Versionskontrolle

Centralized vs Distributed

León Rheinert

Martrikelnummer: IF020017

Exposé für die Lehrveranstaltung
Wissenschaftliches Arbeiten

im Studiengang
Wirtschaftsinformatik

ASW-Gutacher
Prof. Dr. Dieter Hofbauer

31. Mai 2019

Inhaltsverzeichnis

1 Einleitung

Hier werden Probleme die bei einem Softwareentwicklungsprojekt auftreten, wenn als Team und ohne Versionskontrollsystem gearbeitet wird beschrieben und die daraus resultierenden Wünsche zur Lösung als Anschnitt des Funktionsumfangs von Versionskontrollsystemen dargestellt.

2 Was ist ein Version Control System

2.1 Definition

„Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later“[1]

2.2 Verwendung

Ein Versionskontrollsystem speichert demnach jede einzelene Änderung an ausgewählten Dateien in sogenannten Repositories um die Möglichkeit auf eine alte Version zu springen zu ermöglichen und wird vor allem in der Softwareentwicklung eingesetzt

2.3 Funktionsumfang

Mit Versionen werden oft zusätzliche Informationen wie der Bearbeiter und ein Zeitstempel gespeichert um jede Änderung sehr genau zurückverfolgen zu können.
Darüber hinaus gibt es die Möglichkeit innerhalb eines Projekt eine Parallelversion zu starten *(engl. branching)* und verschiedene Branches[2] zu vereinen *(engl. merging bzw. patching)*.

3 Centralized vs Distributed

Versionskontrollsysteme werden hauptsächlich in zentrale (von engl. *centralized*) und verteilte (von engl. *distributed*) Systeme unterschieden. Daneben gibt es noch lokale Versionskontrolle[3], die sich nur auf einen Rechner und meist nur ein Dokument beschränkt.

[1] Scott und S. Ben 2010, S. 1.
[2] Einführung in Branchs und Merging Scott und S. Ben 2010, S. 49-54.
[3] Siehe Scott und S. Ben 2010, S. 1.

3.1 Centralized

Zentralte Versionskontrollsysteme sind mit dem Client-Server abgebildet, wobei der Server meist nur Schnappschüsse, sprich die Versionen von Dateien speichert[4] und die Beteiligten sich nur mit dem Server synchronisieren.

3.2 Distributed

Der Begriff *verteilt* bedeutet in diesem Kontext, dass alle Beteiligten durch lokale Repositories die komplette Historie aller Dateien haben und sich untereinander synchronisieren (Peer-to-Peer).
Man spricht nicht mehr von Versionen, sondern von Revisions, also Änderungen an einer Datei (zb. neue Zeile 10: "Hallo Welt").[5]

3.3 Vor-/Nachteile

Hier werden die jeweiligen Vor- und Nachteile der beiden Systeme betrachtet und als Übersicht in Trade-offs verpackt.
Beispiel zentrale Systeme: Abhängigkeit eines Servers dafür mehr Kontrolle des Workflows (zb. durch Locking[6])
Beispiel verteilte Systeme: Es läuft offline und schnell dafür keine „neuste Version“[7]

4 Praxis

Aufgrund der in der Einführung (1) beschriebenen Probleme bei Softwareentwicklung finden Versionskontrollsysteme in der Praxis ihren Einsatz

4.1 Anwendungsfälle

Dabei kann man die Systeme nicht nur für ihren ursprünglichen Zweck[8], das Versionieren von Quellcode in Softwareprojekten nutzen, sondern auch für allmöglichen Dateien, bei denen man eine Historie haben will und die man ggf. als Team erstellt.

[4] Vgl. Kivanc u. a. 2014, S. 671.
[5] Siehe Azad o. J. für anschauliche Beispiele.
[6] Erklärung des Mechanisumses C. Michael, C.-S. Ben und Brian 2008, S. 3-4.
[7] Vgl. Azad o. J.
[8] Vgl. Blischak, Davenport und Wilson 2016.

4.2 Vorhandene Software

Beide Arten sind nicht nur theoretisch erklärbar, sondern sind bereits technisch umgesetzt und finden in Unternehmen als Standardsoftware ihren Einsatz.

4.2.1 Centralized

Durch das Open-Source-Projekt Concurrent Versions System wurde diese Art erstmalig populär und im Jahr 2000 mit Subversion[9] neu implementiert[10].

4.2.2 Distributed

An dieser Stelle sind vor allem *Git*[11] sowie das Konkurenzprodukt *Mercurial*[12] zu nennen. Beide fallen unter die Kateogrie freie Software[13] doch unterscheiden sich in Komplexität und Erlernbarkeit[14]

5 Fazit

Hier wird die Arbeit zusammengefasst und wieder auf die Stärken der beiden Arten eingegangen um ihre jeweilige Daseinsberechtigung zu untermauern.

[9] siehe C. Michael, C.-S. Ben und Brian 2008, für eine umfangreiche Einführung.
[10] siehe C. Michael, C.-S. Ben und Brian 2008, S. xi Anhang B für Unterschiede der Implementierungen.
[11] Siehe Scott und S. Ben 2010, für eine umfangreiche Einführung.
[12] Siehe Bryan 2009, für eine umfangreiche Einführung.
[13] Definition Grassmuck 2004, S.281-284.
[14] Vgl. Kirmanen 2017.

Literatur

Azad, Kalid (o. J.). *Intro to Distributed Version Control*. URL: https://betterexplained.com/articles/intro-to-distributed-version-control-illustrated/ (besucht am 31. 05. 2019).

Blischak, John D., Emily R. Davenport und Greg Wilson (2016). „A Quick Introduction to Version Control with Git and GitHub“. In: *PLOS Computational Biology* 12.1, S. 1–18. DOI: 10.1371/journal.pcbi.1004668. URL: https://doi.org/10.1371/journal.pcbi.1004668.

Bryan, O'Sullivan (2009). *Mercurial: The Definitive Guide*. O'Reilly Media.

C. Michael, Pilato, Collins-Sussman Ben und W. Brian (2008). *Version Control with Subversion: Next Generation Open Source Version Control*. 2. Aufl. O'Reilly Media.

Grassmuck, Volker (2004). *Freie Software. Zwischen Privat- und Gemeineigentum*. deu. Bundeszentrale für politische Bildung. ISBN: 3893315691. DOI: http://dx.doi.org/10.25969/mediarep/3575. URL: https://mediarep.org/handle/doc/4317.

Kirmanen, Antti (2017). *Git oder Mercurial: Was sollte man nutzen?* URL: https://entwickler.de/online/development/git-mercurial-versionskontrollsystem-vergleich-579792235.html (besucht am 29. 05. 2019).

Kivanc, Muslu u. a. (2014). „Transition from centralized to decentralized version control systems: a case study on reasons, barriers, and outcomes“. In: *ICSE*.

Scott, Chacon und Straub Ben (2010). *Pro Git*. 2. Aufl. Apress.