TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Wirtschaftswissenschaften

Professur für Wirtschaftsinformatik,   
insb. Informationssysteme in Industrie und Handel

Das ist das Thema der Arbeit

Bachelor/Master/Diplomarbeit

zur Erlangung des akademischen Grades

„Bachelor/Master of Science/DipomXYZ“

|  |  |
| --- | --- |
| Name: | Musterfrau, Erika |
| Adresse: | Musterstraße, 01234 Musterstadt |
| Matrikelnummer: | 1234567 |
| Übermittelt an: | Prof. Dr. Muster |
| Übermittlungsdatum: | 28.11.2018 |

Inhaltsverzeichnis

[Abbildungsverzeichnis II](#_Toc516228888)

[Tabellenverzeichnis III](#_Toc516228889)

[Formelverzeichnis III](#_Toc516228890)

[Abkürzungsverzeichnis III](#_Toc516228891)

[1 Einleitung (Überschrift 1) 3](#_Toc516228892)

[2 Verwendung der Vorlage 3](#_Toc516228893)

[2.1 Überschriften (Überschrift 2) 3](#_Toc516228894)

[2.2 Standardtext und Hervorhebungen 3](#_Toc516228895)

[2.3 Aufzählungen und Nummerierungen 3](#_Toc516228896)

[2.4 Abbildungen, Tabellen und Formeln 3](#_Toc516228897)

[2.4.1 Abbildungen (Überschrift 3) 3](#_Toc516228898)

[2.4.2 Tabellen 3](#_Toc516228899)

[2.4.3 Formeln 3](#_Toc516228900)

[2.5 Fußnoten 3](#_Toc516228901)

[2.6 Kopfzeile 3](#_Toc516228902)

[2.7 Abkürzungen 3](#_Toc516228903)

[2.8 Literaturverweise 3](#_Toc516228904)

[2.9 Verzeichnisse 3](#_Toc516228905)

[2.10 Eidesstattliche Erklärung 3](#_Toc516228906)

[Literaturverzeichnis 3](#_Toc516228907)

[Anhangsverzeichnis III](#_Toc516228908)

[Anhang III](#_Toc516228909)

###### Abbildungsverzeichnis

[Abbildung 1: Aussagekräftige Bildunterschrift (Quelle, 1970, S. 25) 3](#_Toc516228910)

[Abbildung 2: Hinweis zum Setzen der korrekten Seitenzahl im Anhang 3](#_Toc516228911)

###### Tabellenverzeichnis

[Tabelle 1: Aussagekräftige Tabellenbeschriftung 3](#_Toc516228912)

[Tabelle 2: Einsatz arabischer und römischer Seitenzahlen 3](#_Toc516228913)

###### Formelverzeichnis

[Formel 1: Biomialformel 3](#_Toc516228914)

###### Abkürzungsverzeichnis

|  |  |
| --- | --- |
| Abk. | Abkürzung |
|  |  |

# Einleitung

Ziel ist es, einen Marktüberblick der verfügbaren Versionskontrollsysteme für die Softwareentwicklung darzustellen. Als Grundlage dafür soll zunächst erläutert werden, was VCS leisten können und warum sie in der modernen Softwareentwicklung eine große Bedeutung haben. Dabei sollen die wichtigsten Begriffe geklärt und eine grundlegende Systematisierung der verschiedenen Konzepte vorgestellt werden.

# Theoretische Grundlagen von Versionskontrollsystemen

Hier sollen die Grundlagen und Prinzipien dargestellt werden, auf denen die Versionskontrolle basiert, welche unterschiedlichen Konzepte existieren

## Was sind VCS? Aufgaben von VCS

Der Entwicklungsprozess moderner Software und Systeme wird in der Regel von mehreren Entwicklern getragen, entstandene Quellcode wird verändert und hinsichtlich neuer Anforderungen angepasst

Die Entwicklung einer Software kann in der Regel nicht als ein linearer Prozess beschrieben werden. Geschriebener Quellcode muss über den Verlauf von Softwareprojekten hinweg verändert, erweitert oder gelöscht werden, um Verbesserungen herbeizuführen oder auf sich verändernde Anforderungen zu reagieren. Diese Anpassungen werden, auf Grund der Komplexität der Softwareprojekte, meist durch mehrere Entwickler umgesetzt, die zu diesem Zweck eng miteinander kooperieren. Im Zentrum dieser Zusammenarbeit steht bei vielen Entwicklerteams ein Versionskontrollsysteme (engl. *Version Control Systems*, VCS).

Es existiert eine Vielzahl an Begrifflichkeiten, die in diesem Zusammenhang genannt werden. Dazu gehören im deutschen Sprachraum Versionsverwaltungssysteme und im englischen Sprachraum *revision control systems, source code management* und *source code control*. All diese Begriffe können synonym verwendet werden und beschreiben die gleiche Art von Systemen und Werkzeugen. Am weitesten verbreitet ist jedoch die Bezeichnung Versionskontrollsystem bzw. *version control system,* weshalb für die vorliegende Arbeit diese Terminologie gewählt werden soll (vgl. Otte, 2009, S. 1).

Für die kollaborative Arbeit von Entwicklern, müssen sie in die Lage versetzt werden, auf die gleichen Dateien zuzugreifen und diese bearbeiten zu können. VCS leisten in diesem Zusammenhang verschiedene wichtige Aufgaben. Sie sorgen dafür, dass vorgenommenen Änderungen der Projektdateien protokolliert werden (mit Benutzer und Zeitstempel), eine Archivierung von unterschiedlichen Versionsständen erfolgt und diese ggf. wiederhergestellt werden können. Nur so kann Transparenz geschaffen werden und die Gefahr von Datenverlusten reduziert werden (vgl. Zolkifli, 2018, S.409). Darüber hinaus bieten VCS das Potenzial, durch verbesserte Abläufe den Softwareentwicklungsprozess erheblich zu Beschleunigen und zu Vereinfachen (vgl. Otte, 2009, S. 1). Dies beinhaltet vor allem auch die Koordination von gemeinsamen Zugriffen und die Möglichkeit der parallelen Arbeit an verschiedenen Entwicklungszweigen. Eine maßgebliche Herausforderung in der Softwareentwicklung besteht darin, den „Change“-Prozess zu skalieren und effizient auf eine Vielzahl an Entwicklern zu verteilen (vgl. de Alwis, 2009, S. 36).

Dabei ist ein besonderes Management notwendig (Integration von Leitung, Organisation, Planung), um

Wenn zwei oder mehr Programmierer auf die selbe Datei zugreifen wollen, kommt es unweigerlich zu einem Konflikt. Es existieren zwei Modelle, die die Lösung dieses Konflikts ermöglichen.

🡪 lock-modify-unlock

🡪 Copy-modify-merge

Es existieren zwei grundlegende Modelle von VCS. Diese sollen im folgenden vorgestellt werden.

## Zentrale Versionskontrollsystemen (CVCS)

## Dezentrale Versionskontrollsysteme (DVCS)

Inkonsistenzen

Das Management von verschiedenen Revisionen (sequentielle/fortlaufende Versionen) und Varianten (simultane Versionen) steht dabei im Fokus.(vgl. Linsbauer, 2017, S. 49).

Somit determinieren VCS maßgeblich, wie die Entwicklung von neuen Funktionen koordiniert wird, mit welcher Regelmäßigkeit und unter welchem Aufwand Entwicklungslinien zusammengeführt werden

* Anforderungen an Softwareentwicklung
* Definition VCS 🡪 Linsbauer
* Verschiedene Varianten von Bezeichnungen, meinen alle das selbe, VCS ist die am weitesten verbreitete Bezeichung (vgl. Zolkifli, 2018, S. 409), (vgl. Otte, 2009, S. 1[Primärquelle])
* Begriffe klären: Version/Revision,
* Abgrenzung zu Variation Control Systems
* Nutzen:
  + Beschleunigung und Vereinfachung des Entwicklungsprozesses (vgl. Zolkifli, 2018, S. 409)
  + Management, Organisation und Koordination der Entwicklung (von Objekten) (vgl. Zolkifli, 2018, S. 409)
  + Effektive Zusammenarbeit von **mehreren Entwicklern** die **parallel** und (geografisch) **verteilt** an **einem Projekt** arbeiten (vgl. Zolkifli, 2018, S. 409)
  + Grundlegende Arbeitsweise mit VCS
  + Beschleunigung und Vereinfachung des Softwareentwicklungsprozesses, ermöglichen neue Workflows/Arbeitsabläufe (Otte, S. 1)

## Funktionsweisen von Versionskontrollsystemen

Wie unterscheiden sich die Architekturen von verteilten und zentralen Versionskontrollsystemen. Außerdem soll betrachtet werden, welche Vor- und Nachteile jeweils damit einhergehen.

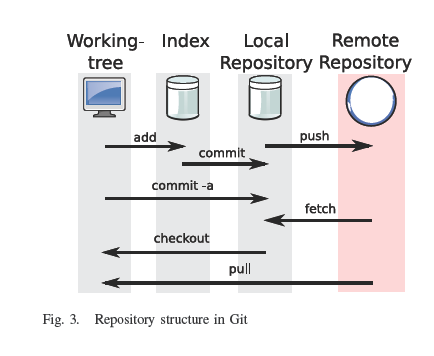
* in einem Repository werden die entsprechenden Dateien mit dazugehörigen Meta-Daten gespeichert. Damit ist es möglich, die einzelnen Veränderungsstufen der Datei über die Zeit nachzuvollziehen. (vgl. Otte, 2009, S. 1). Die Synchronisation von der Versionen übernimmt eine Versionsverwaltungssoftware. Es existieren drei Varianten wie diese genutzt werden können. Neben einer kommandozeilenorientierten Funktionsweise, kann die Synchronisation über eine grafische Benutzeroberfläche oder eine Erweiterung der integrierten Entwicklungsumgebung (IDE) erfolgen.
* Checkout (von Dateien oder ganzen Repositories), Arbeitsverzeichnis (*Working copy*)
* Konzepte: (vgl. Louridas, 2006, S. 105)
  + Lock-Modify-Unlock
    - klassischer Ansatz für eine „Konkurrenz-Situation“, auch „pessimistischer Ansatz“ (Otte, S. 3)
    - Ein Prozess sperrt explizit eine „Ressource“/Datei, nur dann kann die Ressource bearbeitet werden, nach Ende der Bearbeitung wird die Sperrung entfernt 🡪 das Mergen/Zusammenführen von veränderten Dateien ist nicht notwendig, da zu einem bestimmten Zeitpunkt nur durch eine Person Veränderungen an der jeweiligen Datei vornehmen kann (Otte, S. 3)
    - für VCS nicht wirklich möglich, da das Entsperren vergessen werden kann , wird nur in einigen Fällen für CVCS verwendet (Otte, S. 3)
    - Keine Bearbeitung durch Dritte möglich, allerdings kann die Datei mit Leserechten ausgecheckt werden (Louridas, S. 104)
    - Model ist einfach aber restriktiv aber hinderlich bei verteilter Softwareentwicklung
  + Copy-Modify-Merge
    - optimistischer Ansatz
    - Laut Otte arbeiten VCS immer mit diesem Prinzip, da er weniger Administration bedarf und bessere Arbeitsabläufe ermöglicht
    - Eine Ressource kann von verschiedenen Prozessen verändert werden, muss aber im Anschluss zusammengeführt werden
    - egalitarian: weniger restriktiv, jedoch muss es einen einvernehmlichen Prozess zu Konfliktlösung geben
    - unstrukturierte Merge-Tools: Nutzen nur Texterkennung um Konflikte zu identifizieren
    - strukturierte Merge-Tools: arbeiten mit Syntax Erkennung der Programmiersprache
* Branches:
  + können auch in CVCS angelegt werden (vgl. de Alwis, 2009, S. 37)
  + werden verwendet zur Darstellung von *Releases*, Bearbeitung von aufwändigen Fehlerbehebungen (Bug-Fixes) oder umfangreichen Anpassungen, die später mit dem Haupt-Branch zusammengeführt werden sollen (vgl. de Alwis, 2009, S. 37)
* Architekturen
  + lokal, zentral, dezentral/verteilte Systeme (vgl. Zolkifli, 2018, S. 409f), (vgl. Otte, 2009, S.), (vgl. de Alwis, 2009, S.)
    - **Lokal**
      * Anfänge der Versionsverwaltung, nur lokaler Zugriff durch einzelnen Entwickler

## Zentrale Versionskontrollsysteme

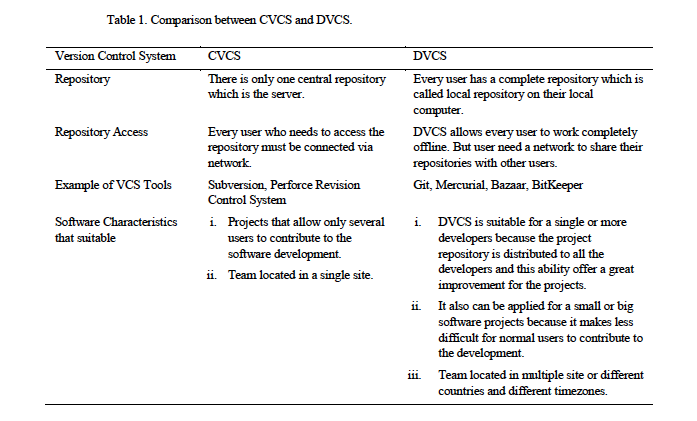
* + - **Zentral CVCS**
      * Erste Ansätze zum kooperativen Arbeiten, Voraussetzung für kooperative Arbeiten
      * Konzept: Trennung der Dateien an denen gearbeitet wird, von der „Master-Kopie“ im Repository, Check-Out 🡪 Veränderung 🡪 Check-In (vgl. Louridas, 2006, S. 104)
      * Sparen von Speicherplatz: es werden nur Veränderungen dokumentiert und mit Versionsnummern versehen(vgl. Louridas, 2006, S. 104)
      * Auf einem Server wird die Master-Kopie des Projekts abgelegt. Dieser Server muss für das Lesen, Abrufen oder Einbringen (Commit) von Veränderungen (Changes) kontaktiert werden. Veränderungen an den Dateien werden über eine Datenbank dokumentiert.

## Dezentrale Versionskontrollsysteme (DVCS)

* + - **dezentral/verteilt**
      * eliminieren einige Nachteile von CVCS (vgl. de Alwis, 2009, S. 36)
      * erstes DVCS: Reliable Software’s Code Co’op in 1997 (vgl. de Alwis, 2009, S. 37)
      * **Jede „ausgecheckte“ Version des originalen Repositories enthält die komplette Versionsgeschichte (= Repository erster Klasse)**
      * Dezentrale VCS können sich wie CVCS verhalten, umgekehrt geht das nicht.
      * Schreibe-Rechte sind nicht mehr relevant, da jeder Entwickler ein Commiter 1. Klasse in seinem Repository ist (vgl. de Alwis, 2009, S. 37)
      * unterstützt durch „ausreichende Informationen“ leichtes Branching und Merging von Branches (vgl. de Alwis, 2009, S. 37)
      * *Feature Branching*: jede potenzielle Anpassung (*Feature*) wird in einem eigenen Branch entwickelt und in die Mainline gemergt anstatt diese direkt auf der „Mainline“ zu entwickeln (vgl. de Alwis, 2009, S. 37)
      * jedes Repository ist ein vollwertiges Repository, weshalb es keinen zwangsläufigen *Master Branch* gibt. Entwickler verständigen sich allerdings auf einen anerkannten („übergeordneten“) Branch
      * mögliche Modelle: **Eigentlich bezogen auf GIT**!
        + ohne Server
        + geteiltes Repository
        + Leutnant-Diktator-Model
        + Integration Manager Model
      * Vorteile
        + positiver Effekt auf Sicherheit der Daten (de Alwis)
        + positiver Effekt auf die Geschwindigkeit von Aktionen
        + hohe Flexibilität, Anpassung an verschiedene Arbeitsabläufe
        + Verfügbarkeit/Mehrwert für non-commiters (de Alwis), besonders wichtig für Open Source Projekte, unterstützt den Community Gedanken besser als CVCS, keine Schreib-Rechte notwendig
        + inkrementelles Speicher von Veränderungen möglich (de Alwis, S. 38)
        + atomare Commits: es werden nur vollständige Commits übertragen oder, im Falle einer Störung, gar kein Bestandteil des Commits, keine Teilung möglich (Baerisch)
        + automatisches Mergen der Commits führt dazu, dass die Entwickler ihre Reps auf dem neuesten Stand halten und verringert außerdem den Aufwand für das Mergen
        + improved support for experimental changes
        + support of disconnected operations
      * GIT:
        + Versionierung wird auf Basis von SHA1 Hash vorgenommen. Eine Versionszuordnung auf Basis eines steigenden Integer-Wertes, wird der nicht-linearen Struktur von verteilten Systemen nicht gerecht und kann damit nicht verwendet werden. Mit dem SHA1 Hash werden die Veränderungen eines Objekts eindeutig identifiziert.
  + Umstrukturierung von CVCS zu DVCS (Quelle nicht gut, aus Zolkifli, 2018, S. 409))🡪 SVN Book, S. XIV



Otte, 2009, S. 9



# Marktüberblick Versionskontrollsystemen und -Software

## Versionskontrollsysteme

* CVS
* GIT
* Mercurial
* SVN
* Bitkeeper

## Versionsverwaltungssoftware

* Beschreibung des Marktes:
  + Viele Open-Source-Software
  + Einige proprietäre Anbieter

## Vergleichskriterien

* K.O.-Kriterien
  + Integration in VS Code
  + Kosten
  + Nutzer
  + Architektur?

## Versionskontrollsoftware Softwareanbieter

* Funktionsumfang

# Vergleich von „relevanten“ Versionskontrollsystemen

* Vorstellung der verschiedenen Systeme
* Zuordnung von Software zu Systemen

Darstellung/Vorstellung oder Vergleich?

Auswahl anhand von:

* Verbreitung
* unterschiedliche Ansätze abdecken (verteilt/zentral)
* Eignung auf Grund von Rahmenbedingungen in gewünschtem Einsatzszenaio

## Vergleichskriterien

###### Literaturverzeichnis

Das Literaturverzeichnis ist mit der Formatvorlage *Quellen* zu verwenden. Angehängt finden Sie ein beispielhaftes Literaturverzeichnis im APA 6th Standard.

Bacharach, S. B. (1989). Organizational Theories: Some Criteria for Evaluation. *Academy of Management Review*, *14*(4), 496–515. <https://doi.org/10.5465/AMR.1989.4308374>

Burton-Jones, A., McLean, E. R., & Monod, E. (2015). Theoretical perspectives in IS research: from variance and process to conceptual latitude and conceptual fit. *European Journal of Information Systems*, *24*(6), 664–679. <https://doi.org/10.1057/ejis.2014.31>

Gregor, S. (2006). The Nature of Theory in Information Systems. *MIS Quarterly*, *30*(3), 611–642.

Shapira, Z. (2011). „I’ve Got a Theory Paper--Do You?“: Conceptual, Empirical, and Theoretical Contributions to Knowledge in the Organizational Sciences. *Organization Science*, *22*(5), 1312–1321. <https://doi.org/10.1287/orsc.1100.0636>

Suddaby, R. (2010). Editor’s Comments: Construct Clarity in Theories of Management and Organization. *Academy of Management Review*, *35*(3), 346–357. <https://doi.org/10.5465/AMR.2010.51141319>

Sutton, R. I., & Staw, B. M. (1995). What Theory is Not. *Administrative Science Quarterly*, *40*(3), 371–384.

Weber, R. (2012). Evaluating and Developing Theories in the Information Systems Discipline. *Journal of the Association for Information Systems*, *13*(1), 1–30.

###### Anhangsverzeichnis

[A 1 Anhang Ebene 1 III](#_Toc516218198)

[A 1.1 Anhang Ebene 2 III](#_Toc516218199)

###### Anhang

#### Anhang Ebene 1

##### Anhang Ebene 2

**Eidesstattliche Erklärung**

Hiermit versichere ich, die vorliegende Arbeit mit dem Titel

„Thema“

selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von mir angegebenen Quellen angefertigt zu haben. Alle aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche gekennzeichnet.

Die Arbeit wurde noch keiner Prüfungsbehörde in gleicher oder ähnlicher Form vorgelegt und wurde noch nicht veröffentlicht.

Dresden, 28.11.2018

|  |  |
| --- | --- |
| Vorname Nachname |  |