Christian-Albrechts-Universität zu Kiel
Institut für Informatik und Praktische Mathematik

Prof. Dr. W.-P. de Roever
Martin Steffen

Softwarepraktikum: Enigma

Sommersemester 2003 **Handout 2** 7. April 2003

<u>Handout 2:</u> Technische Rahmenbedingen: CVS, Makefiles, Spielregeln

Ausgabetermin: 7. April 2003

Java

Um die Abgabe des Quellcodes zu vereinheitlichen und damit die Betreuung und Inspektion der abgelieferten Arbeit von unnotwendigen Komplikationen zu entlasten, werden wir unter "*Concurrent Version Control*" arbeiten, ein verteiltes Versionskontroll- und Managementsystem.[1] Ferner soll jeder Gruppe ein (einfaches) Makefile bereitstellen, damit einfache Tests (wie auf Kompilationsfähigkeit und ähnliches) leicht von uns durchgeführt werden können.

Die wichtigsten Dinge sind auf diesem Handout zusammengefaßt. Falls Sie die Hinweise auf diesem Handout nicht verstehen, bitte fragen Sie nach, wir helfen Ihnen. Weitere Informationen und Links über unsere Webseite.

Verteiltes Arbeiten unter CVS

Jede Gruppe bekommt einen account am cvs-server der Informatik:

`cvs.informatik.uni-kiel.de`

Dort wird sich, für jede Gruppe separat, das jeweilige *Repositorium* befinden, also die verwalteten Quelldateien. Die accounts sind durchnumeriert und heißen

`prakt01`, `prakt02` ...

Jede Gruppe kann nur auf ihren eigenen account zugreifen, wir als Betreuer haben Zugriff auf alle accounts. Dabei gilt —insbesondere für die Abnahmen: alles was außerhalb des Repositorium existiert, existiert nicht wirklich für uns ...

[1] Neben der Entlastung für uns kann sich CVS beim verteilten Arbeiten in der 4er Gruppe als sehr nützlich zeigen. Das hängt davon ab, wie man intern in der Gruppe die Arbeit organisiert, d.h., ob man eher zu "gemeinschaftlichen Koding-Sitzungen" neigt oder nach dem dem Prinzip "getrennt marschieren, gemeinsam zuschlagen" vorgeht, also die Aufgabe so struktiert und entwirft, daß an verschiedenen Teilen getrennt gearbeitet werden kann und nur die Integration gemeinschaftlich gemacht wird. Vor allem im letzteren Arbeitsmodell ist CVS sehr hilfreich.

Technische Vorraussetzungen

Man benötigt zwei Dinge, um von Ferne auf das jeweilige Gruppenrepositorium zyzugreifen:

1. *ssh* (secure shell) für den sicheren Zugriff und
2. *cvs* als Versionskontroll- und Managementwerkzeug.

Diese Tools sind im Uni-Netz vorhanden. Wer daheim (oder von einem sonstigen account) arbeiten will, muß dafür sorgen, daß sie installiert sind. Sie sind kostenlos für Linux/Unix/Windows/MacOS verfügbar.

ssh-Zugriff

Für den ersten Punkt, dem Fernzugriff unter *ssh*, geht man wie folgt vor. Man maile uns den (öffentlichen Teil) seines *ssh*-Schlüssels. Wenn man bereits einen hat, liegt er typischerweise unter `~/.ssh/id_dsa.pub`. Wer noch keinen hat, muß ihn sich generieren und zwar mittels *ssh-keygen* am besten `ssh-keygen -tdsa` für einen ssh-2 Schlüssel.

Angabe des cvs-Servers

Für das korrekte Funktionieren muß man cvs mitteilen, wo sich das Repositorium befindet. Dies kann man wie folgt erreichen (bash-Syntax), wobei `prakt<x>` für die jeweilige Gruppe angepaßt werden muß:

```
export CVSROOT=:ext:prakt<x>@cvs.informatik.uni-kiel.de:/cvs/prakt<x>
export CVS_RSH=ssh
export CVSEDITOR=emacs
```

Ebenso ist `$WORKDIR` ein Platzhalter (!) für das Verzeichnis ist, in dem man arbeiten wird. Die drei Zeilen sind zum korrekten Steuern von cvs gedacht.[2] Anstelle *emacs* kann man auch *emacsclient* oder den Editor seiner Wahl nehmen. *emacsclient* sollte man dann nehmen, wenn man ohnehin emacs verwendet und beim starten des emacs' ein `(server-start)` ausgeführt wird.

Beispiel

Das Beispiel verwendet `$WORKDIR` als Bezeichnung für das *Arbeitsverzeichnis* Als `$WORKDIR` kann sich jeder selbst passend wählen, wo er das Arbeitsverzeichnis haben will. Es spricht nichts dageben, daß man sich auch mehrere Arbeitsverzeichnisse verschafft, zum Beispiel jeder der Arbeitsgruppe eines, oder zum Beispiel eines auf seinem Uni-Account und eines daheim. Mit der Zahl der ausgecheckten Arbeitsverzeichnisse steigt natürlich die Möglichkeiten der Verwirrung. In der Regel kommt man mit einem Arbeitsverzeichnis pro Person und pro "Arbeitsplatz" aus.

- **Auschecken:** So bekommt man zum ersten Mal seine Arbeitskopie des Projektes:

[2] Man kann cvs auch anders steuern, z.B. über `.cvsrc` oder die Option `-d`, bei Bedarf bitte selbst im Manual nachschlagen oder nachfragen.

```
mkdir $WORKDIR
cd $WORKDIR
cvs checkout enigma
```

Die Variable `$WORKDIR` ist hier nur zur Illustration gewählt, cvs kennt sie nicht. Falls man konkret `$WORKDIR` = `~/Projekt` wählt, heißen die Befehle:

```
cd ~
mkdir Projekt
cd Projekt
cvs checkout enigma
```

Das liefert *zwei Unterverzeichnisse,* `enigma<x>` und `CVSROOT`. Ab Beispiel von `prakt15` ergibt das:

```
> cvs checkout enigma
     Enter passphrase for key '/home/<user>/.ssh/id_dsa': ....
     cvs.exe server: Updating CVSROOT...
     cvs.exe server: Updating enigma15
```

Im Folgenden bezeichnet `$WORKDIR` immer das frei wählbare Arbeitsverzeichnis. Die zwei Unterverzeichnisse dienen unterschiedlichen Zwecken

1. `engima<x>`: Dies ist das Unterverzeichnis, innerhalb dessen die Programmentwicklung stattfinden soll. Das bedeutet, falls sie Ihren *Java*-Quellcode in baumartiger Weise strukturieren, d.h. die Paketstrukturierungsmöglichkeiten von *Java* ausnuzten, soll der Quellcode vollständig innerhalb des Verzeichnisses `enigma<x>` liegen! Das `enigma<x>` ist damit gleichzeitig auch der Name des *Java-Paketes* (*package*) ihrer Gruppe.
2. `CVSROOT`: Zentrale CVS-Steuerdatei. Für Sie interessant ist nur die Datei

   `CVSROOT/authorized_keys`.

   Dort wird ssh-Zugriff auf den jeweiligen account mittels der öffentlichen Schlüssel gewährt. Wir tragen einen von jeder Gruppe dort ein, die anderen Teilnehmer können sie entsprechend dem Muster zusätzlich hinzufügen. Was die anderen Dateien betrifft: am Besten: Finger weg! Ebenso sollten in dieses Verzeichnis keine weiteren Dateien oder Unterverzeichnisse hinzugefügt werde, es dient alleine zur Verwaltung.

- **Zurückspeichern:** Wenn man mit den Änderungen durch ist, kann man den gesamten Verzeichnisbaum zurückspeichern:[3]

```
cd $WORKDIR
cvs commit
```

Danach wird man aufgefordert, einen Kommentar bezüglich seiner Änderungen anzugeben. Falls der Kommentar nur kurz ist, kann man schneller auch

[3] Beachte den Wechsel in das Unterverzeichnis.

```
cvs commit -m"Anmerkungen zu den Änderungen"
```

verwenden. Macht man cd \$WORKDIR/enigma<x>; cvs commit -m"Bemerkung"+, dann wird nur das enigma-Unterverzeichnis zurückgespeichert.

- **Updaten:** Den Baum seines Arbeitsverzeichnisses auf den neuesten Stand bringen, geht so

```
cd $WORKDIR
cvs update
```

- **Neue Dateien + Verzeichnisse:** Das geht mit

```
cvs add [filename]
```

  entfernen mit

```
cvs remove [filename],
```

  Anstelle eines Dateinamens kann man auch ein Verzeichnis hinzufügen oder zum Beispiel auch alle neuen *Java*-Dateien eines Verzeichnisses mit `cvs add *.java`.[4]

Version und CLASSPATH

Unser Kurs wird mit dem *Java Development Kit* von Sun arbeiten, und zwar mit der neuesten hier installierten Version *Java 1.4*.

Bevor man damit beginnen kann, sind (a) die Java-Binaries in den Unix-Suchpfad aufzunehmen und (b) dem Java-Interpreter der Ort mitzuteilen, wo benutzerdefinierte Klassen zu finden sind. So geht's:

```
export PATH=/home/java/jdk1.4/bin:$PATH
```

bzw. `setenv PATH "$PATH":/home/java/jdk1.4/bin` für Benutzer der *tcsh* oder verwandten shells. Bitte tragen Sie diese Definitionen zu Beginn des Kurses (also beim ersten Mal, wo Sie sich um die technischen Rahmenbedingungen kümmern) in Ihr `.bashrc` ein (oder für *(t)csh*-Verwender entsprechend in `~/.cshrc` oder ähnliches). Bitte darauf achten, daß nicht versehentlich eine falsche Java-Version *vor* der gewünschten Version im Suchpfad steht.

Es gibt eine Umgebungsvariable zur Steuerung der Javatools: `CLASSPATH`. Um die Paketstruktur auszunutzen, sollte der `CLASSPATH` passend gesetzt werden. Unter der Annahme, daß Sie Ihr Projekt unter `~/Projekt` ausgecheckt haben, d.h., daß Ihre Strukur wie folgt aussieht

```
~/Projekt/enigma<x>
```

setzen Sie bitte Ihren `CLASSPATH` entsprechend auf

```
export CLASSPATH=~/Projet/enigma<x>
```

[4] Mit dem Hinzufügen von Verzeichnissen bitte ein wenig vorsichtig sein, denn man kann sie Verzeichnisse nicht (in offizieller Weise) aus dem Repositorium wieder löschen.

Pakete

Wie oben angedeutet, arbeitet jede Gruppe nicht nur an einem eigenen Account, sondern auch an einem eigenen *Paket*. Z.B. hat Gruppe 5 den Account `prakt05` und erstellt das *Java*-Paket `enigma05`.[5] Insbesondere: bitte nicht die Paketdeklaration

```
package engima<x>;
```

zu Beginn der *Java*-Dateien vergessen (für eventuelle Unterpakete analog).

Makefiles

Um uns (und eventuell Ihnen) das Leben weiterhin zu vereinfachen, soll die Compilation mittel *Makefile*s möglich sein. Das bedeutet, in der Wurzel ihres Projektes soll ein *Makefile* vorhanden sein, z.B.

```
~/Projekt/enigma05/Makefile
```

Dessen erstes Ziel soll `all` sein, das bedeutet, mittels `make all` bzw. einem reinen `make` soll der jeweils aktuelle Stand des Projektes in seiner Gesamtheit kompiliert werden. Mittels `make run` soll das Hauptprogramm gestartet werden. Weitere nützliche Ziele sind `make clean` zum Aufräumen. Wir werden ein einfaches Makefile zur Verfügung stellen, welches jedoch je nach der Strukturierung Ihres Programmens (Unterverzeichnisse/Unterpakete?) angepaßt werden soll.

Konventionen

Wir streben in dem Praktikum nach einem sauberen Entwurf und nach einer sauberen Umsetzung, da “Hauptsache-es-geht” nicht das Motto des professionellen Softwareentwicklers ist (oder zumindest es nicht sein sollte …).

Verschiedenen Punkten sollten sie dabei Beachtung schenken:

- Dokumentieren und kommentieren Sie ihren Code angemessen. Verwenden Sie dabei (auch) *javadoc*.
- Seien sie einheitlich, was die Sprache betrifft, d.h., mischen Sie nicht Englische, Deutsche, und andere Sprachen, was die Bezeichner betrifft. Dies gilt insbesondere für die Schnittstelle zum Benutzer wie z.B. Ausgaben, Menues oder Hilfefunktionen.
- Verwenden Sie angemessene Bezeichner
- Halten sie sich an die *Java*-Coding-Konventionen (Pakete werden klein geschrieben, Klassen groß … .

[5] Der Grund dafür ist, daß wir gegebenenfalls am Ende die verschiedenen Pakete zu einem *gemeinsamen* *Java*-Programm zusammenbinden wollen. Deswegen die zusätzliche Trennung in einzelne Pakete.

Gruppendynamisches

Sie werden in einer Gruppe zu 4 Leuten arbeiten. Die Arbeitsteilung und Organisation der Aufgabe(n) liegt zum großen Teil bei Ihnen. Es ist unvermeidbar und kein Beinbruch, daß Gruppen nicht ganz homogen sind was Programmiererfahrung und speziell Java-Kenntnisse, Arbeitsstil, Temperament usw. angeht. Die Last sollte "gleichmäßig" untereinander aufgeteilt oder bearbeitet werden, auch um Reibereien innerhalb der Gruppe zu vermeiden. Falls es doch zu Unstimmigkeiten kommt ("den X habe ich jetzt 4 Wochen lang nicht mehr gesehen, macht der noch was?", oder umgekehrt "Y weiß alles besser, er macht alles allein, und mein Beitrag geht vollkommen unter") gehen Sie offen damit um, d.h. diskutieren sie es, wobei "offen" nicht heißen soll, daß Sie bei Unzufriedenheit den vermeintlich Schuldigen auf der Stelle bei den Betreuern "anschwärzen" sollen. Verschleppen des Problems bis zur Endabnahme oder bis es in der Gruppe unerträglich wird (es ist selten, aber es kann vorkommen) ist allerdings auch nicht gut. Gruppenarbeit ist Teil des Praktikums, d.h., wir wollen nicht, daß jemand beim leisesten Anflug von Sachdiskussion die Gruppe verläst ("Diese ewigen Diskussionen, ich weiß schon was ich machen muß, am besten spalte ich mich ab und mache ich eine eigene Gruppe, dann bin ich am schnellsten.")

Literatur

[CVS01a] Concurrent versions systems: The open standard for version control. available at `http://www.cvshome.org/`, 2001.

[CVS01b] *Concurrent Versions Systems: CVS Manual*, 2001. available at `http://www.cvshome.org/docs/manual/cvs.html`.

[Fog00] Karl Fogel. *Open Source Projekte mit CVS.* MITP-Verlag, 2000.