with Subversion* 


from *Version Control with Subversion*