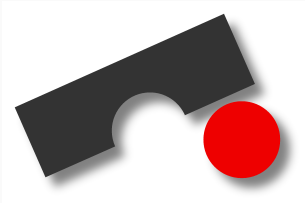
Softwarewerkzeuge

oder **Praxis der Softwareentwicklung**

Prof. Dr.-Ing. Andreas Zeller



Lehrerweiterbildung Informatik, Schloß Dagstuhl, 29.11.2001







Übersicht

Ausgewählte Themen und Werkzeuge der Softwareentwicklung:

- Konfigurationsmanagement mit CVS
- Automatisches Testen mit JUnit
- Extreme Programming







Konfigurationsmanagement

Ein großes Software-Produkt besteht aus

- Tausenden von Komponenten,
- die von Hunderten oder gar Tausenden Personen entwickelt und gewartet werden,
- die oftmals auch noch auf viele Orte verteilt sind,

und an all diesen Komponenten werden von all diesen Personen *Änderungen* vorgenommen.

Die Aufgabe, solche Änderungen zu organisieren und zu kontrollieren, heißt *Software-Konfigurationsmanagement*.







Konfigurationsmanagement: Ursprung

Konfigurationsmanagement: ursprünglich von der US-Raumfahrtindustrie in den 50er Jahren eingeführt

Problem zu dieser Zeit: Raumfahrzeuge unterlagen während ihrer Entwicklung zahlreichen undokumentierten *Änderungen.*

Erschwerend: Raumfahrzeuge wurden im Test normalerweise *vernichtet*

Folge: Nach einem erfolgreichen Test waren Hersteller nicht in der Lage waren, eine Serienfertigung aufzunehmen oder auch nur einen Nachbau durchzuführen.

Konfigurationsmanagement soll solchen *Informationsverlust* verhindern.







Ziele des Konfigurationsmanagements

Rekonstruktion: *Konfigurationen* müssen zu jedem Zeitpunkt wiederhergestellt werden können; eine Konfiguration ist dabei eine Menge von Software-Komponenten in bestimmten Versionen.

Koordination: Es muß sichergestellt werden, daß Änderungen von Entwicklern nicht versehentlich verlorengehen. Dies bedeutet insbesondere das Auflösen von *Konflikten.*

Identifikation: Es muß stets möglich sein, einzelne Versionen und Komponenten eindeutig zu identifizieren, um die jeweils angewandten Änderungen erkennen zu können.







Software-Systeme verwalten mit CVS

CVS = Concurrent Versions System

Weitverbreitetes Werkzeug zum Konfigurationsmanagement

Standard-Werkzeug in der Open-Source-Szene

SourceForge: unterhält CVS-Archive für >25.000 Projekte







CVS-Archive

CVS unterhält ein *Archiv*, in dem die *Versionen* eines Systems aufbewahrt werden.

Versionen werden *platzsparend* gespeichert, und zwar als Urfassung + n Änderungen (nach Datum sortiert).

Will man etwa auf die Fassung vom 1. Februar (1. Juni) zugreifen, nimmt CVS die Urfassung und wendet darauf sämtliche Änderungen bis zum 1. Februar (1. Juni) an.

Auf diese Weise kann der Zustand zu jedem Zeitpunkt wiederhergestellt werden.

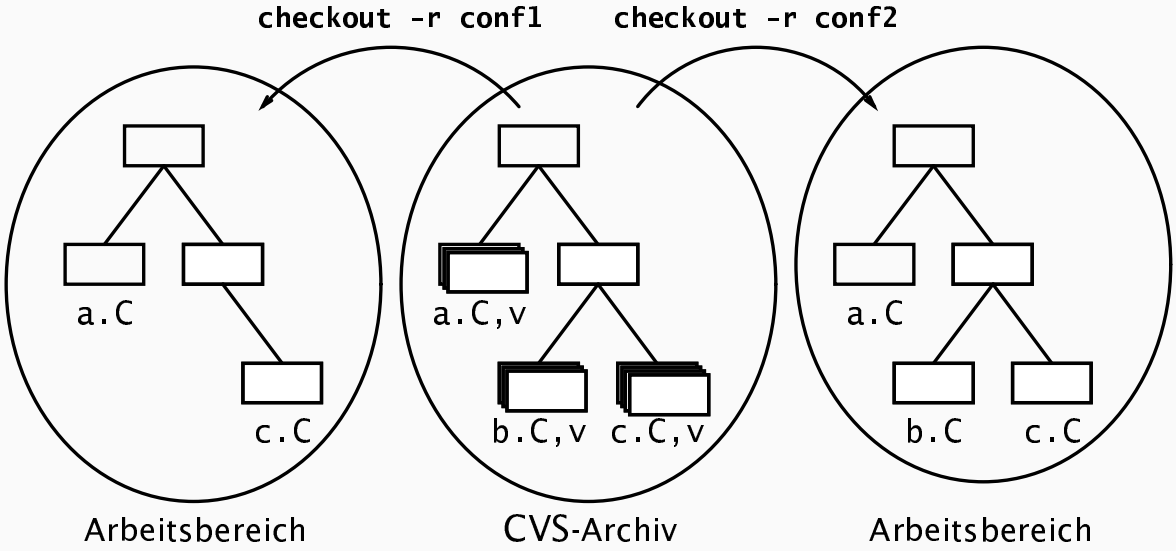






Projekt einrichten: Checkout

Mit einem *check out* wird die letzte Revision in einen *Arbeitsbereich* kopiert:



Projekt einrichten: Checkout (2)

Beispiel: Anke holt sich die aktuelle Fassung des Duden-Projekts:

```
anke$ cvs checkout duden
cvs checkout: Updating duden
U duden/Makefile
U duden/duden.h
U duden/grammatik.C
U duden/woerter.C
anke$ _
```

Mit der Option -D kann Anke beliebige frühere Zustände wiederherstellen:

```
anke$ cvs checkout -D "2001-11-29 14:30" duden
```







Änderungen propagieren: Commit

Anke hat die Datei duden.h geändert und möchte diese Änderung öffentlich machen. Hierbei kann sie die Änderung *begründen*

Möglichkeit 1:

```
anke$ cvs commit duden.h
(Aufruf des Texteditors)
```

Möglichkeit 2:

```
anke$ cvs commit -m "Neu: NUMMERIERUNG" duden.h
Checking in duden.h;
/usr/share/CVS/duden/duden.h,v  <--  duden.h
new revision: 1.2; previous revision: 1.1
done
anke$ _
```

Änderungen propagieren: Commit (2)

Wird nichts angegeben, werden Änderungen im gesamten Verzeichnis propagiert:

```
anke$ cvs commit -m "Neue Deklination"
cvs commit: Examining .
cvs commit: Committing .
Checking in grammatik.C;
/usr/share/CVS/duden/grammatik.C,v <-- grammatik.C
new revision: 1.2; previous revision: 1.1
done
Checking in woerter.C;
/usr/share/CVS/duden/woerter.C,v <-- woerter.C
new revision: 1.4; previous revision: 1.3
done
anke$ _
```







Änderungen nachvollziehen

Mit `cvs log` kann man die Entstehungsgeschichte einer Datei verfolgen:

```
$ cvs log ddd.C
revision 1.632
date: 2001/07/31 16:10:05; author: zeller; state: Exp;lines: +3 -3
Fix: use XawInitializeWidgetSet() to setup Xaw converters;
     Xaw3d has no XawInitializeDefaultConverters().
----------------------------
revision 1.631
date: 2001/07/31 16:00:09; author: zeller; state: Exp;lines: +10 -0
Fix: don't have Xaw converters override our own
----------------------------
revision 1.630
date: 2001/07/30 22:18:27; author: zeller; state: Exp;lines: +18 -6
New: Options are automatically saved upon exiting DDD (can be turned off)
----------------------------
revision 1.629
date: 2001/07/30 21:10:59; author: zeller; state: Exp;lines: +12 -5
New: The Tip of the Day comes with an option to turn it off.
```

. . .

Änderungen nachvollziehen (2)

cvs diff zeigt die Unterschiede zwischen zwei Revisionen:

```
$ cvs diff -r1.631 -r1.632 ddd.C
RCS file: /cvsroot/ddd/ddd/ddd/ddd.C,v
retrieving revision 1.631
retrieving revision 1.632
diff -r1.631 -r1.632
2140,2142c2140,2142
<     // Register Xaw Converters.  This is done before installing our
<     // own converters.
<     XawInitializeDefaultConverters();
---
>     // Initialize Xaw widget set, registering the Xaw Converters.
>     // This is done before installing our own converters.
>     XawInitializeWidgetSet();
```

Konfigurationen benennen

Will Anke die aktuelle Konfiguration *benennen*, (z.B. weil sie an einen Kunden ausgeliefert wird), benutzt sie `tag`:

```
anke$ cvs tag duden1-1
cvs tag: Tagging .
T Makefile
T duden.h
T grammatik.C
T woerter.C
anke$ _
```

Die aktuelle Konfiguration kann nun jederzeit unter diesem Namen *rekonstruiert* werden:

```
anke$ cvs checkout -r duden1-1 duden
```







Konfigurationen benennen (2)

Alternativen zur Rekonstruktion:

- *Der Kunde soll gefälligst die neueste Software benutzen!*
- *Ich weiß auch nicht, warum es gestern noch lief. . .*







Versionen identifizieren

Bei jedem `commit` werden automatisch fortlaufende *Versionsnummern* vergeben, mit denen man später die Konfiguration wiederherstellen kann.

Diese Versionsnummern lassen sich auch in die Dokumente aufnehmen – und zwar mit Hilfe von *Schlüsselwörtern.*

In CVS sind Schlüsselwörter Bezeichner, die in „`$...$`“ eingeschlossen sind; sie werden beim `checkout` automatisch expandiert bzw. beim `update` auf den neuesten Stand gebracht.

Das Schlüsselwort `$Id$` expandiert z.B. zu einem Standard-Dateikopf:

```
$Id: cvsdriver.c,v 1.3 2001/11/21 18:38:08 zeller Exp $
```







Versionen identifizieren (2)

Die Schlüsselwörter lassen sich auch in den Programmtext übernehmen – z.B. als

```
static char version_string[] =
"$Id$";
```

Dieser Code wird beim nächsten checkout zu

```
static char version_string[] =
"$Id: cvsdriver.c,v 1.3 2001/11/21 18:38:08 zeller Exp $
```

expandiert.

Alternative zur Identifikation: *Raten!*

Dateien hinzufügen

Zu den Änderungen, die Entwickler vornehmen, gehören auch das *Hinzufügen* von neuen Dateien.

```
anke$ cvs add ortho.C
cvs add: scheduling file ‘ortho.C’ for addition
cvs add: use ’cvs commit’ to add this file
    permanently
anke$ _
```







Dateien hinzufügen (2)

Beim nächsten `commit` wird `ortho.C` ins CVS-Archiv aufgenommen.

```
anke$ cvs commit -m "Neu: ortho.C - Orthographie"
cvs commit: Examining .
cvs commit: Committing .
RCS file: /usr/share/CVS/duden/ortho.C,v
done
Checking in ortho.C;
/usr/share/CVS/duden/ortho.C,v  <--  ortho.C
initial revision: 1.1
done
anke$ _
```

Dateien löschen

Analog zu cvs add löscht cvs remove Dateien aus dem System:

```
anke$ rm ortho.C
anke$ cvs remove ortho.C
cvs remove: scheduling ‘ortho2.C’ for removal
cvs remove: use ’cvs commit’
to remove this file permanently
anke$ cvs commit -m "ortho.C geloescht"
cvs commit: Examining .
cvs commit: Committing .
Removing ortho.C;
/usr/share/CVS/duden/ortho.C,v  <--  ortho.C
new revision: delete; previous revision: 1.1
done
anke$ _
```

Arbeit beenden

Wird das Arbeitsverzeichnis nicht mehr benötigt, kann es mit `cvs release` gelöscht werden.

```
anke$ cd ..
anke$ cvs release -d duden
You have [0] altered files in this repository.
Are you sure you want to release
directory ‘duden’: yes
anke$ _
```

`cvs release` prüft dabei, ob auch tatsächlich keine offenen Änderungen ausstehen.







Paralleles Arbeiten

Problem: Entwickler arbeiten gleichzeitig auf derselben Datei (bzw. einer Kopie in ihrem Arbeitsbereich).

Wessen Änderungen werden übernommen und wie?

Werkzeuge zum Konfigurationsmanagement kennen zwei Mechanismen:

- Sperren (pessimistisch)
- Integration (optimistisch)







Sperren

Ist in CVS eine Datei (oder ein System) mit `cvs watch` markiert worden, unterhält CVS auf dieser Datei eine *Sperre*:

- Nach einem Checkout ist die Datei nur zum Lesen geöffnet
- Um sie zum Schreiben zu öffnen, muß der Benutzer explizit `cvs edit` eingeben:

  ```
  anke$ cvs edit duden.h
  ```

- Will Petra nun die Datei ebenfalls bearbeiten,

  ```
  petra$ cvs edit duden.h
  ```

  so werden Anke und sie automatisch benachrichtigt – so können sie sich über das weitere Vorgehen verständigen.

Sperren (2)

Andere Konfigurationssysteme als CVS sind da weit restriktiver:

In RCS etwa ist es nicht möglich, eine Datei zu bearbeiten, die bereits von einem anderen bearbeitet wird.

Solche restriktiven Sperren führen zu Problemen bei

- *Hot spots* – Teilen des Systems, die häufig bearbeitet werden
- *Langen Checkouts* – Sperren, die über lange Zeit bestehen







Integration

Die Alternative zu Sperren heißt *Integration:*

Bei jedem `commit` prüft CVS, ob die geänderten Dateien im CVS-Archiv unverändert sind:

- Wenn ja, werden die Änderungen übernommen
- Wenn nein, muß der Benutzer zuvor die neuen Änderungen in seinen Arbeitsbereich *integrieren*.







Integration (2)

`cvs update` überträgt Änderungen aus dem CVS-Archiv in das Arbeitsverzeichnis. Seien

- D die ursprüngliche Fassung der Datei,
- D'_1 die neue Fassung im CVS-Archiv und
- D'_2 die neue Fassung im Arbeitsbereich.

Folgende Fälle können beim `update` auftreten:

1. $D = D'_1 = D'_2$ $\Rightarrow$ Nichts geschieht.
2. $D = D'_1 \neq D'_2$ $\Rightarrow$ D'_2 wird ins CVS-Archiv übernommen.
3. $D = D'_2 \neq D'_1$ $\Rightarrow$ D'_1 wird in den Arbeitsbereich übernommen.
4. $D \neq D'_1 \neq D'_2$ $\Rightarrow$ Die Änderungen werden integriert.

Integration (3)

Ist eine Datei sowohl im CVS-Archiv als auch im Arbeitsbereich geändert worden, so müssen die Änderungen *in dieser Datei* integriert werden.

CVS geht dabei so vor:

- Sind die Änderungen (= Einfügen, Ändern, oder Löschen von Zeilen) mehr als 5 Zeilen voneinander entfernt, werden beide angewandt.
- Ansonsten tritt ein *Konflikt* auf: Beide Änderungen werden (durch Sonderzeichen gekennzeichnet) Teil der Datei.







Ein Konflikt

Beispiel: Anke und Petra haben beide dieselbe Zeile in `duden.h` geändert; Anke hat ihre Änderungen bereits wieder eingespielt.

Nach Petras `update` markiert CVS den Konflikt:

```
<<<<<<< duden.h
Ankes neuer Text
=======
Petras neuer Text
>>>>>>> 1.1.2.1
```

Vor dem nächsten `commit` muß Petra diesen Konflikt auflösen.

Alternative zur Koordination: *Händisches Umkopieren!*

Arbeitsweise mit CVS

1. `cvs checkout`, um eine Kopie des Projekts anzulegen
2. `cvs update`, um aktuelle Änderungen zu übertragen
3. `cvs commit`, um eigene Änderungen zu veröffentlichen
4. `cvs release`, um die eigene Kopie zu löschen.

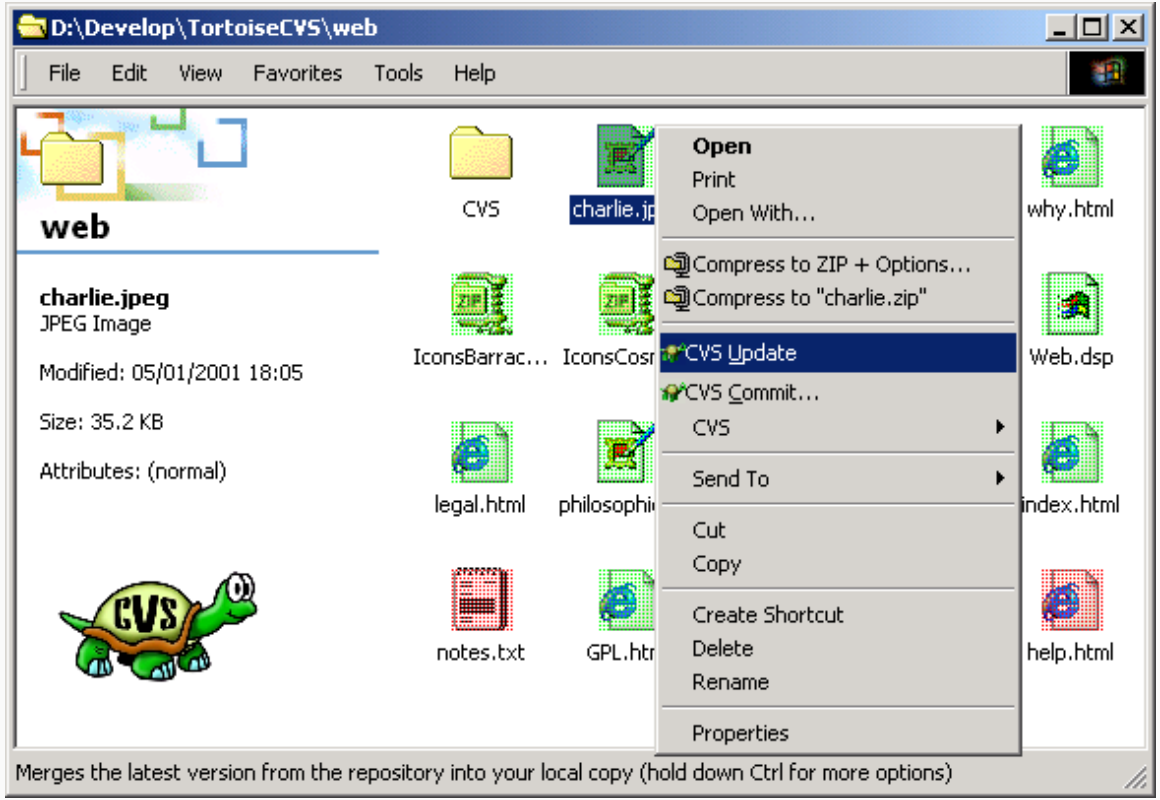
In der Praxis: update / commit / update / commit . . .



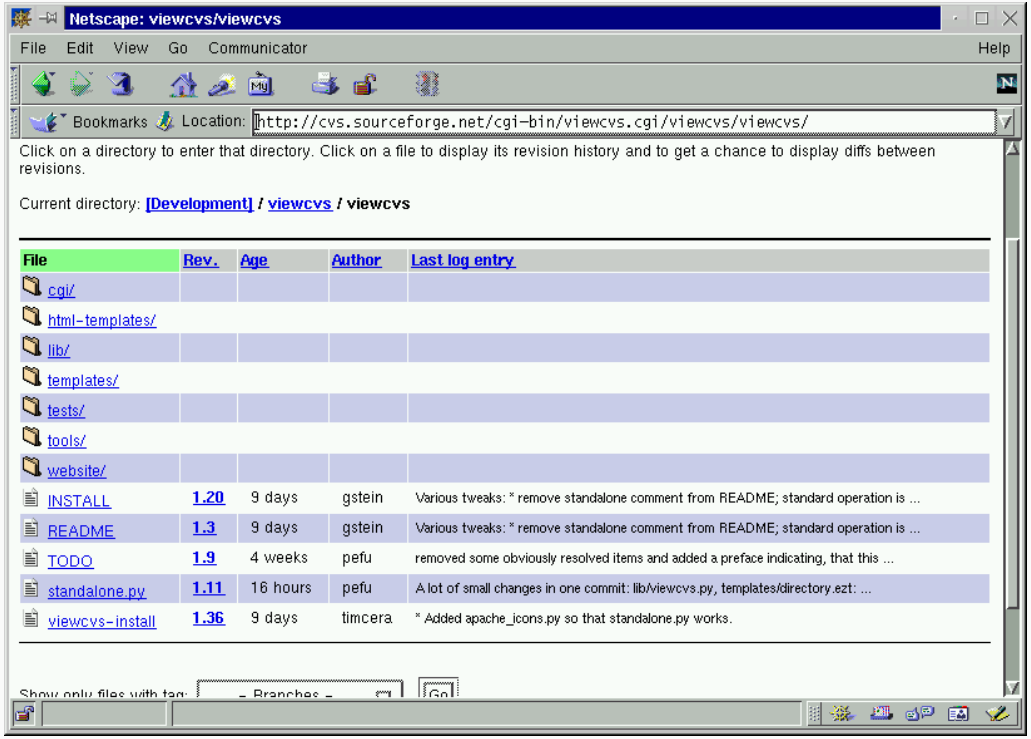




TortoiseCVS



ViewCVS


Netscape: viewcvs/viewcvs
File Edit View Go Communicator Help
Bookmarks Location: http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/viewcvs/viewcvs/
Click on a directory to enter that directory. Click on a file to display its revision history and to get a chance to display diffs between revisions.
Current directory: [Development] / viewcvs / viewcvs
File Rev. Age Author Last log entry
cgi/
html-templates/
lib/
templates/
tests/
tools/
website/
INSTALL 1.20 9 days gstein Various tweaks: * remove standalone comment from README; standard operation is ...
README 1.3 9 days gstein Various tweaks: * remove standalone comment from README; standard operation is ...
TODO 1.9 4 weeks pefu removed some obviously resolved items and added a preface indicating, that this ...
standalone.py 1.11 16 hours pefu A lot of small changes in one commit: lib/viewcvs.py, templates/directory.ezt: ...
viewcvs-install 1.36 9 days timcera * Added apache_icons.py so that standalone.py works.
Show only files with tag: - Branches - Go

Zusammenfassung

- In das CVS-Archiv können Revisionen hinein- und wieder herauskopiert werden (*commit* bzw. *update*).
- Jede Änderung wird vom Entwickler *begründet*; das daraus gebildete *Änderungsprotokoll* dokumentiert die Entstehungsgeschichte einer Komponente.
- In einem CVS-Archiv werden nur die *Unterschiede* zwischen Revisionen gespeichert, was viel Platz spart.
- Das CVS-System ermöglicht die Versionierung von kompletten *Dateibäumen*, indem CVS auch das *Anlegen* und *Löschen* von Dateien berücksichtigt.
- CVS realisiert eine *optimistische Kooperationsstrategie*, die gleichzeitige Änderungen mehrerer Entwickler integriert.