Versionskontrolle: Subversion und Git

Sascha Schulz, sascha@s10z.de

2. März 2017

Zusammenfassung

Dezentrale Arbeitsabläufe sind in der Praxis wenig etabliert. Features, welche dem dezentralen Charakter zugeschrieben werden, erweisen sich jedoch als wichtig. Insbesondere die Möglichkeit, Versionen lokal zu erstellen. Es wird dasselbe Modell des versionierten Dateisystems verwendet. Unterschiede im Versionsmodell zeigen sich als ausschlaggebend für den deutlich höheren Grad an Flexibilität in Bezug auf Branches und den Umgang mit der Versionshistorie.

Inhaltsverzeichnis

1 Einleitung

In der Arbeit mit digitalen Artefakten besteht das Interesse, nachverfolgen zu können, wie diese im Laufe der Zeit geändert wurden. Hierzu wurden erstmalig in den 70er Jahren Versionskontrollsysteme (*engl.*: version control system, VCS) entwickelt, welche letztlich im Werkzeug *Concurrent Versions System*[1] (CVS) mündeten. Dieses verwaltet die Historie auf Basis von Versionen einzelner Dateien und speichert Metadaten zu deren Bearbeitungen. Da jedoch oftmals Dateien in Zusammenhang stehen und auch Änderungen an ihnen im Bezug zu sehen sind, wurde dies langfristig als unzureichend angesehen. Insbesondere im kommerziellen Bereich wurden diverse Konkurrenzprodukte entwickelt, als Ablösung von CVS setzte sich jedoch das im Jahr 2000 entwickelte, freie Tool *Subversion*[2] (SVN) durch [4].

Bei CVS und SVN handelt es sich um zentrale Versionskontrollsysteme (*engl.*: centralized version control system, cVCS). Eine Vielzahl an Open-Source-Projekten mieden die Existenz einer einzelnen zentralen Instanz und setzen auf denzentrale Lösungen (*engl.*: decentralized version control system, dVCS). Insbesondere das kommerzielle Produkt *Bitkeeper*[3] kam dabei zum Einsatz, welches bis zum Jahr 2005 eine spezielle Lizenz hierfür zur Verfügung stellte.

Mit der Beendigung dieser Lizenz und der Notwendigkeit einer Alternative wurde unter anderem *Git*[4] entwickelt, welches zunehmend an Einfluss gewann und schließlich Subversion in der Rolle des de-facto Industrie-Standards ablöste.

Im Rahmen dieser Arbeit werden die Ähnlichkeiten und Unterschiede zwischen Subversion und Git beleuchtet. Nach einem kurzen Abriss der allgemeinen Terminologie wird auf die Auswirkung der Lokalität (zentral vs dezentral) der Versionshoheit eingegangen. Der Fokus dieser Arbeit liegt auf der Untersuchung der verwendeten Modelle: Das gemeinsame Modell des versionierten Dateisystems wird erläutert und der essenzielle Unterschied im Versionsmodell aufgezeigt, sowie die sich daraus ergebenden Implikationen gefolgert.

Auf den Aspekt der Speicherverwaltung (Delta-Kompression) wird nicht eingegangen; hierfür sei auf die Arbeit von P. Baudis [1] verwiesen.

1.1 Terminologie

Eine Version fasst eine Vielzahl an Änderungen zu einer Einheit zusammen. Die Änderungen können dabei auf ein einzelnes Artefakt beschränkt als auch Artefakt-übergreifend sein.

Eine Version wird bei Subversion als *Revision* bezeichnet während in der Sprache von Git hierbei *Commit* verwendet wird. Wir werden vorrangig vom allgemeinen Begriff der Version sprechen und maximal im anwendungsspezifischen Fall auf eine der beiden alternativen Terme zurückgreifen.

[1] http://savannah.nongnu.org/projects/cvs
[2] http://subversion.apache.org/
[3] http://www.bitkeeper.com/
[4] http://www.git-scm.com

Die Software-Entwicklung, in denen Versionskontrollsysteme exzessiv zum Einsatz kommen, hat die Vorstellungen und die verwendete Sprache maßgeblich geprägt. Dabei lässt sich an der Sprache des jeweiligen VCS erahnen, welche Auffassung von den Strukturen der Software-Entwicklung vorherrscht.

Bei Subversion existiert ein *Trunk* (dt.: *Stamm*), vom dem aus *Branches* (dt.: *Zweige*) ausgehen. Dahingegen existieren bei Git ausnahmslos Branches. Per Konvention existiert jedoch ein "Master", welcher in der Idee dem Trunk gleicht. Branches können beliebig aufeinander aufbauen und bilden so ein Geflecht. Der Trunk ebenso wie die Branches wachsen, wenn ihre Entwicklung voran schreitet.

Bei Bedarf werden Trunk und Branches oder auch Branches untereinander zusammengeführt, das sogenannte *merging* als Gegenoperation zum *branching*. Merging bezeichnet darüber hinaus auch das Zusammenführen konkurrierender Versionen zu einem einheitlichen Bild. Weiterführende Informationen zum Thema "merging" findet sich in der Arbeit von T. Mens [3].

Zusätzlich zu Branches existieren in beiden Sprachen *Tags* (dt.: *Markierungen*). Dies sind statische Markierungen, welche spezielle Versionen (z.B. Meilensteile oder Veröffentlichungen) herausstellen.

2 Zentrale und dezentrale Versionskontrolle

Grundlegendes Unterscheidungsmerkmal zwischen Git und Subversion ist die Hoheit über die Versions-Erstellung. Bei Subversion liegt diese bei einer zentralen Instanz während dies bei Git dezentral bei jedem Anwender erfolgt.

Subversion ist ein zentrales Versionskontrollsystem und basiert auf einem zentralen Server, dem so genannten *Repository*. Dieses speichert die Historie des unterstellten Dateisystems. Anwender interagieren mit dem Repository indem sie eine Arbeitskopie erstellen, diese modifizieren und diese Änderungen dem Repository bekannt machen, welches daraus eine neue Version formt.

Die Arbeitskopie wird dabei auf Basis eines versionierten Verzeichnisses erstellt. Per Konvention wird hierzu das Repository auf oberster Ebene durch Projekt-Order gegliedert, welche jeweils ein Unterordner für Trunk, Branches und Tags enthalten. Ein Branch ist – ebenso wie ein Tag – als eigenständiges Verzeichnis realisiert.

Da mehrere Anwender potentiell gleichzeitig mit dem Repository interagieren, kann dies zu Überschneidungen führen. Dies ist von Subversion beabsichtigt, es wird anstelle von Locking-Mechanismen ein "copy-modify-merge"-Ansatz verfolgt [7]. Bei zwei Anwendern kann dabei lokal jeweils eine eigene Modifikation eines Artefakts vorliegen. Zum Zeitpunkt der Veröffentlichung einer Änderung – dem Erstellen einer neuen Version durch das zentrale Repository – wird die erste Änderung übernommen. Wird versucht die zweite, konkurrierende Änderung zu veröffentlichen, meldet das Repository jedoch den Fehler, dass der Ausgangszustand der Datei veraltet ist. Der Anwender muss erst die zwischenzeitlich erfolgten Änderungen in seine Arbeitskopie integrieren, bevor eine neue Version geformt werden kann.

Git hingegen ist ein dezentrales Versionskontrollsystem. Jeder Anwender verfügt über ein voll qualifiziertes lokales Repository, welches sämtliche Versionsinformationen enthält. Der Anwender interagiert mit diesem, um neue Versionen zu erstellen. Neben Anwendern und ihren lokalen Repositories können ebenso *bare repositories* auf Servern existieren. Diese sind dahingehend "nackt", als dass sie kein Arbeitsverzeichnis enthalten [5].

Bei Bedarf ist es möglich, dass Repositories miteinander kommunizieren und so Wissen über Versionen ausgetauscht wird. Hierbei nimmt stets ein Kommunikationsteilnehmer die Rolle des aktiven Clients ein und der andere die Rolle des reaktiven Servers.

Im Fall, dass konkurrierende Änderungen für dasselbe Artefakt existieren, werden vollautomatische Merge-Strategien angewandt, um die Änderungen in Einklang zu bringen. Ist dies nicht möglich, ist der Anwender aufgefordert das Problem zu beheben.

2.1 Implikationen der Lokalität

Die Existenz einer zentralen Versionshoheit ermöglicht einige womöglich wichtige Features, bringt jedoch die entsprechenden Abhängigkeiten zu dieser Instanz mit sich.

Durch das zentrale Repository und die Möglichkeit Arbeitskopien auf Basis eines versionierten Ausgangsverzeichnisses zu erstellen, ermöglicht Subversion die Verwaltung von Zugriffsrechten. Es kann beispielsweise eingeschränkt werden, wer Zugriff auf einen Abschnitt hat, in dem sensible Informationen versioniert sind. Bei Git ist dies im Rahmen eines Repositories nicht möglich, da der Nutzer das gesamte Archiv klont und anschließend vollkommene Kontrolle über die lokale Instanz hat. Es lassen sich lediglich Repository-weite Regeln zur Interaktion definieren. Um dennoch Teilverzeichnisse anders händeln zu können, haben sich Strategien rund um verschachtelte Repositories etabliert.

Durch die zentrale Versionshoheit erkennt Subversion Konflikte, sobald Änderungen veröffentlicht werden. Die Entwicklung schreitet somit nur um einen Schritt voran, bis ein derartiges Problem erkannt wird. Bei Git hingegen wird dieser Umstand erst zum willkürlich gewählten Zeitpunkt der Veröffentlichung bzw. Kommunikation bekannt. Dafür ermöglicht die lokale Versionshoheit bei Git, ohne die Verbindung zu einer zentralen Instanz, lokal (z.B. offline) zu versionieren. Es ist den Anwendern überlassen, ob und wann mit zentralen Servern oder anderen Anwendern (Peer-to-Peer) andere Versionen ausgetauscht werden.

Doch trotz der existierenden Flexibilität legen die Antworten der Git User Survey im Jahr 2016 [6] nahe, dass in den etablierten Arbeitsabläufen nur selten hiervon Gebrauch gemacht wird (siehe Abb. 1). Git wird demzufolge primär in zentralistischen Infrastrukturen eingesetzt. Darüber hinaus antworteten 37% der Befragten, dass sie unter anderem Repositories verwenden, in denen ein einziger Branch existiert. Die Vermutung liegt nahe, dass insbesondere prominente Dienste wie Github[5] oder Bitbucket[6] dies begünstigen, welche oftmals die Rolle

[5] www.github.com
[6] bitbucket.org

der zentralen Instanz übernehmen.

Eine Untersuchung von Brindescu et al. [2] kommt darüber hinaus zu dem Ergebnis, dass insbesondere das lokale Erstellen von Versionen bei dezentralen Versionskontrollsystemen hohen Einfluss auf die Arbeitsweise der Anwender hat. Die so erstellten Versionen weisen eine gesteigerte Kohärenz auf. Dahingegen werden zentrale Versionskontrollsysteme insbesondere auf Grund der geringeren Komplexität und der leichteren Erlernbarkeit bevorzugt.

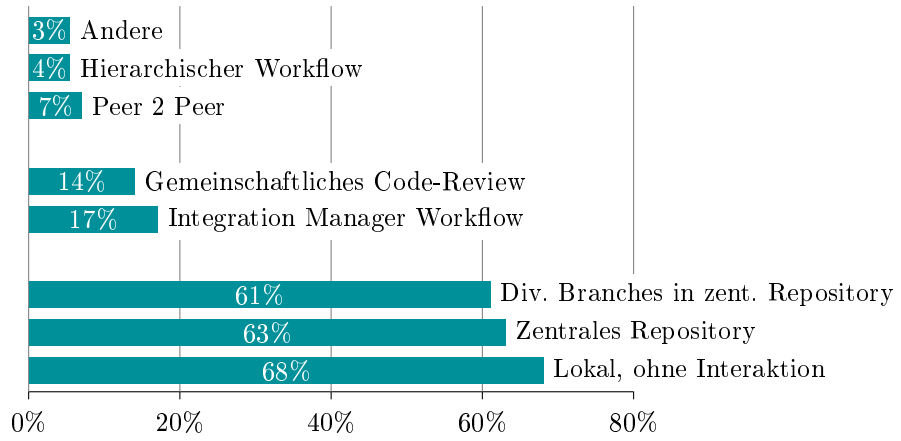

Abbildung 1: Antworten auf die Frage nach den verwendeten Git-Workflows im Rahmen der Git User Survey 2016. Mehrfachnennungen erlaubt.

3 Betrachtung der Modellierung

Im folgenden wird die Modellierung von Subversion und Git betrachtet. Dabei ist bemerkenswert, dass beide Werkzeuge dasselbe Konzept verwenden, um das versionierte Dateisystem abzubilden. Im Versionsmodell lässt sich dahingegen ein kleiner, jedoch essentieller Unterschied feststellen. Im Anschluss wird erläutert, welche enormen Implikationen sich heraus für die Anwendung ergeben.

3.1 Gemeinsames Modell des versionierten Dateisystems

Wie aus den Design-Schriften von Subversion [8] sowie aus dem Git-Handbuch [5] hervorgeht, wird das Dateisystem wie in Abbildung 2 als Baumstruktur betrachtet, welches sich aus Objekten von zwei Typen aufbaut: Verzeichnissen und Dateien. Der Wurzelknoten ist das zu versionierende Verzeichnis, innere Knoten sind ebenfalls Verzeichnisse, Blätter sind Dateien.

Der Bezeichner eines Verzeichnisses oder einer Datei wird an seinem Verweis gespeichert, statt in sich selbst. Ein Verzeichnis ist somit eine Auflistung der Strukturen, die sich darunter befinden, während eine Datei den Datei-Inhalt enthält. Dieser kann als beliebiger Bitstring betrachtet werden, da der Dateityp im Modell abstrahiert wird.

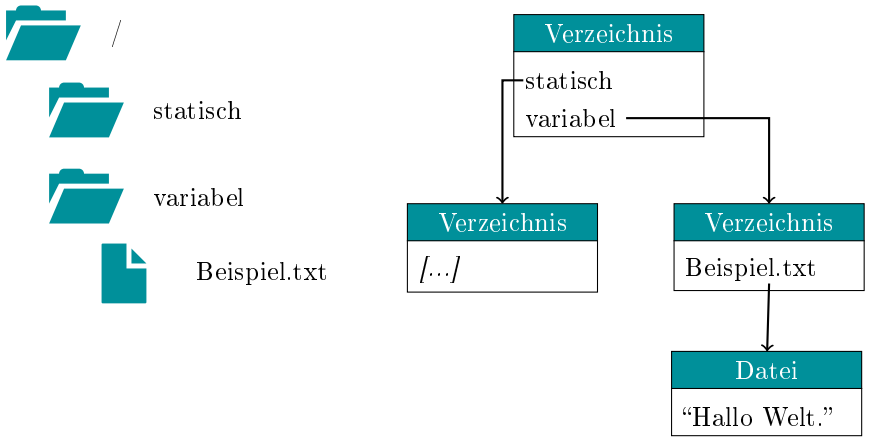

Abbildung 2: Exemplarische Übersetzung einer Verzeichnisstruktur ins Modell. Der Inhalt des Verzeichnisses “statisch” wird aus Übersichtsgründen versteckt.

Findet nun eine Änderung statt, beispielsweise eine Dateiänderung wie in Abbildung 3, so wird vom modifizierten Objekt eine Instanz erzeugt, welche die Änderung abbildet.

Darüber hinaus wird eine neue Instanz des gesamten versionierten Dateisystems erstellt. Hierzu wird vom geänderten Objekt hin zum Wurzelknoten des Baumes von jedem Objekt eine neue Version erzeugt [8]. Von Teilbäumen, welche keinerlei Änderung wiederfuhren (im Beispiel das Verzeichnis “statisch”) wird keine neue Version angelegt, hier wird die bestehende Struktur erneut referenziert.

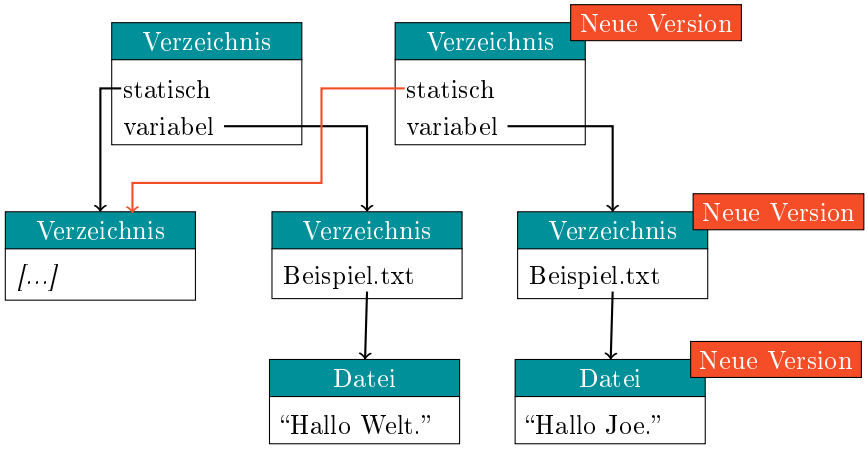

Abbildung 3: Einer Textänderung formt eine neue Version des Dateisystems.

Bei der Umbenennung einer Datei wie in Abbildung 4 zeigt sich, welchen Vorteil die Ausgliederung des Bezeichners eines Objektes bringt: Während sich der Dateiname ändert und die Änderung bis zum Wurzelknoten propagiert wird ist es nicht notwendig, von der Datei selbst eine neue Version anzulegen. Das Umbenennen eines Verzeichnisses funktioniert analog.

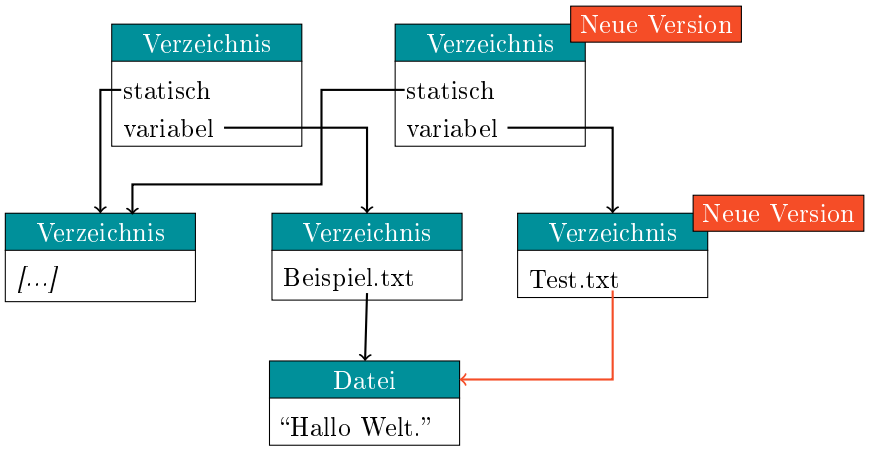

Abbildung 4: Eine Dateiumbenennung formt eine neue Version des Dateisystems.

Auch wenn zwecks Anschaulichkeit die verwendeten Beispiele stets Versionen auf Basis einer einzelnen Änderung aufzeigen: Da stets ein komplett neuer Baum für eine neue Version konstruiert wird, kann dieser Baum auch beliebig viele Änderungen enthalten. Es ist somit hypothetisch auch möglich, dass ein komplett neuer Baum ohne gemeinsame Teilstruktur eine neue Version formt.

Dabei ist zu beachten, dass lediglich die Wiederverwendung von Teilbäumen strukturelle Information besitzt. Zwischen den verschiedenen Versionen einer Datei oder eines Verzeichnisses existiert kein syntaktischer Bezug (wie zum Beispiele eine Referenz, o.ä.). Während sich uns als Betrachter die Semantik in den Beispielen direkt erschließt (durch die Positionierung in der Grafik und der allgemeinen Ähnlichkeit) benötigt es spezielle Verfahren, um dies auf Programmierebene (wieder-)herzustellen.

Im Anschluss existieren somit (von oben gesehen) zwei Einstiegspunkte in das Dateisystem, wobei jeder Einstiegspunkt eine andere Version repräsentiert. Während jede einzelne Version weiterhin das Dateisystem als einen Baum abbildet ist diese Bezeichnung für das Gesamtbild nicht mehr zutreffend; wir erhalten einen ungerichteten, azyklischen Graphen (DAG, directed acyclic graph).

Da für jede Version des Dateisystems ein eigener Einstiegspunkt existiert, ist somit eine Schnittstelle zum Versionsmodell definiert.

3.2 Verschiedene Versionsmodelle

Während Subversion und Git grundsätzlich dasselbe Modell verwenden, um das Dateisystem abzubildenm unterscheiden sie sich geringfügig in ihrem Versionsmodell, was jedoch drastische Unterschiede in den sich daraus ergebenden Möglichkeiten zur Folge hat.

Subversion arbeitet auf Basis von *Revisionen*, wie in Abbildung 5 veranschaulicht wird. Jede Version des Dateisystems hat eine *Revisionsnummer* zugeordnet, anhand derer die Revisionen strikt geordnet sind. Dabei existiert nur ein einzel-

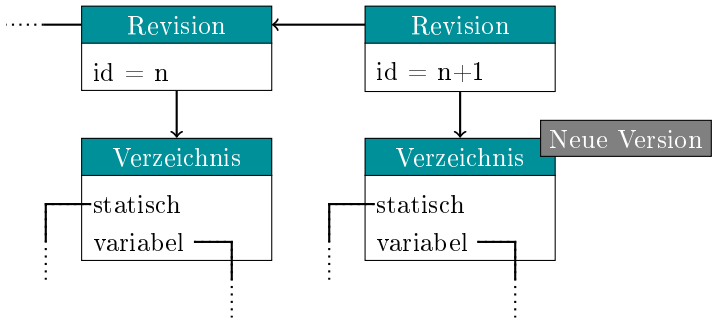

Abbildung 5: Versions-Modell von Subversion

ner Versions-Strang. Somit stellt Subversion ein System dar, bei dem Versionen als nacheinander eintretende Ereignisse in der Zeit betrachtet werden können.

Dies deckt sich insbesondere mit den eingangs erwähnten Anforderungen, wie konkurrierende Versionen vom Anwender in Einklang gebracht werden müssen. Da Subversion ein zentrales Versionsverwaltungssystem ist, existiert auch eine zentrale Instanz, welches die sich aus dem Modell ergebenden Integritätsanforderungen erzwingen kann.

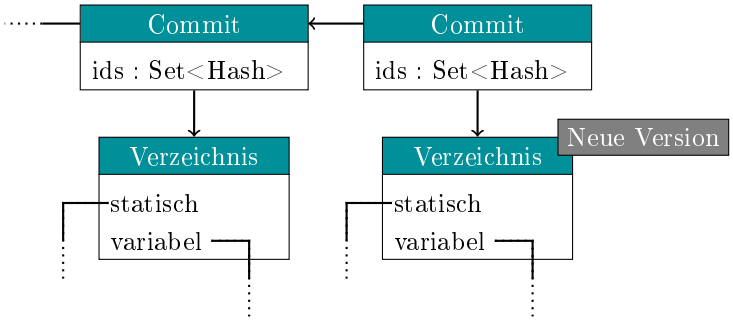

Abbildung 6: Versions-Modell von Git

Git geht dahingegen abstrakter vor und spricht anstelle von Revisionen von *Commits*. An Stelle von Revisionsnummern werden Hashes über Objekte zur Identifikation genutzt und Referenzen auf Vorgängerversionen gespeichert [5].

Dieser Unterschied im Modell erweißt sich letztlich als entscheidend mit weitreichenden Konsequenzen. Zum einen ist es möglich, dass eine Version auf mehrere Vorgänger aufbaut, zum anderen formuliert ein Hash keine implizite Anforderung an die Reihenfolge, in der sich Versionen befinden.

3.3 Implikation der Versionsmodelle

Die Identifikation durch Hashes hat die Auswirkung, dass keine implizite Anforderung an die Reihenfolge gestellt wird, wie dies bei Revisionsnummern der Fall ist. Dadurch sind die Versionen in Git nicht wie in Subversion an eine Striktordnung gebunden, sondern können gleichberechtigt nebeneinander existieren. Dies

hat insbesondere schwerwiegende Auswirkungen darauf, wie sich nebenläufige Entwicklungsstränge realisieren lassen.

In SVN wird für jeden Branch und jedes Tag ein eigenes Verzeichnis angelegt, die koexistenten Stränge werden im Dateisystem abgebildet. Dies ist notwendig, da sich jede Version in einem Branch in die globale Revisionskette einfügen muss. Bereits das Anlegen des Branches selbst stellt daher eine globale Revision dar, ebenso wie eine Änderung im Branch. Die Entwicklungsgeschichte des Branches ist somit mit der des Trunks in der globalen Revisionskette verwoben.

Das Modell des versionierten Dateisystems gibt dabei Aufschluss darüber, wie dabei unnötige Redundanzen auf Serverseite vermieden werden. In der Arbeitskopie werden Artefakte jedoch notwendigerweise pro Branch/Tag dupliziert, was zu einem Overhead führt. Der Nutzer hat die Möglichkeit, dies durch intelligente Auswahl der Weite der Arbeitskopie einzuschränken.

Die Integration eines Branches in den Hauptentwicklungspfad führt schlussendlich zu einer einzelnen neuen Revision, welche sämtliche Änderungen des Branches und deren Auswirkung auf den Trunk zusammenfasst.

Bei Git existieren Branches und Tags virtuell, das Dateisystem zeigt stets nur die aktuell gewählte Perspektive, welche genau einer Version entspricht. Die Versionen stehen lediglich in einer partiellen Ordnung zueinander; und die Branches und Tags enthalten lediglich die Information, welche letzte Version zugehörig ist – sowie einen lesbaren Namen für den Anwender.

Die geringere Kopplung der Versionen ermöglicht es, die Historie neu zu gestalten, beispielsweise in dem die Reihenfolge der Änderungen neu bestimmt wird oder mehrere Versionen zu einer zusammengefasst werden. Da jeder Anwender lokal operiert, gibt es hier keine Einschränkungen an die Integrität. Sofern die Kommunikation mit anderen Repositories angestrebt wird, ist jedoch darauf zu achten, dass gemeinsames Wissen über die Historie zwecks Kompatibilität bewahrt wird.

Unter Git hat jeder Branch seine eigene Historie, welche ab dem Zeitpunkt der Abzweigung separat geschrieben wird. Werden Branches wieder zusammengeführt, können sie sowohl in linearen Zusammenhang gebracht werden, als auch koexistent bleiben und durch in einem "merge-commit" vereinigt werden. Dabei wir die Möglichkeit genutzt, dass ein Commit auf mehrere Vorgänger gleichzeitig verweisen kann.

4 Fazit

Bei Subversion und Git handelt es sich um stark voneinander abweichende Realisierungen von Versionskontrollsystemen. Es ist bemerkenswert, dass sie dennoch im Kern auf dasselbe Modell für das versionierte Dateisystem aufbauen.

Als klaren Vorteil von Subversion gegenüber Git ist der Sicherheitsaspekt hervorzuheben: Benutzerrechte sind im Detail steuerbar und nicht nur auf oberster Ebene. Der Umgang ist leicht zu erlernen und Konflikte werden mit dem Erstel-

len einer Version erkannt. Das Versionsmodell ist einfach gehalten, die Verwendung von Revisionsnummern erfordert eine strikte Sortierung der Versionen.

Git weißt einen deutlich höheren Grad an Flexibilität als Subversion auf. In der Praxis werden die infrastrukturellen Möglichkeiten eines dezentralen Systems anscheinend nur selten genutzt. Essentieller Vorteil ist die Entkopplung von einer zentralen Instanz: Versionen können lokal erstellt werden. Auch wenn dies dazu führt, dass Konflikte erst bei deren Kommunikation erkennbar werden. Darüber hinaus zeigt sich das Versionsmodell von Git als sehr leistungsstark, um nebenläufige Entwicklungen abzubilden und zusammenzuführen, da Commits mehrere Vorgänger referenzieren können und abgesehen von diesen Verweisen in keinem weiteren Bezug zueinander stehen.

Referenzen

[1] P. Baudis, *Current Concepts in Version Control Systems*, CoRR, https://arxiv.org/abs/1405.3496 (Abgerufen am 2017-03-02)

[2] C. Brindescu, M. Codoban, S. Shmarkatiuk and D. Dig, *How Do Centralized and Distributed Version Control Systems Impact Software Changes?*, Proceedings of the 36th International Conference on Software Engineering ICSE 2014, Pages 322-333, 2014

[3] T. Mens, *A State-of-the-Art Survey on Software Merging*, IEEE Trans. Software Eng., Vol. 28, Pages 449-462, 2002

[4] P. Mukherjee, *A fully Decentralized, Peer-to-Peer Based Version Control System*, Ph.D. Thesis, Technische Universität Darmstadt, 2011, http://tuprints.ulb.tu-darmstadt.de/2488/ (Abgerufen am 2017-03-02)

[5] Git Book v2, https://git-scm.com/book/en/v2/Git-Internals-Git-Objects (Abgerufen am 2017-03-02)

[6] Git User's Survey 2016 summary, https://git.wiki.kernel.org/index.php/GitSurvey2016 (Abgerufen am 2017-03-02)

[7] B. Collins-Sussman, B. W. Fitzpatrick, and C. M. Pilato, *Version Control with Subversion For Subversion 1.7*, compiled from r5302, http://svnbook.red-bean.com/en/1.7/svn-book.pdf (Abgerufen am 2017-03-02)

[8] Subversion Design Repository, Section "Bubble-Up Method", https://svn.apache.org/repos/asf/subversion/trunk/notes/subversion-design.html (Abgerufen am 2017-03-02)