Seminarausarbeitung
Evolution von Softwaresystemen
Thema : Konfigurationsmanagement
Vortrag über RCS und CVS
nach dem Buch Programmierwerkzeuge von Andreas Zeller

1 Revisionsverwaltung mit RCS

1.1 Einführung

Im Bereich der Softwareentwicklung werden Werkzeuge benötigt, die es ermöglichen, die an einem Softwareprodukt auftretenden Änderungen zu kontrollieren und zu organisieren. Mit einem solchen Werkzeug, hier speziell RCS, eine Abkürzung für **Revision Control System** (darauf aufbauend auch CVS) muß man eine einmal erzeugte Konfiguration wieder herstellen können, wobei man mit Konfiguration eine Menge von Softwarekomponenten in bestimmten Versionen meint. Dies ist wichtig, weil die Entwickler oftmals sehen möchten, was sich im Laufe der Zeit an einem Programm geändert hat. Die Änderungen an einer Komponente müssen irgendwo gespeichert werden, weil es denkbar ist, daß mehrere Entwickler an einer Komponente gleichzeitig arbeiten und man so auftretende Konflikte wieder beheben kann. Jede Version jeder Komponente wird mit einer Nummer gekennzeichnet, weil man jederzeit die erzeugten Änderungen identifizieren können muß.
Bei Versionen wird ein Unterschied zwischen Revisionen und Varianten gemacht, wobei ersteres Versionen bezeichnet, die an Stelle von existierenden Versionen erzeugt wurden, diese also ersetzen. Sie entstehen hauptsächlich bei der Wartung des Programms. Varianten sind Versionen, die existierende Versionen erweitern. Hier wird die alte Version beibehalten.

1.2 Revisionen speichern und wiederherstellen

Alle Versionen werden in einem zentralen Archiv, dem Versions-Archiv verwaltet, um die Idee der Rekonstruktion verwirklichen zu können. Dabei werden die einzelnen Revisionen über zeitliche Aspekte voneinander unterschieden. Zu den Aufgaben des Revision Control Systems gehört unter anderem, daß die Versionen im Versionsarchiv abgelegt werden können, als auch, daß diese Versionen wieder hergestellt werden können. Jeder Entwickler hat seinen eigenen Arbeitsbereich, der englisch workspace genannt wird. Die sogenannten RCS-Dateien im RCS-Verzeichnis werden von allen Entwicklern geteilt.
Jede einzelne Revision erhält dabei eine Revisionsnummer (Identifizierung), die aus einem Paar zweier Zahlen n.m besteht und die beim check in automatisch vergeben werden. Betrachtet man die so entstandene Ordnung unter den Revisionen, wird man erkennen, daß dies einen Revisionsbaum ergibt, der im einfachsten Fall eine lineare Kette darstellt. Für den RCS-Befehl co (checkout), der eine Datei in einer bestimmten Version aus dem Archiv in den Arbeitsbe-

reich kopiert, hat man in der Regel vier Angabemöglichkeiten. Zum einen kann man die Revisionsnummer mit angeben, die man im zweiten Fall auch verkürzen kann. Ebenfalls möglich ist das Weglassen der Revisionsnummer, wobei dann die letzte Revision genommen wird. Als vierte Möglichkeit kann man einen Zeitpunkt mit angeben.

1.3 Äste und Varianten

Man muß zwei Änderungsarten unterscheiden. Zum einen macht man als Entwickler Fehlerkorrekturen und zum anderen wird man ein Software-Produkt ständig erweitern wollen, wobei wieder neue Fehler entstehen. Deshalb ist eine Revisionskette hier eher sehr unrealistisch. Man erhält in der Regel einen echten Revisionsbaum. Ganz wichtig zu erwähnen ist es, daß es das Konzept der Verzweigung gibt, das das Ziel hat, diese beiden Änderungsarten voneinander zu trennen. Der Nebenpfad wird für die Fehlerkorrekturen benutzt, während der Hauptpfad den Weiterentwicklungen dient.

1.4 Sperren

Ein Problem gibt es, wenn zwei oder mehrere Entwickler die gleiche Revision bearbeiten wollen. Es wäre hierbei leicht möglich, die Arbeiten des oder der anderen zu überschreiben. Aus diesem Grund gibt es bei RCS das Konzept des Sperrens. Es ist nur möglich, eine Revision zu erweitern, wenn man sie zuvor gesperrt hat. Man braucht sich allerdings nicht um das Setzen der Sperren zu kümmern, weil dies beim check out automatisch geschieht. Ebenso wird die Sperre beim check in wieder freigegeben. Aus diesem Grund nennt man die Vorgehensweise beim RCS auch pessimistische Kooperationsstrategie, weil man von vorneherein mit Konflikten rechnet.

1.5 Änderungen vollziehen

Hat man die Datei bearbeitet und ist man an dem Punkt angekommen, an dem man seine Änderungen nun den anderen zugänglich machen will, wird man den RCS-Befehl ci (check in) aufrufen. Dieser wird immer nach dem Abschluß der Änderungen ausgeführt und er erzeugt eine neue Revision mit den Neuerungen. Hierbei ist ein Eintrag für das Änderungsprotokoll möglich, damit man später weiß, aus welchem Grund die neue Revision erzeugt wurde. Beim check in wird immer die nächste freie Revision im Hauptpfad gewählt, falls keine Revisionsangabe vorhanden ist. Gibt man eine Revisionsnummer an, so wird ein Nebenpfad eingerichtet. Dies kann man dazu nutzen, um Sperren zu umgehen.

1.6 Änderungsprotokolle und Schlüsselwörter

Im Laufe der Zeit haben verschiedene Entwickler eine Anzahl von Revisionen dadurch erzeugt, daß sie jedesmal die neuen Dateien mit den darin enthaltenen Änderungen eingecheckt haben. Bei jedem Einchecken wurde ein Kom-

mentar hinzugefügt, wie z.B. Fehler beseitigt. Mit dem Befehl rlog kann man sich nach einer bestimmten Zeit alle Einträge des Änderungsprotokolls ansehen. Dies ist sehr nützlich, weil man sich anhand des Protokolls über die Entwicklungsfortschritte informieren kann. Ebenfalls möglich ist es, mit Hilfe von Schlüsselwörtern Versionsinformationen in Dateien abzulegen. Diese Schlüsselwörter werden in Dollarzeichen eingeschlossen und beim check out automatisch expandiert. Als Beispiele könnte man etwa Revision, Id oder Log angeben.

1.7 Wie RCS Revsionen ablegt

Wird eine Datei sehr oft geändert und somit auch sehr oft eingecheckt, müssen diese unterschiedlichen Revisionen von RCS irgendwie gespeichert werden. Dies können unter Umständen Hunderte von Revisionen sein. Man hätte natürlich die Möglichkeit, alle Revisionen komplett zu speichern, doch dies hätte unweigerlich einen Platzmangel zur Folge. Eigentlich bräuchte man nur die Stellen der Datei zu speichern, an denen sich die Revisionen von ihren Vorgängerrevisionen unterscheiden. Genau das macht RCS. Es wird jeweils nur die letzte Revision des Hauptpfades ganz gespeichert und ansonsten werden nur die Unterschiede zwischen früheren Revisionen abgelegt. Diese Unterschiede werden durch einen DIFF-Algorithmus erzeugt, der sich zeilenweise durch jeweils zwei Dateien durcharbeitet. Die Änderungen werden aus Gründen der Effizienz rückwärts bestimmt, weil auf neuere Revisionen in der Regel viel häufiger zugegriffen wird als auf ältere. Innerhalb der Verzweigungen werden die Änderungen vorwärts abgelegt. RCS benutzt eine PATCH-Algorithmus, um die Änderungen anzuwenden.

1.8 Neuerungen und Weiterentwicklungen

Werkzeuge des Software-Konfigurationsmanagement benutzen meistens eine verschränkte Darstellung, bei der man auf alle Revisionen gleich schnell zugreifen kann. Ein Beispiel für solch eine Darstellung wäre etwa das SCCS, das Source Code Control System. RCS bietet auch einige Erweiterungen wie z.B. graphische Benutzeroberflächen oder den RCE, den Revision Control Engine an, wobei die Vorteile von RCE darin liegen, daß man es von fremden Programmen aus bedienen kann und daß es eine Programmierschnittstelle für RCS-ähnliche Funktionen darstellt. Man kann eine Revision über ihre Kennung öffnen. Als Nachteil hätte man dann aber die Tatsache, daß Programme angepaßt werden müssen. Ein weiterer Fortschritt ist die Abschaffung von Dateisystemen, die durch Datenbanken ersetzt werden

2 Parallele Programmentwicklung mit CVS

2.1 Von Komponenten zu Software-Systemen

Ein weiteres Werkzeug, das auf RCS aufbaut, wäre das CVS, das Concurrent Versions System. Auch dies dient der Versionskontrolle. In der Praxis besteht ein Software-Projekt nicht nur aus einer einzelnen Datei, sondern aus zahlreichen Komponenten, an denen Tausende von Entwicklern ununterbrochen arbeiten. Diese Komponenten sind unterteilt in die Schnittstellen, die die Deklaration darstellen und bestimmte wohldefinierte Dienste anbieten und in die Realisierung oder Implementierung, die die Dienste anderer Komponenten in Anspruch nimmt. Es muß nun auf folgendes hingearbeitet werden : Konsistenz und Flexibilität, wobei ersteres bedeutet, daß die Änderungen miteinander verbunden sein müssen und letzteres, daß beliebig viele Versionen einer Komponente ausgwählt werden können, um diese für eine bestimmte Umgebung passend zu machen. Der Idealfall wäre natürlich, daß über die gesamte Lebensdauer des Systems keine Veränderung der Schnittstellen stattfindet. Allerdings ist dieser Idealfall so gut wie nie gegeben. Man hat also eine Inkonsistenz. Es ist mit CVS möglich, Konfigurationen zu erzeugen, indem man bestimmte Revisionen mehrerer Komponenten zusammenfaßt.

2.2 Dateibäume versionieren mit CVS

Es gibt ein paar grundlegende Unterschiede zum RCS. Beim CVS betrachtet man nicht nur Änderungen in einzelnen Dateien. Es ist möglich, Dateien zu löschen oder hinzuzufügen. Das ganze Software-Projekt wird als eine große Einheit versioniert. Vor der Bearbeitung braucht man sich nicht um irgendwelche Sperren zu kümmern. Dies ist hier nicht nötig. CVS arbeitet auf ganzen Dateibäumen. Um sämtliche Revisionen zu speichern, benutzt CVS ein zentrales Archiv, das sogenannte Repository. Hier werden alle Revisionen sämtlicher Quelldateien verwaltet. Dieses Archiv besteht aus einer Menge von RCS-Archiven. Um mit den dort abgelegten Dateien, bzw. Revisionen der Dateien, arbeiten zu können, gibt es zahlreiche CVS-Befehle. Ich will hier nur einige davon kurz aufzählen. Um Dateien aus dem Archiv in den eigenen Arbeitsbereich zu kopieren, würde man den Befehl check out benutzen. Das Zurückkopieren einer neu erzeugten Revision geschieht mit dem commit-Befehl, der auf einzelne Dateien anwendbar ist. Hier sollte ein Kommentar mit angegeben werden. Es wird dann im CVS-Archiv eine neue Revision der Datei angelegt. Man kann auch völlig auf die Angabe von Dateien verzichten. Möchte man eine Konfiguration mit einem Namen versehen, so wird man den CVS-Befehl tag benutzen. Mit Hilfe des Befehls add kann man Quelldateien hinzufügen. Nach Beendigung der Arbeit ruft man den Befehl release auf, der überprüft, ob alle Änderungen ins Archiv übernommen wurden. Ansonsten wird eine Warnung ausgegeben. Alle diese Befehle werden mit dem Präfix cvs eingeleitet.

2.3 Wie CVS arbeitet

Wie schon gesagt, baut CVS auf RCS auf, es benutzt das RCS-System. Für jedes Projekt wird ein eigenes CVS-Archiv verwaltet. Dieses besteht aus einem Dateibaum mit den gleichen Verzeichnissen und Unterverzeichnissen wie im Original-Projekt. An die Stelle der Original-Dateien treten jetzt RCS-Dateien. Wird der CVS-Befehl check out ausgeführt, so werden die Verzeichnisse mit der gleichen Struktur wie im CVS-Archiv angelegt. Es werden dabei die jeweiligen Revisionen konstruiert und in den Arbeitsbereich gestellt. CVS kennt die Revisionen, die sich im Arbeitsbereich befinden. Deshalb kann ein commit auch korrekt ausgeführt werden. Es wird eine Datei CVS/Entries verwaltet, in der alle Nummern der Revisionen und die checkout-Zeiten festgehalten werden. Will man eine Datei löschen, so ruft man cvs remove plus den Dateinamen auf. Dadurch wird die entsprechende Datei aus dem CVS-Archiv entfernt. Allerdings geht im CVS keine Datei verloren, auch keine gelöschte Datei. Diese Dateien landen im Unterverzeichnis Attic (Dachkammer). Man kann diese jederzeit wieder herstellen.

2.4 Parallele Programmentwicklung

Man braucht beim Auschecken die Dateien nicht zu sperren. Dies liegt daran, daß beim CVS paralleles Arbeiten ausdrücklich zugelassen ist. Dies bringt einige Vorteile mit sich, aber auch Nachteile. Voraussetzung für das parallele Arbeiten ist, daß die Entwickler ständig ihr CVS-Archiv abgleichen. Man muß als Entwickler also ständig auf dem neuesten Stand sein. CVS geht davon aus, daß die Entwickler sich ihre Arbeit selber aufteilen. Deshalb soll es kaum zu Konflikten kommen, was in Wirklichkeit selten der Fall ist. Die Vorgehensweise in CVS wird auch als optimistische Kooperationsstrategie bezeichnet, weil man davon ausgeht, daß kaum Konflikte entstehen.
Es gibt den CVS-Befehl update, um seinen Arbeitsbereich auf den neuesten Stand zu bringen. Dabei werden geänderte Dateien erneut herauskopiert und somit aktualisiert. Ein commit sollte man erst nach einem update machen, um nicht wichtige Änderungen zu überschreiben. Würden zwei Entwickler die gleiche Komponente zur gleichen Zeit ändern, würden die Änderungen beim update verlorengehen, aber update ist so organisiert, daß es die Änderungen integriert. Hierbei wird die Revision nach dem letzten check out zugrundegelegt. Nach einem solchen Vorgang muß immer geprüft werden, ob die Arbeitsumgebung noch konsistent ist.

2.5 Konflikte und Absprachen

Wird dieselbe Stelle derselben Datei geändert, kommt es zwangsläufig zu einem Konflikt. CVS geht nun so vor, daß die Konflikt-Stelle markiert wird und der Konflikt von Hand gelöst werden muß. CVS stellt eine Vorbeugemaßnahme zur Verfügung. Ein Entwickler kann mitteilen, daß er eine Datei beobachtet, ebenso kann ein anderer Entwickler mitteilen, daß er die Datei benutzt. Es werden dann

alle Beobachter der Datei benachrichtigt. Im Text wird allerdings nicht gesagt, wie diese Benachrichtigungen aussehen. Werden E-Mails versendet, oder wie funktioniert das ? Anmelden und Beobachten geschehen mit dem Befehl edit. Die Datei wird zum Bearbeiten angemeldet und gleichzeitig wird der Benutzer als Beobachter eingetragen. Zusätzlich gibt es den CVS-Befehl watch, der eine Datei nur zum Lesen freigibt. Um diese zu ändern, muß edit aufgerufen werden. Dieser Mechanismus des Beobachtens ist ein Mittelweg zwischen pessimistischer und optimistischer Kooperationsstrategie.

3 Kritik

In diesem Text wird nicht beschriebn, wie man RCS oder CVS überhaupt installiert oder wie man das Versions-Archiv anlegt. Es wird auch nicht auf irgendwelche Probleme eingegeangen, wie z.B. wenn mehrere Entwickler verschiedene Betriebssysteme haben. Kann das Versions-Archiv auch auf einem Rechner sein, der an einer anderen Stelle auf der Welt Tausende von Kilometern entfernt steht, sprich kann man CVS auch übers Netz betreiben ? Wie informieren sich die Entwickler gegenseitig, um einen Konflikt zu verhindern ? Gibt es dort nicht irgendwelche Sicherheitsrisiken, wenn sich z.B. jemand unerlaubt Zutritt zum Archiv verschafft und Daten löscht ? Wie werden die Daten übers Netz verschickt ? Auf alle diese Fragen und noch etliche mehr geht der Text nicht ein. Dies würde aber wahrscheinlich auch den Rahmen des Buches sprengen.