

Programmierkonzepte in den Naturwissenschaften

7. Versionsnummern

PD Dr. Till Biskup
Physikalische Chemie
Universität des Saarlandes
Sommersemester 2021





- ❏ Versionsnummern und Versionverwaltung sind zwei voneinander unabhängige Aspekte.
- ❏ Versionsnummern (und ein Versionsverwaltungssystem) sind zwingende Voraussetzung für Nachvollziehbarkeit – und essentiell für saubere wissenschaftliche Datenanalyse.
- ❏ Versionsnummern sollten möglichst sprechend sein und einem klaren Schema folgen.
- ❏ Die Versionsnummer sollte an *genau einem* Ort abgelegt und nach einem festen Schema verändert werden.
- ❏ Schemata sollten konsequent befolgt werden.

Motivation: Reproduzierbarkeit von Prozessabläufen

Voraussetzung: Konsequenz und Disziplin

Kriterien für Versionsnummern-Schemata

Beispiele für Versionsnummern-Schemata

- ▶ zentraler Aspekt der empirischen Wissenschaften:
Reproduzierbarkeit
- ▶ damit zusammenhängende Aspekte:
Selstdokumentation, Überprüfbarkeit
- ▶ Reproduzierbarkeit geht einen Schritt weiter:
 - Konkrete Version *aller* verwendeten Routinen sollte nachvollziehbar sein.
 - Ergebnisse sollten im Rahmen der Möglichkeiten vollständig identisch reproduzierbar sein.
- ☞ Reproduzierbarkeit zu gewährleisten,
liegt in der Verantwortung des durchführenden Wissenschaftlers.

Voraussetzungen für Reproduzierbarkeit

- ▶ **eindeutige Versionsnummer**
 - zur Identifizierung der Version einer Routine bzw. eines Programms oder einer Bibliothek
 - muss in Metadaten zu jedem Prozessschritt abgelegt sein
- ▶ **Versionsverwaltung**
 - Voraussetzung, um eine bestimmte Version einer Routine wiederherstellen zu können
 - entscheidend für die Überprüfbarkeit der Implementierung
- ▶ **selbstdokumentierende Auswertungsroutinen**
 - Dokumentation aller Parameter
 - inklusive der Versionsnummer der verwendeten Routine
 - sollte automatisiert ablaufen

Warum monolithische Skripte keine Alternative sind

- ▶ nicht wiederverwendbar
 - Wiederverwendbarkeit beruht auf Modularität.
 - Codeblöcke werden meist kopiert (statt modularisiert).
 - Anforderungen wandeln sich häufig.
- ▶ nicht aktuell
 - keine Chance herauszufinden, welche Version aktuell ist
 - Fehler werden selten (nie) in allen Versionen behoben
- ▶ mangelnde Code-Qualität
 - sehr umfangreiche Code-Blöcke: schwer durchschaubar
 - schwer überprüfbar bzw. nachvollziehbar
 - schwer/nicht (automatisiert) testbar
 - (meist) nicht sauber getestet

Motivation: Reproduzierbarkeit von Prozessabläufen

Voraussetzung: Konsequenz und Disziplin

Kriterien für Versionsnummern-Schemata

Beispiele für Versionsnummern-Schemata

Voraussetzungen: Einsicht, Konsequenz und Disziplin

- ▶ Versionsnummern sind wichtig
 - Voraussetzung für Reproduzierbarkeit
 - leisten ggf. entscheidenden Beitrag zur Wissenschaftlichkeit
- ▶ Voraussetzung für die Nutzung
 - Bewusstsein für ihre Bedeutung
 - Konsequenz und Disziplin in der Umsetzung
- ▶ zwei Aspekte der Nutzung
 - Entscheidungen im Vorfeld/zu Projektbeginn
 - festgelegte Arbeitsabläufe

Entscheidungen im Vorfeld/zu Projektbeginn

- ▶ Ablageort der Versionsnummer
 - *genau ein Ort*
 - maschinenlesbar
 - unabhängige Unterprojekte mit eigener Versionsnummer
- ▶ Schema für die Versionsnummern
 - Schema festlegen und konsequent umsetzen
 - Wechsel möglich
- ▶ Umgang mit Entwicklerversionen/Veröffentlichungen
 - Entwicklerversionen nicht produktiv einsetzen
 - Entwicklerversionen klar kennzeichnen
 - klarer Ablauf für Veröffentlichungen (*Releases*)
- 👉 Entscheidungen spätestens mit erstem Produktiveinsatz

Arbeitsabläufe

- ▶ Wann wird inkrementiert?
 - Bei jedem Commit?
 - Nur nach Beendigung eines neuen Features?
 - Nur bei Veröffentlichung (*Release*)?
 - Wie wird das in der Versionsverwaltung abgebildet?
- ▶ Wer inkrementiert?
 - Nur der/die Hauptentwickler?
 - konsistente Handhabung sicherstellen
- ▶ Umgang mit Entwicklerversionen
 - klar kennzeichnen
 - ggf. durch Suffix zur Versionsnummer
- 👉 Abläufe weitestgehend automatisieren

Motivation: Reproduzierbarkeit von Prozessabläufen

Voraussetzung: Konsequenz und Disziplin

Kriterien für Versionsnummern-Schemata

Beispiele für Versionsnummern-Schemata

Allgemeine Anmerkungen

- ▶ unterschiedliche Schemata
 - jedes mit seiner Berechtigung
 - transportieren immer semantische Information
 - Marketing: mitunter Nummern im Hintergrund
- ▶ zentrales Kriterium: Eineindeutigkeit
 - jede veröffentlichte Version mit eigener Nummer
 - Voraussetzung für Nachvollziehbarkeit/Reproduzierbarkeit
- ▶ Lesbarkeit für Mensch und Computer
 - semantische Information
 - Vergleich von Versionsnummern
- ☞ Aufgrund des semantischen Gehalts keine automatisierte Überprüfung der Befolgung eines Schemas möglich



- ▶ **Eineindeutigkeit**
 - zentrales Kriterium
 - *raison d'être*
- ▶ **Lesbarkeit**
 - für Mensch und Computer gleichermaßen
 - Hashes (git, mercurial, ...) eher ungeeignet
- ▶ **Vergleichbarkeit**
 - am Einfachsten für Ganzzahlen und Buchstaben
 - wichtig für Kompatibilitätsinformationen
- ▶ **Stabilität**
 - unreifes Projekt
 - Entwicklerversion zu produktiv eingesetztem Projekt
 - für Produktiveinsatz freigegebene Version (*Release*)

- ▶ Auswirkung der Änderungen
 - insbesondere relevant für Abwärtskompatibilität
 - Umfang der Veränderungen weniger wichtig
- ▶ Aktualität
 - Datum der Veröffentlichung einer Version (*Release*)
 - nur bei Aspekten mit „Verfallsdatum“ relevant

Akademischer Kontext

- ▶ Stabilität meist wichtiger als Aktualität
- ▶ Beispiel für Aspekt mit „Verfallsdatum“
 - Werte für physikalische Konstanten (CODATA)
 - hat Auswirkungen auf die Reproduzierbarkeit

Motivation: Reproduzierbarkeit von Prozessabläufen

Voraussetzung: Konsequenz und Disziplin

Kriterien für Versionsnummern-Schemata

Beispiele für Versionsnummern-Schemata

- ▶ keine wirklichen Standards
 - Schemata meistens (zumindest teilweise) numerisch
 - Mehrgliedrigkeit weit verbreitet, meist dreigliedrig
- ▶ semantische Information
 - automatische Überprüfung deshalb *per se* unmöglich
 - Konvention: Versionsnummern werden größer
- ▶ zwei konkrete Schemata
 - SemVer – Betonung der Abwärtskompatibilität
 - CalVer – Betonung des Veröffentlichungsdatums
- 👉 SemVer-Spezifikation mit vielen guten Hinweisen auf allgemeine Prinzipien sauberer Programmentwicklung

Allgemeines Schema

MAJOR.MINOR.PATCH

Kriterien für die Inkrementierung

- ▶ MAJOR
 - inkompatible Veränderungen an der (öffentlichen) API
 - ▶ MINOR
 - neue Funktionalität in abwärtskompatibler Weise
 - ▶ PATCH
 - ausschließlich Bugfixes
- 👉 zusätzliche Bezeichner für Vorveröffentlichungen etc.

- ▶ Informationen zur Kompatibilität
 - lässt sich in Versionsnummernschema ausdrücken
 - verständlich für Mensch und Computer
- ▶ Bewährte Verfahren
 - keine inkompatiblen Änderungen ohne guten Grund
 - überholte Funktionen erst nach Übergangszeit entfernen
- ▶ Nutzung: wenn, dann konsequent
 - Nutzer darauf hinweisen
 - Einhaltung der Semantik konsequent (manuell) überprüfen
- ▶ Behandlung von Entwicklerversionen
 - über Suffix (z.B. „dev“) möglich
 - Nummer ansonsten identisch mit *nächster* Veröffentlichung

Bedeutung der Abwärtskompatibilität

- ▶ allgemeine Bedeutung der Abwärtskompatibilität
 - Programme oftmals von vielen Bibliotheken abhängig
 - Schema zur automatischen Kompatibilitätserkennung erleichtert die Erkennung von Abhängigkeiten
- ▶ Bedeutung in den Naturwissenschaften
 - Daten langlebig (>10 Jahre)
 - ggf. Routinen zur Konversion von Datenstrukturen
 - Dokumentation der Änderungen (Changelog)
- ▶ parallele Pflege mehrerer Versionen
 - bei vielen großen Projekten üblich
 - durch Nutzung von Versionsverwaltungssystemen grundsätzlich relativ komfortabel möglich

Allgemeines Schema

MAJOR.MINOR.MICRO

- ▶ Datum in MAJOR, mitunter zusätzlich in MINOR codiert
 - meist Jahreszahl (YYYY, YY), evtl. zusätzlich Monat (MM)
 - MICRO für (kompatible) Unterversionen reserviert
- ▶ prominente Beispiele
 - Microsoft Windows (95, 98, 2000)
 - Fortran (66, 77, 90, 95, 2000, ...)
 - Ubuntu (10.04, 10.10, ...)
- ☞ deutlich weniger formalisiert als SemVer



- ▶ **Entwicklerversionen**
 - ungerade Zahl bei MINOR in dreigliedrigem Schema
 - Bsp.: Linux-Kernel über Jahre
- ▶ **0.x-Versionen**
 - häufig bei freien Softwareprojekten
 - andere Semantik als bei SemVer
- ▶ **möglichst kleine Zahlen**
 - gegenwärtiges Schema des Linux-Kernels
 - Inkrement von MAJOR ohne semantische Bedeutung
- ▶ **Ziffernfolge irrationaler Zahlen**
 - T_EX: π
 - METAFONT: Eulersche Zahl



- ❏ Versionsnummern und Versionverwaltung sind zwei voneinander unabhängige Aspekte.
- ❏ Versionsnummern (und ein Versionsverwaltungssystem) sind zwingende Voraussetzung für Nachvollziehbarkeit – und essentiell für saubere wissenschaftliche Datenanalyse.
- ❏ Versionsnummern sollten möglichst sprechend sein und einem klaren Schema folgen.
- ❏ Die Versionsnummer sollte an *genau einem* Ort abgelegt und nach einem festen Schema verändert werden.
- ❏ Schemata sollten konsequent befolgt werden.