Institut für Softwaretechnologie
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Fachstudie Nr. 157

Marktanalyse Quellcodeverwaltung

Jakob Jarosch Tobias Kuhn Patrick Strobel

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Stefan Wagner
Betreuer:	Dipl.-Ing. Jan-Peter Ostberg Markus Schmidt (flexis AG)
begonnen am:	23. April 2012
beendet am:	23. Oktober 2012
CR-Klassifikation:	D.2.7

Zusammenfassung

Mit Hilfe einer Quellcodeverwaltung lassen sich Dateien bequem versionieren und sichern. Allerdings entwickeln sich auch Quellcodeverwaltungs-Werkzeug mit der Zeit weiter, woraus sich entscheidende Unterschiede entwickelt haben; zum Beispiel die Unterscheidung zwischen verteilten und zentralen Systemen. Der Industriepartner, die Flexis AG, setzt zur Zeit das zentrale Subversion als Lösung ein. Flexis vermutet aber, dass eine andere Lösung möglicherweise besser die gewünschte Arbeitsweise unterstützt. Daher beschäftigt sich diese Fachstudie mit einer Analyse der meisten auf dem Markt befindlichen Werkzeuge für die Quellcodeverwaltung. Anhand der Anforderungen des Industriepartners wird zunächst eine Vorauswahl getroffen und die relevantesten Werkzeuge detailliert bewertet. Abschließend wird eine Empfehlung für Flexis ausgesprochen.

Abstract

Files can be versioned and secured with the aid of a source code management. However, source code management tools advance over time. Therefore, significant differences have evolved, for example the distinction between distributed and centralized systems. The industry partner, the Flexis AG, is currently using the centralized Subversion as a solution. But Flexis assumes that other solutions might possibly support the desired work process better. Therefore this study deals with an analysis of most source code management tools in the market. By means of the requirements of the industry partner initially a pre-selection is made and the most relevant tools are rated in greater detail. Finally, a recommendation for Flexis is made.

Inhaltsverzeichnis

1 Einleitung

Der Industriepartner, die Flexis AG, ist von der bisherigen Lösung für die Quellcodeverwaltung nicht vollständig überzeugt. Aus diesem Grund interessiert sich Flexis für einen Vergleich zu den Werkzeugen, die auf dem Markt auch verfügbar wären. Diese sollen auf die Kompatibilität zum gewünschten Umgang mit der Quellcodeverwaltung bei Flexis untersucht werden. Hierfür werden die Anforderungen von Flexis an ein solches Werkzeug ermittelt, um passende Werkzeuge ermitteln und bewerten zu können. Daran schließt sich eine Empfehlung an.

Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Grundlagen erklärt die Ausgangslage der Fachstudie und fasst die notwendigen Grundbegriffe zusammen, die nachfolgend verwendet werden.

Kapitel 3 – Marktübersicht zeichnet einen groben Überblick über die auf dem Markt befindlichen Quellcodeverwaltungswerkzeuge.

Kapitel 4 – Momentan verwendetes System beschreibt den aktuellen Ist-Zustand beim Industriepartner Flexis AG.

Kapitel 5 – Bewertungskriterien erklärt die einzelnen Bewertungskriterien, die an die neue Lösung gestellt werden.

Kapitel 6 – Auswertung wertet die ausgewählten Werkzeuge konkret an den zuvor erklärten Bewertungskriterien aus.

Kapitel 7 – Empfehlung spricht eine Empfehlung aus, welches Vorgehen bezüglich der Quellcodeverwaltung bei Flexis empfehlenswert erscheint.

Kapitel 8 – Zusammenfassung und Ausblick fasst die Ergebnisse der Arbeit zusammen und stellt Anknüpfungspunkte vor.

2 Grundlagen

2.1 Begriffe

Quellcodeverwaltung ist ein Werkzeug, das es ermöglicht, Text-Dateien mit mehreren Versionen zu verwalten. Typischerweise wird zu einer Änderung protokolliert, wer diese durchgeführt hat und warum. Tatsächlich gibt es für den Begriff diverse Synonyme, wie z. B. *Revisionskontrolle*, *Versionskontrolle*, *(Software-)Konfigurationsverwaltung* oder *Quellcodekontrolle*. Obwohl diese eigentlich näher zu unterscheiden wären, macht es für die Praxis keinen Sinn, sie genauer zu unterscheiden [O'S09]. Auf alle im Weiteren betrachteten Werkzeuge treffen alle Begrifflichkeiten zu, sodass diese im Rahmen dieser Arbeit austauschbar verwendet werden.

Repository ist das Behältnis in einer Quellcodeverwaltung, das alle Dateien und Versionen eines Projekts enthält. Typischerweise ist ein Repository ein bestimmter Ordner, in dem alle Dateien des Projekts mit der Versionskontrolle verwaltet werden können. [O'S09]

Zentralisierte Quellcodeverwaltung ist eine Quellcodeverwaltung, die ein einzelnes, kanonisches Repository beinhaltet. Dabei arbeiten alle Mitarbeiter an diesem einen Repository. [AS09]

Verteilte Quellcodeverwaltung ist eine Quellcodeverwaltung, die kein von vornherein zentrales Repository mehr enthält. Prinzipiell hat jeder Mitarbeiter sein eigenes, vollständiges Repository. Es kann allerdings trotzdem möglich sein, ein nicht-verteiltes Vorgehen nachzubilden. [AS09]

Revision (oft auch *Changeset* oder *Commit*) bezeichnet eine konkrete Version innerhalb einer Quellcodeverwaltung. [O'S09]

Merge bezeichnet die Operation einer Quellcodeverwaltung, die mehrere gleichzeitige Änderungen an einer Datei wieder zusammenfügt. Können nur zwei Dateien miteinander verglichen werden, spricht man von einem Zwei-Wege-Merge; wird zusätzlich noch die Elterdatei, an der beide Änderungen durchgeführt wurden, einbezogen, so wird das Vorgehen Drei-Wege-Merge genannt. [Men02]

2.2 Themenüberblick

Für die Software-Entwicklung ist heutzutage eine Quellcodeverwaltung nicht mehr wegzudenken. Im Kleinen ermöglicht sie es, Änderungen sicher abzuspeichern und zu doku-

mentieren. Ebenso erlaubt eine Konfigurationsverwaltung es, bei Bedarf ältere Versionen wiederherzustellen oder zu vergleichen. Insbesondere im Großen, wenn mehrere Entwickler koordiniert werden müssen, dient eine Quellcodeverwaltung dazu, die Übersicht zu wahren und vermeidet Datenverluste, da Änderungen nicht einfach „verloren gehen“ können.

Die Flexis AG verwendet bisher das zentralisierte Quellcodeverwaltungs-Werkzeug „Subversion“. Allerdings ist Flexis nicht vollständig sicher, ob diese Lösung voll überzeugen kann. Deswegen soll der Markt auf für Flexis nutzbare Quellcodeverwaltungen untersucht werden. Dabei sollen natürlich die spezifischen Anforderungen bei Flexis einfließen, um passende Werkzeuge zu finden und zu bewerten. Abschließend soll eine Empfehlung ausgesprochen werden.

2.3 Vorgehen

Um einen generellen Überblick über den Bedarf bei Flexis zu gewinnen, haben wir zunächst von unserem Ansprechpartner bei Flexis die groben Anforderungen skizziert bekommen. Hiermit ließ sich ein grober Marktüberblick gewinnen, bei dem wir zunächst 17 Werkzeuge herausgefiltert haben. Anschließend haben wir diese Werkzeuge näher betrachtet und aufgrund einiger Kriterien 9 davon auswählen können, die den Kreis der detailliert erläuterten Werkzeuge in dieser Arbeit bilden.

Anschließend haben wir einen Fragebogen entworfen, mit dessen Hilfe wir bei Flexis etwa 20 Mitarbeiter interviewen konnten. Dabei haben wir sowohl gezielt nach Funktionalität als auch offen nach Wünschen und Anregungen gefragt. Mit diesen Ergebnissen war es uns möglich, die tatsächlichen Anforderungen bei Flexis zu erfassen und die in Frage kommenden Werkzeuge detailliert zu analysieren.

Erste Zwischenresultate wurden in einem Zwischenvortrag zur Diskussion gestellt. Durch die Auswertung aller Werkzeuge konnten wir uns individuell eine Meinung zur besten denkbaren Lösung bilden. Die individuellen Meinungen wurden dann in der Gruppe ausdiskutiert, um zu einem gemeinsamen Konsens zu gelangen, welcher in dieser Ausarbeitung dokumentiert wurde.

3 Marktübersicht

Der Mark der Versionskontrollsysteme lässt sich grob nach deren Architektur unterteilen: Während zentrale Versionsverwaltungssysteme auf einer Client-Server-Architektur basieren, bei denen die Repositorys auf einem separaten (Server-)System gespeichert werden, werden die Repositorys bei verteilten Versionsverwaltungssystemen auf jedem Arbeitsplatz dupliziert. Diese Kapitel soll einen kurzen Überblick über die Entwicklung der Versionskontrollsysteme bieten und einige der bekanntesten bzw. verbreitetsten System, nach Architektur unterteilt, vorstellen.

Auf dem Markt befinden sich neben Open-Source- auch kommerzielle Systeme, bei denen die Lizenzierung meist nach Anzahl der Benutzer erfolgt. Die Gebühren bewegen sich hierbei zwischen, einmalig oder jährlich, wenigen 100 € bis hin zu einigen 1.000 €. Aufgrund der nicht unerheblichen Lizenzkosten werden die kommerziellen Systeme hauptsächlich bei der Entwicklung kommerzieller Produkte eingesetzt.

3.1 Zentrale Quellcodeverwaltung

Der Siegeszug der Versionsverwaltungssysteme in der Softwareentwicklung begann hauptsächlich mit der Entwicklung von *CVS*, dem „Concurrent Versions System“. Die Entwicklung von CVS begann 1989 mit der Zielsetzung, in einem Repository mehrere Dateien speichern zu können und damit *RCS* („Revision Control System“) abzulösen, bei dem dies nicht möglich ist. Insbesondere da in CVS das Einchecken von Änderungen, das Committen, nicht atomar in einer Transaktion ausgeführt wird, ist der Einsatz des Systems mit einigen Risiken verbunden. Durch diese Schwachstelle können Daten korrumpiert werden, wenn Änderungen durch zwei oder mehr Benutzer gleichzeitigen eingecheckt werden oder die Verbindung während der Datenübertragung abbricht. [MB03]

Um die Quellcodeverwaltung zu vereinfachen und hierbei die genannten Risiken zu minimieren, begann im Jahr 2000 die Entwicklung von *Subversion* (meist mit *SVN* abgekürzt) bei CollabNet. Dabei orientierte man sich beim Bedienkonzept am damals sehr populären CVS – denn man wollte hierfür einen Ersatz schaffen. Gleichzeitig wurde u. A. durch Hinzufügen von Transaktionen (also atomaren Commits) bewusst versucht, die bekannten Probleme und Risiken auszumerzen [BCS11]. Die Rechnung ging auf und Subversion konnte CVS zunehmend ersetzen. Mittlerweile zählt Subversion zu dem Versionsverwaltungssystem mit der größten Verbreitung und wird bei zahlreichen Open-Source- und proprietären Projekte verwendet. Einer Untersuchung zufolge betrug der Marktanteil von Subversion 2010 rund

33,4 %, wohingegen der Anteil von CVS auf 11,9 % schrumpfte [Ham10]. Da Subversionen einen guten Ersatz für CVS bietet, lässt sich mutmaßen, dass dessen Marktanteil in Zukunft weiter zurückgehen wird. Subversion wird von einer großen Anzahl bekannter Open-Source-Projekte aber auch bei vielen Firmen eingesetzt [Sub].

Während es sich bei den bereits genannten Systemen – RCS, CVS und Subversion – um Open-Source-Projekte handelt, befinden sich auch einige kommerzielle Produkte auf dem Markt. Einen nennenswerten Anteil konnten dabei *Visual SourceSafe* (12,5 %) und *Team Foundation Server* (8,5 %) von Microsoft sowie *Perforce* der Perforce Software Inc. (6,1 %) und IBM's *ClearCase* (5,4 %) erlangen (Stand 2010 nach [Ham10]). Diese werden nicht nur bei der jeweiligen Herstellerfirma selbst eingesetzt, sondern auch bei Organisation wie Google (Perforce) [Men10] oder Siemens (ClearCase) [SKY07].

Die kommerziellen Versionsverwaltungssysteme werden in der Regel mit einer grafischen Benutzeroberfläche ausgeliefert. Um eine grafische Oberfläche bei den ansonsten über die Kommandozeile zu bedienenden Open-Source-Systeme nachrüsten zu können, stehen für viele solche Oberflächen zur Verfügung. Sehr verbreitet sind hier die *TortoiseX*-Projekte, wobei *X* für das Kürzel des jeweiligen System steht (z. B. TortoiseCVS, TortoiseSVN etc.).

3.2 Verteilte Quellcodeverwaltung

Erste Vertreter der verteilten Versionsverwaltungssysteme waren die Open-Source-Systeme *GNU Arch* und *Monotone*. Arch erschien erstmals 2001, gilt seit 2009 offiziell als veraltet und hat keinen nennenswerten Marktanteil mehr. Stattdessen konzentriert sich die Weiterentwicklung auf den *Arch*-Fork *Bazaar* [Wikc]. *Monotone* konnte sich ebenfalls nicht mehr am Markt behaupten und wurde von neueren Systeme verdrängt [Ham10]. David Roundy begann 2002 die Arbeit an einem neuen Format zur Speicherung der Daten in GNU Arch. Die Ergebnisse flossen zwar nicht in Arch ein, führten aber zur Entwicklung des Patch-basierten Versionskontrollsystems *Darcs*, das schließlich 2003 veröffentlicht wurde [Wikb]. Wohl aufgrund der unkonventionelle Funktionsweise von Darcs, die zwar einige besondere Möglichkeiten bietet aber Probleme mit sich bringt, konnte dieses System auch bis 2010 keinen besonderen Marktanteil erlangen [Ham10]. Weitere Information zum internen Aufbau und den Folgen daraus finden sich in Abschnitt 6.2.3 auf Seite 31.

Einen großen Einfluss auf die Entwicklung nachfolgender Open-Source-Versionsverwaltungssysteme hatte das kommerzielle Produkt *BitKeeper*. BitKeeper wurde 2000 veröffentlicht und wurde anschließend auch zur Versionierung des Linux-Kernels eingesetzt. Dies war möglich, da die Herstellerfirma BitMover Inc. zum damaligen Zeitpunkt eine kostenfreie Nutzung ihres Systems für Open-Source-Projekte ermöglichte. Das zugrunde liegende Lizenzmodell wurde im Jahr 2005 umgestellt. Ab diesem Zeitpunkt war eine kostenfreie Nutzung der Software nicht mehr möglich – für den Linux-Kernel musste daher ein neues Quellcodeverwaltungs-Werkzeug gefunden werden. [Wika]

Da für dieses Projekt kein geeignetes System auf dem Markt gefunden werden konnte, begann Linus Torvalds mit der Entwicklung von *Git*. Eine erste Version des auf den Be-

dürfnissen der Kernel-Entwickler zugeschnittenen Gits erschien bereits im gleichen Jahr (2005). [ITWa] Seitdem konnte die Software viele Anhänger gewinnen und wird nicht nur bei vielen Projekten im Linux-Umfeld verwendet [Gitc]. Der Marktanteil betrug 2010 etwa 2,7 % [Ham10].

Zu etwa der selben Zeit, zu der Torvalds mit der Entwicklung von Git begann, startete Matt Mackall die Entwicklung des ebenfalls quelloffenen *Mercurial*. Das verfolgte Ziel war auch diesmal, eine Alternative zu BitKeeper zu schaffen. [ITWb] Neben Projekten im Linux-Umfeld verwendet u. A. Mozilla Mercurial [Merg].

Im Bereich der kommerziellen Versionsverwaltungssysteme konnten sich neben BitKeeper noch *AccuRev SCM* von AccuRev Inc. (0,4 %) und *Rational Team Concert* der IBM durchsetzen (0,3 %) (Stand 2010 nach [Ham10]).

Wie bei den zentralen Quellcodeverwaltungsystemen verfügen auch, zumindest die letztgenannten kommerziellen Produkte, über eine grafische Benutzeroberfläche. Bei den quelloffenen Systemen lässt sich auch hier mit weiteren Open-Source-Projekten (z. B. TortoiseGit, TortoiseHg) eine grafischen Oberfläche installieren.

4 Momentan verwendetes System

Die Flexis AG setzt seit einigen Jahren Subversion als Versionsverwaltungssytem ein. Dabei wird der Quelltext in zwei verschiedenen Respositorys gespeichert, auf die sich ca. 10 Gigabyte an Quelltext und Binärdateien verteilen.

In den Repositorys sind alle Projekte abgelegt; wird an einem Projekt gearbeitet so wird dieses aus dem Repository ausgecheckt. Da viele Projekte die gleiche Codebasis verwenden, wird das svn:externals-Feature genutzt, um diese Codebasis in das Projekt einzubinden. Außerdem wird Keyword-Expansion eingesetzt, um Dateien mit der aktuellen Version kennzeichnen zu können und so beim Kunden die Möglichkeit zu haben, feststellen zu können welche Version aktuell eingesetzt wird. Aktuell treten beim Committen von Dateien ins Repository regelmäßig Konflikte auf.

Da viele Mitarbeiter keine weitgehende Kenntnis von Kommandozeilen-Tools haben, werden in der Firma hauptsächlich grafische Oberflächen wie TortoiseSVN zur Verwaltung des Repositorys eingesetzt; einige setzen aber auch die Kommandozeile ein. Manche Mitarbeiter arbeiten auch von zuhause aus über eine VPN-Verbindung. Deshalb ist es wichtig, dass die Kommunikation zwischen Repository-Server und Client möglichst ressourcensparend ist.

4.1 Mitarbeiterbefragung

Viele Mitarbeiter sind keine studierten Softwareentwickler sondern in die Softwareentwicklung „reingerutscht“. Die Akzeptanz des aktuellen Systems ist sehr hoch, da alles zumindest für den alltäglichen Bedarf einigermaßen funktioniert. Die Kenntnis der einzelnen Mitarbeiter unterscheidet sich dabei sehr stark. Manche Mitarbeiter nutzen nur die Konsole, um ihre Arbeit mit Subversion zu erledigen, andere können mit dem Repository nur über eine grafische Oberfläche arbeiten und wären ohne diese Unterstützung eher hilflos.

Für Mitarbeiter, die nicht so tief in der Materie stecken wie die anderen, ist ein Wechsel auf ein neues System nur schwer vorstellbar, vor allem weil das aktuelle System in ihren Augen eigentlich ausreichend ist. Sie würden einen Wechsel allerdings nicht ablehnen, sondern, tatsächliche Verbesserungen vorausgesetzt, akzeptieren und auch versuchen sich in das neue System, mit erhoffter Unterstützung, einzuarbeiten.

Mitarbeiter, die eher ein Interesse für die Technik hinter der Softwareentwicklung zeigen, wünschen sich ein System, welches mehr Möglichkeiten bietet, um unabhängig voneinander zu arbeiten und Code untereinander zu tauschen. Vor allem auch mehrere Repositorys für

unterschiedliche Zwecke wie Entwicklung, Test und Deployment aufzusetzen, gehörte zu den vorgeschlagenen Ideen.

Fast alle Mitarbeiter bemängelten die schlechte Behandlung von Konflikten in Subversion und würden sich wünschen, dass diese effektiver funktioniert.

Je nach Abteilung unterscheidet sich auch die Art der Nutzung. Im Marketing beispielsweise ist das aktuelle System schon „kompliziert genug“ und eine Verschlechterung der Benutzerfreundlichkeit wäre nicht gewünscht.

Einige Mitarbeiter schlugen vor, dass eine Kombination aus bestehendem und neuem System vielleicht die beste Lösung sein könnte, sodass sich jeder Mitarbeiter langsam an den Umstieg gewöhnen kann.

5 Bewertungskriterien

In diesem Kapitel wird auf die Kriterien eingegangen, die Einfluss auf die Wahl eines geeigneten Versionsverwaltungssystems haben. Im Rahmen der Mitarbeiterbefragung (siehe Abschnitt 4.1 auf Seite 15) wurden die Bedürfnisse der Mitarbeit näher untersucht, um hieraus die Gewichtung der Kriterien ableiten zu können. Hierbei kristallisierten sich einige K. O.-Kriterien heraus, die von den Mitarbeitern oder der Firmenleitung als unverzichtbar angesehen werden.

Die nachfolgend genannten Funktionen orientieren sich an Subversion, dem bei der Flexis AG bisher verwendeten System. Da die Mitarbeiter mit Subversion vertraut sind, ist davon auszugehen, dass diese von Subversion angebotenen Funktionen intensiv verwendet werden oder für die Arbeit zwingend erforderlich sind.

5.1 K. O.-Kriterien

In diesem Abschnitt werden alle Kriterien aufgelistet, die als unverzichtbar gelten bzw. zwingend erforderlich sind.

5.1.1 JIRA-Integration

Zur Verwaltung der Projekte und zur Aufgabenverteilung wird die kommerzielle Projektmanagement-Software JIRA der australischen Firma Atlassian verwendet. Eine Integration der Versionsverwaltung in JIRA ist somit zwingend erforderlich. Beim bisher verwendeten System ist dies der Fall.

Die Integration in JIRA findet über Plugins statt, so dass für jedes in Frage kommende Versionsverwaltungssystems das Vorhandensein eines solchen Plugins zu prüfen ist. Das Fehlen eine entsprechenden Plugins gilt damit als Ausschlusskriterium.

5.1.2 GUI-Unterstützung

Gerade Open-Source-Versionsverwaltungssysteme verfügen standardmäßig nur über einen Kommandozeileninterpreter. Um eine komfortablere Bedienung zu ermöglichen, sind für viele verbreitete Versionsverwaltungssysteme grafischer Benutzeroberflächen (GUIs) entstanden, die sich meist in den Dateimanager des Betriebssystems integrieren. Die Mitarbeiterbefragung ergab, dass für Subversion die Oberfläche „TortoiseSVN“ auf den Arbeitsplätzen

installiert ist. Von der Mehrheit der befragten Mitarbeiter wird die grafische Oberfläche bevorzugt verwendet und als ein äußerst wichtiges Kriterium angesehen, da es ihren Arbeitsablauf vereinfacht. Insbesondere wird das Beheben von Merge-Konflikten durch die Oberfläche erleichtert.

Da bisher eine grafische Benutzeroberfläche verwendet wird und die Mitarbeiter an eine solche gewöhnt sind, muss für alle in Frage kommenden Systeme eine selbige verfügbar sein. In diesem Zusammenhang ist das verwendete Betriebssystem von Bedeutung, da die grafischen Oberflächen meist nur für Microsoft Windows verfügbar sind. Da die bisherige grafische Benutzeroberfläche nur für Microsoft Windows verfügbar ist und die Mitarbeiter hauptsächlich dieses Betriebssystem verwenden (siehe Abschnitt 5.1.4), muss für die in Frage kommenden Systeme mindestens für Windows eine grafische Oberfläche verfügbar sein.

Von den Mitarbeitern, die ebenfalls unter Linux arbeiten, wird eine grafische Oberfläche nicht als zwingend, allerdings als interessante Zusatzfunktion angesehen. Damit kann die Verfügbarkeit einer GUI unter Linux als Entscheidungshilfe angesehen werden.

5.1.3 IDE-Plugin

Neben einer eigenständigen Benutzeroberfläche (siehe Abschnitt 5.1.2 auf der vorherigen Seite) gaben einige Mitarbeiter an, dass sie unter der integrierten Entwicklungsumgebung (IDE) Eclipse das Subversion-Plugin verwenden. Da sie dieses Plugin als unverzichtbar ansehen, muss auch für alle in Frage kommenden alternativen Systeme ein Plugin für die Entwicklungsumgebung verfügbar sein.

5.1.4 Betriebssysteme

Für die Quellcodeverwaltung selbst ist das verwendete Betriebssystem in der Regel von untergeordneter Bedeutung, da die Systeme im Allgemeinen für die gängigsten Betriebssysteme verfügbar sind. Auf den Firmenarbeitsplätzen wird Microsoft Windows eingesetzt, wobei einige Mitarbeiter auch unter Linux arbeiten. Somit müssen alle in Frage kommenden Versionsverwaltungssysteme sowohl für Linux als auch für Windows verfügbar sein.

5.1.5 Teilweises Auschecken

Da die eingesetzten Repositorys gegenwärtig sehr groß sind (mehrere Gigabytes, siehe Kapitel 4 auf Seite 15) ist das vollständige Herunterladen des Repositorys, wie es bei verteilten Systemen typisch ist, zu vermeiden. Da von den Mitarbeitern auch von einem Heimarbeitsplatz oder bei einem Kunden auf das Repository zugegriffen wird, ist es unzumutbar, dass für die Aktualisierung des Repositorys Daten heruntergeladen werden, die vom Mitarbeiter nicht benötigt werden.

Um die für die Aktualisierung benötigte Zeit kurz zu halten, muss daher vom Versionsverwaltungssystem die Möglichkeit geboten werden, nur tatsächlich benötigte Teile des

Repositorys herunterzuladen bzw. zu aktualisieren. Alternativ kann diesem Problem begegnet werden, sofern das System mit geringem Aufwand das Aufteilen der Repositorys in kleinere Teilrepositorys unterstützt (wodurch nur jeweils ein kleineres, benötigtes Repository heruntergeladen oder aktualisiert werden muss). In diesem Fall muss das System jedoch Funktionen anbieten, die die durch das Aufteilen verlorenen Beziehungen innerhalb des ehemaligen Repositorys nachbilden.

5.1.6 Externals

Subversion bietet über die s. g. „Externals“ die Möglichkeit, Verzeichnisse aus anderen Repositorys in ein eigenes Repository einzubinden. Ändern sich die Daten im eingebundenen Repository, werden diese im eigenen automatisch ebenfalls aktualisiert. Diese auch als „Vendor Branches“ bekannte Funktion wird daher oft dazu verwendet, Komponenten von Drittfirmen und Zulieferern in das firmeninterne Repository zu übernehmen und aktuell zu halten – so auch bei der Flexis AG.

Von alternativen Systemen wird daher eine Funktionalität erwartet, mit der sich die genannten Beziehungen zwischen verschiedenen Repositorys nachbilden lassen.

5.1.7 Export

Die Konfiguration und der Zustand der Arbeitskopie der Repositorys wird bei vielen Versionsverwaltungssystemen meist in besonderen Konfigurationsverzeichnissen oder -dateien gespeichert. Diese befinden sich oft im Hauptverzeichnis der Arbeitskopie, gelegentlich aber auch (etwa bei Verwendung eines älteren Subversion-Clients) in jedem Verzeichnis der Arbeitskopie. Gerade in letztgenanntem Fall ist das Erstellen einer sauberen Kopie der Daten, etwa zur Auslieferung oder für den Build-Prozess, mit nicht unerheblichem Aufwand verbunden, wenn diese Konfigurationsverzeichnisse bzw. -dateien manuell aus der angelegten Kopie entfernt werden müssen.

Um das Erstellen einer Kopie der versionierten Daten zu erleichtern, existiert in Subversion die Export-Funktion, die zu einer gegebenen Revisions-Nummer eine Kopie der zugehörigen Daten erstellt. Unter Verwendung dieser Funktion lassen sich zudem diverse Prozess-Schritte automatisieren. Da dies bei der Flexis AG der Fall ist, müssen alternative Systeme eine ähnlich einfache Möglichkeit bieten, über entsprechende Kommandos oder alternativ über Zusatzprogramme eine solche, saubere Kopie zu generieren.

5.1.8 Benutzerfreundlichkeit

Schon in Abschnitt 5.1.2 auf Seite 17 ist klar geworden, dass viele Benutzer wert auf eine möglichst simple, verständliche und grafische Benutzerführung legen. Konkret wurde von vielen Mitarbeitern gefordert, dass die Benutzerführung mindestens so einfach wie in Subversion sein soll. Das bedeutet z. B. auch, dass gewöhnte Arbeitsweisen sich im

neuen Werkzeug möglichst ähnlich wiederfinden sollten ohne den Prozess übermäßig kompliziert zu machen. Da die Benutzer bisher hauptsächlich über den Dateimanager Windows-Explorer auf die Versionsverwaltung zugreifen, müssen sich auch alternative Systeme in den Dateimanager integrieren und hierüber zumindest auf die geläufigsten Funktionen Zugriff gewähren. Ebenfalls problematisch ist der Fall, wenn zwar eine grafische Unterstützung existiert, die große Mehrheit der Benutzer oder Dokumentation aber die Arbeit über die Konsole forciert.

5.1.9 Konversion von Subversion

Ein logischer Schritt in der Erwägung eines anderen Versionskontrollsystems ist natürlich die Frage, wie gut sich das vorhandene System in das neue umwandeln lässt. Fehlt diese Möglichkeit, kämen die Konsequenzen einem Datenverlust gleich – schließlich müsste jede alte Änderung separat in Subversion begutachtet werden oder es käme zu Konflikten zwischen alt und neu. Allerdings kann auch diese Konversion nicht ganz ideal sein, bspw. wenn sie nur Teile aus dem alten Repository übernimmt, z. B. wenn nur die Revisionen, nicht aber die Commit-Beschreibungen konvertiert würden.

5.2 Entscheidungskriterien

Bei den in diesem Abschnitt genannten Kriterien handelt es sich um Anforderungen, die von einigen Mitarbeitern oder Firmenleitung genannt und gewünscht, jedoch nicht als zwingend angesehen wurden. Damit erleichtern es diese Kriterien, zwischen zwei Systemen, die beide die K. O.-Kriterien erfüllen, zu entscheiden.

5.2.1 Lizenz

Beim bisher verwendeten Versionsverwaltungssystem „Subversion“ und der dazugehörigen Benutzeroberfläche „TortoiseSVN“ handelt es sich um Open-Source-Projekte, wodurch sich die Anwendungen unentgeltlich einsetzen lassen. Für die Mitarbeiter von untergeordneter Rolle, für die Firmenleitung aber umso wichtiger, ist daher die Frage nach der Lizenz, unter der die Systeme verwendet werden dürfen.

Prinzipiell wurde uns mitgeteilt, dass für die Flexis AG sowohl Open-Source- als auch kommerzielle Systeme in Frage kommen. Aus dem finanziellen Gesichtspunkt wird jedoch eine Open-Source-Lösung bevorzugt. Bei der Untersuchung der kommerziellen Systeme spielen die Lizenzkosten (Grundkosten und Kosten je Arbeitsplatz, ggf. pro Jahr) daher eine wichtige Rolle. Durch eine Hochrechnung der anfallenden Kosten bei Ausrüstung aller Arbeitsplätze (ca. 100 laut Befragung, siehe Abschnitt 4.1 auf Seite 15) mit dem entsprechenden Versionsverwaltungssystem ergibt sich ein Überblick über den finanziellen Aufwand einer etwaigen Umrüstung. Eine Entscheidung soll sich damit erleichtern lassen.

5.2.2 Dateiberechtigungen

Da besonders unter Linux Dateiberechtigungen eine wichtige Position einnehmen, wird eine Mitversionierung dieser Berechtigungen von den Mitarbeitern, die auch unter Linux arbeiten, als wünschenswert angesehen. Dieser Wunsch bezieht sich dabei hauptsächlich auf die Speicherung des Executable-Bits, d. h. der Kennzeichnung einer Datei als ausführbare Anwendung.

5.2.3 Keywords

Über die Subversion-Funktion „Keyword Expansion“ lassen sich in Text-Dateien (Quellcode etc.) definierte Schlüsselwörter durch Informationen aus dem Repository ersetzen. Ein gängiger Anwendungsfall ist etwa die Verwendung des Schlüsselworts „ID“ welches beim Aktualisieren der Arbeitskopie automatisch durch Informationen über die zugehörige Datei (u. a. Revisionsnummer der letzten Änderung und Benutzer, der diese Änderung durchgeführt hat) ersetzt wird.

Von den Mitarbeitern wird diese Funktion aufgrund einiger mit einhergehender Probleme kontrovers diskutiert, da etwa beim automatischen Vergleichen der Daten identische Dateien

durch eine unterschiedlicher Ersetzung der Schlüsselworte als verschieden markiert werden. Da diese Funktion dennoch von einigen Mitarbeitern verwendet wird, sollte sie auch von alternativen Systemen unterstützt werden.

5.2.4 File-Lock

Durch das Setzen von File-Locks auf eine Datei kann der Benutzer in Subversion verhindern, dass diese Datei von einem anderen Benutzer geändert werden kann. Erst durch eine explizite Freigabe der Sperre lässt sich die Datei durch andere Benutzer verändern. Da diese Funktion damit dem zugrunde liegenden Gedanken einer Versionsverwaltung, nämlich dem gleichzeitigen Arbeiten an den versionierten Daten, darunter eben auch das gleichzeitige Arbeiten an ein und derselben Datei, widerspricht, wird diese Funktion hauptsächlich nur verwendet, wenn ein gleichzeitiges Bearbeiten einer Datei nicht möglich ist. Dies ist typischerweise bei Binär-Dateien der Fall.

Da in den Repositorys der Flexis AG nur wenige Binär-Dateien gespeichert sind und diese zudem nur selten geändert werden, wird diese Funktion von keinem der befragten Mitarbeitern verwendet. Daher wird von den alternativen Systemen das Vorhandensein einer Lock-Funktion nicht verlangt.

5.2.5 Sammeln bzw. Zwischenspeichern von Commits

Um die in den Repositorys gespeicherten Quelldaten lauffähig zu halten, sind die Mitarbeiter bestrebt, durchgeführte Änderungen erst hochzuladen, wenn diese möglichst fehlerfrei sind. Gleichzeit entsteht aber auch der Wunsch, bereits kleinere Änderungen in die Versionsverwaltung einzuchecken (etwa um nach nicht zufriedenstellenden Änderungen auf einen definierten Zwischenstand zurückspringen zu können).

Um beide Seiten vereinen zu können, erweist es sich als wünschenswert, wenn von alternativen Versionsverwaltungssystemen die Möglichkeit geboten wird, das Einchecken von Änderungen und das finale Hochladen dieser in ein zentrale Repository zu trennen. Das heißt die Fähigkeit, Commits zwischenzuspeichern und die gesammelten Commits schließlich in das Repository einzustellen

Ebenfalls bieten einige Systeme die Möglichkeit, Änderungen in einem eigenen lokalen Bereich zwischenzuspeichern. Hierdurch lässt sich weiterhin das lokale Repository bzw. die Arbeitskopie aktualisieren, ohne das nicht eingecheckte Änderungen verloren gehen oder zu Konflikten während des Vorgangs führen.

Da beide Funktionen die Arbeit mit kleineren lokalen oder trotz kleinerer lokaler Änderungen erleichtern, kann deren Verfügbarkeit zur Entscheidungshilfe bei der Wahl eines alternativen Systems herangezogen werden.

5.2.6 Suchen von Changesets

Gelegentlich entsteht der Bedarf, nach einem Changeset zu suchen, in dem eine bestimmte Änderung an einer Datei vorgenommen wurde. Ein solcher Fall stellt beispielsweise die Suche nach dem betreffenden Changeset dar, in dem ein bestimmter Fehler in die Anwendung eingefügt wurde.

Die befragten Mitabeiter gaben an, dass die Suche einer bestimmten Revision gelegentlich vorkommt. In diesem Fall bedienen sie sich des Logs der betreffenden Datei und durchsuchen dieses manuell nach der betreffenden Änderung. Einige Versionsverwaltungssysteme bieten hierfür Funktionen an, die den Aufwand bei dieser Suche verringern können. Somit handelt es sich bei dieser Suche um eine wünschenswerte Funktion.

5.2.7 Effizienz

Auf die Frage nach der Effizienz erhielten wir von den befragten Mitarbeitern meist die Antwort, dass das bisherige System (d. h. Subversion) ausreichend schnell sei. Dies lässt sich z. T. auch darauf zurückführen, da hauptsächlich innerhalb des Firmengebäudes, somit innerhalb des lokalen Netzes, auf die Repositorys zugegriffen werden. Der durch Subversion benötigte Speicherplatz stellte ebenfalls kein Problem dar.

Für alternative Systeme bedeutet dies, dass ihre Effizienz mindesten auf Subversion-Niveau sein muss.

6 Auswertung

6.1 Bewertungsübersicht

Zu jedem in Frage kommenden Werkzeug wird eine kurze Übersichtsgrafik dargestellt, die möglichst schnell repräsentieren soll, ob eine aus Kapitel 5 relevante Anforderung voll erfüllt (●), gar nicht erfüllt (○) oder teilweise erfüllt (◐) wird.

Dabei wurde platzbedingt versucht, die Kategorien durch kleine Symbole und Abkürzungen darzustellen. K. O.-Kriterien sind unterstrichen. Die Verweise auf die zugehörigen Bewertungskritieren finden sich hochgestellt dahinter. Eine solche Grafik sieht z. B. so aus:

JIRA	GUI	IDE	OS	✂	⋈	⇒	☺	⇝	©	rwx	Keywords	🔊	Suchen	🕒
○	○	○	○	○	○	○	○	○	○	○	○	○	○	○

JIRA steht für die Integration in Atlassian JIRA, wie in Abschnitt 5.1.1 beschrieben.

GUI steht für die Unterstützung einer graphischen Benutzeroberfläche, wie in Abschnitt 5.1.2 beschrieben.

IDE steht für das Vorhandensein einer IDE-Integration, wie in Abschnitt 5.1.3 beschrieben.

OS steht für die Unterstützung der Betiebssysteme, wie in Abschnitt 5.1.4 beschrieben.

✂ steht für ein teilweises Auschecken von Repositorys, wie in Abschnitt 5.1.5 beschrieben.

⋈ steht für die Unterstützung einer SVN-externals-artigen Einbindung von Dritt-Repositorys, wie in Abschnitt 5.1.6 beschrieben.

⇒ steht für die Möglichkeit eines sauberen Exportierens, wie in Abschnitt 5.1.7 beschrieben.

☺ steht für die Einschätzung, dass das Werkzeug sehr benutzerfreundlich ist, wie in Abschnitt 5.1.8 beschrieben.

⇝ steht für die Möglichkeit, ein vorhandenes SVN-Repository in ein Repository des genannten Werkzeugs zu überführen, wie in Abschnitt 5.1.9 beschrieben.

© steht für eine möglichst günstige Lizenz, wie in Abschnitt 5.2.1 beschrieben.

rwx steht für die Speicherung von Dateiberechtigungen, wie in Abschnitt 5.2.2 beschrieben.

Keywords steht für die Unterstützung von Keyword Expansion, wie in Abschnitt 5.2.3 beschrieben.

steht für die Möglichkeit, Commits lokal zwischenzuspeichern und zu bearbeiten, wie in Abschnitt 5.2.5 beschrieben.

Suchen steht für die Möglichkeit, Commits effizient zu suchen, wie in Abschnitt 5.2.6 beschrieben.

steht für die Effizenz des Werkzeugs, wie in Abschnitt 5.2.7 beschrieben.

6.2 In Frage kommende Werkzeuge

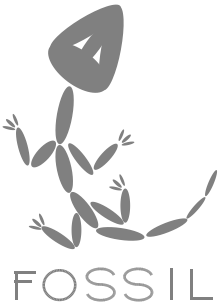
Wie in Kapitel 3 erläutert, befinden sich unzählige Quellcodeverwaltungs-Systeme auf dem Markt. In diesem Kapitel werden deshalb nur Systeme vorgestellt die trotz einer Vorauswahl weiterhin in Frage kommen.

AccuRev SCM ist ein kommerzielles Produkt der AccuRev Inc. Die offizielle Webseite befindet sich auf www.accurev.com/accurev.html.

Bazaar ist ein Open-Source-Projekt und unter der GPL v2-Lizenz veröffentlicht. Die offizielle Webseite befindet sich auf bazaar.canonical.com.

Darcs ist ein Open-Source-Projekt und unter der GPL-Lizenz veröffentlicht. Die offizielle Webseite befindet sich auf www.darcs.net.

Fossil ist ein Open-Source-Projekt und unter der BSD-Lizenz veröffentlicht. Die offizielle Webseite befindet sich auf www.fossil-scm.org.



Git ist ein Open-Source-Projekt und unter der GPL v2-Lizenz veröffentlicht. Die offizielle Webseite befindet sich auf www.git-scm.org.

IBM Rational Team Concert ist ein kommerzielles Produkt der IBM Corporation. Die offizielle Webseite befindet sich auf ibm.com/software/rational/products/rtc/.

Mercurial ist ein Open-Source-Projekt und unter der GPL v2-Lizenz veröffentlicht. Die offizielle Webseite befindet sich auf mercurial.selenic.com.

Perforce ist ein kommerzielles Produkt der Perforce Software Inc. Die offizielle Webseite befindet sich auf www.perforce.com.

Plastic SCM ist ein kommerzielles Produkt von Codice Software. Die offizielle Webseite befindet sich auf www.plasticscm.com.

6.2.1 AccuRev SCM

AccuRev SCM ist ein kommerzielles Versionsverwaltungssystem der AccuRev Inc. Es wird seit 2002 stetig weiterentwickelt und von namhaften Firmen wie der NASA, Siemens oder Sony eingesetzt.

JIRA	GUI	IDE	OS	✂	⋈	⇒	☺	⇝	©	rwx	Keywords	🔊	Suchen	🕒
●	●	●	●	●	●	◐	●	●	○	●	●	●	○	●

System

Bei AccuRev SCM handelt es sich um ein kommerzielles Produkt, für dessen Einsatz eine kostenpflichtige Lizenzierung erforderlich ist [Acca]. Pro Nutzer (Lizenz) fällt eine einmalige Gebühr in Höhe von 1.151,81 € ($1.500) an. Auf 100 Benutzer hochgerechnet entspricht dies Kosten in Höhe von ca. **115.181 €**. Die Integration von AccuRev in JIRA kostet pro Server 764,03 € ($995).

Im Gegensatz zu den meisten anderen vorgestellten Systeme verfolgt AccuRev, wie Subversion, den zentralen Ansatz. Die gesamte Kommunikation geschieht über einen zentralen Server. Trotzdem ist es möglich in kleinen Teams in einer Kopie, getrennt vom Rest des Repositorys, in sogenannten *Streams* zu arbeiten und anschließend die Ergebnisse wieder im Haupt-Stream zu integrieren. Außerdem kann man lokale Änderungen mit dem Server synchronisieren ohne diese einzuchecken, damit ist beispielsweise das Weiterarbeiten an einem anderen PC möglich (*Private Checkin*).

AccuRev SCM bietet für JIRA ein Plugin namens „AccuSync“ an, mit welchem die Versionsverwaltung direkt in JIRA integriert werden kann.

Außerdem gibt es sowohl für das Kontextmenü des Explorers unter Windows (ähnlich zu „TortoiseSVN“) als auch für Eclipse eine Erweiterung, die das Arbeiten mit dem Repository sehr einfach möglich machen soll. Das Eclipse-Plugin wird direkt von AccuRev SCM angeboten und ist kostenlos.

Das Herzstück des AccuRev-Versionsverwaltungssystem ist der eigenständige Client. Mit diesem können alle Funktionen, die das AccuRev-System beherrscht, ausgeführt werden. Trotz allem unterstützt AccuRev auch das Arbeiten mit der Konsole.

Funktionsumfang

AccuRev Inc. bewerben ihr Produkt als leistungsstarkes Konzept zur mehrstufigen kontinuierlichen Integration. Damit ist gemeint, dass AccuRev SCM die Möglichkeit bietet, Änderungen am Quelltext über mehrere Ebenen zu verwalten und so je nach Entwicklungsfortschritt verschiedene Repositorys zu nutzen.

Außerdem bietet es nativ die Unterstützung zum einfachen lokalen Branchen und Mergen. Das bedeutet, dass man lokal Änderungen vornehmen kann und diese dann nach ausgiebigen Tests wieder zurück in den Hauptzweig des Repositorys mergt.

Laut Webseite ist auch eine sehr komfortable Verwaltung mehrerer Versionen möglich [Accb].

Mit AccuRev ist durch den zentralen Ansatz auch ein teilweises Auschecken des Repositorys möglich.

Es ist nur möglich das Execution-Bit der Dateiberechtigungen zu setzen, alle anderen Berechtigungseinstellungen werden vom AccuRev-System ignoriert. Außerdem wird eine Keyword-Expansion wie in Subversion unterstützt.

Der Export einer bestimmten Revision ist nach dem Checkout der Revision durch einfaches Kopieren des Ordners möglich. Eine binäre Suche nach Revisionen ist hingegen nicht möglich.

Einen direkten Externals-Support wie Subversion bietet AccuRev nicht. Es kann aber mit einfachen symbolischen Links, die AccuRev mit in das Repository einchecken kann, gearbeitet werden um ein sehr ähnliches Verhalten zu erreichen.

6.2.2 Bazaar

Bei Bazaar, auch kurz bzr, handelt es sich um ein frei verfügbares, unter der GPL lizenziertes, verteiltes Konfigurationsverwaltungs-Werkzeug, das von Canonical Ltd. gefördert wird.

JIRA	GUI	IDE	OS	✂	⋈	⇒	☺	⇝	©	rwx	Keywords	🔊	Suchen	🕒
◐	●	◐	●	○	●	●	●	●	●	○	●	●	●	●

System

Als GPL-Werkzeug ist Bazaar frei verfügbar. Bazaar selbst wirbt mit Anpassungsfähigkeit an verschiedene Arbeitsmodelle, Effizienz, Flexibilität und besonderer Benutzerfreundlichkeit. Zahlreiche Features werden in Bazaar über Plugins realisiert. Trotz der Tatsache, dass Bazaar u. a. mit seiner Community wirbt und schon seit längerer Zeit existiert, soll sich ein Buch darüber nur „in Entwicklung“ befinden. [Bzrc]

Bazaar steht für diverse Betriebssysteme zur Verfügung, u. a. für Linux und für Windows [Bzra]. Es gibt auch ein Plugin für Eclipse, allerdings kann dies zu einem Absturz von Eclipse führen, wenn auf der Kommandozeile Authentifizierungsinformationen abgefragt werden [Bzrh]. Es gibt eine Tortoise-GUI namens TortoiseBzr, die sich ähnlich wie TortoiseSVN verhält [Bzri]. Dazu kommt auch ein JIRA-Plugin [Bzrg], dessen Dokumentation allerdings eher spärlich ausfällt und das seit 2008 auch nicht mehr weiterentwickelt wurde.

Mit Hilfe von bzr-svn [Bzrf] lassen sich Subversion-Repositorys von Bazaar aus bearbeiten, so als würde es sich um Bazaar-Repositorys handeln. Ebenso lassen sich damit auch SVN-Repositorys nach Bazaar konvertieren.

Funktionsumfang

Als verteiltes Versionskontrollsystem besitzt jeder Entwickler in Bazaar ein lokales Repository, in das er zunächst committen kann. Die Änderungen müssen dann gesondert je nach Kollaborationsstil entweder an die Repositorys anderer Entwickler oder das Haupt-Repository weitergeleitet werden. Dafür ist ein teilweiser Checkout nicht möglich. [Bzra]

Bazaar kennt ein zu Subversion vergleichbares Export-Kommando, mit dem Teile des Repositorys in beliebiger Form (Verzeichnisstruktur oder Archiv) exportiert werden können. Als Dateiberechtigung wird nur das Execution Bit, also die Erlaubnis zur Ausführung, mitversioniert. [Bzra]

Keyword-Expansion [Bzre], Externals [Bzrd] und *Bisect* [Bzrb], also die schnelle, binäre Suche nach einem bestimmten Commit, stehen in Bazaar ebenfalls als Plugins zur Verfügung.

6.2.3 Darcs

Darcs, hierbei handelt es sich um ein Akronym für „Darcs adavanced revision control system“, zählt zu den verteilten Systemen.

JIRA	GUI	IDE	OS	✂	⋈	⇒	☺	⇝	©	rwx	Keywords	🔊	Suchen	🕒
○	◐	●	●	◐	◐	●	◐	◐	●	○	○	●	●	○

System

Darcs kann für alle gängigen Betriebssysteme (darunter auch Windows und Linux) von der Homepage heruntergeladen werden [Dara]. Hierin enthalten ist allerdings nur der Kommandozeilen-Interpreter. Die Kommandos die zur Bedienung des Systems benötigt werden, wurden möglichst einfach und intuitiv gehalten: Werden den Kommandos nicht alle benötigten Parameter übergeben, so wird der Benutzer interaktiv abgefragt, wie das Kommando ausgeführt werden soll (beim Einchecken von Änderungen wird der Benutzer bei fehlender Angabe von Parametern etwa für jede geänderte Datei gefragt, ob er diese mit Einchecken möchte). Da es sich um ein Open-Source-Projekt handelt, fallen für die Nutzung keine Gebühren an.

Zur Integration in den Windows Explorer und als eigenständige Benutzeroberfläche lässt sich gegenwärtig lediglich das Open-Source-Projekt „TortoiseDarcs“ [HT] finden, bei dem die ähnlich lautende CVS-Oberfläche übernommen und an Darcs angepasst wurde. Die Benutzeroberfläche integriert sich in den Windows Explorer und ähnelt im Bedienkonzept der in der Firma verwendeten Oberfläche „TortoiseSvn“. Zwar steht für Darcs auf der Homepage ein eigenes Wiki und eine ausführliches Handbuch [Rou] zur Verfügung, die Beschreibungen orientieren sich jedoch ausschließlich an der Verwendung der Kommandozeile. Da die neuste Veröffentlichung der grafischen Oberfläche zudem aus dem Jahr 2006 stammt, ist davon auszugehen, dass diese mit der Weiterentwicklung von Darcs nicht schritthalten wird. Für Eclipse 3.x existiert zur Integration von Darcs das Plugin „EclipseDarcs“ [FG], das im Gegensatz zur zuvor erwähnten Benutzeroberfläche eine höhere Entwickleraktivität aufweist.

Die Suche nach einem Darcs-Plugin für das Projektmanagementsystem JIRA im Atlassian Marktplatz [Atlc] sowie eine allgemeine Internetsuche nach eben diesem ergab keine Treffer – eine Integration des Versionsverwaltungssystems ist damit ohne Weiteres nicht möglich.

Darcs hebt sich insbesondere durch seinen internen Aufbau von den meisten Versionsverwaltungssytemen ab: Diese speichern die Änderungen meist intern in einem Baum. Werden Änderungen in das System eingecheckt, so wird dem Baum ein neuer Eintrag mit den betreffenden Änderungen angehängt. Hierdurch stehen alle Änderungen, und damit auch die gespeicherten Versionen, in einer chronologischen Folge. Darcs hingegen fußt auf der Patch-Theorie [Rou]. Für jede eingecheckte Änderung wird eine Patch-Datei generiert, in der die durchgeführten Änderungen gespeichert werden. Somit existiert für jeden Commit, in Darcs als *record* bezeichnet, ein eigener Patch. Die Reihenfolge des Eincheckens selbst, d. h.

die chronologische Abfolge, wird vom System nicht gespeichert. Damit existieren zwischen den verschiedenen Patches per se keine zeitliche Abhängigkeiten.

Wurden die Änderungen via *record* eingecheckt, können diese an ein anderes (z. B. zentrales Repository) übertragen werden (*push*). Patches, die auf einem externen Repository liegen, jedoch noch nicht in das eigene, lokale Repository übernommen wurden, lassen sich mit dem Befehl *pull* in dieses herunterladen und damit die Arbeitskopie aktualisieren. Da es sich somit um ein verteiltes Versionsverwaltungssystem handelt, ist das Einchecken von Änderungen und das Zurverfügungstellen dieser von Haus aus getrennt.

Da wie eingangs beschrieben zwischen den Patches keine Reihenfolge definiert ist, werden von Darcs während des Herunterladens die Abhängigkeiten zwischen den einzelnen Patches berechnet um sicherstellen zu können, dass alle Dateien korrekt wiederhergestellt werden. Dies führt dazu, dass die Patches in einer Reihenfolge angewendet werden, die der Reihenfolge des Eincheckens nicht mehr zwingend entspricht.

Funktionsumfang

Dank des patch-zentrierten Aufbaus und dem Nicht-Vorhandensein einer strikten Reihenfolge zwischen den Patches bietet Darcs im Vergleich zu anderen Quellcodeverwaltungssystemen einige besondere Funktionen, die die tägliche Arbeit erleichtern können. Gleichzeitig kann die durch die Patches entstehende Komplexität zu Problemen führen. Diese Aspekte sollen nachfolgend, vor allem in Bezug auf die Bewertungskriterien (siehe Kapitel 5 auf Seite 17), näher betrachtet werden.

Darcs bietet beim initialen Kopieren bzw. Herunterladen eines Repositorys auf das lokale System mit dem Kommando *get* die Möglichkeit an, eine „lazy copy“ des Repositorys zu erstellen [Hö11]. Hierbei werden durch Angabe des Parameters *lazy* nur diejenigen Patches aus dem externen Repository geladen, die zum Generieren einer Arbeitskopie mit den aktuellsten Daten benötigt werden. Werden weitere Patches benötigt (z. B. um einen älteren Arbeitsstand einer Datei wiederherzustellen), werden diese vom System bei Bedarf automatisch heruntergeladen.

Bereits aus dem internen Aufbau heraus stehen die s. g. spontaneous Branches zur Verfügung. Dabei wird der Namen der Patches (d. h. die beim Record angegebene Beschreibung) zur Generierung von Branches verwendet [Hö11]. Um die Funktion nutzen zu können, muss die Benennung nach einem bestimmten Schema erfolgen, wobei alle zusammengehörigen Patches (alle Patches, die einen Brach repräsentieren sollen) etwa mit dem selben Namensanfang beginnen müssen. Über einen zusätzlichen Parameter kann nun beim Aktualisieren des Repositorys (*pull*) oder beim Übertragen der Daten an ein externes Repository (*push*) angegeben werden, dass ausschließlich Patches mit einem bestimmten Namensanfang heruntergeladen und auf die Arbeitskopie angewendet bzw. hochgeladen werden sollen.

Beide Funktionen ließen sich etwa wie folgt anwenden, um dem Problem der bei der Flexis AG verwendeten, sehr großen Repositories zu begegnen: Soll das Repository vollständig auf einen Arbeitsplatz übertragen werden, um mit einer vollständige Arbeitskopie arbeiten zu

können, so lässt sich mit einem „lazy-get“ die zu übertragende Datenmenge reduzieren, weil nicht die vollständige Historie heruntergeladen werden muss. Werden zudem alle Patches konsequent so benannt, dass deren Name stets mit dem Namen bzw. Kürzel des Teilprojekts beginnen, zu dem sie gehören, lässt sich die zu übertragende Datenmenge weiter reduzieren, wenn nur die Daten zu einem bestimmten Teilprojekt benötigt werden. Hierbei genügt es dann, beim Aktualisieren bzw. beim Übertragen des Repositories, eben diesen Namen anzugeben, wodurch nur die nötigsten Patches heruntergeladen werden.

Aufgrund der nicht vorhandenen Chronologie erlaubt Darcs es, Patches und somit die Daten aus verschiedenen Repositorys in ein eigenes Repository zu laden. Bei diesem Vorgang wird standardmäßig die Vereinigungsmenge der Patches gebildet, wodurch alle Daten der verschiedenen Repositorys gemeinsam zur Verfügung stehen. Dieses Verfahren lässt sich damit zur Einbindung von Fremdrepositorys (also Externals bzw. Vendor Branches, siehe Abschnitt 5.1.6 auf Seite 19) verwenden. Neben der Vereinigungsmenge lässt sich alternativ auch die Schnitt- oder die Komplementärmenge bilden (siehe auch die Beschreibung des *pull* Kommandos in [Rou]). Zur Einbindung externer Repositpories sind diese beiden allerdings unbedeutend. Damit externe Repositories eingebunden werden können, müssen diese verständlicherweise als Darcs-Repositories geführt werden. Hierfür bietet sich das Open-Source Tool „Tailor“ an, mit dem sich Repositorys verschiedener Systeme ineinander konvertieren lassen [Tai]. Liegt das externe Repository somit nicht als Darcs-Repository vor, muss dieses zunächst mittels Tailor in ein Darcs Repository umgewandelt werden. Zwar lässt es sich nun in das eigene Repository einbinden, durch die manuell durchgeführte Konvertierung werden am original Repository durchgeführte Änderungen später aber nicht übernommen, da hierfür eine erneute Konvertierung nötig wäre. Sollen Änderungen also automatisch übernommen werden, so kann eine Lösung darin bestehen, die Konvertierung regelmäßig automatisch zu wiederholen und das konvertierte Repository auf einem eigenen Server zur Einbindung bereitzustellen.

Da es sich um ein verteiltes Versionsverwaltungssystem handelt, ist für das Einchecken von Änderungen (also das Generieren von Patches) und das Betrachten der Historie ohne Netzwerkkommunikation möglich. Dadurch ist Darcs im Vergleich zu Subversion bei den häufigsten Anweisungen und bei kleineren Repositorys oft schneller. Bei größeren Repositorys schneidet Darcs hingegen, insbesondere wenn die Kommunikation zwischen Subversion Client und Server über eine schnelle Netzwerkverbindung stattfindet, schlechter ab [Darb]. Beim Herunterladen und Mergen von Änderungen (also beim Zusammenführen mehrerer Patches, in denen Änderungen an einer gleichen Datei vorgenommen wurden) über das *pull*-Kommando kam es insbesondere vor der Version 2 häufig zu einem von der Anzahl der Konflikte abhängigen exponentiellen Laufzeitverhalten. Das Durchführen der Aktualisierung konnte dabei über eine Stunde dauern [Thr]. Dieses Problem konnte mit Darcs-2 abgeschwächt werden, wo dieses Verhalten seltener zu Tage tritt. Gänzlich verhindern lies es sich jedoch nicht [Darc].

Mit dem *dist* Kommando lassen sich der aktuelle Zustand oder ältere Zustände des Repositorys in ein tar-Archive exportieren. Nicht eingecheckte Änderungen und das Darcs-Konfigurationsverzeichnis werden dabei nicht übernommen.

Das Setzen von Dateiberechtigungen auf einzelne Datei wird von Darcs nicht unterstützt. Ebenso wird das automatische Ersetzen von Schlüsselwörtern (Keyword Expansion) nicht unterstützt.

Mit dem Kommando *trackdown* stellt Darcs die Möglichkeit zur Verfügung, die Patches des Repositorys nach einer bestimmten Änderung zu durchsuchen. Durch Angabe des Parameters *bisec* führt das System statt der linearen Suche eine binäre Suche durch, womit sich die benötigte Zeit reduzieren lässt. Das Kommando selbst wurde absichtlich recht einfach gehalten: Für die Suche wird dem Kommando für den benötigten Vergleich lediglich ein gültiger Kommandozeilenbefehl oder ein Skript übergeben. Die Suche wird nun solange durchgeführt, bis der erste Patch gefunden wurde, bei der der Befehl bzw. das Skript einen Erfolg zurückmeldet. Um z. B. die letzte Version der Datei „Record.lhs“ zu finden, in der diese noch den Eintrag „FIXME“ enthält, könnte das Kommando in der Form *darcs trackdown 'grep FIXME Record.lhs'* verwendet werden. Dieses und weitere Beispiele finden sich im Darcs Benutzerhandbuch [Rou].

Die Konvertierung eines Subversion-Repositorys nach Darcs ist nur über die Zusatz-Anwendung „Tailor“ (Open-Source) möglich. Bei der Konvertierung scheint Tailor jedoch Probleme mit Dateien und Verzeichnissen zu haben, die mit dem Subversion-Kommando *move* verschoben wurden. In einem Wiki-Artikel [Dard] wird beschrieben, wie die Konvertierung durchgeführt und dieses Problem umgangen werden kann.

6.2.4 Fossil SCM

Das Versionsverwaltungssystem Fossil ist ein verteiltes System welches von D. Richard Hipp, der auch Autor der SQLite-Datenbank ist, entwickelt wurde. Fossil vereint neben der Versionsverwaltung auch einen Bug-Tracker und eine Wiki-Software.

JIRA	GUI	IDE	OS	✂	⋈	⇒	☺	⇝	©	rwx	Keywords	🔊	Suchen	🕒
○	●	○	●	○	○	◐	◐	●	●	●	○	●	●	●

System

Fossil ist unter der BSD Open-Source-Lizenz veröffentlicht und kann auf allen gängigen Betriebssystemen genutzt werden. Fossil bietet neben dem Modul der Versionsverwaltung einen Webservice an der das Repository direkt über das HTTP-Protokoll verfügbar macht. So kann ein Server sehr einfach mit nur einem Befehl eingerichtet werden. Die Konfiguration des Repositorys geschieht dann über eine Weboberfläche.

Weitere Features, die Fossil von anderen Versionsverwaltungssystemen wie Git abhebt, sind, dass in Fossil direkt ein Bug-Tracker integriert ist, der auch über den Webservice erreichbar ist. Außerdem lässt sich auch ein Wiki-System, das in das Versionsverwaltungssystem integriert ist, über den Webservice anzeigen.

Fossil bietet im Gegensatz zu Subversion kein zentrales Repository an, sondern erlaubt es dem Nutzer ein eigenes Repository aufzubauen, das dann in ein anderes Repository *gepusht* werden kann. Unter *Pushen* versteht man das Übertragen des eigenen Entwicklungsstand in ein anderes Repository. Nach einem abgeschlossenen *Push* sind die beiden Repositorys dann auf dem gleichen Stand. Dadurch bietet sich im Vergleich zu Subversion die Möglichkeit über mehrere Instanzen eine Änderung von einem Test-Repository in ein Repository mit stabilen Versionen zu übertragen.

Fossil erlaubt dadurch auch kein „Auschecken“ mehr, sondern nur das komplette *Klonen* eines bestehenden Repositorys. Dadurch ist auch kein partieller Checkout möglich, was aber durch eine laut [Fosc] sehr effiziente Kompression des Repositorys kompensiert werden kann.

Fossil bietet durch den direkt integrierten Bug-Tracker kaum Anbindungen an andere Ticket-Systeme, das betrifft auch das nicht vorhandene JIRA-Plugin. Eine Integration in Eclipse bietet Fossil auch nicht, dafür gibt es eine GUI, die eine sehr einfache Verwaltung des Fossil-Repositorys erlauben soll [Fosb]. Außerdem gibt es den bereits erwähnten Webservice, der auch eine sehr umfangreiche Verwaltung des Repositorys zulässt.

Laut Internetseite wird Fossil von einigen größeren Projekten wie SQLite und TCL genutzt. Die Repositorys dieser Projekte sind mehrere GB groß und sollen der Webseite nach trotzdem noch sehr performant laufen, was dann die Anforderungen an die Effizienz erfüllt.

Eine weitere gute Funktion, die Fossil bietet, ist die automatische Synchronisation des Repositorys mit einem anderen Repository. So werden mögliche Konflikte frühzeitig erkannt und können dadurch einfacher behoben werden.

Die Speicherung des Fossil-Repositorys wird in einer SQLite-Datenbank vorgenommen. Dies erlaubt atomare Änderungen (Commits) und durch automatische Checks wird eine unbemerkte Korrumpierung der Datenbank verhindert.

Funktionsumfang

Fossil bietet die Möglichkeit, das Execute-Flag der Dateiberechtigungen im Repository zu speichern. Weitere Flags der Dateien werden aber nicht übernommen.

Fossil bietet keine Export-Funktion, um einzelne Revisionen zu exportieren. Es kann aber einfach der aktuelle Checkout kopiert werden, da im Vergleich zu alten Subversion-Versionen keine Ordner oder Ähnliches angelegt werden. Eine Keyword-Expansion ist hingegen nicht vorhanden und Fossil bietet dafür auch keinen Alternativ-Mechanismus.

Fossil bietet auf der Kommandozeile den Befehl *bisect*. Mit dessen Hilfe kann man eine binäre Suche starten, um für eine gesuchte Änderung einen Commit zu finden, in der zum Beispiel ein Bug zum ersten Mal aufgetreten ist. Damit kann in logarithmischer Zeit der Commit gefunden werden, in dem das Verhalten zum ersten Mal aufgetreten ist.

Um ein bestehendes Subversion-Repository nach Fossil zu portieren benötigt es einen Zwischenschritt über ein Git-Repository, da Fossil von Haus aus nur aus Git importieren kann. Des Weiteren werden der Anleitung nach auch nur „einfache“ Subversion-Repositorys unterstützt [Fosa].

6.2.5 Git

Das verteilte Versionsverwaltungssystem Git oder auch git wurde ursprünglich von Linus Torvalds für die Entwicklung des Linux-Kernels geschrieben, wobei das Hauptaugenmerk auf Geschwindigkeit lag.

JIRA	GUI	IDE	OS	✂	⋈	⇒	☺	⇝	©	rwx	Keywords	🔊	Suchen	🕒
●	●	●	●	○	●	◐	◐	●	●	◐	◐	●	●	●

System

Als Linux-Werkzeug ist Git unter einer Open-Source-Lizenz verfügbar. Git ist prinzipiell nur verteilt, ermöglicht aber verschiedene Workflows. Zum Beispiel lässt sich auch verteilt ein „Subversion-Style Workflow“ mit einem einzigen zentralen Repository nachbilden, aber auch eine hierarchische Staffelung von verschiedenen Repositorys ist möglich. Ein „Integration Manager“ kann z. B. nur bestimmte Änderungen in ein ‚hochheiliges‘ Repository übernehmen. [Gita]

Dabei tritt ein sehr markantes Merkmal von Git zu Tage: Es gibt kein „Auschecken“ im herkömmlichen Sinne mehr, die einzige Möglichkeit besteht darin, das Repository zu *klonen*. Da somit ein lokales Git-Repository immer eine vollständige Kopie mit allen Commits beinhaltet, ist auch kein partielles Auschecken möglich. Allerdings wird dies von Git selbst explizit als Feature gewertet, da somit bei jedem Entwickler stets ein vollständiges Backup bereitsteht. Ebenso sollen ganze Git-Repositorys durch Kompression nicht erheblich größer sein als ein ausgechecktes Subversion-Repository. [Gita]

Git ist auf vielen Betriebsystemen verfügbar, wozu auch die geforderten zählen. Es existieren diverse GUIs für den Umgang mit Git, darunter auch TortoiseGit [Tora], das TortoiseSVN nachempfunden ist. Mit EGit [EGi] existiert auch ein Eclipse-Plugin für Git. Es sollte allerdings erwähnt werden, dass sich, bedingt durch die Herkunft, z. B. viele Erklärungen zu Git auf Konsolenein- und ausgaben beziehen. Atlassian bietet selbst ein Werkzeug für die direkte Verwaltung von Git für „Enterprise-Teams“ an, welches für 100 Lizenzen etwa 6000 US-$ kostet [Atla]. Es gibt allerdings auch sowohl kostenfreie [JIRb] als auch kommerziell betreute [JIRa] Addons für JIRA, hier lägen die Kosten um 1000 US-$.

Git versteht sich selbst als ein Versionskontrollsystem, in dem Branchen und Mergen besonders einfach und schnell funktioniert. Dadurch soll es im Prinzip erleichtert werden, Änderungen auszuprobieren ohne mit Konsequenzen für die Lauffähigkeit des Projekts rechnen zu müssen. Hierfür unterstützt Git fünf verschiede Merge-Strategien, die gezielt konfiguriert werden können. Weiterhin stechen zwei Features von Git stechen dabei hervor: Es gibt zunächst eine sogenannte *Staging Area*, in der ein Commit nach Belieben zusammengebaut und geändert werden kann, bevor er letztendlich tatsächlich durchgeführt wird. Es muss nicht mal ein vollständiger Commit durchgeführt werden, sondern die Änderungen können auch in einen lokalen *Stash* zwischengespeichert werden, um temporär zurück zum Originalzustand zu gelangen und z. B. erst einmal andere Änderungen zu begutachten. Auch

wenn er durchgeführt wird, gelangt ein Commit auch nur in das lokale Repository, und muss erst mittels *pushen*, in ein fremdes Repository, oder *pullen*, von einem anderen Repository aus, verteilt werden. [Gita]

Git kennt keine Revisionen, sondern nur *Commits*, die auch nicht inkrementell nummeriert werden, sondern über ihre Hashsumme identifiziert werden. Dies hat mehrere Vorteile, wie z. B. wird dadurch die Integrität der Daten sichergestellt: Der Repository-Inhalt kann nicht korrumpiert werden ohne, dass sich dabei die Commit-Hashes ändern. Dabei basiert der Hash nicht nur auf dem Inhalt, der sich im Repository befindet, sondern auch auf allen vorangegangenen Commits. [Gita]

Da Git für den Linux-Kernel entwickelt wurde, der selbst kein kleines Repository darstellt, war die Geschwindigkeit, mit der Git arbeitet, ein zentraler Aspekt. Durch die „Trennung" der Repositorys wird die Übertragung von Daten im Vergleich zu Subversion minimiert, womit viele Operationen wesentlich schneller durchgeführt werden können, als wenn die Informationen erst vom zentralen Server abgerufen werden müssten. Git wirbt allerdings damit, dass fast alle Operationen schneller sind als in Subversion – bis auf das initiale Klonen. [Gita]

Funktionsumfang

Für einen detaillierteren Einstieg in Git empfiehlt sich ein Blick in das kostenfrei online verfügbare „Pro Git" [Cha09]. Dort wird auch der Befehl *bisect* erklärt, mit dessen Hilfe eine binäre Suche nach einem bestimmten Commit durchgeführt werden kann, in der bspw. erstmals ein Bug aufgetreten ist. Dabei ist es nur nötig, den Suchraum einzugrenzen und eine logarithmische Anzahl von Commits als „gut" oder „schlecht" zu bewerten, damit Git automatisch den Commit ausfindig macht, in dem das schlechte Verhalten zum ersten Mal aufgetreten ist.

Ebenfalls findet sich dort eine Erläuterung von Externals für Git, die dort *Submodules* heißen und nativ unterstützt werden. Dabei wird der exakte Commit des Submoduls im übergeordneten Repository mitgespeichert, sodass sich die Umgebung exakt wiederherstellen lässt. Um ein fremdes Repository einbinden zu können, muss dieses als Git-Repository angeboten werden. Daher ist es ggf. nötig, die „Git-Version" eines nicht in Git verwalteten Repositorys nach dessen Konvertierung an einem zentralen Punkt anzubieten [Sta].

Etwas anders sieht es bei der Keyword-Expansion aus: Diese wird von Git nicht unterstützt, da Git Dateien wie Commits über einen eindeutigen Hash identifiziert. Git bildet den Hash einer Datei aber vor dem Commit, wobei der Commit-Hash auch wieder vom Hash der Datei abhängt. Das prinzipielle Verhalten lässt sich aber mit Git *Attributes* nachbilden, wobei hier während des Aus- bzw. Eincheckens die Information hinzugefügt bzw. gelöscht werden. Somit werden Änderungen nicht mitversioniert, aber die tatsächlich verwendeten Dateien enthalten nützliche Informationen. [Cha09]

Es existiert keine beliebige Export-Funktionalität in Git. Es gibt allerdings diverse Konsolenbefehle und Skripte, die ähnliche Ergebnisse erzielen. TortoiseGit unterstützt die native

Export-Funktion, mit deren Hilfe die gesamte ausgecheckte Verzeichnisstruktur mit Dateien in ein ZIP-Archiv exportiert werden kann.

Git erlaubt nur zwei Arten von Datei-Berechtigungen: Entweder Lesen und Schreiben für den Besitzer, Nur-Lesen für alle anderen („644“[1]), oder für jeden zusätzlich noch ausführen („755“). Dabei werden vorhandene Berechtigungen immer in eine der beiden umgewandelt.

Mit *git-svn* unterstützt Git die Konversion eines Subversion-Repositorys. Eine umfangreiche Erklärung zur Konversion findet sich nicht direkt, aber z. B. in Internet-Blogs [Alb10]. Allerdings unterstützt *git-svn* auch den bidirektionalen Fluss von Änderungen von und zu Subversion und kann somit selbst quasi als Subversion-Client genutzt werden [Cha09]. Dies würde auch ein einfacheres Testen und eine reibungsärmere Übergangsphase ermöglichen, da Git so schrittweise angetestet bzw. eingeführt werden könnte, ohne, dass ein vorhandenes Repository kopiert und ein neuer Server aufgesetzt werden müsste.

[1] `chmod`-Notation, siehe z. B. [chm]

6.2.6 IBM Rational Team Concert

Bei IBM Rational Team Concert handelt es sich um ein kommerzielles kollaborations- und integrationsunterstützendes Werkzeug. Die Version „RTC Developer“ bietet dabei eine verteilte Quellcodeverwaltung.

JIRA	GUI	IDE	OS	✂	⋈	⇒	☺	⇝	©	rwx	Keywords	🔊	Suchen	🕒
●	●	●	●	●	○	○	●	●	○	◐	○	●	○	?

System

IBM Rational Team Concert steht für Windows und Linux zur Verfügung und unterstützt auch Eclipse. Ebenso lässt sich JIRA bidirektional integrieren. [IBM]

Viele Informationen zu Rational Team Concert finden sich auf den Hilfeseiten von IBM oder bei Jazz [Jaz], auf dem Rational Team Concert aufsetzt.

Rational Team Concert wird über den Browser oder IDE-Plugins gesteuert, sodass hier auch eine benutzerfreundliche GUI zur Verfügung steht. Es ist möglich, ein Subversion-Repository für Rational Team Concert zu übernehmen. [Jaz]

Laut Angaben der IBM-Webseite kostet eine Einzelplatzlizenz zwischen 2000 und 5000 US-$, was höchstwahrscheinlich über dem Investitionsrahmen von Flexis liegen dürfte. Dafür handelt es sich um ein volles Kollaborationswerkzeug mit seinen eigenen Funktionen, die aber weder Bestandteil dieser Analyse waren noch zum Prozess von Flexis passen müssen.

Funktionsumfang

Rational Team Concert ermöglicht es, Änderung aus dem aktuellen Bearbeitungsstand, der *Sand Box*, zunächst in einem eigenen *Repository Workspace* zu speichern, das allerdings auch auf dem Server gespeichert wird. Von dort aus kann dann der Code an einen *Stream* übergeben werden, sodass ihn andere bearbeiten können. Streams sind mit Branches aus den gängigen Versionskontrollsystemen vergleichbar. Ebenso definiert Rational Team Concert einige weitere eigene Begriffe wie *Baseline/Snapshot* für Tags und *Change Set* für Revision. [Jaz]

Es ist für Entwickler möglich, nur gewisse Komponenten zur Bearbeitung zu „laden“, was sinngemäß bedeutet, dass Repositoryinhalte teilweise ausgecheckt werden können. Eine Unterstützung für ein „Externals“-artiges Verhalten scheint zwar in Planung zu sein, aber derzeit nicht zu existieren. Keyword Expansion wird nicht unterstützt. Bezüglich der Berechtigungen erfasst Rational Team Concert nur das Benutzer-Executable-Bit. [Jaz]

Zum effizienten Suchen von bestimmten Änderungen und zum teilweisen Export ließen sich keine vertrauenswürdigen Quellen finden, weswegen wir annehmen, dass diese Funktionalität nicht vorhanden ist. Auch zur Effizienz lassen sich keine vergleichbaren Aussagen treffen.

6.2.7 Mercurial

Bei dem in Python implementierten Mercurial handelt es sich um ein verteiltes Versionsverwaltungssystem, das oft auch mit „hg“ abgekürzt wird; Hg steht im Periodensystem der Elemente für Quecksilber, im Englischen „Mercury“.

JIRA	GUI	IDE	OS	✂	⋈	⇒	☺	⇝	©	rwx	Keywords	🔊	Suchen	🕒
●	●	●	●	◐	◐	●	●	●	●	◐	●	●	●	●

System

Von der offiziellen Homepage [Mera] lässt sich die Software für alle verbreiteten Betriebssysteme herunterladen, da bei der Entwicklung ein großer Augenmerk auf Plattformunabhängigkeit gelegt wurde. Da es sich um ein Open-Source Projekt handelt, kann Merurial unentgeltlich genutzt werden. Für gängige Linux Distributionen steht Mercurial in deren eingebauten Paketverwaltungen zur Verfügung. Unter Windows bietet sich der Download der Installationsdatei mit integrierter Benutzeroberfläche „TortoiseHg“ an [Torb]. Die Installation beinhaltet u. a. einen Python-Interpreter, so dass zur Ausführung keine weiteren Komponenten installiert werden müssen.

TortoiseHg lässt sich wie das verwandte TortoiseSvn bedienen und integriert sich ebenfalls in den Windows Explorer. Von TortoiseHg existiert neben einer Windows-Version auch eine Version für Linux und Mac OS X, wobei sie sich unter Linux in den Dateimanager „Nautilus“ integriert und hierin den selben Funktionsumfang bietet wie unter Windows.

Bei der Entwicklung von Mercurial wurde auf eine möglichst einfache Bedienung geachtet. Beispielsweise wird für alle Kommandos ein einheitlicher Aufbau verwendet und soweit möglich die selben Parameter. Das System verfügt über einen eingebauten Webserver, der über das Kommando *server* gestartet wird und über HTTP einen Zugriff auf die Repositorys ohne Installation eines separaten Webserver ermöglicht. Über CGI lässt sich aber auch ein separater Webserver (Apache HTTPD etc.) zum Zugriff auf Mercurial Repositorys einrichten. Alternativ kann auf die Repositorys auch über SSH zugegriffen werden. Einen guten Einstieg in die Benutzung und die Administration von Mercurial bietet das Buch „Mercurial: The Definitive Guide“ [O'So9].

Zur Integration in die Entwicklungsumgebung Eclipse bietet sich das Plugin „MercurialEclipse“ an [Merb]. Das Plugin gilt als ausgereift und bietet Zugriff auf alle Funktionen von Mecurial [Gen11].

Eine Integration in das Projektmanagementsystem JIRA kann auf zwei Wegen erfolgen: Atlassian bietet mit dem kommerziellen Produkt „FishEye“ eine Oberfläche, mit der sich Repositorys durchsuchen und überwachen lassen. FishEye lässt sich in JIRA integrieren und unterstützt neben Subversion unter Anderem auch Mercurial [Atlb]. Alternativ findet sich im Atlassian Marktplatz ein Plugin, über das ein Zugriff auf Mercurial Repositorys ohne FishEye möglich ist [Cus].

Funktionsumfang

Zur Einbindung externer Repositorys unterstützt Mercurial s. g. „Subrepositories“. Hiermit lassen sich fremde Repositorys innerhalb eines selbst festgelegten Verzeichnisses in ein eigenes Repository einhängen – damit entspricht es weitestgehend der Externals-Funktion in Subversion. Neben Mercurial-Repositorys lassen sich mit der Subrepository-Funktion auch Git- und Subversion-Repositories einbinden. In [ara] wird die Verwendung dieser Funktion detailliert beschrieben.

Leider zählen Subrepository zu den „Features of Last Resort“ [Merc]. Es handelt sich also um Funktionen, die möglichst vermieden werden sollten. Die Gründe hierfür liegen u. a. im nicht ausreichend definierten Verhalten von rekursiven Mercurial-Kommandos, also Kommandos die eigenständig auf dem gesamten Repository inklusive der eingebundenen Subrepositorys operieren (wie etwa das Kommando *status* zur Anzeige der geänderten Daten). Darüber hinaus versucht das *push* Kommando zuallererst die eingebundenen Repositorys zu aktualisieren. Sollten diese zeitweise nicht erreichbar sein, schlägt *push* somit fehl. Um diese Probleme zu vermeiden wird empfohlen, keine Fremdrepositorys in Repositorys einzubinden, die Quelldateien enthalten. Stattdessen sollte ein „Master Repository“ angelegt werden in dem ausschließlich mittels Subrepositorys sowohl die eigenen Repositorys als auch die Repositorys aus fremden Quellen eingebunden werden. [Merh]

Mercurial bietet keine Funktion, um nur Teile eines Repositorys auf das lokale System herunterzuladen. Um zu vermeiden, dass das vollständige Repository zur Arbeit heruntergeladen werden muss, bietet sich die Verwendung der bereits vorgestellten Subrepositorys an. Um den oben erwähnten Problemen auszuweichen, empfiehlt es sich auch hier, ähnlich wie bei der Einbindung fremder Repositorys vorzugehen: Nachdem das große Repository in mehrere kleinere aufgeteilt wurde (etwa nach Projekten oder Komponenten), wird anschließend ein „Master Repository“ angelegt, in dem die zuvor angelegten, kleineren Teilrepositorys eingebunden werden. Durch diese Vorgehensweise lässt sich das sehr große Repository aufteilen, ohne die Beziehungen der Projekte bzw. Komponenten untereinander zu verlieren. Gleichzeitig genügt es, nur das Repository herunterzuladen oder zu aktualisieren, das die für die Arbeit benötigten Daten enthält.

Ein unter Linux gesetztes Executable-Bit wird von Mercurial mitversioniert. Da sich dieses Bit allerdings nur mit dem entsprechenden Unix-Befehl, nicht aber mit einem Mercurial-Kommando ändern lässt, ist eine Änderung der Berechtigung unter Windows nicht möglich. Weitere Dateiberechtigungen werden allerdings nicht versioniert. [Mere]

Mit *archive* lassen sich die Daten der aktuellsten oder eine älteren Revision exportieren. Als Exportziel werden dabei neben einem Verzeichnis auch verschiedene Archivtypen unterstützt.

Ebenso zu den „Features of Last Resort“ [Merc] zählt das automatische Ersetzen von Schlüsselwörtern – die Keyword-Expansion – aufgrund der mit dieser Funktion einhergehenden Probleme (siehe hierzu auch Abschnitt 5.2.3 auf Seite 21). Sollte diese Funktion dennoch benötigt werden, lässt sie sich nach dem Aktivieren der bei der Installation von Mercurial bereits mitgelieferten Keyword-Erweiterung verwenden. In der Konfigurationsdatei des

betreffenden Repositorys lassen sich nun die Dateien festlegen, in denen die Schlüsselwortersetzung durchgeführt werden sollen. Standardmäßig unterstützt die Erweiterung die aus CVS bekannten Schlüsselwörter, wie etwa Id oder $Revision$. Bei Bedarf können jedoch auch weitere Schlüsselwörter und deren Zieltext in der Konfigurationsdatei des Repositorys definiert werden. Weitere Informationen hierzu und zur allgemeinen Verwendung der Erweiterung finden sich in der Mercurial Wiki [Merd].

Bei Einsatz von TortoiseHg ist der Zugriff auf das Repository direkt aus dem Dateimanager möglich. TortoiseHg bietet Zugriff auf alle für die Arbeit benötigten Funktionen von Mercurial. Beim Aktualisieren des Repositorys oder bei einem manuellen *Merge* von Branches versucht Mercurial, soweit möglich, die verschiedenen Dateirevision automatisch mit einem Drei-Wege-Merge zusammenzuführen. Sollte das automatische Zusammenführen nicht gelingen, wird zur Behebung des Konflikts ein grafische Drei-Wege-Merge-Werkzeug gestartet. Zur vereinfachten Behebung von auftretenden Konflikten beim Mergen wird auch aus der Kommandozeile heraus, sofern installiert, standardmäßig ein grafisches Tool gestartet. Unter Linux und Windows (zumindest bei installiertem TortoiseHg) wird zur Konfliktbehebung das Werkzeug kdiff3 verwendet. Zu TortoiseHg wird im Internet eine umfangreiche Dokumentation [Be] angeboten, die die Arbeit mit Mercurial über diese Oberfläche erläutert.

Durch die Architektur ist das Einchecken von Änderungen und das Teilen dieser mit anderen Mitarbeitern bereits getrennt. Um zusätzlich private Änderungen zwischenspeichern zu können, etwa um das lokale Repository mit einem *Pull* aktualisieren zu können ohne das Konflikte mit den nicht eingecheckten Dateien entstehen, wurde die Shelve-Erweiterung entwickelt [Merf]. Die Erweiterung erlaubt es, lokale Änderungen in einem geschützten Bereich zwischenzuspeichern und innerhalb der Arbeitskopie die entsprechende Datei auf eine gespeicherte Version zurückzusetzen. Bei Bedarf lassen sich unterschiedliche Dateiversionen in verschieden benannten Shelves sichern.

Mit *bisec* ist eine binäre Suche nach Changesets möglich, die bereits nach wenigen Durchläufen ein Ergebnis liefert. Diesem Kommando wird dafür über einen Parameter bei jedem Aufruf mitgeteilt, ob sich die gesuchte Änderung in der momentan in der Arbeitskopie angezeigten Changeset enthalten ist. Im Anschluss wechselt das Kommando die Arbeitskopie auf ein anderes Changeset. Dieser Vorgang wird solange wiederholt, bis das Changeset gefunden wurde, in dem die gesuchte Änderung eingefügt wurde. [O'S09]

Bryan O'Sullivan hat die Geschwindigkeit von Mercurial mit Subversion verglichen und im Abschnitt „Mercurial Compared with Other Tools“ seines Buches [O'S09] die Ergebnisse beschrieben. Bei den durchgeführten Tests schnitt Mercurial bei allen durchgeführten Operationen besser ab als Subversion.

Mercurial wird mit einer Erweiterung installiert, die mit dem Kommando *convert* das Importieren der Daten und Historie u. A. aus Subversion erlaubt. Auch eine inkrementelle Konvertierung ist möglich, bei der nur die Daten konvertiert werden, die sich seit der letzten Ausführung des Kommandos geändert haben.

6.2.8 Perforce

Perforce ist ein Versionsverwaltungssystem, das von Perforce Software, Inc. entwickelt wird. Es wird von Firmen wie EADS, IBM oder SAP eingesetzt.

JIRA	GUI	IDE	OS	✂	⋈	⇒	☺	⇝	©	rwx	Keywords	🔊	Suchen	🕒
●	●	●	●	○	◐	◐	●	●	○	●	●	●	○	●

System

Bei Perforce handelt es sich um ein kommerzielles Produkt, für dessen Einsatz auf mehr als 20 Arbeitsplätzen oder mehr als 20 Workspaces eine kostenpflichtige Lizenzierung erforderlich ist [Perc]. Zur Lizenzierung bietet Perforce Software zwei Modelle an: Eine normale Lizenz, oder eine Lizenz inkl. Support, welche in den ersten 12 Monaten obligatorisch ist. Pro Nutzer fällt eine jährliche Gebühr in Höhe von 490,99 € ($640) an. Auf 100 Benutzer hochgerechnet entspricht dies jährlichen Kosten in Höhe von ca. **49.098 €**. Für den ab dem 2. Jahr optionalen Support fallen jährlich weitere 122,74 € ($160) je Nutzer an. Was dann im ersten Jahr ca. **61.372 €** entspricht.

Ein Plugin für JIRA („FishEye“) ist auch als kommerzielles Produkt erhältlich, welches einmalig 3.068,66 € ($4.000) kostet und ein Jahr Support bietet, jedes weitere Jahr Support kostet 1.534,33 € ($2.000).

Es existiert für alle gängigen Betriebssysteme eine GUI- als auch eine Konsolenversion des Clients, über den alle Operationen, die das System unterstützt, ausgeführt werden können. Für Windows werden sogar zwei unterschiedliche grafische Oberflächen angeboten. Des Weiteren gibt es für alle Plattformen auch einen Administrations-Client, der die Workspaces verwalten kann.

Es werden auch Plugins für Eclipse und Visual Studio angeboten, welche wie Subclipse das Versionverwaltungssystem direkt in die IDE integrieren [Perb].

Peforce selbst ist mehr ein zentrales Versionsverwaltungssystem als ein verteiltes. Es gibt allerdings die Möglichkeit, Repositorys über mehrere Server zu verteilen, um die Last besser zu verteilen oder auch Repositorys je nach Standort nur dort verfügbar zu machen. Die Repositorys können automatisch zwischen den Servern synchronisiert werden.

Die Daten des Repositorys sind in zwei Teile aufgeteilt. Zum einen gibt es eine proprietäre Datenbank, in der alle Informationen über Versionierung, Konfiguration, Nutzer, Commit-Nachrichten, etc. gespeichert werden. Und zum anderen gibt es ein Verzeichnis auf dem Server, in dem alle eingecheckten Dateien abgelegt werden. Deltas der Dateien werden im RCS-Format gespeichert. Die Datenkbank ist über MD5-Prüfsummen mit den Dateien verknüpft. Außerdem bietet die Datenbank, sofern konfiguriert, eine Wiederherstellungsfunktion an, um im Falle eines Hardwaredefekts diese wiederherstellen zu können.

Ein Import des bestehenden SVN-Repositorys ist mittels des Scripts *SVN2P4* möglich [Perd].

Funktionsumfang

Trotz des zentralen Ansatzes bietet Perforce mittels *P4Sandbox* dem Nutzer die Möglichkeit, eigene private Repositorys zu erstellen und dort auch nur lokale Branches zu erstellen und somit unter anderem nur lokal zu committen.

Perforce bietet keine direkte zu SVN-Externals äquivalente Funktion an. Es gibt allerdings die Möglichkeit in einem Workspace Teile von mehreren Repositorys einzubinden. So kann, wenn die Externals-Quelle in einem Repository vorhanden ist, diese direkt in den Workspace eingebunden werden [Pera].

Wie Subversion bietet Perforce eine Keyword-Expansion an, die es ermöglicht in eingecheckten Dateien automatisch Revision, Datum, etc. zu ergänzen.

Perforce biete keine Möglichkeit nur einen Teil des Repositorys auszuchecken. Durch die Verwendung von Workspaces ist es allerdings möglich ein großes Repository in mehrere kleine logisch getrennte Teil-Repositorys zu unterteilen, die dann mittels des Workspaces wieder zusammengefasst werden.

6.2.9 Plastic SCM

Das Versionsverwaltungssystem Plastic SCM wird von der spanischen Firma „Codice Software“ entwickelt und vertrieben.

JIRA	GUI	IDE	OS	✂	⋈	⇒	☺	⇝	©	rwx	Keywords	🔊	Suchen	🕒
●	●	●	●	●	◐	◐	●	◐	○	○	○	◐	○	◐

System

Plastic SCM lässt sich aufgrund seiner Architektur als hybrides Versionsverwaltungssystem bezeichnen, das sich sowohl zentral als auch verteilt verwenden lässt. Eine Kombination beider Ansätze ist ebenfalls möglich. Das System verwendet eine Client-Server-Architektur, die der Architektur typischer zentralisierter Systeme (wie etwa Subversion) stark ähnelt. Die Repositorys und deren verschiedene Revisionen werden dabei von einem Server-Prozess in einer SQL-Datenbank verwaltet. Je nach Anforderungen an die Datenbank lassen sich verschiedene Datenbanksysteme einsetzten. Die Klienten erhalten über eine Netzwerkverbindung Zugriff auf die gespeicherten Daten bzw. Revisionen [Cod12c]. Im Gegensatz zu typischen rein zentralisierten Systemen bietet Plastic SCM jedoch Funktionen, mit denen sich die auf einem Server gespeicherten Repositorys leicht replizieren lassen, wobei sich die Entwickler bei diesen Funktionen an den verteilter Systeme orientiert haben.

Dies zeigt sich nicht nur in der gleichlautenden Benennung der Befehle (*push*, *pull*), sondern auch im ähnlichen Vorgehen beim Beheben von Konflikten, die beim Replizieren entstehen können (typischerweise wenn eine Datei des selben Branches auf beiden Servern auf unterschiedlich Art geändert wurde). So werden etwa die bei einem *pull* heruntergeladenen Changesets in einem Teilbranch abgelegt, der anschließend mit dem Hauptbranche zusammengeführt (via *merge*) werden muss. Während bei verteilten Versionsverwaltungssystemen aber meist das vollständige Repository repliziert wird, erlaubt Plastic SCM es, lediglich bestimmte Branches auf einen anderen Server zu übertragen oder mit diesem abzugleichen. [Cod12a]

Diese Kombination aus einem Serverprozess, der allein für die Verwaltung der Repositorys zuständig ist, und den angebotenen Funktionen zur Replikation erlauben die Verwendung verschiedener Server-Topologien: Wird lediglich ein Server-Prozess verwendet, auf den alle Klienten über eine Netzwerkverbindung zugreifen, lässt sich das System ähnlich wie Subversion einsetzten. Alternativ ist es jedoch auch möglich, auf jedem Arbeitsplatz neben der Client-Version auch einen (eingebetteten) Server zu installieren. In diesem Fall können die Benutzer ein auf dem zentralen Server abgelegtes Repository in ihren lokalen Server replizieren und haben somit auch ohne eine bestehende Netzwerkverbindung Zugriff auf alle Revisionen und Funktionen des Versionsverwaltungssystems. Das System lässt sich dann also wie ein verteiltes Versionsverwaltungssystem einsetzten. Auch verschiedene Zwischenformen (z. B. mehrere Server an verschiedenen Standorten) sind möglich.

Die Herstellerfirma von Plastic SCM, Codice Software, bietet im Atlassian Marktplatz ein Plugin an, mit dem sich die Versionsverwaltung in JIRA integrieren lässt. Von der Herstellerfirma wird ebenfalls ein Plugin zur Integration in die Entwicklungsumgebung Eclipse angeboten [Cod12b]. Sowohl Client als auch Server des Versionsverwaltungssystems sind sowohl in einer Windows-Version als auch in einer Linux-Version verfügbar. Die grafische Benutzeroberfläche von Plastic SCM ist zentraler Bestandteil des Systems. Es existiert zwar auch ein Kommandozeileninterpreter, über den sich das System bedienen lässt, die Bedienung konzentriert sich aber auf das Verwenden der grafischen Oberfläche, die unter Windows und Linux die selben Funktionen bietet [Cod12c]. Während der Installation wird auf Wunsch eine Integration in den Windows Explorer mit installiert, die Zugriff auf die wichtigsten Funktionen des Systems bietet.

Bei Plastic SCM handelt es sich um ein kommerzielles Produkt, für dessen Einsatz auf mehr als 15 Arbeitsplätzen eine kostenpflichtige Lizenzierung erforderlich ist [Cod]. Zur Lizenzierung bietet Codice Software zwei Modelle an: Abonnement oder unbefristete Lizenz. Beim Abonnement fällt pro Nutzer eine jährliche Gebühr in Höhe von 222,45 € ($279) an. Auf 100 Benutzer hochgerechnet entspricht dies jährlichen Kosten in Höhe von ca. **22.245 €**. Die Kosten der unbefristeten Lizenz betragen einmalig bei 51 bis 100 Nutzern (da Mengenrabatt) 426,55 € je Benutzer. Für den optionalen Support fallen ab dem zweiten Jahr jährlich weitere 94,88 € ($119) je Nutzer an. Für die unbefristete Lizenz fallen somit bei 100 Benutzern einmalig etwa **42.655 €** sowie ggf. weitere jährliche Gebühren in Höhe von ca. **9.488 €** für den (optionalen) Support an.

Funktionsumfang

Das Anlegen einer Arbeitskopie und das Übertragen der Daten in diese Kopie ist in Plastic SCM getrennt. In eine neu erstellte Arbeitskopie werden die Daten über die *update*-Funktion heruntergeladen. Nach Auswahl des zur Arbeit benötigten Changesets werden nur die Daten übertragen, die in dem zugehörigen Branch enthalten sind. Der Hersteller Codice Software empfiehlt, für jedes Arbeitspaket einen eigenen Branch zu erstellen („Branch per Task“). Wurde die Arbeit an diesem Paket und damit an dem Branch beendet, solle der Branch wieder in den Hauptbranch integriert (*merge*) werden. Wird dieser Empfehlung gefolgt, lässt sich die auf die Arbeitsstation zu übertragende Datenmenge reduzieren, sofern der benötigte Branch alle für die Arbeit benötigten Daten enthält. Da wie weiter oben beschrieben das Replizieren auf Branch-Ebene stattfindet – d. h. beim Klonen des Repositorys mit *pull* lässt sich der zu replizierende Branch auswählen – lässt sich auch hierbei die Datenmenge durch geeignetes Anlegen von Branches reduzieren.

Um zu vermeiden, dass Dateien in die Arbeitskopie geladen werden, die für die durchzuführende Arbeit nicht benötigt werden oder von denen sich bereits eine ältere Revision in der Arbeitskopie befindet, die zur Arbeit genügt, lassen sich Dateien und Verzeichnisse in eine *cloak*-Datei eintragen. Alle darin gelistete Dateien werden beim *Update* ignoriert und somit vom Server nicht mehr übertragen.

Mit der Funktion *xlink* besteht die Möglichkeit, externe Repositorys (in Subversion als Externals bezeichnet, siehe Abschnitt 5.1.6 auf Seite 19) in ein anderes Repository einzuhängen [Cod12d]. Von *xlink* werden allerdings ausschließlich externe Plastic SCM Repositorys unterstützt. Sollen Repsitorys eingebunden werden, die nicht mit Plastic SCM verwaltet werden, müssen diese zuvor konvertiert und als Plastic SCM Repository bereitgestellt werden. Dieses Vorgehen führt jedoch dazu, dass Änderungen an den Fremdrepositorys nicht automatisch übernommen werden, weil hierzu die Konvertierung erneut durchgeführt werden muss. Es bietet sich in diesem Fall an, die Konvertierung in regelmäßigen Abständen automatisiert auf einem Serversystem durchzuführen, um einen möglichst aktuellen Datenbestand zu gewährleisten.

Zur Konvertierung eines Repositorys bedient sich Plastic SCM dem in Git verwendeten *fast-export* Format [Cod12c]. Mit dem Werkzeug „svn-all-fast-export“ [Mac] ist das Erstellen einer solchen Exportdatei aus Subversion heraus möglich. Anschließend kann die erstellte Datei über die Kommandozeile mit dem *fast-import* Kommando in ein Plastic SCM Repository importiert werden. Sofern das Ausgangs-Versionsverwaltungssystem das Erstellen einer inkrementellen Export-Datei unterstützt (hierbei wird anstelle der vollständigen Historie nur die Historie ab einer bestimmten Revision exportiert), lässt sich ein bereits konvertiertes Repository mit geringerem Aufwand auf den aktuellen Stand bringen.

Plastic SCM verfügt über ein recht ausführliches Rechtesystem, über das sich der Zugriff auf die Dateien des Repositorys steuern lässt [Cod12c]. So lässt sich beispielsweise festlegen, welcher Benutzer eine neue Revision einer Datei in das Repository einchecken darf. All diese Rechte finden jedoch auf der Ebene der Versionsverwaltung statt und sind unabhängig von den im Dateisystem des Betriebssystems gesetzten Dateiberechtigungen. Auf das Versionieren dieser Dateiberechtigungen geht die Dokumentation des Versionsverwaltungssystems nicht ein. Auch ließen sich außerhalb der Dokumentation keine Hinweise auf eine Speicherung der Berechtigungen finden und im Test wurde nach dem Ändern einer Berechtigung die betreffende Datei nicht als geändert markiert. Somit ist davon auszugehen, dass dies von Plastic SCM nicht unterstützt wird.

Ebenso konnten keine Hinweise auf eine Export-Funktion in der Dokumentation gefunden werden. Auch in der praktischen Erprobung des Versionsverwaltung konnte eine solche Funktion nicht gefunden werden. Anzumerken ist allerdings, dass das System lediglich im Hauptverzeichnis der Arbeitskopie einen Konfigurationsverzeichnis („.plastic“) anlegt, was das manuelle Exportieren erleichtert. Von Plastic SCM ebenfalls nicht unterstützt ist das automatische Ersetzen von Schlüsselwörtern [Pos11].

Wie zu Beginn des Abschnittes beschriebe, ist die grafische Benutzeroberfläche ein zentraler Bestandteil von Plastic SCM. In die Oberfläche integriert ist eine grafische Darstellung der Changesets und Branches sowie deren Beziehungen zueinander. Hierin lässt sich durch Auswahl eines Changesets im gewünschten Branch leicht ein weiterer Unterbranch anlegen oder dieser mit einem anderen Branch zusammenführen (*merge*). Beim *Mergen* versucht die Versionsverwaltung Konflikte soweit möglich automatisch zu lösen. Ist dies nicht möglich, wird ein integriertes grafisches Drei-Wege-Merge-Werkzeug geöffnet, in dem der Konflikt behoben werden kann.

In einem als *Shelves* bezeichneten Bereich lassen sich in Plastic SCM die an einer Datei durchgeführten Änderungen lokal zwischenspeichern. Die Änderungen werden damit auch ohne Einchecken der Dateien gesichert und es lässt sich somit bei Bedarf eine gesicherte Version wiederherstellen. Da beim Einchecken (*Commit*) die Daten grundsätzlich an einen Server übertragen werden, ist ein lokales Zwischenspeichern bzw. Trennen von Commits und deren Upload, nur möglich, wenn auf dem lokalen System neben dem Client auch der Plastic SCM Server installiert ist (siehe auch die Erläuterung der Architektur im Abschnitt „System“). Wurde das System auf diese weise Eingerichtet und befindet sich eine Kopie des Repositorys im lokalen System, können Änderungen auch ohne Netzwerkverbindung eingecheckt und später gemeinsam übertragen werden. Die Arbeitsweise entspricht damit derer klassischer verteilter Versionsverwaltungssysteme.

Eine binäre Suche nach Änderungen und den zugehörigen Changesets wird von dem Versionsverwaltungssystem nicht unterstützt. Eine solche Funktion wird weder in der Dokumentation erwähnt, noch konnte sie bei der Erprobung von Plastic SCM gefunden werden.

7 Empfehlung

7.1 Übersicht

In der folgenden Tabelle werden nochmals alle Bewertungen grob zusammengefasst, sodass ein schneller Überblick möglich ist. Für die Bedeutung siehe Kapitel 6.

Name	JIRA	GUI	IDE	OS	✂	⋈	⇒	☺	⇝	©	rwx	Keywords	🔊	Suchen	🕒
AccuRev	●	●	●	●	●	●	◐	●	●	○	●	●	●	○	●
Bazaar	◐	●	◐	●	○	●	●	●	●	●	○	●	●	●	●
Darcs	○	◐	●	●	◐	◐	●	◐	◐	●	○	○	●	●	○
Fossil	○	●	○	●	○	○	◐	◐	●	●	●	○	●	●	●
Git	●	●	●	●	○	●	◐	◐	●	●	◐	◐	●	●	●
IBM RTC	●	●	●	●	●	○	○	●	●	○	◐	○	●	○	?
Mercurial	●	●	●	●	◐	◐	●	●	●	●	◐	●	●	●	●
Perforce	●	●	●	●	○	◐	◐	●	●	○	●	●	●	○	●
Plastic	●	●	●	●	●	◐	◐	●	◐	○	○	○	◐	○	◐

7.2 Diskussion

Betrachtet man die Bewertungen der Werkzeuge in vertikaler Richtung, fällt sehr schnell auf, dass alle Werkzeuge auf den gewünschten Betriebssystemen verfügbar sind und fast alle, zumindest teilweise, lokal committen und von SVN importieren können. Auch bietet jedes Werkzeug mehr oder weniger eine grafische Unterstützung.

Horizontal gesehen erfüllen nur wenige Werkzeuge wirklich alle K. O.-Kriterien. Erstaunlich ist dabei, dass die kommerziellen Werkzeuge nicht unbedingt mehr Anforderungen erfüllen als die frei verfügbaren.

Weiterhin ist uns bei der Auswertung aufgefallen, dass die Keyword Expansion von vielen Werkzeugen sehr kritisch gesehen wird. Gerade in verteilten Systemen gibt es oft parallele Versionen, die nicht einheitlich sequentiell wie bei Subversion durchnummeriert werden können. Dazu stört das Einsetzen von zusätzlichen Informationen die Erfassung der tatsächlichen Änderungen, weswegen die allgemeine Empfehlung dahingeht, die Informationen während des Builds hinzuzufügen und nicht im Quellcodemanagement.

Am besten schneidet eindeutig Mercurial ab, das sowohl alle Kriterien wenigstens teilweise erfüllt als auch frei verfügbar ist. Als zweite Alternative würden wir Git hervorheben, das lediglich das teilweise Auschecken nicht unterstützt, was sich aber ähnlich wie bei Mercurial

(aber komplizierter) umgehen lässt und in einem verteilten System fast unumgänglich ist. Dazu kommt, dass Git ebenso wie Mercurial kostenfrei erhältlich ist und ansonsten auch alle Kriterien erfüllt. Als einziges Werkzeug neben Mercurial erfüllt AccuRev zwar alle K. O.-Kriterien, allerdings muss dafür auf die binäre Suche verzichtet werden und ein nicht unerheblicher Preis gezahlt werden, weswegen es vermutlich die dritte Wahl darstellt.

Mercurial und Git haben auch den Vorteil, dass sich, da sie frei verfügbar sind, allgemein auch mehr Informations- und Dokumentationsquellen dazu finden. Ebenso ermöglicht dies eine größere Auswahl an Dritthersteller-Software und Erweiterungen, die bei proprietären Systemen oft nur gegen weitere Gebühren beim Hersteller zu erhalten sind.

Die beiden Alternativen zwischen Mercurial und Git stellt gewissermaßen auch eine Art Glaubensfrage dar. Während Git meistens schneller ist und zahlreiche komplexe Möglichkeiten und Funktionen bietet, ist Mercurial wesentlich benutzerfreundlicher und die Repositorys meistens etwas kleiner. [Gitb, Git10]

7.3 Einführungsstrategie

Um ein neues System einzuführen sind wieder Mercurial und Git im Vorteil, da diese nicht nur ein kontinuierliches Konvertieren ermöglichen, sondern auch (bei Mercurial als Erweiterung) als Subversion-Client benutzt werden können, sodass ein bidirektionales Austesten möglich ist. Sollte sich kein neues System durchsetzen können, könnte sich bereits diese Arbeitsweise für technisch mehr versierte Mitarbeiter als Lösung anbieten.

Dabei kann man sich mit der Bedienung der Werkzeuge vertraut machen. Unter Umständen sollte testweise ein Repository erst konvertiert werden, bevor Geschwindigkeit und Größe beim Auschecken verglichen werden. Dadurch lässt sich die individuelle Präferenz herausfinden.

Funktioniert dies zufriedenstellend, so kann das gesamte System eingesetzt werden. Dabei sind wahrscheinlich die Repositorys in kleinere Einzelteile aufzugliedern und entsprechend neu zu konfigurieren.

7.4 Mögliche Probleme bei der Umstellung

Da die Umstellung einer Quellcodeverwaltung naturgemäß sehr gravierend ausfällt, dürfte das Hauptproblem die Akzeptanz der Entwickler sein. Da viele Mitarbeiter mit dem aktuellen System weitgehend zufrieden sind, werden sie eine Umstellung nicht unbedingt befürworten. Außerdem sinkt natürlich temporär durch die Anpassung an die Umstellung die allgemeine Produktivität.

Daher sollte eine Umstellung mit Rücksicht auf die Konsequenzen genau abgewägt werden und genau geprüft werden, ob die erhofften Verbesserungen dadurch eintreten werden oder sich nicht neue Probleme auftun.

8 Zusammenfassung und Ausblick

In dieser Arbeit wurde der Flexis AG ein passendes Quellcodeverwaltungs-Werkzeug empfohlen. Dazu wurden zunächst Grundlagen der erhältlichen Quellcode- und Versionsverwaltungen besprochen und anschließend eine grobe Marktübersicht dargelegt. Im weiteren Vorgehen wurden die konkreten Anforderungen von Flexis, sowohl zwingend erforderliche als auch gewünschte, durch eine Mitarbeiterbefragung erhoben. Die Kriterien wurden erläutert und eine gezielte Auswahl an 9 Werkzeugen getroffen, die eingehend auf die Anforderungen beleuchtet wurden.

Schließlich wurde eine Darstellung entwickelt, um alle Werkzeuge miteinander in einer Tabelle vergleichen zu können. Es wurden Auffälligkeiten bei den Werkzeugen und den Anforderungen erläutert. Dazu wurde unsere Top-3-Empfehlung ausgesprochen und begründet. Zuletzt wurde eine Möglichkeit zur inkrementellen Einführung dargelegt und mögliche Probleme dabei aufgezeigt.

Ausblick

Die Untersuchung erhebt selbstverständlich keinen Anspruch auf Vollständigkeit; obwohl versucht wurde, alle verbreiteten und nützlichen Werkzeuge auszusuchen, besteht die Möglichkeit, dass in der Zukunft neue Quellcodeverwaltungs-Werkzeuge hinzukommen oder alte sich verändern. Praktisch dürfte dies allerdings nicht allzu bald der Fall sein. Auf praktischer Seite ist ein nächster, denkbarer Schritt in naher Zukunft sicherlich die Überprüfung der Vorschläge, um gewünschte Verbesserungen nachzuweisen und eventuelle Probleme im Betrieb aufzudecken. Auch aus akademischer Sicht macht eine (breitere) Untersuchung der Brauchbarkeit der Funktionalitäten der Versionsverwaltungssysteme im realen Betrieb und eine mögliche Weiterentwicklung der Werkzeuge durchaus Sinn.

Literaturverzeichnis

[Acca] AccuRev Licensing and Pricing. http://www.accurev.com/licensing.html. (Zitiert auf Seite 28)

[Accb] AccuRev Whitepaper - Top 10 Reasons Why Software Development Is Better With AccuRev. http://www.accurev.com/sites/default/files/document/top10-reasons-software-development-better-with-accurev.pdf. (Zitiert auf Seite 29)

[Alb10] J. Albin. Converting a Subversion repository to Git. http://john.albin.net/git/convert-subversion-to-git, 2010. (Zitiert auf Seite 39)

[ara] aragost Trifork. Subrepositories. http://mercurial.aragost.com/kick-start/en/subrepositories. (Zitiert auf Seite 43)

[ASo9] B. de Alwis, J. Sillito. Why are software projects moving from centralized to decentralized version control systems? In *Cooperative and Human Aspects on Software Engineering, 2009. CHASE '09. ICSE Workshop on*, S. 36 – 39. 2009. (Zitiert auf Seite 9)

[Atla] Atlassian Stash. http://www.atlassian.com/software/stash/overview/jira-dvcs-repository-integration. (Zitiert auf Seite 37)

[Atlb] Atlassian. FishEye. http://www.atlassian.com/software/fisheye/overview. (Zitiert auf Seite 42)

[Atlc] Atlassian. Marketplace. https://marketplace.atlassian.com. (Zitiert auf Seite 31)

[BCS11] C. M. P. Ben Collins-Sussman, Brian W. Fitzpatrick. *Version Control with Subversion*. O'Reilly Media, 2011. URL http://svnbook.red-bean.com. (Zitiert auf Seite 11)

[Be] S. Borho, et al. *Welcome to TortoiseHg's documentation!* URL http://tortoisehg.bitbucket.org/manual/1.1/index.html. (Zitiert auf Seite 44)

[Bzra] Bazaar 2.5 Documentation. http://doc.bazaar.canonical.com/bzr.2.5/en. (Zitiert auf Seite 30)

[Bzrb] Bazaar Bisect Plugin. https://launchpad.net/bzr-bisect. (Zitiert auf Seite 30)

[Bzrc] Bazaar Documentation. http://wiki.bazaar.canonical.com/Documentation. (Zitiert auf Seite 30)

[Bzrd] Bazaar Externals Plugin. https://launchpad.net/bzr-externals. (Zitiert auf Seite 30)

[Bzre] Bazaar KeywordExpansion. http://wiki.bazaar.canonical.com/KeywordExpansion. (Zitiert auf Seite 30)

[Bzrf] Bazaar Subversion Plugin. https://launchpad.net/bzr-svn. (Zitiert auf Seite 30)

[Bzrg] bzr-jira. https://launchpad.net/bzr-jira. (Zitiert auf Seite 30)

[Bzrh] BzrEclipse. http://wiki.bazaar.canonical.com/BzrEclipse. (Zitiert auf Seite 30)

[Bzri] TortoiseBzr. http://wiki.bazaar.canonical.com/TortoiseBzr. (Zitiert auf Seite 30)

[Cha09] S. Chacon. *Pro Git*. Apress, 1 Auflage, 2009. URL http://git-scm.com/book. (Zitiert auf den Seiten 38 und 39)

[chm] FreeBSD Man Page CHMOD(1). http://www.freebsd.org/cgi/man.cgi?query=chmod&sektion=1. (Zitiert auf Seite 39)

[Cod] Codice Software. Licensing and pricing. https://www.plasticscm.com/buy.aspx. (Zitiert auf Seite 48)

[Cod12a] Codice Software. *Plastic SCM Distributed*, 2012. URL http://www.plasticscm.com/releases/4.1/manuals/en/distributedsystem.pdf. (Zitiert auf Seite 47)

[Cod12b] Codice Software. *Plastic SCM IDE integrations*, 2012. URL http://www.plasticscm.com/releases/4.1/manuals/en/idesguide.pdf. (Zitiert auf Seite 48)

[Cod12c] Codice Software. *Plastic SCM Introduction*, 2012. URL http://www.plasticscm.com/releases/4.1/manuals/en/userguide.pdf. (Zitiert auf den Seiten 47, 48 und 49)

[Cod12d] Codice Software. *Plastic SCM Xlinks guide*, 2012. (Zitiert auf Seite 49)

[Cus] CustomWare. Mercurial Plugin for JIRA. https://marketplace.atlassian.com/plugins/com.consultingtoolsmiths.jira.plugin.ext.mercurial.mercurial-jira-plugin. (Zitiert auf Seite 42)

[Dara] Darcs. http://darcs.net. (Zitiert auf Seite 31)

[Darb] Darcs Wiki. DifferencesFromSubversion. http://darcs.net/DifferencesFromSubversion. (Zitiert auf Seite 33)

[Darc] Darcs Wiki. Frequently Asked Questions (Performance). http://darcs.net/FAQ/Performance. (Zitiert auf Seite 33)

[Dard] Darcs Wiki. MigratingFromSubversion. http://darcs.net/MigratingFromSubversion. (Zitiert auf Seite 34)

[EGi] EGit. http://www.eclipse.org/egit. (Zitiert auf Seite 37)

[FG] L. Frenzel, R. Grzanka. EclipseDarcs. http://eclipsedarcs.sourceforge.net. (Zitiert auf Seite 31)

[Fosa] Convert simple SVN-Repository to Fossil. http://bens.me.uk/2011/convert-simple-svn-to-fossil. (Zitiert auf Seite 36)

[Fosb] Fossil GUI - Fuel. http://code.google.com/p/fuel-scm. (Zitiert auf Seite 35)

[Fosc] Fossil Performance. (Zitiert auf Seite 35)

[Gen11] E. Gentz. Die neue Freiheit bei der Versionskontrolle, 2011. URL http://www.heise.de/developer/artikel/Die-neue-Freiheit-bei-der-Versionskontrolle-1224755.html. (Zitiert auf Seite 42)

[Gita] About Git. http://git-scm.com/about. (Zitiert auf den Seiten 37 und 38)

[Gitb] Git vs. Mercurial: Please Relax. http://importantshock.wordpress.com/2008/08/07/git-vs-mercurial. (Zitiert auf Seite 52)

[Gitc] Git Wiki. GitProjects. https://git.wiki.kernel.org/index.php/GitProjects. (Zitiert auf Seite 13)

[Git10] Google Code – Analysis of Git and Mercurial. http://code.google.com/p/support/wiki/DVCSAnalysis, 2010. (Zitiert auf Seite 52)

[Hö11] M. Höher. *darcs/camp*. Technische Universität Dresden, 2011. (Zitiert auf Seite 32)

[Ham10] J. Hammond. Forrester Databyte: SCM Tool Adoption. http://blogs.forrester.com/application_development/2010/01/forrester-databyte-developer-scm-tool-adoption-and-use.html, 2010. (Zitiert auf den Seiten 12 und 13)

[HT] K. Hoijarvi, E. Thomson. TortoiseDarcs. http://tortoisedarcs.sourceforge.net. (Zitiert auf Seite 31)

[IBM] IBM – Rational Team Concert. http://www-01.ibm.com/software/rational/products/rtc. (Zitiert auf Seite 40)

[ITWa] ITWissen. Git. http://www.itwissen.info/definition/lexikon/Git-Git.html. (Zitiert auf Seite 13)

[ITWb] ITWissen. Mercurial. http://www.itwissen.info/definition/lexikon/Mercurial-Mercurial.html. (Zitiert auf Seite 13)

[Jaz] . https://jazz.net/library. (Zitiert auf Seite 40)

[JIRa] Git Version Control Viewer. https://marketplace.atlassian.com/plugins/com.xiplink.jira.git.jira_git_plugin. (Zitiert auf Seite 37)

[JIRb] JIRA Git Plugin. https://studio.plugins.atlassian.com/wiki/display/JGIT/JIRA+Git+Plugin. (Zitiert auf Seite 37)

[Mac] T. Macieira. svn-all-fast-export. http://repo.or.cz/w/svn-all-fast-export.git. (Zitiert auf Seite 49)

[MB03] K. F. Moshe Bar. *Open Source Development with CVS*. Paraglyph Press, Inc., 2003. (Zitiert auf Seite 11)

[Men02] T. Mens. A State-of-the-Art Survey on Software Merging. *Software Engineering, IEEE Transactions on*, 28(5):449 – 462, 2002. (Zitiert auf Seite 9)

[Men10] G. Mendal, Herausgeber. *Developing and Maintaining a Strategic Perforce Plan at Google*. 2010. URL http://www.perforce.com/perforce/conferences/eu/2010/Presentations/Geoff_Mendal-Strategic_Plan.paper.pdf. (Zitiert auf Seite 12)

[Mera] Mercurial. http://mercurial.selenic.com. (Zitiert auf Seite 42)

[Merb] MercurialEclipse. http://javaforge.com/project/HGE. (Zitiert auf Seite 42)

[Merc] Mercurial Wiki. Features of Last Resort. http://mercurial.selenic.com/wiki/FeaturesOfLastResort. (Zitiert auf Seite 43)

[Merd] Mercurial Wiki. Keyword Extension. http://mercurial.selenic.com/wiki/KeywordExtension. (Zitiert auf Seite 44)

[Mere] Mercurial Wiki. Mercurial Frequently Asked Questions. http://mercurial.selenic.com/wiki/FAQ. (Zitiert auf Seite 43)

[Merf] Mercurial Wiki. Shelve Extension. http://mercurial.selenic.com/wiki/ShelveExtension. (Zitiert auf Seite 44)

[Merg] Mercurial Wiki. Some Projects that Use Mercurial. http://mercurial.selenic.com/wiki/ProjectsUsingMercurial. (Zitiert auf Seite 13)

[Merh] Mercurial Wiki. Subrepository. http://mercurial.selenic.com/wiki/Subrepository. (Zitiert auf Seite 43)

[O'So9] B. O'Sullivan. *Mercurial: The Definitive Guide*. O'Reilly Media, 2009. (Zitiert auf den Seiten 9, 42 und 44)

[Pera] Multiple Repository Sync in Perforce. https://confluence.atlassian.com/display/BAMKB/Multiple+Repository+Sync+in+Perforce. (Zitiert auf Seite 46)

[Perb] Perforce Eclipse Plugin. http://www.perforce.com/product/components/eclipse_plugin. (Zitiert auf Seite 45)

[Perc] Perforce Preise - Internetseite. http://www.perforce.com/purchase/licensing-pricing-options. (Zitiert auf Seite 45)

[Perd] SVN2P4 - Von SVN zu Perforce. http://public.perforce.com/wiki/SVN2P4. (Zitiert auf Seite 45)

[Pos11] J. Posner. *Migrating from Perforce to Plastic SCM*. Codice Software, 2011. URL http://www.plasticscm.com/releases/3.0.1/migration-guides/perforce_migration.pdf. Abschnitt 2.7. (Zitiert auf Seite 49)

[Rou] D. Roundy. *Darcs User Manual*. URL http://www.darcs.net/manual/darcs.pdf. (Zitiert auf den Seiten 31, 33 und 34)

[SKY07] SKYTEC AG. Case Study: Siemens AG. www.skytecag.com/unternehmen/references/case-studies/single-referenz/archive/2007/article/siemens-ag-1, 2007. (Zitiert auf Seite 12)

[Sta] Stack Overflow. Is it possible to have a subversion repository as a git submodule? http://stackoverflow.com/questions/465042/is-it-possible-to-have-a-subversion-repository-as-a-git-submodule. (Zitiert auf Seite 38)

[Sub] Subversion Testimonials. http://svn.apache.org/repos/asf/subversion/branches/1.6.x/www/testimonials.html. (Zitiert auf Seite 12)

[Tai] Tailor. http://progetti.arstecnica.it/tailor. (Zitiert auf Seite 33)

[Thr] Three of Coins. Darcs vs Git: mathematician versus engineer. http://3ofcoins.net/2008/12/16/darcs-vs-git-mathematician-versus-engineer. Kommentar von „Maciej“ vom 24.12.2008. (Zitiert auf Seite 33)

[Tora] TortoiseGit. http://code.google.com/p/tortoisegit. (Zitiert auf Seite 37)

[Torb] TortoiseHg. http://tortoisehg.bitbucket.org. (Zitiert auf Seite 42)

[Wika] Wikipedia. BitKeeper. http://en.wikipedia.org/w/index.php?title=BitKeeper&oldid=510265555. (Zitiert auf Seite 12)

[Wikb] Wikipedia. Darcs. http://en.wikipedia.org/w/index.php?title=Special:Cite&page=Darcs&id=517379492. (Zitiert auf Seite 12)

[Wikc] Wikipedia. GNU arch. http://en.wikipedia.org/w/index.php?title=GNU_arch&oldid=511941266. (Zitiert auf Seite 12)

Alle URLs wurden zuletzt am 21. 10. 2012 geprüft.

Erklärung

Hiermit versichern wir, diese Arbeit
selbständig verfasst und nur die angegebenen
Quellen benutzt zu haben.

(Jakob Jarosch Tobias Kuhn Patrick Strobel)