C:\Users\haller\Desktop\Logo_HFU.tif

Bachelor-Thesis

in

Wirtschaftsinformatik

Verteilte Versionsverwaltung

Eine Tool-Integration in Java-Programmier-Vorlesungen

|  |  |
| --- | --- |
| Referent: | Prof. Dipl.-Inform. J. Anton Illik |
| Korreferent: | Prof. Dr.-Ing. Stefan Noll |
| Vorgelegt am: | 31.08.2017 |
| Vorgelegt von: | Ertugrul Özkara |
|  | 245269 |
|  | Steigstraße 54  78554 Aldingen |
|  | ertugrul.oezkara@hs-furtwangen.de |

# Abstract

[Englisch, 100 -120 Worte]

[Deutsch, 100 – 120 Worte]

# Inhaltsverzeichnis

[Abstract III](#_Toc482796945)

[Inhaltsverzeichnis V](#_Toc482796946)

[Abbildungsverzeichnis IX](#_Toc482796947)

[Tabellenverzeichnis XI](#_Toc482796948)

[Abkürzungsverzeichnis XIII](#_Toc482796949)

[1 Einleitung 15](#_Toc482796950)

[2 Grundlagen 16](#_Toc482796951)

[2.1 Definition 16](#_Toc482796952)

[2.2 Was ist Versionsverwaltung 17](#_Toc482796953)

[2.3 Vorteile einer Versionsverwaltung 18](#_Toc482796954)

[2.4 Geschichte der Versionsverwaltung 19](#_Toc482796955)

[2.4.1 SCCS 20](#_Toc482796956)

[2.4.2 CVS 20](#_Toc482796957)

[2.4.3 SVN 21](#_Toc482796958)

[2.5 Bekannte Versionsverwaltungssysteme 22](#_Toc482796959)

[2.5.1 CVS 23](#_Toc482796960)

[2.5.2 SVN 23](#_Toc482796961)

[2.6 Funktionsweise 23](#_Toc482796962)

[2.6.1 Probleme(nicht sicher) 23](#_Toc482796963)

[2.6.2 Lösungswege(nicht sicher) 23](#_Toc482796964)

[2.7 Arten der Versionsverwaltung 23](#_Toc482796965)

[2.7.1 Lokale Versionsverwaltung 23](#_Toc482796966)

[2.7.2 Zentrale Versionsverwaltung 23](#_Toc482796967)

[2.7.3 Lokal versus zentral 23](#_Toc482796968)

[2.7.4 Verteilte Versionsverwaltung 23](#_Toc482796969)

[2.8 Verschiedene Konzepte der Versionsverwaltung 23](#_Toc482796970)

[2.8.1 Lock Modify Write 23](#_Toc482796971)

[2.8.2 Copy Modify Merge 23](#_Toc482796972)

[2.8.3 Objektbasierte Versionisierung 23](#_Toc482796973)

[2.9 Strategien der Versionsverwaltung 23](#_Toc482796974)

[2.10 Zusammenfassung 23](#_Toc482796975)

[3 Git 25](#_Toc482796976)

[3.1 Entstehung 25](#_Toc482796977)

[3.2 Grundlagen 25](#_Toc482796978)

[3.3 Eigenschaften von Git 25](#_Toc482796979)

[3.4 Interner Aufbau von Git 25](#_Toc482796980)

[3.5 Funktionsweise von Git 25](#_Toc482796981)

[3.6 Vorteile von Git 25](#_Toc482796982)

[3.7 Nachteile von Git 25](#_Toc482796983)

[3.8 Wichtige Begriffe innerhalb von Versionsverwaltungssystemen 25](#_Toc482796984)

[3.9 Branches kurz erklärt(nicht sicher ob hier oder in 3.7) 25](#_Toc482796985)

[3.10 Branching und Merging(nicht sicher ob hier oder in 3.7) 25](#_Toc482796986)

[3.11 Protokollierung 25](#_Toc482796987)

[3.12 Wiederherstellung 25](#_Toc482796988)

[3.13 Archivierung 25](#_Toc482796989)

[3.14 Koordinierung 25](#_Toc482796990)

[3.15 Gleichzeitiges Entwickeln 25](#_Toc482796991)

[3.16 Verteilte Abläufe 25](#_Toc482796992)

[3.17 Hosting über GitHub 25](#_Toc482796993)

[3.18 Befehle 25](#_Toc482796994)

[3.19 Kommandozeile zur Bedienung von Git(nicht sicher) 25](#_Toc482796995)

[3.20 Zusammenfassung 25](#_Toc482796996)

[4 Versionsverwaltungssysteme 27](#_Toc482796997)

[4.1 Ziele der Versionsverwaltung 27](#_Toc482796998)

[4.2 Verschiedene Versionsverwaltungssysteme im Überblick 27](#_Toc482796999)

[4.2.1 Darcs 27](#_Toc482797000)

[4.2.2 Git 27](#_Toc482797001)

[4.2.3 GNU arch 27](#_Toc482797002)

[4.2.4 Apache Subversion 27](#_Toc482797003)

[4.2.5 Fossil 27](#_Toc482797004)

[4.2.6 Mercurial 27](#_Toc482797005)

[4.2.7 Bazaar 27](#_Toc482797006)

[4.2.8 Gegenüberstellung verschiedener Systeme 27](#_Toc482797007)

[4.3 Zusammenfassung 27](#_Toc482797008)

[5 Umsetzung/Integration des Tools in die Vorlesung 27](#_Toc482797009)

[5.1 Git installieren 27](#_Toc482797010)

[5.2 GitHub 27](#_Toc482797011)

[5.3 Repository aufbauen 27](#_Toc482797012)

[5.4 Veränderung an Repository melden 27](#_Toc482797013)

[5.5 Commit-Verlauf einsehen 27](#_Toc482797014)

[5.6 Rückgängig machen 27](#_Toc482797015)

[5.7 Wartung und Wiederherstellung 27](#_Toc482797016)

[5.8 Git als PlugIn in Eclipse(nicht sicher) 27](#_Toc482797017)

[5.9 Zusammenfassung 28](#_Toc482797018)

[6 Resumee 28](#_Toc482797019)

[Literaturverzeichnis 29](#_Toc482797020)

[Stichwortverzeichnis 31](#_Toc482797021)

[Eidesstattliche Erklärung 33](#_Toc482797022)

[A. [Anhang] 35](#_Toc482797023)

# Abbildungsverzeichnis

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

Das Abbildungsverzeichnis muss vor der Finalen Abgabe entfernt werden, wenn keine Abbildungen in die Arbeit eingefügt wurden.

# Tabellenverzeichnis

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

Das Tabellenverzeichnis muss vor der Finalen Abgabe entfernt werden, wenn keine Tabellen in die Arbeit eingefügt wurden.

# Abkürzungsverzeichnis

|  |  |
| --- | --- |
| VCS | Version Control System |
| SCCS | Source Code Control System |
| CVS | Concurrent Versions System |
| SVN | Subversion |
| RCS | Revision Control System |
|  |  |
|  |  |
|  |  |

# Einleitung

Versionsverwaltungen oder auch Versionskontrollsysteme (kurz: VCS) verwalten die Änderungen an beliebig vielen Dateien, die im Laufe der Zeit entstehen. Sie ermöglichen es diese in eine ältere Version zurückzuführen ohne dass es dabei eine Rolle spielt, ob es sich um Bilddateien, Quellcode oder jegliche andere Art von Dateien handelt.[[1]](#footnote-1)

Versionskontrolle ist vor allem im Bereich der Software-Entwicklung ein immer wichtiger werdendes Thema. Höchstwahrscheinlich erreicht jedes Projekt irgendwann den Punkt, an dem ältere oder gelöschte Stände noch ein-mal eingesehen werden müssen, es zu verstehen gilt, warum zu einem bestimmten Zeitpunkt eine bestimmte Funktion in ein Programm eingefügt worden ist oder das etwas kaputt ist und man in die Ausgangssituation zurück möchte.  
Diese bieten nicht nur einen Ausweg für die beschriebenen Probleme, sondern sie sind ebenso eine Erleichterung im Voranschreiten des Entwicklungsprozesses in Solo- und auch Großprojekten.[[2]](#footnote-2) Darüber hinaus besteht jederzeit die Gefahr, dass man vor allem in großen Projekten mit mehreren Mitgliedern, die zuvor verrichtete Arbeit eines anderen überschreibt und diese damit verloren geht.

Mit immer größer und anspruchsvoller werdenden Programmen steigt auch die Anzahl der benötigten Entwickler. Diese wiederum haben die Aufgabe Anfälligkeiten auf Fehler so gut wie möglich zu verhindern. Dies jedoch stellt eine immer größer werdende Herausforderung dar. Abhilfe schaffen hier Versionsverwaltungssysteme, welche sich deshalb auch am schnellsten in dieser Branche verbreitet haben.[[3]](#footnote-3)

Software-Entwicklung findet nicht nur in Unternehmen statt, sondern wird auch in den verschiedensten Einrichtungen gelehrt. Dabei reicht das Spektrum von weiterführenden berufsbezogenen Schulen, bis hin zu Universitäten. Daher ist es nicht unwahrscheinlich, dass die oben aufgeführten Probleme sich auch in Unterrichtsräumen und Hörsälen zu erkennen geben und Lehrende sowie auch Lernende vor Herausforderungen stellen.

Aufgrund dieser Erkenntnis ist es das Ziel dieser Arbeit Versionsverwaltung Studenten, Schülern sowie Lehrkörpern näher zu bringen, diese dafür zu begeistern und die benötigte Versionskontrolle in Programmier-Vorlesungen zu verankern.

Gegenstand von Kapitel 2 ist es dem Leser fundamentales Wissen über Versionsverwaltung zu vermitteln und zu verstehen aus welcher Notwendigkeit heraus diese entstanden ist. Des Weiteren werden verschiedene Konzepte und auch Funktionsweisen angesprochen. Nachfolgend werden verschiedene und gängige Versionsverwaltungssysteme unter die Lupe genommen und entsprechende wichtige Begriffe erklärt, um sich ein Bild darüber machen zu können, welche Systeme am geeignetsten für die Arbeit in Programmier-Vorlesung sind. Anschließend wird auf die Versionsverwaltungssoftware Git eingegangen Den Abschluss bildet die Integration von Git in die Vorlesung und die zugehörige Handhabung.

# Grundlagen

Dieses Kapitel geht auf wichtiges und fundamentales Wissen im Bereich von Versionsverwaltung und auch verteilte Versionsverwaltung ein. Es wird erklärt was Versionsverwaltung ist, warum es Notwendig ist, welche Konzepte dahinter stecken und warum heutzutage verteilte Versionsverwaltung bevorzugt wird.

## Definition

„Eine Versionsverwaltung hat die Aufgabe, den Werdegang eines Projektes zu protokollieren. Sie übernimmt die Versionierung und die Archivierung der Daten und ermöglicht den gemeinsamen Zugriff darauf. Dabei kann jede Änderung und jeder Stand verfolgt, rückgängig gemacht oder wiederhergestellt werden.“ (GlossarWiki 2016, S. 1)

„Eine Versionsverwaltung ist ein System mit dem Änderungen an Dateien auf Befehl der Benutzer aufgezeichnet werden und bei Bedarf jede der vorherigen Dateiversionen wiederhergestellt werden kann. Der zentrale Ort an dem die Änderungen der Dateien aufgezeichnet werden nennt sich Repository. Zusätzlich zur Benutzung eines Repositories von lediglich einem Benutzer, bieten die meisten Versionsverwaltungen die Möglichkeit, dass mehrere Benutzer gleichzeitig an Dateien arbeiten können, ohne dass dabei eine Änderung eines Nutzers verloren geht. Die Kombination dieser Eigenschaften bietet die Grundlage der Softwareentwicklung im Team. Dies bedeutet jedoch keineswegs, dass Versionsverwaltungen lediglich für die Softwareentwicklung interessant sind. Ebenso kann man die Systeme dazu nutzen, um Änderungen an Bildern, Audiodateien oder sogar Binärdateien nachzuverfolgen und alte Versionen wiederherzustellen. Gerade große Projekte in der Bild- und Tonbearbeitung brauchen viel Platz auf der Festplatte und im Arbeitsspeicher, da sie selbst die Option bieten alle Änderungen nichtdestruktiv 1 durchzuführen und somit eine eigene Versionsverwaltung enthalten. Die Anforderungen an die Rechner könnten jedoch immens gesenkt werden, wenn man lediglich die aktuell nötigen Daten bereithält, anstatt des gesamten Werdegangs einer Datei. Dies kann man erreichen, indem man die programmeigene Versionsverwaltung deaktiviert und stattdessen auf ein System wie Git oder Subversion setzt.“ (Mauel 2013, S. 5)

## Was ist Versionsverwaltung

„Arbeit, die Sie nicht mit einem Versionskontrollsystem verwalten, ist Arbeit, die Sie möglicherweise noch einmal machen müssen - sei es weil Sie versehentlich eine Datei löschen oder Teile als obsolet betrachten, die Sie später doch wieder benötigen.“ (Haenel/Plenz 2011, S. 14)

Das obige Zitat unterstreicht das Argument, wie wichtig die Arbeit mit Versionsverwaltung in manchen Bereichen sein kann. Versionsverwaltung kann unter anderem dafür sorgen, dass bei einer sorgfältigen Planung viel Zeit gespart wird. Beispielsweise können verlorengeglaubte Dateien wiederhergestellt werden. Dadurch muss bei einem unterlaufenen Fehler, die bereits getane Arbeit nicht wiederholt werden.

Versionsverwaltung bedeutet im Prinzip nichts anderes, als das von Dateien Versionen erstellt werden können, um diese dann je nach Gebrauch zu verwalten. Jede noch so kleine Änderung an einer Datei erstellt eine neue Version dieser. Der Begriff »Version« kann vieles Bedeuten, doch wenn man im Bereich der Versionskontrolle davon spricht, handelt es sich um die Version einer einzelnen Datei oder der Ansammlung mehrerer Dateien.[[4]](#footnote-4)

Dadurch soll das gemeinsame Arbeiten an einer Vielzahl von Dokumenten ermöglicht werden. Dabei gilt die höchste Priorität der Vermeidung von Datenverlust bei gleichzeitiger Entwicklung durch mehrere Personen.[[5]](#footnote-5)

Egal ob eine reine Textdatei bearbeitet wird, ein Bild oder ein Video, es ist üblich, dass der aktuelle Stand immer wieder gesichert wird, damit die bereits verrichtete Arbeit nicht verloren geht oder damit man bei Bedarf auf eine ältere Version zurückgreifen kann. Die Vorgehensweise bei der Benennung dieser "Sicherheitskopien" ist von Mensch zu Mensch unterschiedlich. Manche geben die aktuelle Versionsnummer der Datei bei, andere wiederum erzeugen verschiedene Ordner mit dem entsprechenden Datum.  
Diese Vorgehensweisen sind zwar nicht ganz falsch, doch weder sind sie praktisch noch sind sie resistent gegen Fehleranfälligkeit. Wenn zu viele Dateien versioniert werden müssen und diese keine eindeutigen Namen zugewiesen bekommen, dann steht man vor einem Problem.[[6]](#footnote-6)

Zur Lösung dieser Probleme wurden Versionsverwaltungssysteme entwickelt, die es erlauben, genau nachzusehen wer etwas wann und weshalb geändert hat, um bei Bedarf die entsprechenden Änderungen rückgängig zu machen. Das System notiert sich nämlich, wer eine Datei geändert hat mit der dazugehörigen Uhrzeit und verweist auf den Abschnitt in der Datei, die geändert worden ist. So entsteht nach und nach mit jeder Änderung eine Änderungshistorie, die dem Arbeiten an mehreren Dateien Erleichterung bringen. Diese Änderungshistorie erlaubt es genau nachzuverfolgen ab welchem Punkt ein Projekt in die falsche Richtung geht oder Fehler aufweist, sodass diese Fehler schnell ausfindig gemacht und auch behoben werden können.[[7]](#footnote-7)

## Vorteile einer Versionsverwaltung

Vor allem bei großen Projekten ist die Nutzung eines Versionsverwaltungssystems fast unumgänglich. Diese bringen nämlich erhebliche Erleichterungen mit an Bord. Die meisten Vorteile die eine Versionskontrolle mit sich bringt wurden zwar weiter oben schon fast alle genannt, dennoch werden hier noch einmal einige wichtige Punkte zusammengefasst.

Wie schon erwähnt, können Änderungen lokalisiert und rückgängig gemacht werden. Der große Vorteil hierbei ist, dass diese Änderungen innerhalb des gesamten Projekts vorgenommen werden können. Durch Fehler wird nicht das endgültige Schicksal eines Projektes beschlossen. Das Wissen, dass man jederzeit zu einem bestimmten Zeitpunkt zurückspringen kann, verleiht Mitarbeitern innerhalb eines Projektes eine gewisse Ruhe. Das Team arbeitet dadurch ungezwungener und schneller, denn die Möglichkeit Fehler schnell wieder beheben zu können, verleiht das Gefühl von Sicherheit. Versionsverwaltungssysteme verhalten sich in diesem Rahmen so ähnlich wie eine Zeitmaschine, die es uns erlaubt an einem bestimmten Datum zu einem bestimmten Punkt des Projektes zurückzuspringen.[[8]](#footnote-8)

Ein weiterer Vorteil, den Versionskontrolle mit sich bringt, ist das gleichzeitige Entwickeln. Mehrere Entwickler können gleichzeitig und kontrolliert an demselben Quellcode arbeiten, ohne dass die Arbeit des jeweils anderen dadurch überschrieben wird.[[9]](#footnote-9)

Des Weiteren hilft die Protokollierung dabei gemachte Änderungen ausfindig zu machen und zu verstehen wer im Team welche Änderung aus welchem Grund und an welcher Stelle des zu bearbeitenden Programms gemacht hat.[[10]](#footnote-10) Dies ist unter anderem auch dann nützlich, falls man ein bestimmtes Fragment des selbst erzeugten Quellcodes nicht mehr nachvollziehen kann.

Nicht nur gleichzeitiges Entwickeln ist möglich sondern auch das gleichzeitige Veröffentlichen von Software. Das Team ist dadurch nicht gezwungen kurz vor einem Release seine Arbeiten zu unterbrechen und kann ungehindert auf dem Hauptzweig des Projektes die Arbeit fortsetzen.[[11]](#footnote-11)

## Geschichte der Versionsverwaltung

Zu versuchen die Geschichte der Versionsverwaltung in vollem Maße darzustellen, würde den Rahmen dieser Arbeit übersteigen. Anhand verschiedener wegweisender Systeme ist es jedoch möglich zu der Entstehung der Entwicklung von Git hinzuführen und diese nachzuvollziehen.[[12]](#footnote-12)

### SCCS

Das Source Code Control System wurde Anfang der 1970er Jahre durch M. J. Rochkind unter Unix entwickelt. Es handelt sich hierbei um das wahrscheinlich erste VCS, welches auf dem Unix-System verfügbar war.[[13]](#footnote-13)

Es wurde ein zentraler Speicher zur Verfügung gestellt, welcher als Repository bezeichnet wurde. Dieses Konzept hat sich durchgesetzt und existiert heute immer noch. Das SCCS bot natürlich zu der damaligen Zeit nur einige Grundfunktionalitäten. Entwickler konnten Dateien ausführen und diese Testen ohne sie beim Auschecken sperren zu müssen. Beim Bearbeiten einer Datei wiederum war dies unumgänglich. Nach getaner Arbeit wurde die Datei wieder eingecheckt und die vorherige Sperrung wieder aufgehoben.[[14]](#footnote-14)

### CVS

Walter Tichy entwickelte Anfang der 80er Jahre das Revision Control System (RCS). Durch das RCS konnte man auf eine effiziente Art und Weise verschiedene Überprüfungen, die man an Dateien vollzogen hatte, speichern.[[15]](#footnote-15)

Auf der Grundlage des RCS-Modells wurde einige Jahre später das Concurrent Versions System entwickelt. Dieses bot einige Vorteile gegenüber seinem Vorgänger auf Das CVS war sehr beliebt und wurde eine lange Zeit als die beliebteste Versionsverwaltung in der Open Source-Gemeinde angesehen.[[16]](#footnote-16) Hauptsächlich fand die Bedienung hier über eine Kommandozeile statt.[[17]](#footnote-17), was alle Versionsverwaltungen im Grunde gemeinsam haben.

Durch CVS wurde bei dem Umgang mit Locks (Sperrung) eine neue Ideologie geschaffen. Dies ist wahrscheinlich die wichtigste Erneuerung, welche es mit sich gebracht hat. Erstmals war es möglich private Arbeitskopien von Dateien zu erstellen, an denen man arbeiten wollte. Dies hatte den Vorteil, dass die jeweilige Datei nicht gesperrt werden musste. Somit konnten andere Entwickler ebenfalls zu derselben Zeit an derselben Datei arbeiten, ohne dabei die Arbeit des anderen zu verhindern.

Diese verschiedenen Änderungen durch verschiedene Entwickler konnten später zusammengeführt werden, ohne dass die Arbeit des Anderen dadurch verloren ging.

Der einzige Konflikt bestand in dem Fall das Zwei verschiedene Entwickler dieselbe Zeile bearbeiten wollten. In diesem Fall wurden auf beide Veränderungen hingewiesen und es musste sich für eine Veränderung entschieden werden.[[18]](#footnote-18)

### SVN

Subversion ist ein zentrales Versionsverwaltungssystem und basiert auf einer Client- Server-Architektur.[[19]](#footnote-19)

Typischerweise entstand Subversion aus den Mängeln und Fehlern, die bei CVS ausgemacht wurden. Der Unterschied zum Vorgänger bestand darin, dass Änderungen direkt gespeichert wurden und der Umgang mit den verschiedenen Entwicklungszweigen sichtbar besser zu handhaben war.  
Aufgrund dessen wurde Subversion nach seiner Vorstellung schnell zu einer der beliebtesten Versionsverwaltungen in der Softwaregemeinde.[[20]](#footnote-20)

Die wohl bekannteste und gängigste Versionsverwaltung unserer Zeit ist die Software Git, welche nach Subversion entstanden ist und das eigentliche Thema dieser Arbeit ist. Auf diese Software wird in Kapitel 3 näher eingegangen und ist dementsprechend an dieser Stelle nicht aufgeführt.

## Bekannte Versionsverwaltungssysteme

Wie schon bei der Geschichte der Versionsverwaltungssysteme, würde eine ausführliche Aufführung aller vorhandenen Systeme den Rahmen dieser Arbeit sprengen, da im Laufe der Zeit viele Open-Source-Produkte, sowie auch viele proprietäre Software durch bekannte Unternehmen, entwickelt worden sind.

Folglich werden zwei der bekanntesten Programme aus dem Bereich der Versionskontrolle aufgeführt, welche vor der Existenz von Git die wahrscheinlich gängigsten in diesem Bereich waren und auch heute noch teilweise verwendet werden.

### CVS

Obwohl es heutzutage nicht mehr weiterentwickelt wird, gilt das Concurrent Versions System als *das* klassische Versionskontrollsystem. Die Verwaltung läuft entweder lokal über den eigenen Rechner oder über einen Webserver. Das CVS wurde ursprünglich als ein reines Kommandozeilen-Programm entwickelt. Später kam aber auch eine Benutzeroberfläche hinzu.[[21]](#footnote-21)

Ein wichtiger Aspekt bei CVS ist die Verankerung in der Open-Source-Gemeinde. Die Software kann ohne den Anspruch auf Lizenzen frei im Internet erhalten werden. Wie schon weiter oben erwähnt, ist es historisch als der Nachfolger des Revision Control Systems anzusehen.[[22]](#footnote-22)

Das Repository verhält sich hier wie ein Dateiserver mit einem Verzeichnisbaum als Inhalt. Ein Verzeichnis für die allgemeine Bedienung und eines für jedes angelegte Projekt. Jede Datei wird in das entsprechend zugehörige Verzeichnis abgelegt. Zusätzlich enthalten die Verzeichnisse Informationen, welche die Unterschiede zu den Vorgängerversionen aufweisen sowie eine Zeitangabe wann die Datei geändert worden ist. Darüber hinaus ist es möglich für jede Version Kommentare zu hinterlassen.[[23]](#footnote-23)

Von der Nutzung dieser Software ist heutzutage allerdings abzuraten, da es inzwischen sehr veraltet ist und neuere Systeme existieren. Diese weisen in vielen Bereichen Verbesserungen auf, sowohl in Anwenderfreundlichkeit als auch in grundlegenden Funktionalitäten.[[24]](#footnote-24)

### SVN

Apache Subversion gilt als der Quasi-Nachfolger von CVS.[[25]](#footnote-25) Da Subversion ein zentrales Versionsverwaltungssystem ist, wird das Repository auf einem zentralen Server gespeichert. Der wesentliche Unterschied zu verteilten Versionskontrollsystemen wie z.B. zu Git liegt darin, dass die gesamte Historie auf dem Server gespeichert und auch nur dort benötigt wird. Es ist nicht notwendig das Repository zu klonen oder zu kopieren um damit arbeiten zu können. Das Arbeitsverzeichnis kann mit einem Checkout direkt vom Server erstellt werden.[[26]](#footnote-26)

Der Vorteil hierbei ist, dass bei großen Repositories ein Checkout schneller ausgeführt werden kann, da immer nur ein einziger Zustand des Projektes von dem Client übernommen werden muss. Auch wird dadurch weniger Speicherplatz auf dem Client beansprucht und sobald das Projekt auf dem Server abgelegt wird, steht es allen Clienten mit Serververbindung zur Verfügung.[[27]](#footnote-27)

Nachteile dieser zentralen Architektur können jedoch sein das ein Commit fehlschlagen kann, wenn das Netzwerk nicht verfügbar oder der Server offline ist. Änderungen, welche evtl erst getestet werden sollten, können nicht gemacht werden, bis das Repository wieder erreichbar ist.[[28]](#footnote-28)

Ein wichtiger Unterschied zwischen SVN und CVS ist der, dass bei SVN auch ganze Verzeichnisse versioniert werden können.[[29]](#footnote-29) Durch Subversion werden Änderungen an Dateien und Verzeichnissen durch Revisionen verwaltet. Dabei werden immer nur die Unterschiede zwischen den Dateien übertragen.[[30]](#footnote-30) Auch wenn nur einzelne Dateien geändert wurden, beziehen sich die Revisionen immer auf das gesamte Repository. Die Nummerierung der einzelnen Revisionen erfolgt aufsteigend und beginnt bei 0. Jedes Mal, wenn eine Änderung an das Repository übermittelt wird, zählt die Anzahl der Revision des gesamten Repository um eins hoch. Dies vereinfacht die Wiederherstellung von älteren Ständen, da nun zu jedem Zeitpunkt eine genaue Revision zugeordnet werden kann.[[31]](#footnote-31) SVN erlaubt das Verschieben und Umbenennen von Dateien.[[32]](#footnote-32) Auch können gelöschte Dateien mühelos wiederhergestellt werden, da die Versionsgeschichte erhalten bleibt.  
Darüber hinaus sind Logdateien verschiedener Benutzer sowohl für einzelne Dateien oder Verzeichnisse, als auch für Revisionen abrufbar.  
Weitere Merkmale von Subversion sind die Fähigkeit gleichzeitig entwickeln zu können und das Erkennen und Behandeln von Konflikten.[[33]](#footnote-33)

Wie schon bei CVS ist Subversion beim Open-Source-Segment angesiedelt und wird hauptsächlich über eine Kommandozeile bedient. Allerdings wurden auch hier zahlreiche verschiedene grafische Oberflächen für die Bedienung von SVN entwickelt. Die Bedienung zum Vorgänger ist sehr ähnlich. Die erste lauffähige Version von SVN wurde Anfang 2004 veröffentlicht.[[34]](#footnote-34)

Bei Subversion unterscheidet man zwischen drei verschiedenen Entwicklungszweigen. Der erste ist der Hauptentwicklungszweig, welcher auch Trunk genannt wird. Aus diesem Trunk resultieren die Nebenzweige und sind auch bekannt als Branches. Als letztes gibt es die Tags. Diese werden auch als Momentaufnahmen bezeichnet. Sie werden nicht mehr verändert und erweisen sich dann als nützlich, wenn man auf spezielle Versionen zugreifen möchte, ohne die Revisionsnummer zu kennen.[[35]](#footnote-35)

Eine weitere Besonderheit bei Subversion ist das es abgesehen vom Trunk, zwischen diesen Zweigen technisch gesehen keinen Unterschied gibt. Jeder einzelne dieser Zweige beginnt seine Versionsgeschichte beim Hauptentwicklungszweig oder bei einem anderen bereits existierenden Nebenzweig. Die Einordnung welcher Zweig was darstellt erfolgt letztendlich über die Benennung der Verzeichnisse im Repository und wird dementsprechend verwendet.[[36]](#footnote-36)

Bei der Arbeit mit Subversion wird zunächst einmal eine Arbeitskopie von der Datei mit der man arbeiten möchte erstellt. Diese Arbeitskopie wiederum wird aus dem Repository ausgecheckt. Mit dieser kann man wie gewohnt arbeiten ohne dafür Subversion verwenden zu müssen. Lediglich das Hinzufügen oder das Entfernen von Dateien erfolgen über Subversion, da dieses auch ganze Verzeichnisse versioniert. Für den Fall das man Änderungen verwerfen möchte, wird im Hintergrund eine ursprüngliche Version der aktuellen Arbeitskopie gespeichert. Abgeschlossene Änderungen müssen mit dem Repository zusammen aktualisiert werden, um Veränderungen im Repository mitzubekommen, welche in der Zwischenzeit evtl. getätigt wurden. Die gemachten Veränderungen werden durch Subversion automatisch in die veränderte Arbeitskopie übernommen. Eine Ausnahme besteht hier allerdings bei Konflikten. Wenn sich Änderungen überschneiden, müssen diese manuell gelöst und SVN mitgeteilt werden. Als letzter Schritt ist ein Commit notwendig. Diesem wird eine kurze Erklärung beigefügt, was sich nun alles in der Revision geändert hat. Diese ist später im Log-Protokoll wiederzufinden. Wie schon weiter oben erklärt, aktualisiert Subversion das Repository und erhöht die Revision um eins.[[37]](#footnote-37) Das Repository ist durch das lokale Dateisystem über ein Interface zu erreichen. Die Protokollierung der Änderungen erfolgt entweder über eine Datenbank oder direkt über das Dateisystem.[[38]](#footnote-38)

## Funktionsweise

Ohne zu verstehen wie Versionskontrolle funktioniert und aus welcher Notwendigkeit heraus die Versionskontrolle entstanden ist, ist es schwer nachzuvollziehen was genau bei einer Versionsverwaltungssoftware passiert und was bestimmte Begriffe innerhalb dieser Systeme Bedeuten und auch bewirken.

Zunächst werden auf Probleme eingegangen, welche bei der Bearbeitung einer einzelnen Datei oder bei der Arbeit in einem Großprojekt auftauchen würden ohne die Verwendung einer Versionskontrolle.

Daraufhin werden für die entsprechenden Problemstellungen Lösungen aufgezeigt, welche die Funktionsweise der heutigen Versionskontrolle darstellen.

### Probleme(nicht sicher)

### Lösungswege(nicht sicher)

## Arten der Versionsverwaltung

### Lokale Versionsverwaltung

### Zentrale Versionsverwaltung

### Lokal versus zentral

### Verteilte Versionsverwaltung

## Verschiedene Konzepte der Versionsverwaltung

### Lock Modify Write

### Copy Modify Merge

### Objektbasierte Versionisierung

## Strategien der Versionsverwaltung

## Zusammenfassung

# Git

## Entstehung

## Grundlagen

## Eigenschaften von Git

## Interner Aufbau von Git

## Funktionsweise von Git

## Vorteile von Git

## Nachteile von Git

## Wichtige Begriffe innerhalb von Versionsverwaltungssystemen

## Branches kurz erklärt(nicht sicher ob hier oder in 3.7)

## Branching und Merging(nicht sicher ob hier oder in 3.7)

## Protokollierung

## Wiederherstellung

## Archivierung

## Koordinierung

## Gleichzeitiges Entwickeln

## Verteilte Abläufe

## Hosting über GitHub

## Befehle

## Kommandozeile zur Bedienung von Git(nicht sicher)

## Zusammenfassung

# Versionsverwaltungssysteme

## Ziele der Versionsverwaltung

## Verschiedene Versionsverwaltungssysteme im Überblick

### Darcs

### Git

### GNU arch

### Apache Subversion

### Fossil

### Mercurial

### Bazaar

### Gegenüberstellung verschiedener Systeme

## Zusammenfassung

# Umsetzung/Integration des Tools in die Vorlesung

## Git installieren

## GitHub

## Repository aufbauen

## Veränderung an Repository melden

## Commit-Verlauf einsehen

## Rückgängig machen

## Wartung und Wiederherstellung

## Git als PlugIn in Eclipse(nicht sicher)

## Zusammenfassung

# Resumee

# Literaturverzeichnis

Fügen Sie hier ihre verwendete Literatur ein. Beachten Sie dabei die Vorgaben zu den Zitierstilen

# Stichwortverzeichnis

#### **Keine Indexeinträge gefunden.**

Das Stichwortverzeichnis ist optional. Wenn Sie kein Stichwortverzeichnis in Ihrer Arbeit verwenden wollen können Sie dieses Kapitel entfernen!

# Eidesstattliche Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbständig verfasst und hierzu keine anderen als die angegebenen Hilfsmittel verwendet habe. Alle Stellen der Arbeit die wörtlich oder sinngemäß aus fremden Quellen entnommen wurden, sind als solche kenntlich gemacht.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt oder an anderer Stelle veröffentlicht.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben kann.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[Ort, Datum Name]

Auch hier müssen die Platzhalter mit den korrekten Daten ersetzt werden.

# [Anhang]

Der Anhang ist optional. Wenn Sie keinen Anhang in Ihrer Arbeit verwenden wollen können Sie dieses Kapitel entfernen!

1. Vgl. Denker, Merlin/Stefan Srecec, »Versionsverwaltung mit Git. Fortgeschrittene Programmierkonzepte in Java, Haskell und Prolog« (2015), S. 1 [↑](#footnote-ref-1)
2. Vgl. Vijayakumaran, Sujeevan, *Versionsverwaltung mit Git*, 1. Auflage, Frechen 2016, S. 14 [↑](#footnote-ref-2)
3. Vgl. Bechara, John, »Revisionssichere Archivierung. Verteiltes Dokumentenmanagement« (2009), S. 5 [↑](#footnote-ref-3)
4. Vgl. Vijayakumaran, Sujeevan, *Versionsverwaltung mit Git*, 1. Auflage, Frechen 2016, S. 17 [↑](#footnote-ref-4)
5. Vgl. taentzer, »Versionsverwaltung von Softwareartefakten« (2014), S. 43–74, hier S. 43 [↑](#footnote-ref-5)
6. Vgl. Vijayakumaran, Sujeevan, *Versionsverwaltung mit Git*, 1. Auflage, Frechen 2016, S. 17 [↑](#footnote-ref-6)
7. Vgl. dass., *Versionsverwaltung mit Git*, 1. Auflage, Frechen 2016, S. 17-18 [↑](#footnote-ref-7)
8. Vgl. Thomas, David/Andrew Hunt/Falk Lehmann/Uwe Petschke, »Der pragmatische Programmierer«, *Der pragmatische Programmierer, Hunt, Andrew, [2. Aufl.]. - München [u.a.] Hanser* (2003), S. 1–7, hier S. 2 [↑](#footnote-ref-8)
9. Vgl. dies., »Der pragmatische Programmierer«, *Der pragmatische Programmierer, Hunt, Andrew, [2. Aufl.]. - München [u.a.] Hanser* (2003), S. 1–7, hier S. 1 [↑](#footnote-ref-9)
10. Vgl. dies., »Der pragmatische Programmierer«, *Der pragmatische Programmierer, Hunt, Andrew, [2. Aufl.]. - München [u.a.] Hanser* (2003), S. 1–7, hier S. 1-2 [↑](#footnote-ref-10)
11. Vgl. dies., »Der pragmatische Programmierer«, *Der pragmatische Programmierer, Hunt, Andrew, [2. Aufl.]. - München [u.a.] Hanser* (2003), S. 1–7, hier S. 2 [↑](#footnote-ref-11)
12. Vgl. Lämmer, Frank, Traditionelle Webentwicklung vs Versionsverwaltung, 06.04.2017, *http://t3n.de/news/traditionelle-webentwicklung-versionsverwaltung-303580/*, S. 5 [↑](#footnote-ref-12)
13. Vgl. Loeliger, Jon/Matthew McCullough, *Version control with Git. [powerful tools and techniques for collaborative software development ; covers GitHub]*, 2. ed., Sebastobol Calif. u.a. 2012, S. 5 [↑](#footnote-ref-13)
14. Vgl. dies., *Version control with Git. [powerful tools and techniques for collaborative software development ; covers GitHub]*, 2. ed., Sebastobol Calif. u.a. 2012, S. 5 [↑](#footnote-ref-14)
15. Vgl. dies., *Version control with Git. [powerful tools and techniques for collaborative software development ; covers GitHub]*, 2. ed., Sebastobol Calif. u.a. 2012, S. 5 [↑](#footnote-ref-15)
16. Vgl. dies., *Version control with Git. [powerful tools and techniques for collaborative software development ; covers GitHub]*, 2. ed., Sebastobol Calif. u.a. 2012, S. 5 [↑](#footnote-ref-16)
17. Fischer, Lars, »Werkzeuge zum Versions- und Variantenmanagement:. CVS/Subversion vs. ClearCase«, S. 3 [↑](#footnote-ref-17)
18. Vgl. Loeliger, Jon/Matthew McCullough, *Version control with Git. [powerful tools and techniques for collaborative software development ; covers GitHub]*, 2. ed., Sebastobol Calif. u.a. 2012, S. 5 [↑](#footnote-ref-18)
19. Vgl. Denker, Merlin/Stefan Srecec, »Versionsverwaltung mit Git. Fortgeschrittene Programmierkonzepte in Java, Haskell und Prolog« (2015), S. 5 [↑](#footnote-ref-19)
20. Vgl. Loeliger, Jon/Matthew McCullough, *Version control with Git. [powerful tools and techniques for collaborative software development ; covers GitHub]*, 2. ed., Sebastobol Calif. u.a. 2012, S. 5 [↑](#footnote-ref-20)
21. Vgl. Dr.-Ing. Mathias Magdowski, »Versionskontrolle mit Apache Subversion« (2015), S. 27 [↑](#footnote-ref-21)
22. Vgl. Fischer, Lars, »Werkzeuge zum Versions- und Variantenmanagement:. CVS/Subversion vs. ClearCase«, S. 3 [↑](#footnote-ref-22)
23. Vgl. ders., »Werkzeuge zum Versions- und Variantenmanagement:. CVS/Subversion vs. ClearCase«, S. 5 [↑](#footnote-ref-23)
24. Vgl. Lämmer, Frank, Traditionelle Webentwicklung vs Versionsverwaltung, 06.04.2017, *http://t3n.de/news/traditionelle-webentwicklung-versionsverwaltung-303580/* [↑](#footnote-ref-24)
25. Vgl. Dr.-Ing. Mathias Magdowski, »Versionskontrolle mit Apache Subversion« (2015), S. 28 [↑](#footnote-ref-25)
26. Vgl. Denker, Merlin/Stefan Srecec, »Versionsverwaltung mit Git. Fortgeschrittene Programmierkonzepte in Java, Haskell und Prolog« (2015), S. 11 [↑](#footnote-ref-26)
27. Vgl. Denker, Merlin/Stefan Srecec, »Versionsverwaltung mit Git. Fortgeschrittene Programmierkonzepte in Java, Haskell und Prolog« (2015), S. 11–12 [↑](#footnote-ref-27)
28. Vgl. Mauel, Volker, »Vergleich von Git und SVN« (2013), S. 8 [↑](#footnote-ref-28)
29. Vgl. Fünten, Alexander aus der, »GIT & SVN. Versionsverwaltung« (2012), S. 5 [↑](#footnote-ref-29)
30. Vgl. Dr.-Ing. Mathias Magdowski, »Versionskontrolle mit Apache Subversion« (2015), S. 28 [↑](#footnote-ref-30)
31. Vgl. Fünten, Alexander aus der, »GIT & SVN. Versionsverwaltung« (2012), S. 6 [↑](#footnote-ref-31)
32. Vgl. Dr.-Ing. Mathias Magdowski, »Versionskontrolle mit Apache Subversion« (2015), S. 28 [↑](#footnote-ref-32)
33. Vgl. Sieverdingbeck, Ingo/Jasper van den Ven, »Versionsverwaltung mit SVN« (2008), S. 9 [↑](#footnote-ref-33)
34. Vgl. Fischer, Lars, »Werkzeuge zum Versions- und Variantenmanagement:. CVS/Subversion vs. ClearCase«, S. 3 [↑](#footnote-ref-34)
35. Vgl. Fünten, Alexander aus der, »GIT & SVN. Versionsverwaltung« (2012), S. 6 [↑](#footnote-ref-35)
36. Vgl. ders., »GIT & SVN. Versionsverwaltung« (2012), S. 6 [↑](#footnote-ref-36)
37. Vgl. ders., »GIT & SVN. Versionsverwaltung« (2012), S. 6–7 [↑](#footnote-ref-37)
38. Vgl.Mauel, Volker, »Vergleich von Git und SVN« (2013), S. 7 [↑](#footnote-ref-38)