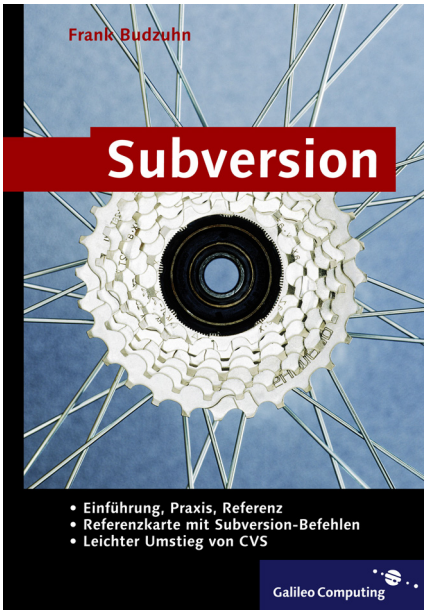
www.galileocomputing.de - Leseprobe - Subversion



Auf einen Blick

Inhalt

6 Der Entwicklungsprozess im Detail 75

Vorwort

Über viele Jahre hinweg war der Begriff Versionsmanagement im Bereich der Open Source Entwicklung praktisch synonym mit CVS, dem Concurrent Versions System. Das System leistet – in vielen Entwicklungsteams auch heute noch – im Wesentlichen das, was man von einem solchen Werkzeug erwartet: es speichert den Sourcecode des Entwicklungsteams an zentraler Stelle, erlaubt es, alte Versionsstände zu reproduzieren, führt die Änderungen mehrerer Entwickler zusammen und kann Verzweigungen in der Entwicklungslinie verwalten.

Im Bereich der quelloffenen Systeme gab es lange Zeit kein anderes System, das derart leistungsfähig war wie CVS. Dies trifft zwar nicht für kommerzielle Alternativen zu CVS zu, jedoch werden Open Source Projekte auch meist mit Open Source Werkzeugen entwickelt. CVS ist ein bereits recht altes Programm, die Wurzeln reichen bis in das Jahr 1986 zurück. Die Basis für die heutige C-Implementierung von CVS wurde im Jahr 1989 gelegt. Das Alter ist CVS deutlich anzumerken. Die Implementierung entspricht nicht dem Stand der Softwaretechnik, und das Werkzeug hat viele »Macken«, an die man sich als Entwickler im Laufe der Zeit gewöhnt hat. Zwei Dinge haben CVS jedoch über die Jahre hinweg am Leben erhalten: der Mangel an nichtkommerziellen Alternativen und sein offenes Entwicklungsmodell, das der Open Source Gemeinschaft in ihrem Entwicklungsstil stark entgegen kommt. CVS sperrt den Sourcecode einzelner Entwickler nicht gegeneinander ab, wie es viele kommerzielle Werkzeuge tun, sondern es erlaubt jedem Entwickler den gesamten Sourcecode in dessen eigener privater Kopie beliebig zu verändern. Anschließend versucht CVS selbstständig alle Änderungen wieder zusammenzuführen.

Ein potenzieller Nachfolger für CVS sollte also drei Anforderungen erfüllen: er sollte quelloffen sein, er sollte das gleiche offene Entwicklungsmodell wie CVS haben und er sollte die Macken und Fehler von CVS beheben. Daneben wäre eine moderne Softwarearchitektur beim Aufbau des Systems selbst wünschenswert. In der Tat wurde die Entwicklung eines solchen Programms im Jahre 2000 begonnen. Seine Entwickler haben ihm den Namen *Subversion* gegeben. Die Version 1.0 von Subversion wurde am 23.2.2004 freigegeben. Seitdem hat Subversion viel Aufmerksamkeit auf sich gezogen und es ist jetzt schon abzusehen, dass Subversion seinen Vorgänger CVS ablösen wird.

Dieses Buch möchte Sie in die Arbeit mit einem Versionsmanagement im Allgemeinen und mit Subversion im Speziellen vertraut machen. Sollten Sie zuvor bereits mit CVS gearbeitet haben, so finden Sie in Kapitel 9, »Subversion für CVS-Benutzer«, eine komprimierte Darstellung für CVS-Umsteiger, die auf die wichtigsten Unterschiede zwischen CVS und Subversion eingeht.

Besonderer Dank für die Unterstützung dieses Buchprojekts geht – nicht zum ersten Mal – an meine Frau Elke Szurowski, die alle Zeichnungen für dieses Buch erstellt hat und nicht müde geworden ist, das Manuskript in den verschiedenen Stadien der Entstehung Korrektur zu lesen. Vielen Dank an dieser Stelle auch an das Team von Galileo Press und dort besonders an meine Lektorin Judith Stevens-Lemoine für die erprobt gute und unkomplizierte Zusammenarbeit.

Wenn Sie Anregungen, Kritik oder Korrekturen zu diesem Buch haben, so zögern Sie bitte nicht, mir eine E-Mail an *autor@subversionbuch.de* zu schicken und schauen Sie auch mal auf der Webseite zum Buch unter *http://www.subversionbuch.de* vorbei!

Hamburg, 30.10.2004
Frank Budszuhn

Der Autor

Frank Budszuhn (E-Mail: autor@subversionbuch.de), Jahrgang 1966, ist Diplom-Informatiker und arbeitet seit 1994 als Softwareentwickler. Nach dem Studium der Informatik in Hamburg und in England arbeitete er mehrere Jahre für den Audiospezialisten Steinberg, wo er unter anderem die Grundlagen für das Audiorestaurationsprogramm »Clean« legte. Ab 1999 entwickelte er komplexe Webapplikationen für die Internetagenturen Fluxx.com, Netmatic und SinnerSchrader. Seit 2002 erstellt er in seiner eigenen Firma, der AlsterContor GmbH, Internetanwendungen im Kundenauftrag. Frank Budszuhn ist außerdem Autor der Bücher »Visual C++, Windows Programmierung mit den MFC« (Addison Wesley) und »CVS« (Galileo Computing).

3 Der Entwicklungsprozess mit Subversion

Dieses Kapitel stellt zunächst ein paar Gedanken zum Verhältnis zwischen Sourcecode und Entwickler in einem Team an, betrachtet wahrscheinliche Umstände des ersten Kontakts zwischen Entwickler und Subversion und zeigt dann den typischen Entwicklungszyklus mit Subversion.

3.1 Modell des kooperativen Entwickelns

Einzelner Entwickler

Um es vorwegzunehmen: Es kann durchaus sinnvoll sein, Subversion als einzelner Entwickler einzusetzen, denn auch ein einzelner Entwickler muss Versionsstände verwalten und hat gegebenenfalls das Problem der Verzweigung in seiner Entwicklungslinie. Doch die Verwendung von Subversion nur durch einen Entwickler ist nicht der Normalfall. Meist wird Subversion neben der reinen Versionsverwaltung auch zur Distribution und Synchronisation des Sourcecodes mehrerer Entwickler eingesetzt.

Sourcecode wird geteilt

Die grundsätzlichen Änderungen, die einen Entwickler bei der Verwendung von Subversion betreffen, sind in Kapitel 1 bereits dargestellt worden. Die erste Änderung beim Einsatz eines Versionsmanagementsystems wie Subversion sollte jedoch im Kopf des Entwicklers stattfinden: Der Sourcecode gehört nicht dem einzelnen Entwickler, sondern dem Team als Ganzen. Er wird nicht vom einzelnen Entwickler erstellt, sondern vom gesamten Team. Der einzelne Entwickler sollte sich von dem Gedanken trennen, dass der von ihm erstellte Sourcecode sein Eigentum ist (rein rechtlich ist er es meist sowie so nicht, da viele angestellte Programmierer die Rechte an von ihnen geschaffener Software per Angestelltenvertrag an ihren Arbeitgeber abtreten).

Arbeit auf Kopien

Der Entwickler eines Teams arbeitet immer auf einer lokalen Kopie des Sourcecodes, niemals auf der Master-Kopie. Die Master-Kopie wird im Repository des Subversion-Servers gespeichert. Weil jeder Entwickler nur eine Kopie erhält, gibt der einzelne Entwickler auch die Kontrolle über den Sourcecode an Subversion ab. Er muss sich weder um die Sicherung noch um das Aufbewahren von Versionsständen kümmern.

Code-Distribution

Der Entwickler muss sich daran gewöhnen, dass Subversion auch die Distribution von Sourcecode übernimmt. Es treffen laufend Änderun-

gen anderer Entwickler ein (so man Subversion nicht alleine nutzt) und die eigenen Änderungen werden umgekehrt genauso weiterverteilt. Das kann anfangs etwas ungewohnt sein. Den Zeitpunkt des Abgleichs mit Subversion bestimmt der Entwickler allerdings immer noch weitgehend selbst.

3.2 Betrachtungen zum ersten Kontakt

Viele Entwickler haben ihren ersten Kontakt zu einem Versionsmanagementsystem wie Subversion innerhalb eines Firmenprojekts, in dem die Zeit knapp bemessen ist. Das Projekt benötigt neue Ressourcen, also mehr Programmierer, ist sowieso schon hinter dem Zeitplan und allgemein hat niemand der Projektmitglieder die Muße, einen in die Geheimnisse des Versionsmanagement einzuführen. Oft bekommt der Entwickler vom Projektleiter eine kurze mechanische Einweisung, welche Befehlsfolgen auszuführen sind, um Sourcecode aus und in das Versionsmanagementsystem zu bekommen. Ein echtes Verständnis für das System bildet sich dabei meist nicht heraus, genauso mechanisch wie die Befehlsfolgen erlernt worden sind, werden sie auch ausgeführt. Es bleibt die Unsicherheit, etwas »kaputt zu machen« und das System nicht richtig zu bedienen. Meist macht ein einzelner Entwickler zwar aus Unwissenheit wenig »kaputt«, jedoch kann das Unverständnis des Systems durchaus zu Problemen und Mehrarbeit führen. Deswegen ist es wichtig, vor dem ersten Einsatz von Subversion ein grundsätzliches Bild des Systems im Kopf zu haben. Dieses soll helfen, die durchgeführten Operationen und Befehle zu begreifen und in ihren Auswirkungen auf das Repository zu verstehen.

3.3 Der Entwicklungszyklus mit Subversion

Arbeitsschritte im Entwicklungsprozess

Genau wie die Softwareentwicklung selbst ein zyklischer Prozess ist (im kleinsten ist das oft der Edit-Compile-Run-Zyklus), so ist auch die Arbeit mit Subversion ein zyklischer Prozess. Die Arbeitsschritte in diesem Prozess sind nicht streng vorgegeben, normalerweise folgen sie aber etwa diesem Ablauf:

- Eine Arbeitskopie vom Subversion-Server anfordern

oder

- Eine bestehende Arbeitskopie mit dem Repository abgleichen
- Auflösung von aufgetretenen Konflikten

dann

- Entwicklung auf der Arbeitskopie bis eine neue Teilversion erreicht ist
- Übernahme der Änderungen aus der lokalen Arbeitskopie in das Repository

Mit Beginn des zweiten Durchlaufs wird im ersten Schritt normalerweise eine bestehende Arbeitskopie mit dem Repository abgeglichen, statt eine neue lokale Arbeitskopie anzufordern. Abbildung 3.1 zeigt den Entwicklungszyklus.

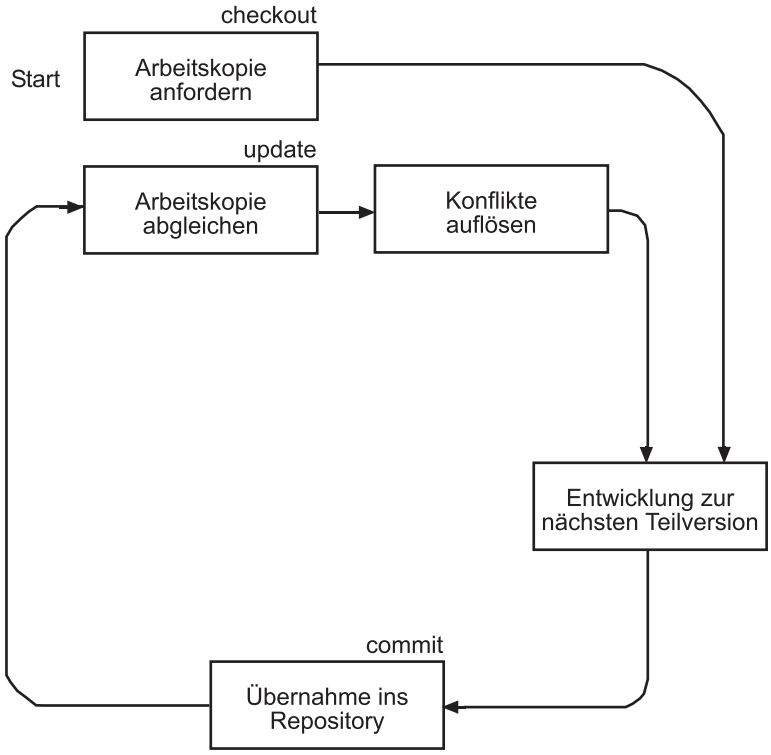


Abbildung 3.1 Der Entwicklungszyklus mit Subversion

Die einzelnen Schritte sollen nun im Folgenden detailliert erläutert werden.

3.3.1 Eine Arbeitskopie von Subversion anfordern: Checkout

Es wird immer auf einer Kopie gearbeitet

Der erste Schritt für einen Entwickler, vorausgesetzt es gibt bereits Sourcecode im Repository, ist eine lokale Arbeitskopie des Sourcecodes vom Subversion-Server anzufordern. Ein Entwickler arbeitet **immer** auf einer Kopie des Sourcecodes, niemals auf denen im Repository befindlichen Originaldaten. Die Anforderung von Sourcecode aus dem Repository wird als *Checkout* bezeichnet. Der Checkout liefert einen Baum von Sourcecode-Dateien und Verzeichnissen. Wie der Sourcecode ins Repository gekommen ist, soll an späterer Stelle beschrieben werden (siehe dazu Abschnitt 6.7, »Ein neues Projekt beginnen: import«). Beim Checkout fordert der Entwickler eine Kopie aller (Unter-)Verzeichnisse und Dateien eines Verzeichnisses im Repository an. Wichtig für das Verständnis von Subversion ist dabei zu wissen, dass ein Checkout keine Zustandsänderung auf dem Server bewirkt, der Subversion-Server führt kein Buch über die Checkouts. Der Entwickler kann also eine Arbeitskopie anfordern, sie löschen und nochmals wieder eine Kopie anfordern. Subversion ist es »egal« wie viele Kopien im Umlauf sind, das Versionsmanagementsystem »weiß« es nicht. Die Arbeit mit lokalen Kopien bedeutet auch, dass alle Änderungen, die an den lokalen Dateien und Verzeichnissen vorgenommen werden, zunächst für niemand anderes sichtbar werden. Es werden auch keine Dateien gesperrt, so dass jeder Entwickler jederzeit alles in seiner lokalen Kopie ändern kann.

3.3.2 Abgleich der Arbeitskopie mit dem Repository: Update

In den meisten Fällen besitzt der Entwickler bereits eine lokale Arbeitskopie. In diesen Fällen wird der Entwickler kein Checkout durchführen, sondern er wird die vorhandene lokale Arbeitskopie mit dem Repository abgleichen. Dieser Abgleich wird als *Update* bezeichnet. Beim Update werden die Änderungen aus dem Repository in die lokale Arbeitskopie übernommen. Es wird ausschließlich die lokale Arbeitskopie verändert, das Repository wird nicht berührt. Auch wird auf dem Subversion-Server keine Zustandsänderung durch das Update bewirkt, genau wie ein Checkout ist auch ein Update ohne Auswirkung auf andere Entwickler.

Das Update überträgt alle Änderungen aus dem Repository in die lokale Arbeitskopie. Nicht veränderte Dateien und Verzeichnisse werden

übersprungen. Nur im Repository veränderte Dateien und neue Dateien und neue Verzeichnisse im Repository werden einfach in die lokale Arbeitskopie übernommen, alte Dateien in der lokalen Arbeitskopie werden dabei überschrieben. Ausschließlich in der lokalen Arbeitskopie veränderte Dateien werden bei einem Update nicht angefasst. Sollten sich Änderungen an einer Datei sowohl im Repository als auch in der lokalen Arbeitskopie ergeben haben, so versucht Subversion die beiden Dateien zu einer zusammenzuführen. Dieser Vorgang wird als *Merging* bezeichnet.

Merging

Beim Merging vergleicht Subversion zeilenweise die Datei aus dem Repository mit der Datei aus der lokalen Arbeitskopie und erzeugt daraus eine neue Datei. Haben sich Änderungen an einer Stelle nur in einer der beiden Versionen ergeben, so werden diese in die Ergebnisdatei übernommen. Gibt es dagegen Änderungen an gleicher Stelle in beiden Dateien, so muss Subversion aufgeben. Es lässt sich automatisch nicht mehr bestimmen, welche Version korrekt ist. Dieser Vorfall wird als *Konflikt* bezeichnet. Die Tabelle zeigt alle Fälle, die beim Update einer Datei auftreten können.

lokale Arbeitskopie	Repository	Aktion	Zustand danach in der lokalen Arbeitskopie
Datei nicht verändert	Datei nicht verändert	Keine Aktion: Überspringen	Nicht modifiziert
Datei verändert	Datei nicht verändert	Keine Aktion	Modifiziert
Datei nicht verändert	Datei verändert	Kopieren	Nicht modifiziert
Datei verändert	Datei verändert	Merge	Falls Merge erfolgreich: modifiziert, sonst Konflikt
Datei oder Verzeichnis nicht vorhanden	Neue Datei oder neues Verzeichnis vorhanden	Kopieren bzw. anlegen	Nicht modifiziert
Datei oder Verzeichnis nicht verändert	Datei oder Verzeichnis gelöscht	Löschen	Nicht vorhanden

lokale Arbeitskopie	Repository	Aktion	Zustand danach in der lokalen Arbeitskopie
Neue Datei oder neues Verzeichnis vorhanden	Datei oder Verzeichnis nicht vorhanden	Keine Aktion	Datei oder Verzeichnis unbekannt. Muss erst durch den Befehl **add** dem Repository hinzugefügt werden.

3.3.3 Auflösung von aufgetretenen Konflikten

Konfliktmarker

Bei einem Konflikt gibt Subversion eine entsprechende Kennzeichnung aus und platziert so genannte *Konfliktmarker* innerhalb der Datei: Es werden beide Versionen in die Datei übernommen und durch deutlich erkennbare Strichlinien voneinander und vom Rest des Sourcecodes abgetrennt. Die Konfliktmarker sind so gewählt, dass sie in allen gängigen Programmiersprachen Fehlermeldungen beim Kompilieren oder bei der Ausführung verursachen.

Konflikte auflösen

Sind Konflikte aufgetreten, so müssen diese vom Entwickler in der lokalen Arbeitskopie aufgelöst werden. Meist entscheidet er sich in Absprache mit seinen Entwicklerkollegen für eine der beiden Versionen oder baut ein Gesamtwerk aus den beiden Varianten. Ist der Konflikt beseitigt, so muss dies Subversion explizit mitgeteilt werden, so dass es den Zustand der betroffenen Dateien zurück setzt. Konflikte betreffen nur die lokale Arbeitskopie, auf dem Server selbst gibt es keinen Konflikt.

Konflikte sind selten

Allen Subversion-Anfängern sei an dieser Stelle gesagt, dass Konflikte seltener auftreten, als man zunächst vermuten sollte. Schließlich arbeiten die einzelnen Entwickler eines Entwicklerteams meist an unterschiedlichen Teilaspekten und damit auch an unterschiedlichen Stellen im Programmcode. Konflikte deuten daher meist auf mangelnde Absprache unter den Entwicklern hin.

3.3.4 Entwicklung auf der Arbeitskopie bis neue Teilversion erreicht ist

Nach dem Checkout oder dem Update tut der Entwickler, was er auch ohne Subversion tun würde. Er arbeitet ganz normal mit seiner Entwicklungsumgebung auf den Dateien der lokalen Arbeitskopie. Es fin-

det keine Interaktion mit Subversion und damit auch nicht mit anderen Entwicklern statt.

Zeitpunkt der Übernahme ins Repository

Irgendwann erreicht der Entwickler einen Punkt, an dem er meint, dass alle anderen Entwickler seines Teams seine Änderungen übernehmen könnten. Wann dieser Punkt erreicht ist, liegt im Ermessen des Entwicklers selbst. Welchen Zustand die Software zu diesem Zeitpunkt haben sollte, wird meist durch den Projektleiter festgelegt. So sollten sich generell alle Änderungen übersetzen (kompilieren) lassen und andere Entwickler sollten in ihrer Arbeit durch die Neuerungen nicht behindert werden. Hat der Entwickler beispielsweise zu Testzwecken die Hauptmenüleiste ausgebaut, so ist dies nicht unbedingt eine Version, die in das Repository übernommen werden sollte, da dann andere Entwickler nicht mehr testen können.

Zwei mögliche Herangehensweisen

Beim Erreichen einer stabilen Version hat der Entwickler zwei Möglichkeiten: Entweder versucht er seine Änderungen direkt in Subversion zu übernehmen oder er gleicht zunächst seine Änderungen mit dem Repository ab. Schließlich können sich in der Zwischenzeit Änderungen durch andere Entwickler ergeben haben. Die erste Variante sollte nur gewählt werden, wenn die Entwicklungszeit kurz war und die im Repository gehaltene Version sich vermutlich kaum verändert hat. Die zweite Variante ist auf jeden Fall die defensivere Variante, da der Entwickler hier zunächst seine Änderungen mit denen der anderen Entwickler zusammenführt und auch testen kann, bevor er das Gesamtwerk in das Repository überführt. Ob der Entwickler vor der Übernahme seiner Änderungen ins Repository noch ein Update durchführt, wird oft auch von Erfahrungswerten bestimmt. Diese ergeben sich zwangsläufig nach einer gewissen Zeit bei der Arbeit an einem konkreten Projekt und dem Einsatz von Subversion.

3.3.5 Übernahme der Änderungen aus der lokalen Arbeitskopie in das Repository: Commit

Nun ist der Entwickler so weit, dass er seine Änderungen in das Repository überführen möchte. Bis jetzt hat alles was er gemacht hat keinerlei Auswirkungen auf das Repository oder auf andere Entwickler gehabt. Würde er an dieser Stelle seinen Sourcecode einfach löschen, so würde niemand bemerken, dass er überhaupt an der Entwicklung beteiligt war. Erst mit der Übernahme der Änderungen aus der lokalen Kopie in das Repository wird dieses verändert und damit bekommen auch alle anderen Entwickler diese Neuerungen mit ihrem nächsten

Update in ihre jeweiligen Arbeitskopien überführt. Die Übernahme aus der lokalen Kopie in das Repository wird als *Commit* oder auch als *Einchecken* bezeichnet. Ein Commit wird normalerweise nur auf Dateien und Verzeichnissen ausgeführt, die in der lokalen Kopie auch tatsächlich verändert worden sind.

Subversion kann das Einchecken verweigern

Ein Commit kopiert die Änderungen des Entwicklers in das Repository. Für den Fall, dass eine Datei im Repository Änderungen enthält, die vom Entwickler noch nicht übernommen worden sind, verweigert Subversion das Commit. Die Zurückweisung des Commits durch Subversion darf nicht mit einem Konflikt verwechselt werden. Ein Konflikt kann grundsätzlich nur beim Update auftreten. Die Ablehnung des Commits durch Subversion erfolgt nur, weil die Revision im Repository neuer ist, als in der lokalen Arbeitskopie. Subversion kann aber nur ein Commit auf der aktuellen im Repository gespeicherten Revision durchführen, daher erzwingt es das Update um die aktuelle Revision aus dem Repository in die lokale Arbeitskopie zu übertragen. Erst dieses Update kann zu einem Konflikt führen, muss es aber nicht! Nach dem Update ist der Entwickler aufgefordert, eventuelle Konflikte zu beseitigen und dann das Commit erneut durchzuführen.

Ein neuer Zyklus

Nach dem Commit kann der Entwickler einfach auf der Arbeitskopie weiterarbeiten. Oft führt ein Entwickler ein Commit zum Abschluss seines Arbeitstages durch, so dass er nach dem Commit Feierabend macht und die Arbeit unterbricht. Am nächsten Arbeitstag führt der Entwickler dann ein Update auf der lokalen Arbeitskopie aus, um die Änderungen zu erhalten, die sich in der Zwischenzeit durch die Arbeit der anderen Entwickler ergeben haben. Damit leitet der Entwickler einen neuen Arbeitszyklus ein.

3.4 Der Entwicklungszyklus in der Zusammenfassung

Nun soll der Zyklus der Entwicklung mit Subversion nochmals zusammengefasst dargestellt werden:

- Wenn ein Entwickler mit der Entwicklung an einem bereits bestehenden Projekt beginnt, so besitzt er noch keinen lokalen Sourcecode des Projekts. Dieser ist nur im Repository des Subversion-Servers vorhanden. Dieser Zustand ist in Abbildung 3.2 dargestellt.

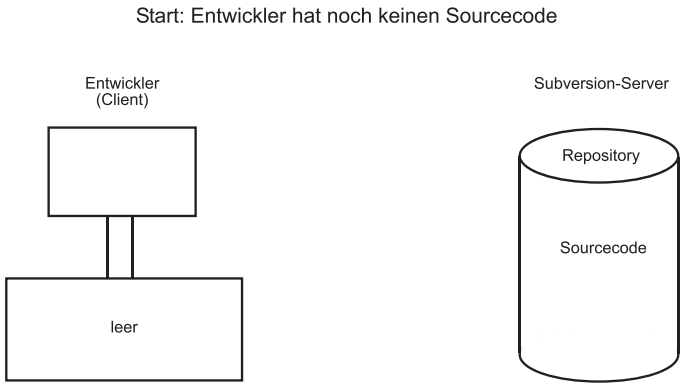


Abbildung 3.2 Entwickler besitzt noch keinen Sourcecode

- Um mit der Entwicklung zu beginnen, fordert der Entwickler den Sourcecode vom Subversion-Server an. Dazu verwendet er den Befehl **checkout**. Der Sourcecode wird aus dem Repository in das Arbeitsverzeichnis des Entwicklers kopiert, es wird eine lokale Arbeitskopie angelegt. Dieser Vorgang wird nach dem gleichnamigen Befehl als *Checkout* bezeichnet und ist in Abbildung 3.3 zu sehen.

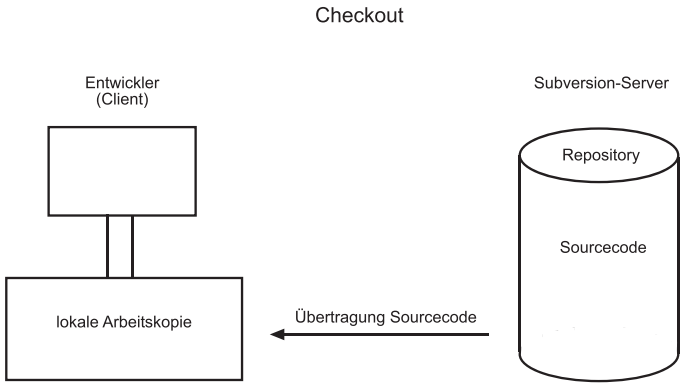


Abbildung 3.3 Entwickler fordert Sourcecode an

- In den meisten Fällen besitzt der Entwickler schon eine Kopie des Sourcecodes in Form einer lokalen Arbeitskopie. In diesem Fall führt der Entwickler keinen Checkout durch. Stattdessen wird die bereits vorhandene lokale Arbeitskopie zu Beginn eines neuen Zyklus mit dem Repository abgeglichen. Dazu verwendet der Entwickler den Befehl **update**. Der Vorgang wird – entsprechend dem Namen des Befehls – ebenfalls als Update bezeichnet. Die Änderungen aus dem

Repository werden in die lokale Arbeitskopie des Entwicklers kopiert. Der Ablauf ist in Abbildung 3.4 gezeigt.

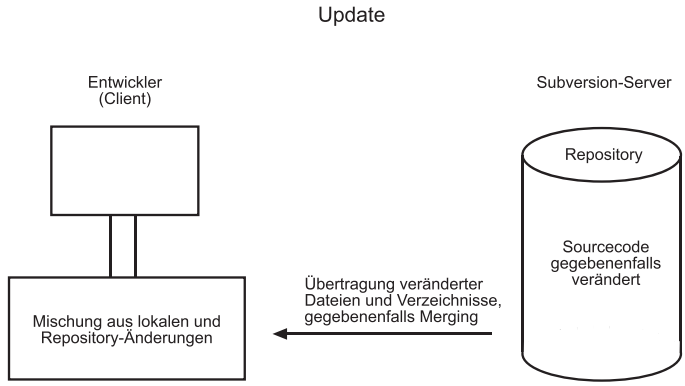


Abbildung 3.4 Abgleich des Repositories mit der lokalen Arbeitskopie

- Sind vor dem Update einzelne Dateien sowohl im Repository als auch in der lokalen Arbeitskopie verändert worden, so versucht Subversion bei diesen Dateien, sie automatisch zusammenzuführen. Subversion führt einen *Merge* auf den Dateien durch. Nun kann es sein, dass dabei Konflikte auftreten. Diese muss der Entwickler manuell in der lokalen Arbeitskopie beseitigen. Danach sollte der Entwickler aus dem Sourcecode eine neue Programmversion erstellen und diese testen. Schließlich können beim Merging oder auch nur beim Update Programmteile verschiedener Entwickler zusammengekommen sein, die von der Programmlogik her nicht miteinander harmonieren und im fertigen Programm Probleme verursachen. Konfliktbeseitigung und Test werden in Abbildung 3.5 gezeigt.

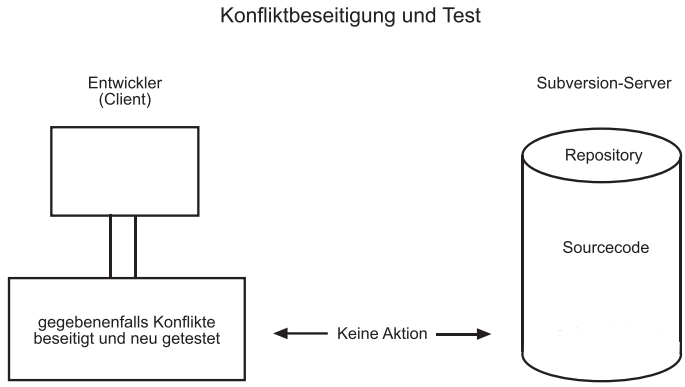


Abbildung 3.5 Konflikte beseitigen und testen

- Wenn der Entwickler einen Konflikt beseitigt hat, so muss er dies Subversion explizit mitteilen. Dazu ruft er den Befehl **resolved** auf. Solange der Befehl nicht aufgerufen worden ist, lassen sich keine Änderungen in das Repository übernehmen, da sich die vom Konflikt betroffenen Dateien weiterhin im Konfliktzustand befinden. Erst der Befehl **resolved** setzt diesen Zustand zurück. Dies ist in Abbildung 3.6 zu sehen.

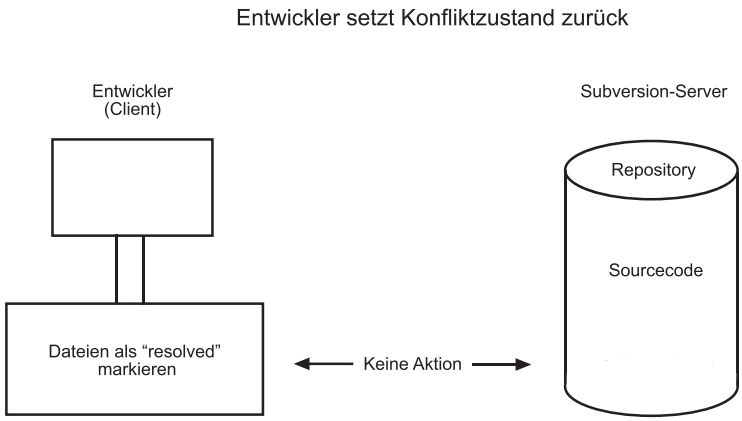


Abbildung 3.6 Den Konfliktzustand zurücksetzen

- Nun arbeitet der Entwickler auf seiner lokalen Arbeitskopie weiter bis er eine neue stabile Teilversion erreicht hat. Während dieser Zeit findet keine Interaktion mit dem Subversion-Server statt. Dies zeigt Abbildung 3.7.

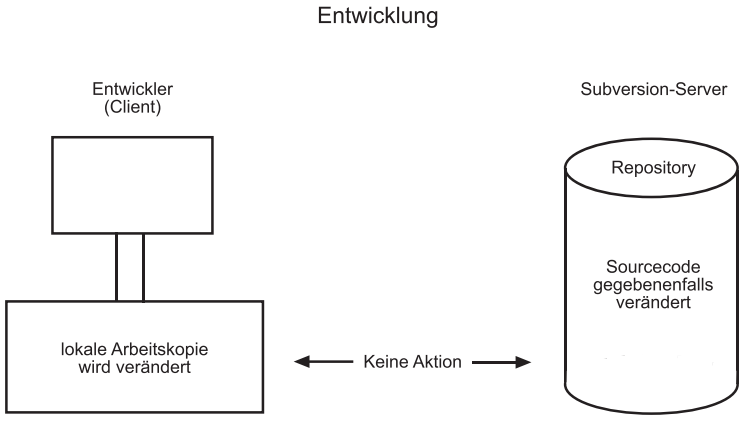


Abbildung 3.7 Keine Aktion während der Entwicklung

- Hat die Entwicklung der neuen Teilversion eine gewisse Zeit gebraucht, so kann der Entwickler nun nochmals seine Arbeitskopie mit dem Repository abgleichen (vergleiche Abbildung 3.4). Eventuell dabei auftretende Konflikte muss er beheben (vergleiche Abbildungen 3.5 und 3.6). Er kann sich allerdings auch dazu entschließen, gleich zum nächsten Schritt zu gehen und seine Änderungen einzuchecken.
- Der Entwickler überführt nun seine Änderungen in das Repository. Dieser Vorgang wird als *Einchecken* oder als *Commit* bezeichnet und ist in Abbildung 3.8 gezeigt. Die Änderungen des Entwicklers werden aus der lokalen Arbeitskopie in das Repository kopiert. Dazu verwendet der Entwickler den Befehl **commit**. Hat der Entwickler zuvor keinen Abgleich mit dem Repository durchgeführt und eine Datei oder ein Verzeichnis wurde in der lokalen Arbeitskopie und im Repository verändert, so verweigert Subversion das Commit. In diesem Fall muss der Entwickler auf jeden Fall den Befehl **update** auf der Datei beziehungsweise auf dem Verzeichnis durchführen, bevor er die diese einchecken kann. Das Update ist notwendig, um die aktuelle Revision der betreffenden Ressource in die lokale Arbeitskopie zu übernehmen. Ein Commit wird nämlich von Subversion nur auf der jeweils aktuellen Revision akzeptiert. Treten bei diesem Update Konflikte auf, so müssen diese beseitigt werden, bevor das Commit durchgeführt werden kann. Die Konflikte treten folglich immer beim Update in der lokalen Arbeitskopie auf, niemals beim Commit im Repository. Das Commit wird von Subversion eben genau dann verweigert, wenn es zu Konflikten kommen *könnte*.

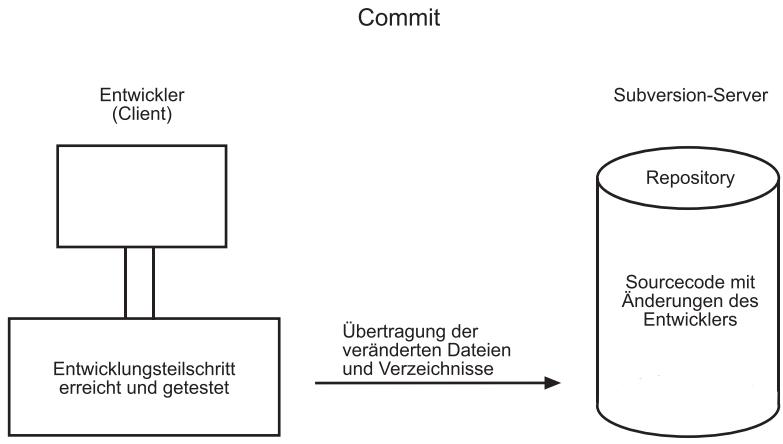


Abbildung 3.8 Änderungen ins Repository übernehmen

Nach dem Einchecken sind keine weiteren Schritte notwendig. Der Entwickler arbeitet einfach auf der bestehenden lokalen Arbeitskopie weiter. Der Entwicklungszyklus beginnt von vorn.

3.5 Der Entwicklungszyklus mit mehreren Entwicklern

Innerhalb eines Entwicklungsteams greift jeder Entwickler auf den Subversion-Server aus seinem eigenen Entwicklungszyklus heraus zu. Damit laufen mindestens genauso viele Entwicklungszyklen parallel ab, wie es Entwickler im Team gibt (es kann durchaus noch mehr Zyklen geben, wenn beispielsweise zu Testzwecken weitere Arbeitskopien aus dem Repository »gezogen« werden). Welcher Zyklus sich in welchem Zustand befindet, ist nicht bestimmbar, es ist jederzeit jede Kombination von Zuständen denkbar. Es ist weder für Subversion noch für einen einzelnen Entwickler überhaupt feststellbar, wie viele Arbeitskopien und damit Entwicklungszyklen existieren. Subversion führt über Arbeitskopien kein Buch. Jeder Zyklus kann jederzeit beendet werden, es können jederzeit neue Zyklen entstehen. Wenn ein Entwickler dies wünscht, kann er auch auf mehreren Arbeitskopien gleichzeitig arbeiten und damit mehrere Zyklen führen.

Daher lässt sich folgern, dass ein Entwickler jederzeit davon ausgehen muss, dass sich das Repository seit dem letzten Zugriff auf den Subversion-Server verändert hat. Von den bisher beschriebenen Befehlen verändert nur **commit** das Repository. Zwar kennt Subversion noch mehr Befehle, die das Repository verändern, jedoch ist es der Befehl **commit**, der die Änderungen am Sourcecode über das Repository an die anderen Entwickler verteilt. Dies ist daher auch der Befehl, bei dem man sich vorher genau überlegen sollte, wann man ihn aufruft!

Lokale Arbeitskopien können jederzeit gelöscht werden

Da der Server über lokale Kopien kein Buch führt, können diese jederzeit gefahrlos entsorgt werden. Weder der Server noch andere Entwickler werden davon betroffen.

Synchronisiert werden mehrere Entwicklungszyklen letztendlich durch die Befehle **update** und **commit**. Jeder ist dabei für eine Richtung zuständig. Aus Sicht des Entwicklers führt **update** die eingehende Synchronisation durch und **commit** die ausgehende.

Abbildung 3.9 zeigt einen Subversion-Server mit drei Entwicklungszyklen.

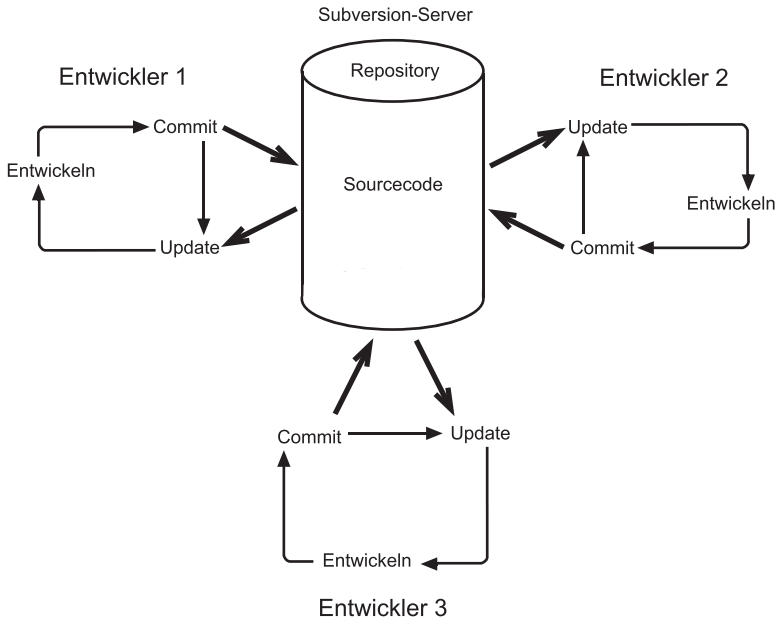


Abbildung 3.9 Subversion-Server mit drei Entwicklungszyklen

3.6 Subversion und Kommunikation

Subversion kann die Kommunikation in einem Entwicklerteam nicht ersetzen. Im Gegenteil: Die Kommunikation der Entwickler untereinander ist unabdingbar! Eine gute Kommunikation und eindeutig verteilte Aufgaben der Entwickler helfen Konflikte zu vermeiden. Wenn jeder Entwickler ein ungefähres Bild der Aufgaben und Arbeitsgebiete der anderen Entwickler im Kopf hat, so wird er hoffentlich den zuständigen Entwickler fragen, bevor er sich an die Veränderung von Dateien macht, die eigentlich gar nicht in seinem Zuständigkeitsbereich liegen.

3.7 Regeln im Umgang mit Subversion

Für die Entwickler eines Teams ist es meist sinnvoll zur Verwendung von Subversion gewisse Regeln aufzustellen, um eine reibungslose Zusammenarbeit zu fördern. Oft werden diese Regeln durch den Projektleiter aufgestellt (siehe Anhang B). Generell sollte der Zustand der in Subversion eingecheckten Sourcecode-Dateien geregelt sein. So ist es sinnvoll, nur kompilierbare Dateien einzuchecken und dafür Sorge zu tragen, dass die Lauffähigkeit des Gesamtsystems gegeben ist.

Ein paar Gedanken sollte sich der Entwickler auch dazu machen, wann und wie oft er Updates vornimmt und wann er seine eigenen Änderungen zurückführt (eincheckt). Häufige Updates halten einen immer auf den neuesten Stand können bei konzentrierter Entwicklung aber auch stören, da sie zu sofortigen Änderungen zwingen und damit aus dem eigentlichen Problem herausreißen können. Zu seltene Updates können viele Konflikte auslösen, schließlich hat sich in der Zwischenzeit viel getan und damit steigt auch die Wahrscheinlichkeit für Konflikte. Auch ein Commit sollte wohl überlegt erfolgen. Meist bietet es sich an, wenn ein Teilproblem vollständig gelöst wurde und wenige Auswirkungen auf den Rest der Entwicklung zu erwarten sind. Idealerweise findet das gesamte Entwicklerteam zu einem gemeinsamen Rhythmus.

3.8 Zusammenfassung

Dieses Kapitel hat das theoretische Basiswissen vermittelt, das ein Entwickler besitzen sollte, wenn er mit Subversion arbeiten möchte. Ein besonderer Schwerpunkt ist auf die Beschreibung der grundsätzlich ablaufenden Prozesse und auf die Auswirkungen der Zusammenarbeit mehrerer Entwickler gelegt worden. Nun ist es an der Zeit, erste praktische Schritte zu unternehmen. Dazu werden in den nächsten beiden Kapiteln zunächst die Installation des notwendigen Client-Programms vorgenommen und danach eine erste Sitzung mit Subversion durchgespielt.

4 Installation

Subversion lässt sich auf vielen Betriebssystemen installieren. In diesem Kapitel soll die Installation auf Windows und auf Linux gezeigt werden. Obwohl Client und Server zusammen installiert werden, geht es in diesem Kapitel schwerpunktmäßig um die Einrichtung des Client-Programms.

4.1 Installation unter Windows

4.1.1 Installation durch das Installationsprogramm

Die einfachste Art Subversion unter Windows zu installieren ist es, das dafür vorgesehene Installationsprogramm zu verwenden. Das Programm führt Windows-typisch in mehreren Schritten durch die Installation. Es ist nach dem Schema **svn-<versionsnummer>-setup.exe** benannt, also beispielsweise **svn-1.0.5-setup.exe** bei Version 1.0.5. Das Programm wird durch einen Doppelklick gestartet. Nach einer Abfrage, ob man Subversion auch wirklich installieren möchte, erscheint das in Abbildung 4.1 gezeigte Dialogfeld.

Abbildung 4.1 Startdialog des Subversion-Installationsprogramms

Lizenzbedingungen

Im zweiten Schritt der Installation müssen die Lizenzbedingungen bestätigt werden. Bei der Lizenz handelt es sich um eine leicht abgewandelte Apache-style Open Source Lizenz.

Im dritten Schritt wird die verwendete Version der Berkeley DB ausgegeben und es wird darauf hingewiesen, dass ein eventuell als Windows-Dienst laufender Apache Webserver vor der Installation deaktiviert werden muss. Das Installationsprogramm nimmt dies nicht selbst vor.

Installationsverzeichnis

In Schritt vier der Installation wird das Installationsverzeichnis bestimmt. Bei einer normalen Windows-Installation steht die Vorgabe auf **c:\Programme\Subversion**.

Im fünften Schritt wird der Name für den Eintrag in das Startmenü von Windows ausgewählt. Die Vorgabe hier ist »Subversion«. Möchte man keinen Eintrag im Startmenü erzeugen – im Startmenü werden nur Links auf die Dokumentation und den Deinstaller eingetragen, nicht jedoch auf die Programme selbst – so setzt man in diesem Schritt ein Häkchen bei »Don't create a Start Menu Folder«.

Der sechste Schritt erlaubt die Anlage von Verknüpfungen auf die Subversion-Dokumentation auf dem Desktop und in der Schnellstartleiste. Möchte man diese Verknüpfungen nicht anlegen, so deaktiviert man die beiden Auswahlboxen. Abbildung 4.2 zeigt diesen Schritt.

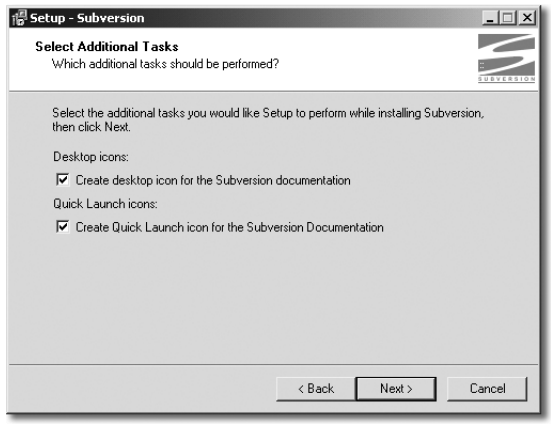


Abbildung 4.2 Anlage von Verknüpfungen auf die Dokumentation

Im Schritt sieben wird nochmals eine Zusammenfassung der vorzunehmenden Installationsaufgaben angezeigt. Ein Klick auf die Schaltfläche **Install** startet die Installation. Die Installation wird durch einen Fortschrittsbalken angezeigt.

Windows 9x

Nachdem alle Dateien installiert worden sind, werden noch einige Informationen zur Installation angezeigt. So werden Windows 9x und ME-Benutzer aufgefordert, die Umgebungsvariable `APR_ICONV_PATH`

in der Datei **AUTOEXEC.BAT** auf das Verzeichnis **iconv** der Subversion-Installation zu setzen, also beispielsweise folgendermaßen:

```
SET APR_ICONV_PATH="c:\Programme\Subversion\iconv"
```

Zum Schluss wird noch die Bestätigung angezeigt, dass die Installation beendet worden ist. Ein Neustart ist nicht erforderlich.

4.1.2 Subversion deinstallieren

Eine mit dem Installationsprogramm angelegte Subversion-Installation lässt sich einfach wieder deinstallieren. Dazu ruft man entweder direkt den im Startmenü eingetragenen Deinstaller auf oder man geht in die Systemsteuerung und wählt unter **Software** den Eintrag Subversion aus. Ein Klick auf die Schaltfläche **Ändern/Entfernen** entfernt Subversion aus dem System. Abbildung 4.3 zeigt diesen Schritt.

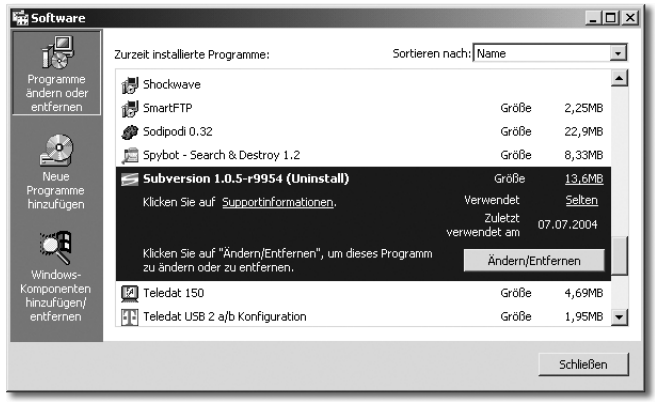


Abbildung 4.3 Subversion deinstallieren

4.1.3 Manuelle Installation

Man kann Subversion unter Windows auch recht einfach manuell installieren. Dazu wird Subversion in Form eines ZIP-Archivs ausgeliefert. Das Archiv ist nach dem Schema **svn-win32-<Versionsnummer>.zip** benannt, also beispielsweise **svn-win32–1.0.5.zip** bei Version 1.0.5. Das ZIP-Archiv enthält bereits eine Verzeichnisstruktur, die sich direkt – beispielsweise nach **c:\Programme** – auspacken lässt. Nach dem Auspacken erhält man ein Verzeichnis mit dem Namen **svn-win32-<Versionsnummer>** und den in Abbildung 4.4 gezeigten Unterverzeichnissen.

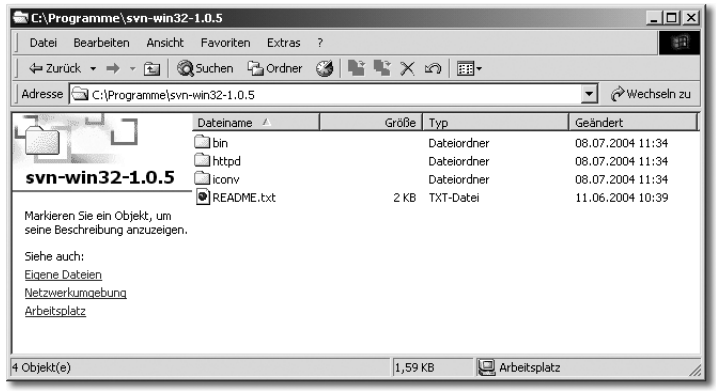


Abbildung 4.4 Unterverzeichnisse der Subversion-Installation

Die Datei **README.txt** beschreibt die installierte Version und die verwendeten Bibliotheken. Das Verzeichnis **bin** enthält die ausführbaren Programme und die benötigten DLLs. Das Verzeichnis **httpd** enthält Module, die für den Apache Webserver benötigt werden. Für den Subversion-Client werden diese Module nicht benötigt. Im Verzeichnis **iconv** sind allerlei Module der *Apache Portable Runtime* (APR) enthalten, einer betriebssystemunabhängigen Bibliothek auf deren Basis Subversion implementiert worden ist.

Windows-Path-Variable

Damit die Subversion-Kommandozeilenprogramme einfach verwendet werden können, sollte man das Unterverzeichnis **bin** in die Umgebungsvariable `PATH` aufnehmen. Auf Windows 2000 und Windows XP geht man dazu in der Systemsteuerung auf das Symbol **System** und klickt dann auf dem Reiter **Erweitert** auf die Schaltfläche **Umgebungsvariablen**. Daraufhin wird das in Abbildung 4.5 gezeigte Dialogfeld geöffnet.

Man wählt im unteren Bereich Systemvariablen den Eintrag **Path** aus und klickt auf die Schaltfläche **Bearbeiten...**. Es öffnet sich das in Abbildung 4.6 gezeigte Dialogfenster.

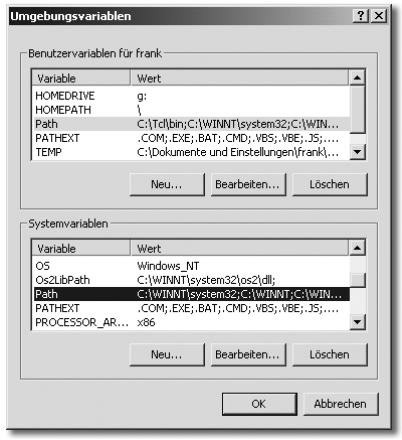


Abbildung 4.5 Änderung der Path-Variablen

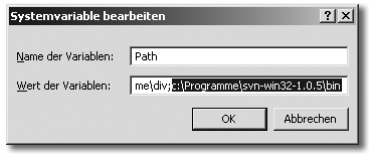


Abbildung 4.6 Pfad für Subversion hinzufügen

Nun geht man an das Ende des Eintrags und fügt dort, durch ein Semikolon abgetrennt, das Verzeichnis an, in dem sich die ausführbaren Subversion-Dateien befinden, also beispielsweise **C:\Programme\svn-win32–1.0.5\bin**. Man sollte beachten, dass man auf Windows NT, Windows 2000 und Windows XP Administratorrechte benötigt, um diese Änderung durchzuführen. Besitzt man keine Administratorrechte, so kann man den Pfad auch für den eigenen Benutzer setzen. Dazu legt man die Benutzervariable **Path** für den eigenen Benutzernamen an (oberes Feld in Abbildung 4.5, Schaltfläche **Neu...**), wenn sie noch nicht vorhanden ist. Man sollte nicht vergessen, an dieser Stelle auf die systemweite Path-Variable zu verweisen, also beispielsweise folgendermaßen:

```
%Path%;C:\Programme\svn-win32-1.0.5\bin
```

Damit ist Subversion einsatzbereit! Man kann dies testen, indem man in einer Eingabeaufforderung

```
svn help
```

eingibt. Subversion sollte dann eine Liste aller seiner Unterbefehle ausgeben.

4.2 Installation unter Debian Linux

Unter Debian Linux lässt sich Subversion komfortabel über die eingebaute Paketverwaltung installieren. Damit dies reibungslos funktioniert, muss zuvor allerdings eine weitere Quelle für die Paketverwaltung konfiguriert werden, da Subversion in der normalen Debian-Distribution nicht enthalten ist.

apt

Die Quellen der Paketverwaltung **apt** werden in die Datei **sources.list** eingetragen. Diese Datei findet man normalerweise im Verzeichnis **/etc/apt**. Dort fügt man eine zusätzliche Zeile hinzu:

```
deb http://people.debian.org/~adconrad woody subversion
```

Damit ist **apt** in der Lage aktuelle Subversion-Pakete zu laden. Voraussetzung dafür ist natürlich eine bestehende Internetverbindung. Sollte der Computer, auf dem Subversion installiert werden soll, keine Internetverbindung besitzen, so muss man die Installationspakete zuvor mittels eines andern Rechners von der angegebenen Adresse herunterladen und dann auf einem Medium speichern, das von dem Computer gelesen werden kann, auf dem Subversion installiert werden soll. In der Datei **sources.list** trägt man dann den entsprechenden lokalen Pfad ein.

Installiert wird Subversion durch den Aufruf von

```
apt-get install subversion
```

Die Paketverwaltung von Debian installiert anschließend alle Pakete, die zur Ausführung von Subversion notwendig sind. Dazu gehören die Apache Portable Runtime (APR), die Berkeley DB und einige andere Bibliotheken. Zur anschließenden Verwendung des Client-Programms **svn** ist keine weitere Konfiguration notwendig!

4.3 Installation auf anderen Linux- und Unix-Systemen

Die Installation auf Debian wurde beispielhaft für alle Linux-Distributionen beschrieben. Auf anderen Linux-Derivaten ist meist ein RPM-Paket zu installieren. Dies sollte ähnlich reibungslos funktionieren wie auf Debian. Andere Unix-Systeme wie beispielsweise die BSD-Systeme verwenden wieder andere Paketformate. Daher kann an dieser Stelle keine allgemein gültige Installationsanleitung gegeben werden. Nach der Installation ist keine Konfiguration am Subversion-Client vorzuneh-

men, er sollte einfach funktionieren. Wer für sein System kein Installationspaket auftreiben kann, der kann Subversion auch aus dem Sourcecode selbst übersetzen. Das Kompilieren von Subversion aus dem Sourcecode wird an späterer Stelle in Abschnitt 8.3, »Subversion selbst kompilieren«, beschrieben.

4.4 Die Programme und Module von Subversion

4.4.1 Die Kommandozeilenprogramme

Eine Subversion-Installation besteht aus sechs Kommandozeilenprogrammen:

- **svn** ist das Clientprogramm für den Entwickler. Zur Verwaltung von Sourcecode wird ausschließlich dieses Programm benötigt, alle anderen Programme dienen administrativen Zwecken.
- **svnadmin** legt Repositories an und verwaltet diese. Das Programm kann nur auf dem Server ausgeführt werden, die Verwaltung von Repositories über ein Netzwerk ist nicht vorgesehen.
- **svnlook** dient zur Inspektion von Subversion-Repositories. Auch dieses Programm muss direkt auf dem Server ausgeführt werden und kann nicht über ein Netzwerk arbeiten. **svnlook** besitzt nur lesende Optionen und kann ein Repository nicht verändern.
- **svnversion** gibt die in einer lokalen Arbeitskopie vertretenen Revisionen aus.
- **svnserve** ist ein kleiner, einfacher Server, der Subversion-Repositories über das Netzwerk zugänglich macht. Der Server verwendet ein simples, proprietäres Protokoll und kann recht einfach konfiguriert werden. Der Server wird in Abschnitt 8.4.3, »svnserve einrichten«, beschrieben.
- **svndumpfilter** ist ein Filterprogramm, mit dem durch das Programm **svnadmin** erstellte Dumps des Repositories nach bestimmten Kriterien gefiltert werden können. Ein solchermaßen gefilterter Dump kann wiederum in ein Repository importiert werden. Auf diese Weise können Repositories nachträglich um bestimmte Informationen bereinigt werden.

4.4.2 Das Apache Modul

Das Apache-Modul **mod_dav_svn** dient zum Zugriff auf Subversion-Repositories über den Apache Webserver. Dabei übernimmt Apache

sowohl den Transport der Daten über das Netzwerk, wie auch die Authentifizierung der Benutzer. Zur Übertragung der Daten wird das WebDAV-Protokoll verwendet, eine Erweiterung von HTTP die auch den schreibenden Zugriff erlaubt. Dabei können Übertragungen auch per SSL verschlüsselt werden.

4.5 Die Verbindung zum Repository herstellen

Um eine Subversion-Installation zu testen, sollte man versuchsweise auf ein bestehendes Repository zugreifen und dort ein Verzeichnis auschecken. Wer von seinem Entwicklungsrechner Zugriff auf das Internet hat, kann für diesen Versuch den Sourcecode von Subversion selbst auschecken. Der Zugriff auf ein Repository wird bei Subversion durch eine URL definiert. Dabei sind verschiedene Zugriffsprotokolle möglich (HTTP, HTTPS, SVN, SVN+SSH, FILE). Das Repository muss bei Befehlen, die nicht aus einer lokalen Arbeitskopie heraus ausgeführt werden, angegeben werden. Dies ist bei den Befehlen **import** und **checkout** immer der Fall. Um den Sourcecode von Subversion auszuchecken gibt man ein:

```
svn checkout http://svn.collab.net/repos/svn/trunk svn
```

Subversion sollte daraufhin eine Ausgabe folgender Art machen:

```
A  svn\Makefile.in
A  svn\ac-helpers
A  svn\ac-helpers\install.sh
A  svn\ac-helpers\install-sh
A  svn\build.conf
A  svn\win-tests.py
A  svn\www
A  svn\www\release-history.html
A  svn\www\testing-goals.html
...
```

Der Vorgang kann gefahrlos abgebrochen werden, indem man Strg + Untbr (Windows) oder Strg + C (Unix) drückt oder auf Windows die Eingabeaufforderung schließt (nicht alle Windows-Versionen von Subversion reagieren auf die Eingabe von Strg + Untbr). Wer keinen Zugang zum Internet hat, der testet Subversion mittels eines Zugriffs auf den eigenen Server. Für Leser, die keinen Zugriff auf einen eigenen Subversion-Server haben, wird in Anhang A der lokale Zugriff auf ein Repository als Alternative beschrieben. Wer stattdessen selbst einen

Subversion-Server betreiben möchte, findet dazu eine ausführliche Anleitung in Abschnitt 8.1, »Einen Subversion-Server aufsetzen«.

4.6 Zusammenfassung

Nach den theoretischen Überlegungen zum Entwicklungszyklus mit Subversion und der Installation der Software ist es nun an der Zeit, erste praktische Schritte mit dem System durchzuführen. Das nachfolgende Kapitel startet mit einer einfachen Beispielsitzung.

Index

T