Open Source-Versionsverwaltungssysteme: Konzepte und Anwendungen

Ralf Geschke
<ralf@kuerbis.org>

Hausarbeit im Fach
Betriebsinformatik III

Version 1.1 vom 30.11.2008

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Vorwort

Die vorliegende Arbeit entstand im Rahmen der Veranstaltung „Betriebsinformatik III“ im Sommersemester 2008 an der Fachhochschule für Oekonomie & Management (FOM). Bis auf das Deckblatt und dieses Vorwort ist sie im Vergleich zur vorgelegten und benoteten Fassung unverändert geblieben. Wie allgemein bekannt, schreitet die Entwicklung insbesondere im Bereich Open Source Software stetig voran. So wurden seit Erstellung dieses Textes teilweise mehrere neue Versionen der hier begutachteten Systeme veröffentlicht. Diese beheben Fehler oder wurden in wesentlichen Punkten verbessert, z.B. im Bereich der Anwendungs--Performance oder der zur Verfügung stehenden Dokumentation. Für eine Evaluierung des Einsatzes wäre somit eine neuerliche Betrachtung der aktuellen Versionen notwendig. Dennoch bietet der Text eine Übersicht der Grundlagen und eine Einführung in die nach wie vor gültigen Konzepte und eine Darstellung des Ist-Standes zum Zeitpunkt Juli 2008.

Erftstadt, im November 2008

Ralf Geschke

1. Einleitung

Versionsverwaltungssysteme sind aus der modernen Softwareentwicklung nicht mehr wegzudenken. Dabei reicht der Einsatzbereich von kleinen Ein-Mann-Projekten bis zur weltweit verteilten Programmierung umfangreicher Open Source-Software wie Linux oder Firefox oder innerhalb kommerzieller Softwareprojekte. In der folgenden Arbeit findet sich eine Darstellung der Konzepte und Anwendungen von Open Source-Versionsverwaltungssystemen. Nach der Begriffsbestimmung und Einordnung folgt zunächst eine Betrachtung unter wirtschaftlichen Aspekten, anschließend wird der Begriff Open Source erörtert. Danach werden Anwendungsgebiete skizziert sowie Konzepte und Grundlagen des Arbeitsablaufs mit Versionsverwaltungssystemen erläutert. Die letzten Kapitel befassen sich einem näheren Blick auf die verschiedenen Systeme und Hinweisen zur Entscheidungsfindung, den Abschluss bildet ein Fazit und eine Handlungsempfehlung.

2. Versionsverwaltungssysteme

2.1. Begriffsbestimmung

Unter einem Versionsverwaltungssystem wird Software verstanden, mit der sich die Veränderung von Dateien nachvollziehen lässt. Dabei werden die Veränderungen protokolliert, so dass der Zugriff auf unterschiedliche Versionen einer Datei erfolgen kann. Diese Versionen sind sowohl einem Zeitpunkt als auch einer Person zuzuordnen[1]. Dabei kann jederzeit auf jede frühere Version der jeweiligen Datei zurückgegriffen werden. Versionsverwaltungssysteme werden ebenfalls zur Lösung des Problems der gleichzeitigen, sich gegenseitig überschreibenden Änderungen aufgrund des gleichzeitigen Zugriffs mehrerer beteiligter Personen eingesetzt[2]. Weiterhin dienen Versionsverwaltungs-systeme zur Kommunikation zwischen Mitarbeitern, da zusätzliche Informationen zur jeweiligen Version gespeichert werden[3]. Die übergreifende Disziplin ist das Software Configuration Management (SCM), wozu die Teilbereiche Versionskontrolle, Änderungs-management (Change Management) , Build- und Release Management gehören[4] [5].

2.2. Einordnung Software-Entwicklungsprozess

Die Unterteilung in verschiedene Phasen ist eine “wesentliche Voraussetzung zur

1 Vgl. Baerisch (2005), S. 6

2 Vgl. Hill (2005), S. 1

3 Vgl. Hill (2005), S. 1

4 Vgl. Hill (2005), S. 3

5 Vgl. Sommerville (2007), S. 739 ff.

wirtschaftlichen Durchführung von Projekten"[6]. Allgemein gliedern sich Projekte in die Phasen Problemanalyse, Konzeption und Grundlegung, Ausarbeitung von Details und Gestaltung, Realisierung sowie Nutzung. Innerhalb der Realisierungsphase findet dabei die Programmierung statt[7]. Eine Ableitung des allgemeinen Modells für die Entwicklung von Software-Projekten stellt das Wasserfall-Modell dar. Dieses auch Softwarelebenszyklus genannte Modell besteht aus einer Kaskade von Phasen, die in sich abgeschlossen sind und aufeinander aufbauen. Nach der Analyse und Definition der Anforderungen, die die Systemspezifikation bilden, folgt der System- und Softwareentwurf, wodurch die allgemeine Systemarchitektur festgelegt wird. Anschließend wird innerhalb der Implementierungs- und Komponententest-Phase der Entwurf durch die Erstellung von Programmen umgesetzt. Dazu gehört auch ein Komponententest, der sicherstellt, dass die Spezifikationen erfüllt werden. Nach der Zusammenführung der Programme in der Integrations- und Systemtest-Phase wird das System in Betrieb genommen. In dieser Phase ist weiterhin die Wartung anzusiedeln, wozu neben Fehlerkorrektur auch die Verbesserung des Systems aufgrund neuer Anforderungen gehört[8]. Versionsverwaltungssysteme werden dabei während der Implementierungsphase eingesetzt. Diese Phase ist gekennzeichnet durch die Erstellung und Bearbeitung von Quellcode-Dateien[9]. Der Einsatz ist dabei nicht auf ein bestimmtes Vorgehensmodell beschränkt, denn eine Implementierungs- oder Entwicklungsphase existiert in jedem Modell, etwa der evolutionären Entwicklung, im komponentenbasierten Software-Engineering, der inkrementellen Entwicklung[10] oder auch agilen Methoden[11].

2.3. Wirtschaftliche Betrachtung

Die wirtschaftlichen Vorteile bei der Nutzung von Versionsverwaltungssysteme stützen sich auf zwei Aspekte. Zum einen sind die Lizenzkosten der hier betrachteten Systeme zu nennen, eine nähere Betrachtung dazu folgt im nächsten Abschnitt. Zum anderen entstehen Vorteile während des Softwareentwicklungsprozesses an sich. Bei der Bearbeitung der Quelltexte können Fehler auftreten, etwa eine falsche oder fehlerhafte Implementierung beim Einbau neuer Funktionen, oder versehentliches Löschen von Textabschnitten und ganzen Dateien. Diese unerwünschten Änderungen müssen zur weiteren Bearbeitung festgestellt und rückgängig gemacht werden, was eine zentrale Aufgabe von

6 Litke (2007), S. 28

7 Vgl. Litke (2007), S. 27

8 Vgl. Sommerville (2007), S. 96 f.

9 Vgl. Baerisch (2005), S. 6

10 Vgl. Sommerville (2007), S. 98 ff.

11 Vgl. Sommerville (2007), S. 430 ff.

Versionsverwaltungssystemen darstellt[12] [13]. Dadurch haben Versionierungssysteme Projekten "viele Personenjahre unnützer Arbeit erspart"[14].

Anhand des folgenden Szenarios soll die Zeit- und somit Kostenersparnis eines typischen Anwendungsfalls verdeutlicht werden. Ein Entwicklerteam besteht aus zehn Programmierern, davon fünf Festangestellte und fünf Freiberufler. Alle Entwickler arbeiten in einem gemeinsamen Verzeichnis. Wer welche Quelldatei bearbeitet, wird durch persönliche Absprache geregelt. Ein Entwickler löscht versehentlich eine zentrale Datei, was einen Testlauf des Systems verhindert. Ohne dieses Testverfahren ist die Weiterentwicklung nicht möglich, da die umfassenden Änderungen des letzten Entwicklungszyklus geprüft werden müssen. Ohne ein Versionsverwaltungssystem muss die fehlende Datei aus dem Backup wiederhergestellt werden. Diese Aufgabe ist den Entwicklern meist nicht möglich, sondern erfolgt durch die Systemadministratoren. Eine Anfrage bei einem Administrator eines mittelständischen Unternehmens ergab, dass der vollständige Backup-Prozess von Benachrichtigung über das Auffinden und Wiederherstellen einer Datei ungefähr 20 Minuten in Anspruch nimmt. In dieser Zeit steht der Entwicklungsprozess weitestgehend still, was einen Ausfall von zehn Mitarbeiten à 20 Minuten bedeutet. Während der Stundensatz respektive die Kosten pro Lohnminute für fest angestellte Mitarbeiter eines Unternehmens von vielen Faktoren abhängig ist und nur für das jeweilige Unternehmen berechnet werden kann, ist die Ermittlung der Stundensätze von Freiberuflern transparenter. In der Auswertung des Online-Portals zur Projektvermittlung GULP[15] von Februar 2008 wird ein durchschnittlicher Stundensatz von 71 Euro genannt[16]. Damit ergibt sich nur für den Ausfall der Freiberufler im Projekt folgende Rechnung.

Ausfallzeit		20 min
Stundensatz	71 EUR / h	1,18 EUR / min
Anzahl Freiberufler		5
Ergebnis		118,3 EUR

Tabelle 1: Beispielsrechnung Verlust durch Ausfallzeit

Das versehentliche Löschen der Datei hätte das Unternehmen aus dem Beispiel 118,30 EUR gekostet. Unberücksichtigt bleiben hierbei die Kostensätze der fest angestellten Mitarbeiter sowie der Administrationsaufwand, welcher intern verrechnet wird. Dieses einfache Beispiel verdeutlicht bereits die Vorteile, die sich durch den Einsatz von

12 Vgl. Baerisch (2005), S. 7

13 Vgl. Lau (2008), S. 182

14 Lau (2008), S. 182

15 http://www.gulp.de

16 Vgl. http://www.gulp.de/kb/st/stdsaetze/sstext_f.html, Stand 03.07.2008

Versionsverwaltungssoftware ergeben, denn die Wiederherstellung einer Datei aus dem Versionssystem dauert nur wenige Sekunden.

2.4. Open Source

Open Source Software ist zunächst Software, die den Bestimmungen der Open Source Definition der Open Source Initiative (OSI)[17] unterliegt[18]. Die Nutzungsbedingungen werden dabei durch verschiedene Lizenzformen geregelt. Die grundlegenden Eigenschaften dieser Software sind die Benutzbarkeit ohne Einschränkungen, die Verfügbarkeit des Quellcodes, die Weitergabe ohne Einschränkungen und Zahlungsverpflichtungen sowie die Möglichkeit zur Veränderung der Software und Weitergabe in der veränderten Form[19]. Während diese Aspekte vor allem für das Verständnis des Konzeptes Open Source eine Rolle spielen, sind beim Einsatz in den Unternehmen weitere Merkmale von entscheidender Bedeutung. Bereits durch den Wegfall der Lizenzkosten ergibt sich häufig ein geringerer Total Cost of Ownership (TCO). Der Anteil der Lizenzkosten an der TCO wird zwar als vergleichsweise gering eingeschätzt, dennoch finden sich in Vergleichsstudien Angaben zur Ersparnis zwischen 19 Prozent und 36 Prozent gegenüber kommerziellen Softwareanbietern.. Dabei gilt, dass das Einsparpotential stark vom Einsatzgebiet der Software abhängt[20].

3. Anwendungen

3.1. Softwareentwicklung

Primär werden Versionsverwaltungssysteme in der Softwareentwicklung eingesetzt. Das System dient dabei als zentraler Datenspeicher für alle Quelldateien der beteiligten Softwareentwickler. Verschiedene Versionen der Dateien können jederzeit wiederhergestellt werden. Versionsverwaltungssysteme unterstützen die Arbeit an mehreren Entwicklungszweigen, etwa zur Pflege einer als stabil gekennzeichneten Programmversion sowie einer Variante, in der sämtliche neuen Entwicklungen durchgeführt werden. Die Verteilung des zu bearbeitenden Quellcodes an die Programmierer ist ebenfalls Aufgabe eines Versionskontrollsystemes. Des weiteren kann eine gleichzeitige Bearbeitung derselben Quellcode-Datei stattfinden. Konflikte, die durch diese Arbeiten entstehen, werden weitestgehend durch das Versionskontrollsystem

17 http://www.opensource.org

18 Vgl. Grassmuck (2002), S. 230 f.

19 Vgl. Grassmuck (2002), S. 233

20 Vgl. Vetter et al. (2007), S. 206

behoben, so dass nur in wenigen Fällen ein manueller Eingriff notwendig ist[21]. Versionsverwaltungssysteme ersetzen jedoch nicht die Kommunikation der Teammitglieder untereinander. Falls etwa bei der Bearbeitung von Projekten grundlegende Dateien geändert werden müssen, ist eine Absprache im Vorfeld sinnvoll. Dabei können alle Arten von Kommunikationsmedien eingesetzt werden, von persönlichen Gesprächen bis hin zu E-Mail[22].

3.2. Weitere Anwendungsgebiete

Zwar ist das Hauptanwendungsgebiet die Softwareentwicklung, jedoch stehen durch erweiterte Funktionalitäten, die über den Umgang mit Quelltext-Dateien hinaus gehen, weitere Möglichkeiten offen. Moderne Versionsverwaltungssysteme können für alle Arten von Dateien, dies betrifft auch Binärdateien, eingesetzt werden. Dadurch erschließen sich Anwendungsgebiete wie die Versionsverwaltung für den Einsatz im Design-Bereich zur Verarbeitung von Grafik-Dateien[23] sowie das Management für allgemeine Texte, von Diplomarbeiten bis hin zu Büchern. Eine weitere Anwendung ist der Einsatz als Content Management System für eine Website. Die Dokumente liegen dabei an zentraler Stelle vor, ähnlich einem Redaktionssystem können die Dateien, aus denen die Website besteht, von mehreren Benutzern bearbeitet werden. Die Archivfunktion ist dabei integraler Bestandteil von Versionsverwaltungssystemen[24]. Ein eher ungewöhnlicher Bereich ist die Verwendung in einer Applikation zur Aufnahme des Baufortschritts in verteilter Umgebung. In diesem Konzept dient ein zentraler Objektspeicher zur Aufnahme von versionierten Objekten[25]. Diese Anwendung ist dem Bereich Projektmanagement zuzuordnen. Ebenfalls können Konfigurationsdateien, die bei der Systemadministration verwendet werden, mittels eines Versionsverwaltungssystems gespeichert werden[26].

4. Konzepte

Zwar unterscheiden sich die im weiteren Verlauf betrachteten Versionsverwaltungssysteme im Detail, jedoch existieren nur wenige Basiskonzepte, die die Grundlage der Systeme darstellen. Diese werden nachfolgend näher betrachtet.

21 Vgl. Budszuhn (2007), S. 30

22 Vgl. Budszuhn (2007), S. 350

23 Vgl. Hill (2005), S. 1

24 Vgl. Merkle (2006), S. 7 f.

25 Vgl. Schweigel (2004), S. 42 ff.

26 Vgl. Budszuhn (2007), S. 38

4.1. Grundlagen

4.1.1. Repository

Zur Speicherung der versionierten Dateien müssen diese an einem bestimmten Ort abgelegt werden. Dies kann mittels regulärer Dateien in einem Dateisystem geschehen, des weiteren kann eine Datenbank dafür benutzt werden[27] [28]. Dieser Speicherort wird als Repository bezeichnet. Gespeichert werden dabei abhängig vom System die jeweiligen abgelegten Dateien beziehungsweise die Unterschiede zwischen den verschiedenen Versionen einer Datei[29]. Des weiteren finden Meta-Informationen wie Kommentare und Eigenschaften der Dateien Platz[30].

4.1.2. Zugriff

Für den Lese- und Schreibzugriff auf die versionierten Dateien dienen wiederum abhängig vom verwendeten System spezielle Anwendungen, auch Clientprogramme genannt. Damit stehen dem Entwickler alle Funktionen des Versionsverwaltungssystems offen.

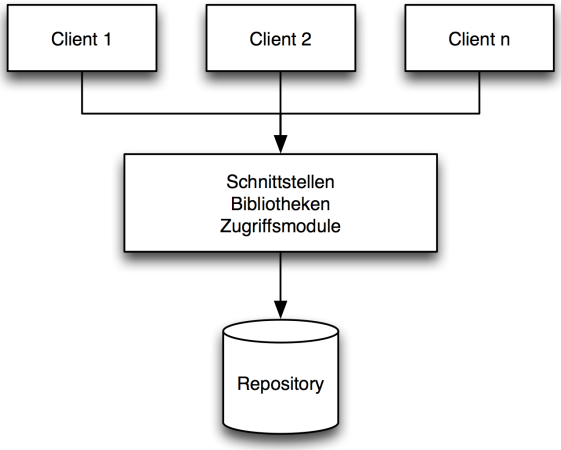

Abbildung 1: Repository-Zugriff

Ebenfalls abhängig vom System erfolgt der Zugriff über einen spezialisierten Server, in dem die Zugriffsmodule integriert sind, oder mittels Schnittstellen und Bibliotheken, die im Clientprogramm implementiert sind[31]. Für jedes System existieren mehrere Arten von Clientprogrammen, diese können aus einer Kommandozeilen-Anwendung bestehen, des

27 Vgl. Baerisch (2005), S. 32

28 Vgl. Budszuhn (2007), S. 39 f.

29 Vgl. Baerisch (2005), S. 32

30 Vgl. Budszuhn (2007), S. 28

31 Vgl. Budszuhn (2007), S. 38 f.

weiteren gibt es grafische Frontends[32] [33]. Auch in Entwicklungsumgebungen (IDEs) gibt es Unterstützung zur Versionsverwaltung, so dass sich diese Systeme fest in den Arbeitsablauf der Programmerstellung integrieren[34].

4.1.3. Arbeitskopie

Bei der Arbeitskopie handelt es sich um eine lokale Kopie der zur Entwicklung notwendigen Dateien. Diese können das gesamte Repository oder eine Teilmenge davon umfassen. Die Bearbeitung der Quelldateien findet ausschließlich innerhalb der Arbeitskopie statt[35]. Ebenso werden neu angelegte Dateien zunächst innerhalb der Arbeitskopie angelegt und in einem zweiten Schritt in das Repository zur Versionierung eingestellt.

4.1.4. Versionierungsstrategien

Während der Arbeit in einem Projekt kann es dazu kommen, dass mehrere Entwickler gleichzeitig an denselben Dateien arbeiten. Daher kann es passieren, dass die Änderungen in den Dateien gegenseitig überschrieben werden. Eine Aufgabe des Versionsverwaltungssystems ist es, diese bei der Bearbeitung entstehenden Konflikte zu verhindern beziehungsweise eine Lösungsmöglichkeit zu schaffen. Dazu gibt es zwei unterschiedliche Ansätze, die Sperren-Verändern-Entsperren-, auch Lock-Modify-Unlock-Lösung genannt, sowie die Kopieren-Verändern-Zusammenführen-Lösung, auch Copy-Modify-Merge bezeichnet[36] [37] [38].

Beim Sperren-Verändern-Entsperren wird eine Datei von einem Entwickler zur Bearbeitung als gesperrt markiert. Andere Entwickler können nur lesend auf die Datei zugreifen, diese jedoch nicht selbst verändern. Erst wenn die Bearbeitung der Datei durch den ersten Entwickler abgeschlossen und diese ins Repository überführt worden ist, wird sie wieder für andere Entwickler freigegeben. Damit wird eine gleichzeitige Änderung bereits im Vorfeld verhindert.

Die Kopieren-Verändern-Zusammenführen-Lösung arbeitet grundlegend anders. Hierbei erfolgt keinerlei Sperrung im Versionsverwaltungssystem. Die Entwickler können gleichzeitig an derselben Datei arbeiten. Wenn die veränderte Datei ins Repository gestellt

32 Vgl. Budszuhn (2007), S. 34 ff.

33 Vgl. Merkle (2005), S. 5 ff.

34 Vgl. Budszuhn (2007), S. 36 f.

35 Vgl. Baerisch (2005), S. 9 f.

36 Vgl. Hill (2005), S. 10 ff.

37 Vgl. Merkle (2006), S. 2

38 Vgl. Collins-Sussman et al. (2007), S. 2 ff.

wird, erfolgt innerhalb dieses Prozesses durch den Server eine Prüfung, ob die Datei zwischenzeitlich durch einen anderen Entwickler verändert wurde. In diesem Fall wird der Vorgang abgebrochen und der Entwickler entsprechend informiert. Währenddessen versucht das System, die veränderte Datei aus dem Repository mit der bearbeiteten Datei innerhalb der Arbeitskopie zusammenzuführen. Gelingt dies nicht automatisch, muss der Entwickler die Zusammenführung manuell durchführen. In diesem Konfliktfall obliegt dem Entwickler die Entscheidung, welche Änderungen tatsächlich in das Repository übernommen werden sollen und welche verworfen werden. Die automatisierte Zusammenführung ist nur mit Textdateien möglich, daher ist bei der Bearbeitung von Binärdateien, etwa bei der Arbeit mit Grafikdateien, die Sperren-Verändern-Entsperren-Lösung vorzuziehen.

4.1.5. Kennzeichnung von Entwicklungsständen

Versionskontrollsysteme bieten die Möglichkeit, den vollständigen Stand eines Projektes im Ganzen zu kennzeichnen. Diese Kennzeichnung wird durch Markierungen, auch Tags genannt, realisiert. Eine Anwendung kann der Übergang von der Implementierungs- in die Betriebsphase sein. Damit lässt sich die Weiterentwicklung von denjenigen Ergänzungen abgrenzen, die in der Wartungsphase durchgeführt werden. Eine weitere Anwendung ist die Kennzeichnung von Versionsständen, die Kunden zur Prüfung vorgelegt werden[39].
Ein ähnliches Verfahren ist die Verzweigung von Projekten. Dies wird mittels Branches umgesetzt. Diese dienen dazu, parallel die Bearbeitung an alternativen Entwicklungslinien zu ermöglichen. Beispielsweise werden neue Bestandteile zu einem Projekt in einem experimentellen Entwicklungszweig hinzugefügt, während die normale Weiterentwicklung im allgemeinen Zweig stattfindet. Zu gegebener Zeit können diese Zweige wieder zusammengeführt werden, wobei sich widersprechende Änderungen als Konflikte auftreten und dementsprechend gelöst werden müssen[40] [41].
Die Unterschiede zwischen Tags und Branches bestehen vor allem in der Definition und im praktischen Einsatz.

4.1.6. Systemarchitektur

Grundlegend lassen sich Versionsverwaltungssysteme in zwei Kategorien einteilen, den zentralisierten und dezentralen Ansatz. Dabei betreffen die Unterschiede gleichermaßen das Entwicklungsmodell. Bei einem zentralen System gibt es ein zentrales Repository zur

39 Vgl. Hill (2005), S. 25

40 Vgl. Baerisch (2005), S. 13 f.

41 Vgl. Hill 82005), S. 25

Datenspeicherung. Die Entwickler beziehen ihre Arbeitskopie aus diesem Repository. Der Zugriff auf das Repository wird oftmals durch einen Serverprozess geregelt. Nachdem die Entwickler Dateien innerhalb der Arbeitskopie geändert haben, werden diese Änderungen in das Repository übertragen und stehen allen Beteiligten zur Verfügung[42] [43]. Die wichtigsten Vertreter der zentralen Systeme sind CVS und Subversion.

Beim dezentralen Ansatz mittels verteilter Versionsverwaltungssysteme (DVCS) ist ein flexiblerer Arbeitsablauf denkbar. Jeder Entwickler besitzt ein eigenes Repository. Dies steht insbesondere auch lokal zur Verfügung, was das Arbeiten mit dem Versionsverwaltungssystem ohne Zugriff auf den Server ermöglicht[44]. Dezentrale Systeme können dabei auch ein weiteres Repository für den zentralen Zugriff enthalten, müssen dies aber nicht. Die Arbeitskopie entnimmt der Entwickler aus dem lokalen Repository. Verteilte Versionsverwaltungssysteme „betrachten ihren versionierten Datenbestand als Aggregierung von Veränderungen“[45].

Die Entwickler überführen die veränderten Dateien in ihr jeweils eigenes Repository. Dadurch entstehen Unterschiede zwischen den einzelnen Repositories. Zum Abgleich zwischen den Repositories werden diese Unterschiede in so genannten Changesets gesammelt. Die ChangeSets können wiederum zwischen den beteiligten Entwicklern ausgetauscht oder in einem weiteren Repository, welches den Hauptentwicklungszweig darstellt, zusammengeführt werden[46]. Bei der Arbeit mit verteilten Versionsverwaltungssystemen ist die Etablierung eines für das jeweilige Projekt zweckmäßigen Workflows durch den Projektleiter zwingend notwendig[47].

4.1.7. Atomare Operationen

Mittels atomarer Operationen wird sichergestellt, dass die Integrität von Arbeitskopie und Repository erhalten bleibt. Wenn ein Entwickler mehrere Dateien innerhalb der Arbeitskopie verändert hat und diese Änderungen ins Repository stellt, kann es zu Problemen kommen. Neben den durch Änderungen an denselben Dateien verursachten Konfliktfällen können dies auch Störungen während der Übertragung sein, die etwa durch Netzausfälle verursacht wurden. Eine Operation wird atomar genannt, wenn sichergestellt ist, dass die Änderungen aller Dateien erfolgreich übertragen wurden und somit vollständig sind, oder wenn im Fall des Scheiterns überhaupt keine Änderung erfolgt ist, weder in der

42 Vgl. Baerisch (2005), S. 32 f.

43 Vgl. Hill (2005), S. 4

44 Vgl. Lau (2008), S. 183

45 Baerisch (2005), S. 32 f.

46 Vgl. Baerisch (2005), S. 32 f.

47 Vgl. Lau (2008), S. 187

Arbeitskopie, noch im Repository. Dadurch wird verhindert, dass sich das Repository nach einer Operation in einem undefinierten Zustand befindet[48].

4.1.8. Weitere Merkmale

Es gibt weitere Aspekte, die bei der Differenzierung von Versionsverwaltungssystemen eine Rolle spielen. Dazu gehört die Unterstützung verschiedener Betriebssystem-Plattformen, dies betrifft sowohl den Client als auch Server. Die Laufzeitumgebung ist dann relevant, wenn in Unternehmen bereits Entwicklungsumgebungen existieren, in die das Versionsverwaltungssystem integriert werden muss. Ebenso stellt sich die Frage nach den notwendigen Hardware- und Netzwerk-Anforderungen. Bei Systemen, die sehr wenige Netzzugriffe benötigen, wird mehr lokaler Speicherplatz verwendet als von der Software, die für jede Aktion eine Datenübertragung initiiert[49]. Ebenfalls relevant ist der Aspekt der Skalierbarkeit, also das Verhalten unter Last beim gleichzeitigen Zugriff von vielen Personen. Daneben ist die technische Zuverlässigkeit wichtig, das System darf durch Fehlbedienung keinerlei Datenverlust erleiden. Zur Ausgereiftheit des Systems gehört weiterhin die Stabilität der Schnittstellen, sowohl in Bezug auf die Benutzung durch die Anwender, als auch im Zusammenhang mit der Entwicklung von zusätzlicher, unterstützender Software[50]. Nicht zuletzt spielen die Rechtevergabe und Authentifizierung sowie die sichere Datenübertragung eine Rolle. Der Aspekt der Granularität behandelt die Frage, wie fein die Zugriffsrechte abgestuft werden können, so dass den Entwicklern nur der Zugang zu den jeweils erlaubten Projekten, Verzeichnissen oder Dateien ermöglicht wird[51]. Da bei der Arbeit in verteilten Teams die einzelnen Mitglieder oftmals an verschiedenen Orten arbeiten, welche über das prinzipbedingt unsichere Internet verbunden sind, ist ferner eine Verschlüsselung der Datenübertragung erwünscht.

4.2. Arbeitsablauf

Da sich nicht nur die Konzepte, sondern auch Basisfunktionen verschiedener Versionsverwaltungssysteme ähnlich sind, wird im folgenden auf die wichtigsten Arbeitsvorgänge näher eingegangen. Zur Betrachtung kommt jeweils die Kopieren-Verändern-Zusammenführen-Strategie, da diese von den meisten modernen Versionsverwaltungssystemen unterstützt wird.

48 Vgl. Baerisch (2005), S. 28

49 Vgl. Baerisch (2005), S. 25 f.

50 Vgl. Baerisch (2005), S. 26 f.

51 Vgl. Baerisch (2005), S. 25

4.2.1. Erzeugung der Arbeitsumgebung

Diese Aufgabe gehört zum Bereich der Systemadministration und ist nur bei der Einführung eines Versionsverwaltungssystems oder je nach System zu Beginn eines Projektes notwendig. Dabei wird zunächst das zentrale Repository erstellt und eine grundlegende Konfiguration durchgeführt. Diese umfasst unter anderem die Festlegung der Zugriffsmethoden, die Einrichtung des Servers und die Vergabe der Rechte für die Entwickler. Daneben erlauben manche Systeme den Einsatz verschiedener Speichertypen. Ob ein Dateisystem oder eine Datenbank zum Einsatz kommt, wird ebenfalls in diesem Schritt festgelegt[52]. Vielfach werden neben dem eigentlichen Versionsverwaltungssystem auf dem Server weitere Tools verwendet. Dazu gehören Web-Frontends, die den Einblick ins Repository mittels Browser ermöglichen. Bei manchen Systemen ist dies bereits im Lieferumfang enthalten und muss aktiviert werden. Andere Tools wie ViewVC[53], WebSVN[54] [55] oder Chora[56] müssen separat eingerichtet werden. Darüber hinaus gehende Funktionalität bietet Trac[57], welches ein Wiki, ein Ticketsystem zur Fehlerverfolgung, eine Timeline, eine Roadmap und einen Repository-Browser integriert und somit grundlegende Voraussetzungen für das Projektmanagement bietet. Während die Dokumentation im Wiki stattfindet, stellt die Timeline eine Liste aller Änderungen dar, diese beinhaltet Tickets, Meilensteine und die Wiki-Seiten selbst[58]. Mingle[59] ist ein System zur agilen Softwareentwicklung und integriert ebenfalls den Zugriff auf das Repository.
Der Projektleiter sollte in der Vorbereitung weitere Vorkehrungen treffen. Dazu gehört die Festlegung bestimmter Personen zur Durchführung administrativer Aufgaben wie die Verwaltung von Tags und Branches. Des weiteren muss ein Ansprechpartner zur Verfügung stehen, der bei Problemen mit der Versionsverwaltungssoftware Maßnahmen zur Lösung einleiten kann[60].

4.2.2. Import

Beim Import wird ein zuvor nicht unter Versionierung stehendes Projektverzeichnis in das Versionsverwaltungssystem eingestellt. Durch diesen Schritt wird gleichzeitig das initiale

52 Vgl. Merkle (2006), S. 4

53 http://www.viewvc.org

54 http://websvn.tigris.org

55 Vgl. Budszuhn (2007), S. 179 ff.

56 http://www.horde.org/chora/

57 http://trac.edgewall.org

58 Vgl. Huhmann (2007), S. 78-80

59 http://studios.thoughtworks.com/mingle-project-intelligence

60 Vgl. Budszuhn (2007), S. 350

Verzeichnislayout innerhalb des Repositories festgelegt. Beim Import können auch Dateien ausgeschlossen werden, welche keinen Platz im Repository finden sollen. Dies sind beispielsweise generierte Dateien, ausführbarer Objektcode oder automatisch angelegte Kopien der Quellcode-Dateien[61].

4.2.3. Checkout

Das Konzept der lokalen Arbeitskopie wurde bereits erläutert. Als Checkout wird der Vorgang bezeichnet, die Arbeitskopie aus dem Repository zu erzeugen. Zwar ist es möglich, die Arbeitskopie aus jeder Revision der Dateien zu erstellen, in den meisten Fällen erfolgt jedoch der Checkout der neuesten Entwicklungsversion. Die Arbeitskopie beinhaltet alle Dateien und Verzeichnisse, die sich im angeforderten Projekt befinden[62]. Ebenso ist ein partieller Checkout einzelner Verzeichnisse möglich.

4.2.4. Status

Mittels der Statusabfrage wird eine Übersicht der verschiedenen Zustände von Dateien der Arbeitskopie im Vergleich zum Repository angezeigt. Dateien können geändert oder hinzugefügt worden sein, ebenso möglich wäre das Entfernen einer Datei aus der Arbeitskopie. Falls es bei einem Bearbeitungsschritt zu Konflikten innerhalb einer Datei gekommen ist, wird dieser Status ebenfalls angegeben.

4.2.5. Update

Beim Update erfolgt ein Vergleich zwischen Arbeitskopie und Repository. Das Ziel ist es hierbei, die Arbeitskopie zu aktualisieren. Alle Änderungen aus dem Repository werden in die Arbeitskopie automatisch eingepflegt. Die Veränderung von Dateien erfolgt ausschließlich innerhalb der Arbeitskopie, nicht jedoch im Repository. Während des Update-Vorgangs können verschiedene Situationen entstehen. Falls eine Datei nur in der Arbeitskopie, nicht jedoch im Repository verändert wurde, verbleibt die modifizierte Datei in der Arbeitskopie. Falls eine Änderung sowohl in Arbeitskopie als auch im Repository vollzogen wurde, erfolgt der Versuch des automatisierten Zusammenführens, das so genannte Merging. Falls eine Datei oder ein Verzeichnis aus dem Repository gelöscht wurde, gilt dies nach dem Update auch für die Arbeitskopie, sofern das gelöschte Element nicht zwischenzeitlich in der lokalen Kopie modifiziert wurde. Falls hingegen Dateien oder Verzeichnisse nur aus der Arbeitskopie entfernt wurden, sind diese nach dem Update-

61 Vgl. Budszuhn (2007), S. 92 ff.

62 Vgl. Baerisch (2005), S. 9 f.

Prozess wieder vorhanden[63] [64].

4.2.6. Commit

Die Übergabe der geänderten Dateien aus der Arbeitskopie in das Repository wird als Commit oder auch Einchecken bezeichnet. Diese Operation führt somit im Erfolgsfall zu einer Änderung innerhalb des Repository. Nach dem Commit stehen die geänderten Dateien allen Entwicklern zur Verfügung[65]. Manche Systeme erlauben nur dann die Durchführung eines Commit, wenn die Arbeitskopie dem Stand des Repositories entspricht. Falls nicht, wird der Commit abgelehnt und der Entwickler dazu aufgefordert, zunächst ein Update durchzuführen, damit die aktualisierten Inhalte des Repositories in die Arbeitskopie übernommen werden[66]. Je nach System gehören Commits zu den atomaren Operationen. Bei atomaren Commits werden entweder alle geänderten Dateien in das Repository übertragen, oder der Vorgang wird abgebrochen, so dass sich keinerlei Änderung wiederfindet[67]. Nach einem erfolgreichen Commit-Vorgang kann die Arbeit ohne Unterbrechung auf der Arbeitskopie fortgesetzt werden[68]. Beim Commit erhält der Entwickler die Möglichkeit, einen Kommentar beizufügen, der zusätzlich zu Angaben wie Version und Zeit im Repository gespeichert wird.

4.2.7. Add

Wenn bei der Entwicklung neue Dateien oder Verzeichnisse in der Arbeitskopie hinzugefügt wurden, werden diese bei einem Commit nicht automatisch ins Repository übernommen. Das Versionsverwaltungssystem hat bis zu diesem Moment keinerlei Kenntnis über diese Dateien. Daher müssen neue Dateien oder Verzeichnisse explizit hinzugefügt werden, diese Add-Operation ist vergleichbar mit einem Import, wirkt sich jedoch nur auf die einzeln hinzugefügten Elemente aus. Nach dem Add werden durch ein Commit die neuen Dateien oder Verzeichnisse an das Repository übergeben[69].

4.2.8. Merge

Das Zusammenführen von Änderungen aus dem Repository mit den bearbeiteten Dateien aus der Arbeitskopie wird als Merge bezeichnet. Dabei werden die zwei Dateiinhalte

63 Vgl. Budszuhn (2007), S. 46 f.

64 Vgl. Baerisch (2005), S. 11

65 Vgl. Baerisch (2005), S. 10

66 Vgl. Budszuhn (2007), S. 49

67 Vgl. Baerisch (2005), S. 28

68 Vgl. Budszuhn (2007), S. 50

69 Vgl. Baerisch (2005), S. 10

zeilenweise verglichen. Das Versionsverwaltungssystem versucht, eine aktualisierte Ergebnisdatei zu erstellen. Änderungen, die an unterschiedlichen Stellen derselben Datei auftreten, werden in die aktualisierte Fassung übernommen. Sollten hingegen die Änderungen an gleicher Stelle vorgenommen worden sein, kann das Versionsverwaltungssystem nicht über die Validität entscheiden, es kommt zu einem Konflikt. Dieser muss anschließend durch manuellen Eingriff des Entwicklers gelöst werden. Versionsverwaltungssysteme benutzen zur Kennzeichnung der Konflikte innerhalb der Dateien Konfliktmarker. Dabei handelt es sich um bestimmte Zeichenfolgen, die an die Stelle in der Datei geschrieben werden, an der der Konflikt aufgetreten ist[70]. Für die Lösung der Konflikte ist der Entwickler verantwortlich, in dessen Arbeitskopie der Konflikt auftritt. Die widersprüchlichen Veränderungen müssen integriert oder einer Fassung der Vorzug gegeben werden[71]. Falls nicht bereits in der Projektvorbereitung geschehen, sollten nach einer zügigen Konfliktlösung Vorkehrungen getroffen werden, deren Einhaltung die Entstehung von Konflikten verhindert. Durch Absprachen der Entwickler untereinander und Festlegung klar abgegrenzter Arbeitsbereiche wird die Notwendigkeit der gleichzeitigen Arbeit an denselben Dateien verringert[72].

4.2.9. Export

Einige Versionsverwaltungssysteme speichern in der lokalen Arbeitskopie nicht nur die zum Projekt gehörenden Dateien und Verzeichnisse, sondern zusätzliche Informationen wie Server und Pfad des Repositories bis hin zu Meta-Daten. Da bei einer Weitergabe des Projektes, etwa an Kunden, diese zusätzlichen Dateien unerwünscht sind, existiert die Möglichkeit, eine Kopie von Inhalten des Repositories ohne Hinterlegung von Versionsinformationen durchzuführen. Dieser Vorgang wird als Export bezeichnet. Da die zur Verwaltung relevanten Informationen fehlen, kann diese Kopie jedoch nicht als Arbeitskopie eingesetzt werden.

4.2.10. Der Entwicklungszyklus im Überblick

Die Arbeit mit einem Versionsverwaltungssystem kann als zyklischer Prozess dargestellt werden.

70 Vgl. Budszuhn (2007), S. 47 f.

71 Vgl. Baerisch (2005), S. 13

72 Vgl. Budszuhn (2007), S. 48

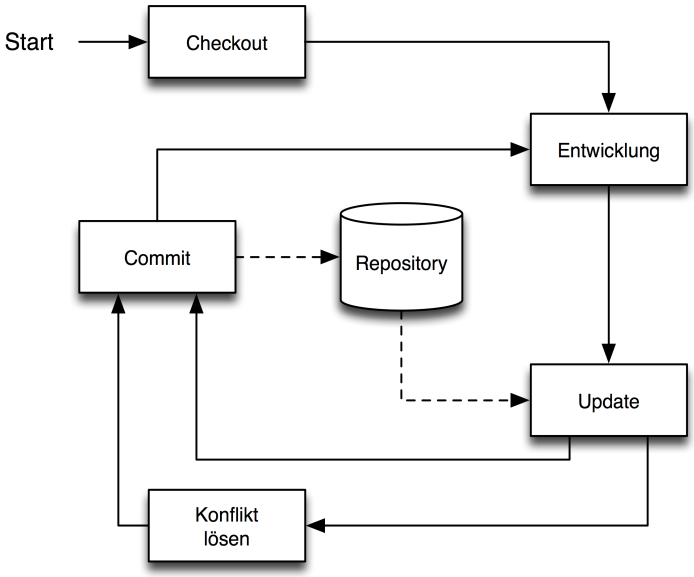

Abbildung 2: Entwicklungszyklus

Zu Beginn fordert der Entwickler eine Arbeitskopie aus dem Repository an. Sämtliche Softwareentwicklung findet ab diesem Zeitpunkt ausschließlich innerhalb der Arbeitskopie statt. Zu einem geeigneten Zeitpunkt sollen die Änderungen in das Repository übernommen werden. Dies kann in regelmäßigen Abständen, etwa täglich, oder am Ende einer Arbeitssitzung geschehen.

5. Systeme

Eine Betrachtung aller verfügbaren Versionsverwaltungssysteme würde den Umfang dieser Arbeit sprengen. Daher werden im folgenden sowohl diejenigen Systeme vorgestellt, die eine weite Verbreitung erfahren haben als auch im Bereich der verteilten Versionsverwaltungssysteme neue Produkte, denen ein großes Potenzial zugesprochen wird. Listen von Versionsverwaltungssystemen finden sich in der Wikipedia als „Comparison of revision control software“[73] oder auf der Seite „Version Control Systems for Linux“[74].

5.1. Historischer Abriss

Bereits in den 1970er Jahren wurde ein erstes System namens SCCS (Source Code Control System) von den Bell Telephone Laboratories entwickelt. Quelltexte konnten dabei versioniert hinterlegt werden, die Speicherung erfolgte als eine „Serie von Inkrementen“[75]. Im Open Source-Bereich wurde eine Verbesserung von SCCS durch das System RCS (Revision Control System) entwickelt. Dabei kam die Strategie des Sperren-Verändern-

73 http://en.wikipedia.org/wiki/Comparison_of_revision_control_software, Stand 03.07.2008

74 http://linuxmafia.com/faq/Apps/vcs.html, Stand 03.07.2008

75 Hill (2005), S. 4

Entsperren zum Einsatz. Das System verwaltet den Zugriff von mehreren Benutzern durch Zugriffslisten und bietet bereits alle Grundfunktionalitäten von Versionsverwaltungssystemen. Einschränkungen bestehen in den Möglichkeiten der Verwaltung bei der Arbeit mit vielen Entwicklern[76]. Daher begann Ende der 1980er Jahre die Entwicklung von CVS (Concurrent Versions System)[77].

5.2. CVS

Die Grundlagen von CVS[78] liegen in einer Sammlung von Shell-Skripten als Frontend für RCS. Wenig später wurde es in der Programmiersprache C neu geschrieben. CVS implementiert die Kopieren-Verändern-Zusammenführen-Lösung, die das gleichzeitige Arbeiten von mehreren Entwicklern an derselben Datei ermöglicht. CVS gehört zur Kategorie der Systeme mit einem zentralen Repository. CVS unterstützt die Versionierung hierarchischer Verzeichnisstrukturen. Die Versionierungsinformationen lagert CVS im Repository als Bestandteil der darin eingestellten Dateien. Bei CVS erhält jede Datei eine eigene Revisionsnummer. Beim Tagging werden die Tags als Textmarkierungen verstanden, welche an mehrere Dateien beigefügt werden. Die Zusammenfassung zu einem Entwicklungsstand geschieht anhand dieser Markierungen. Zur Realisierung von Branches fügt CVS der Revisionsnummer zwei weitere Zahlen hinzu, dabei beschreibt die erste Zahl die Verzweigung und die zweite die Revisionsnummer des jeweiligen Branches. CVS ist ein Client-Server-System, wobei das Serverprotokoll eine spezifische Implementierung darstellt. CVS entwickelte sich zu einem de-facto-Standard bei der Entwicklung von Open Source Software, wird aber ebenso in kommerziellen Entwicklungsprojekten eingesetzt[79] [80]. Angesichts der weiten Verbreitung und der langen Reifezeit gibt es eine Vielzahl von weiteren Tools, welche die Arbeit mit CVS unterstützen. Dazu gehören Web-Frontends ebenso wie die Integration in Entwicklungsumgebungen oder generell Werkzeuge, die den Umgang mit CVS erleichtern[81]. CVS hat jedoch auch Schwächen. Die Entwicklung des Systems an sich ist nicht durchgängig verlaufen, sondern es wurden mit der Zeit weitere Funktionalitäten hinzugefügt. Dies führte dazu, dass konzeptionelle Probleme bestehen blieben. CVS bietet keine Versionierung für Verzeichnisse. Das Löschen oder Verschieben von Verzeichnissen ist nicht möglich, was beispielsweise das Reengineering von Projekten erschwert. Ein Commit ist bei CVS nicht atomar, sondern wird dateiweise durchgeführt.

76 Vgl. Loukides, Oram (1997), S. 207 ff.

77 Vgl. Merkle (2006), S. 2

78 http://www.nongnu.org/cvs/

79 Vgl. Tilkov (2003), S. 116-118

80 Vgl. Budszuhn (2007), S. 235 ff.

81 Vgl. Bar, Fogel (2003), S. 239 ff.

Wenn beim Commit ein Konflikt in einer Datei auftritt, werden die anderen Dateien trotzdem ins Repository übertragen. Teilweise Änderungen, die zu einem inkonsistenten Zustand führen, sind daher möglich. Der Umgang mit Binärdateien ist bei CVS schwierig, zur Erkennung muss die Konfiguration explizit vorbereitet werden, außerdem werden Binärdateien immer als Ganzes in der kompletten Größe im Repository abgelegt[82] [83]. Diese Schwächen führten zur Entwicklung von weiteren Versionsverwaltungssystemen, die die Vorteile von CVS übernehmen und sich in der Bedienung an CVS anlehnen. Für die Anwender bedeutet dies eine nur geringe Hürde beim Umstieg auf ein anderes System.

5.3. Subversion

Subversion (SVN)[84] wurde als inoffizieller Nachfolger von CVS konzipiert. Angesichts der Einschränkungen von CVS suchte das amerikanische Softwareunternehmen CollabNet Corporation[85] nach einer Alternative, wurde jedoch nicht fündig. Deshalb stellte CollabNet im Jahr 2000 ein Entwicklerteam zusammen, welches ein System erstellen sollte, was sich grundsätzlich an den Eigenschaften von CVS orientiert, jedoch die konzeptionellen Probleme löst und neuen Anforderungen gerecht wird. Das System sollte der modernen Softwareentwicklung Rechnung tragen und eine modulare Struktur bieten. Nach einer Entwicklungszeit von 14 Monaten konnten die Entwickler den CVS-Server ersetzen und das neue System zur eigenen Softwareentwicklung in Betrieb nehmen. Die Version 1.0 ist 2004 erschienen, es findet eine stetige Weiterentwicklung statt. Subversion gehört zur Kategorie der Systeme mit einem zentralen Repository. Neben dem Kopieren-Verändern-Zusammenführen-Konzept beherrscht Subversion seit der Version 1.3 auch die Sperren-Verändern-Entsperren-Lösung. Die wichtigsten Merkmale von Subversion sind die Beseitigung der Schwachstellen von CVS bei gleichzeitiger weitestgehender Kompatibilität in der Bedienung. Verzeichnisse werden versioniert, so dass die Historie beim Verschieben, Löschen oder Umbenennen erhalten bleibt. Der Umgang mit Binärdateien wurde verbessert, so dass nicht mehr der vollständige Dateiinhalt, sondern ebenso wie bei Textdateien die jeweiligen Unterschiede zwischen den Versionen im Repository abgelegt werden. Weiterhin können beliebige mit einem Objekt verbundene Attribute, die sich als Schlüssel-Wert-Paare darstellen, mit Dateien und Verzeichnissen verknüpft werden. Dies wird beispielsweise dazu benutzt, um Dateien von der Versionierung auszuschließen. Das Repository erlaubt die Benutzung verschiedener Speicherarten, zum Einsatz kommt wahlweise eine Datenbank-basierte Lösung oder ein

82 Vgl. Budszuhn (2007), S. 235 ff.

83 Vgl. Tilkov (2003), S. 118

84 http://subversion.tigris.org

85 http://www.collab.net

Dateisystem. Jedes dieser Systeme bietet spezifische Vor- und Nachteile, die je nach Einsatzzweck Relevanz erlangen. So ist beispielsweise die Geschwindigkeit der Operationen bei der Datenbank-basierten Lösung höher, während das Dateisystem besser skaliert. Als Standard gilt seit Version 1.2 die FSFS genannte Dateisystem-Lösung. Es stehen mehrere Zugriffsmethoden zur Verfügung, Subversion bietet den Entwicklern Zugriff über ein lokales Dateisystem, über einen eigenen svnserve genannten Server, der auch mit SSH zur sicheren Datenübertragung kombiniert werden kann, sowie unter Zuhilfenahme eines Apache-2-Webservers, wobei das WebDAV-Protokoll verwendet wird. Commits sind bei Subversion atomar, diese Operation wird entweder ganz oder gar nicht ausgeführt, was Inkonsistenzen im Repository vermeidet. Subversion bietet eine Programmierbibliothek (API), welche von den Clientprogrammen zur Nutzung aller Funktionen von Subversion verwendet werden kann. Durch die Verwendung des Tools SWIG existieren auch Anbindungen aus den Programmiersprachen Java, Perl, Tcl und Python. Damit ist die Programmierung von ergänzender Software auf einfache Art und Weise möglich. Eine Änderung im Vergleich zu CVS stellt die Nummerierung der Revisionen dar. Bei Subversion gibt es eine „Global Revision Number“, die sich auf den Zustand des gesamten Repositories bezieht. Dadurch kann jeder Stand eines Projektes einer Revisionsnummer zugeordnet werden. Darüber hinaus gibt es Tags und Branches, die intern als Kopie behandelt werden. Dateikopien werden bei Subversion als so genannte „Cheap Copy“ realisiert. Für unveränderte Dateien legt Subversion nur eine Referenz an, ein echter Kopiervorgang erfolgt erst bei Änderung, wobei dabei nur die Unterschiede zwischen den Dateien gespeichert werden. Tags und Branches werden daher als Verzeichnisstruktur im Repository eines Projektes angelegt. Die Benutzung erfolgt durch vom Projektleiter festgelegte Konventionen. Subversion besteht neben dem Server- und Client-Anwendung aus weiteren Modulen, die etwa zur Administration oder Inspektion des Repositories verwendet werden. Dazu gehört auch ein Programm zur Konvertierung bestehender CVS-Datenbestände. Angesichts der umfassenden Verbesserungen im Vergleich zu CVS sind viele Projekte auf Subversion umgestiegen. Es existiert eine Vielzahl von Tools von Drittanbietern, angefangen von Web-Frontends über verschiedene grafische Clientanwendungen, die auf unterschiedlichen Betriebssystemen laufen, bis hin zur Integration in Editoren oder Entwicklungsumgebungen[86] [87] [88] [89].

86 Vgl. Budszuhn (2007), S. 32

87 Vgl. Budszuhn (2007), S. 235 ff.

88 Vgl. Tilkov (2003), S. 116-118

89 Vgl. Leibl (2004), S. 78-80

5.4. Mercurial

Mercurial[90], abgekürzt Hg, gehört zur Kategorie der verteilten Versionsverwaltungssysteme. Der Entwickler arbeitet mit einem eigenen Repository, was sich aus einem Verzeichnis auf dem lokalen Datenträger bildet. Die Meta-Informationen werden dabei in Dateien in einem weiteren, versteckten Unterverzeichnis abgelegt. Die Speicherung von Dateirevisionen erfolgt in einem speziellen Format, genannt Revlog. Ebenfalls sind darin die Unterschiede (Deltas) gegenüber der vorhergehenden Revision abgelegt. Anhand einer Index-Datei kann schnell auf die benötigten Teile zugegriffen werden. Ein Platzvorteil gegenüber Subversion ergibt sich durch die komprimierte Speicherung der Deltas. Commits sind atomar, die Menge der Änderungen wird als Changeset bezeichnet. Ein Commit führt zu einer neuen, global eindeutigen Kennung des Repositories. Anstelle Tags und Branches werden verschiedene Verzweigungsmodelle verwendet. Dabei beziehen sich Verzweigungen immer auf das gesamte Repository. Im globalen Modell wird die Verzweigung Klon genannt. Dabei wird ein entferntes Repository vollständig kopiert. Das lokale Modell wirkt sich nur auf das lokale Repository aus. Nachdem eine lokale Verzweigung initiiert wurde, wird die Arbeitskopie diesem Zweig zugeordnet. Das Zusammenführen geschieht dadurch, dass die Änderungen aus einer Verzweigung in die Hauptentwicklungslinie zurückgeführt wird. Beim Abgleich mit anderen Repositories werden die Änderungen im ersten Schritt nur in das lokale Repository übertragen, nicht jedoch in die Arbeitskopie. Dies kann durch einen zweiten Schritt ermöglicht werden. Mercurial bietet verschiedene Zugriffsmethoden, lokal über ein Dateisystem, sowie entfernt mittels SSH oder dem eingebauten Webserver. Daneben lassen sich Changesets bündeln und etwa per E-Mail an weitere Entwickler senden. Diese gebündelten Changesets können in weitere Repositories übernommen werden. Die Migration vorhandener Repositories aus anderen Systemen gestaltet sich schwierig, ebenso ist die Einrichtung eines Mercurial-Servers im Vergleich zu Subversion relativ komplex. Entscheidend für die Verwendung eines verteilten Systems wie Mercurial ist eine geeignete Organisation des Arbeitsablaufs[91] [92].

5.5. Bazaar

Bazaar[93] gehört zur Klasse der verteilten Versionsverwaltungssysteme. Obwohl es noch

90 http://www.selenic.com/mercurial/

91 Vgl. Lau (2008), S. 183 ff.

92 Vgl. Hein (2006), Stand 16.05.2008

93 http://bazaar-vcs.org

nicht die Verbreitung wie Mercurial erreicht hat[94] wird Bazaar bereits von vielen Open Source Projekten verwendet[95]. Bazaar erlaubt flexible Arbeitsabläufe. Die Spanne reicht dabei vom ausschließlich lokalen Betrieb ohne Server über ein zentralisiertes System mit Server bis hin zu Mischformen wie ein zentralisiertes Repository mit lokalen Kopien, verschiedenen lokalen Branches und der Möglichkeit, Änderungen automatisiert oder nach expliziter Kontrolle in das zentrale Repository zu übernehmen[96]. Die Entwicklung wurde vom Unternehmen Canonical Ltd.[97] vor dem Hintergrund initiiert, aus den Erfahrungen mit anderen Versionsverwaltungssystemen zu lernen und ein besseres System zu implementieren. Die Schwerpunkte wurden dabei auf die Merkmale Einfachheit in der Anwendung, Benutzerfreundlichkeit, Geschwindigkeit, Erweiterbarkeit, Sicherheit und die Möglichkeit der Einbettung in andere Systeme gelegt[98]. Die zugrunde liegenden Kommandos orientieren sich an denen von CVS und Subversion, was den Umstieg erleichtert. Ebenso können andere Versionsverwaltungssysteme transparent integriert werden[99].

6. Entscheidungsfindung

Ein umfangreicher Vergleich der einzelnen Systeme ist an dieser Stelle aufgrund des begrenzten Umfangs dieser Arbeit nicht möglich. Daher konzentriert sich die folgende Darstellung auf die beim Einsatz in Unternehmen ebenso wichtige Frage nach dem Support und der Unterstützung durch Lösungsanbieter. Dieser Aspekt ist vor allem dann relevant, wenn im Unternehmen selbst keine oder nur wenige Kenntnisse über Versionsverwaltungssysteme vorhanden sind.

6.1. Grundsatzentscheidungen

Zunächst muss die Frage nach dem Einsatz eines Versionsverwaltungssystems geklärt werden. Findet die Softwareentwicklung nur vor Ort im Unternehmen selbst statt, oder ist die Einbindung externer Mitarbeiter notwendig? Erfolgt die Bearbeitung häufig von mobilen Arbeitsplätzen, an denen keine Verbindung zum Internet zur Verfügung steht? Ist ein sehr flexibler Arbeitsablauf notwendig, welcher nicht durch zentrale Systeme geboten wird? In solchen Fällen empfiehlt sich die Verwendung eines dezentralen

94 Vgl. Lau (2008), S. 183

95 Vgl. http://bazaar-vcs.org/WhoUsesBzr, Stand 07.06.2008

96 Vgl. http://bazaar-vcs.org/Workflows, Stand 07.06.2008

97 http://www.canonical.com

98 Vgl. http://bazaar-vcs.org, Stand 07.06.2008

99 Vgl. Wikipedia, http://de.wikipedia.org/wiki/Bazaar, Stand 16.05.2008

Versionsverwaltungssystems[100]. Andernfalls kann auf eine Lösung eines zentralen Repositories zurückgegriffen werden, deren Vorteile sich vor allem auf eine breite Unterstützung und Ausgereiftheit der Systeme stützen.

6.2. Dokumentation

Die Dokumentation erhält sowohl bei der Einführung eines Systems als auch als Referenz für die tägliche Arbeit einen hohen Stellenwert. Die Qualität und Quantität unterscheidet sich bei den hier betrachteten Systemen. CVS ist aufgrund seines Alters sehr gut dokumentiert. Dies umfasst sowohl die Systemdokumentation, die online zur Verfügung steht[101], als auch die Menge der Literatur. Das Buch „Open Source Development with CVS“ kann dabei kostenlos heruntergeladen werden[102]. Ähnliches gilt für Subversion, neben Artikeln in Fachzeitschriften existiert auf dem Büchermarkt ein umfassendes Angebot. Auch hier existiert ein kostenfrei herunter ladbares Buch namens „Version Control with Subversion“[103], eine Übersetzung in die deutsche Sprache wurde begonnen. Das Äquivalent zu Mercurial heißt „Distributed revision control with Mercurial“ und steht ebenfalls kostenfrei zur Verfügung[104]. Darüber hinaus ist Mercurial weniger gut dokumentiert, es findet sich keine weitere Literatur in Buchform. Auf der Projekt-Website, die aus einem Wiki besteht, finden sich Texte und Tutorien, welche Konzepte und Erläuterungen für den Einstieg bieten sowie auf fortgeschrittene Lösungen wie die Verwendung von Erweiterungen eingehen[105]. Bei Bazaar besteht die Dokumentation ebenfalls aus einem Wiki[106], welches neben Texten zum Einstieg auch ein Benutzer- sowie Entwicklerhandbuch enthält. Diese werden jedoch ausschließlich in Form der Website angeboten, gedruckte Literatur gibt es bislang nicht.

6.3. Support- und Lösungsanbieter

Der klassische Supportweg von Open Source Produkten erfolgt über das Internet. Eine Gemeinschaft (Community), die sich aus Anwendern und Programmierern rund um das jeweilige Produkt bildet, bietet freiwillige Hilfestellung bei Problemen oder Fragen helfen.

---

100 Vgl. Lau (2008), S. 183

101 http://ximbiot.com/cvs/manual/

102 http://cvsbook.red-bean.com

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

104 http://hgbook.red-bean.com

105 http://www.selenic.com/mercurial/wiki/index.cgi/Mercurial

106 http://bazaar-vcs.org/Documentation

Diese Unterstützung wird in Form von Mailinglisten, FAQs, Foren oder Chats geleistet[107] [108]. Bei allen der hier genannten Produkten stehen derartige Möglichkeiten zur Verfügung. Während das Interesse sowie damit einher auch die Unterstützung von CVS langsam zurückgeht, ist die Community bei den übrigen Systemen aktiv und motiviert.

Eine weitere Möglichkeit besteht im Bezug gewerblicher Support-Dienstleistungen. Auf den jeweiligen Projekt-Websites sind Hinweise und Links zu entsprechenden Unternehmen zu finden. Wegen der geringer werdenden Bedeutung von CVS existieren nur noch wenige Anbieter[109].

Der Initiator und Hauptsponsor von Subversion, das Unternehmen CollabNet, bietet neben Dienstleistungen wie Support, Training und Beratung zu Subversion auch kommerzielle Software an, die auf Subversion basiert[110]. Die Projekt-Website listet weitere Anbieter kommerzieller Dienstleistungen[111]. Das steigende Interesse spiegelt auch die seit 2007 stattfindende Konferenz zum Thema Subversion namens SubConf[112] wider, auf deren Aussteller-Seite weitere Lösungsanbieter für Subversion genannt werden. Als Beispiele seien hier für den deutschsprachigen Raum die Hood GmbH[113], die Argumentum GmbH[114] und die elego Software Solutions GmbH[115] angeführt. Damit verfügt Subversion über eine im Vergleich größere Anzahl von Anbietern kommerzieller Dienstleistungen.

Eine ähnliche Situation ergibt sich für die Software Mercurial. Neben den Dienstleistungen des projektleitenden Unternehmens Selenic Consulting[116] stehen weitere Unternehmen für Beratung, Training und Schulung zur Verfügung. Im deutschsprachigen Raum ist dies beispielsweise die Intevation GmbH[117]. Im Vergleich zu Subversion wird jedoch der Eindruck erweckt, dass es sich um eher kleinere Unternehmen handelt, auch die Anzahl ist insgesamt geringer.

Als Dienstleister für Bazaar wird auf der entsprechenden Webseite ausschließlich das Unternehmen Canonical[118] genannt, welches Supportleistungen anbietet[119].

---

107 Vgl. Grassmuck (2002), S. 235 ff.

108 Vgl. Hill (2005), S. 13 f.

109 Vgl. http://ximbiot.com/cvs/wiki/Commercial%20Support, Stand 04.07.2008

110 Vgl. Gellien, Leibl (2008), S. 84

111 http://subversion.tigris.org/commercial-support.html

112 http://www.subconf.de

113 http://www.hood-group.com/de/

114 http://argumentum.de

115 http://www.elego.de

116 http://www.selenic.com/

117 http://intevation.net/

118 http://www.canonical.com/

119 Vgl. http://bazaar-vcs.org/BzrSupport, Stand 07.06.2008

7. Fazit und Handlungsempfehlung

Die Entscheidung pro oder contra Versionsverwaltungssystem bei der Softwareentwicklung kann nur zugunsten des Einsatzes ausfallen. Dafür sprechen Vorteile, die nicht nur im Entwicklungsprozess selbst begründet sind, sondern auch wirtschaftliche Aspekte umfassen. Dazu gehört auch die Nutzung von Open Source Produkten, welche keine Lizenzkosten verlangen. Im nächsten Schritt stellt sich die Architekturfrage, welche jedoch nicht pauschal zu beantworten ist. Je nach Einsatzzweck kann ein verteiltes oder ein zentralisiertes Versionsverwaltungssystem notwendig erscheinen. Bei letzterem lautet die Empfehlung Subversion. Dafür sprechen der Funktionsumfang, die Stabilität, die konsequente Weiterentwicklung, die Marktakzeptanz sowie Verfügbarkeit von Dokumentation und kommerzieller Dienstleistungen. Beim Einsatz eines verteilten Versionsverwaltungssystems ist Mercurial die erste Wahl, die Gründe dafür sind die bessere Integration in eine Arbeitsumgebung unter Windows, der kommerzielle Support mehrerer Anbieter, und die zur Zeit umfassendere Dokumentation.

Glossar

CMS (Content Management System): eine Software zur gemeinschaftlichen Erzeugung und Bearbeitung von Inhalten, etwa zur Verwendung des Aufbaus und Pflege einer Website.

DVCS (Distributed Version Control System): verteiltes Versionskontrollsystem.

FAQ (Frequently Asked Questions): Eine Sammlung von Dokumentationen in einem Frage-Antwort-Schema.

IDE (Integrated Development Environment, integrierte Entwicklungsumgebung): Eine Sammlung von Werkzeugen zur Entwicklung von Software. Die grundlegenden Bestandteile sind Editor, Compiler, Debugger und Laufzeitumgebung, weitere Module können integriert werden.

SSH (Secure Shell): Netzwerkprotokoll zur verschlüsselten Verbindung und einer sicheren Authentifizierung zu einem anderen Computer.

SWIG (Simplified Wrapper and Interface Generator): Ein Werkzeug, welches Funktionen aus Modulen, die in C oder C++ geschrieben wurden, anderen Programmiersprachen zugänglich macht.

WebDAV (Web-based Distributed Authoring and Versioning): ein Standard zur Bereitstellung und verteilten Bearbeitung von Daten.

Wiki: Eine Software zur Bearbeitung von Webseiten, bei der die Inhalte gemeinschaftlich aufgebaut und gepflegt werden. Die Bearbeitung kann von jedem Benutzer der Website erfolgen, diese Offenheit grenzt ein Wiki gegenüber einem CMS ab.

Literaturverzeichnis

Baerisch, S. (2005): Versionskontrollsysteme in der Softwareentwicklung, Bonn 2005, http://www.gesis.org/publikationen/berichte/iz_arbeitsberichte/pdf/ab_36.pdf (18.05.2008, 04:04)

Bar, M., Fogel, K. (2003): Open Source Development with CVS, 3. Aufl., Scottsdale 2003

Budszuhn, F. (2007): Subversion, 2. Aufl., Bonn 2007

Collins-Sussman, B., Fitzpatrick, B., Pilato, C., (2007): Version Control with Subversion (Compiled from r2866), http://svnbook.red-bean.com/ (18.05.2008, 04:04)

Gellien, K., Leibl, A. (2008): Sprung nach vorn, in: iX Magazin für professionelle Informationstechnik, 2008, Ausgabe 04, S. 84-86

Grassmuck, V. (2002): Freie Software zwischen Privat- und Gemeineigentum, Bonn 2002

Hill, F. (2005): Einführung einer Revisionskontrolle mit Subversion in einer heterogenen Systemumgebung für Design und Programmierung, Mannheim 2005, http://docs.tx7.de/BA-Mannheim/diplomarbeiten/2005-08-08_diplomarbeit_friedel_hill_freigegeben.pdf (18.05.2008, 23:03)

Hein, T. A. (2006): Mercurial Distributed SCM – Die verteilte Alternative zu CVS, 2006, http://www.intevation.de/~thomas/mercurial-lt2006 (03.07.2008 20:36)

Huhmann, J. (2007): Spuren im Wald, in: iX Magazin für professionelle Informationstechnik, 2007, Ausgabe 04, S. 78-80

Lau, O. (2008): Offline versionieren, in: c't Magazin für Computertechnik, 2008, Ausgabe 12, S. 182-187

Leibl, A. (2004): CVS reloaded, in: iX Magazin für professionelle Informationstechnik, 2004, Ausgabe 08, S. 78-80

Litke, H.-D. (2007): Projektmanagement, München 2007

Loukides, M., Oram, A. (1997): Programmieren mit GNU Software, Köln 1997

Lutterbeck, B., Bärwolff, M., Gehring, R. (Hrsg.) (2007): Open Source Jahrbuch 2007, Berlin 2007

Schweigel, S. (2004): Aufnahme des Baufortschritts in verteilter Umgebung, http://stefan.schweigel-net.de/diplom/DASchweigel.pdf (18.05.2008, 04:04)

Sommerville, I. (2007): Software Engineering, 8. Aufl., München 2007

Merkle, C. (2006): Die eierlegende Wollmilchsau, Augsburg 2006, http://cms.hs-augsburg.de/report/2006/Merkle_Christian__Subversion/Subversion.pdf (18.05.2008, 15:26)

Tilkov, S. (2003): Eins, zwei, drei, in: iX Magazin für professionelle Informationstechnik, 2003, Ausgabe 02, S. 116-118

Vetter, M., Renner, T., Kett, H., Rex, S.: Open-Source-Software – Einsatzstrategien, Reifegrad und Wirtschaftlichkeit, in: Lutterbeck, B., Bärwolff, M., Gehring, R. (Hrsg.) (2007): Open Source Jahrbuch 2007, Berlin 2007, S. 195-207