21. Konfigurationsmanagement

Prof. Dr. rer. nat. Uwe Aßmann
Lehrstuhl Softwaretechnologie
Fakultät Informatik
Technische Universität Dresden
Version 13-1.0, 15.06.13

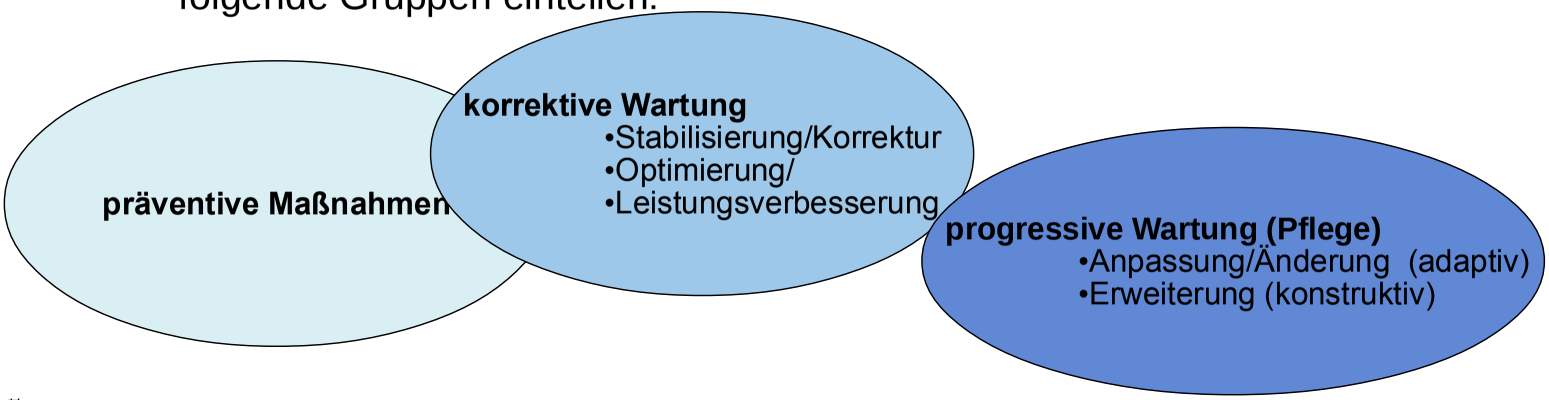
Referenzierte Literatur

- Subversion Portal http://subversion.tigris.org
- debian Dokumentation
- Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato. Version Control with Subversion. Creative Commons Attribution License. http://svnbook.red-bean.com/
- The GIT book http://git-scm.com/book
- Another free GIT book http://it-ebooks.info/book/145/
- Jan Dittberner. Communardo Software GmbH, Dresden.Vortrag “Verteilte Versionsverwaltung mit Git. Vortrag JUG Saxony
  - http://www.jugsaxony.org/wp-content/uploads/2011/07/65180676-Jug-Saxony-Git-2011.pdf

21.1 Software-Wartung

Software-Wartung (1)

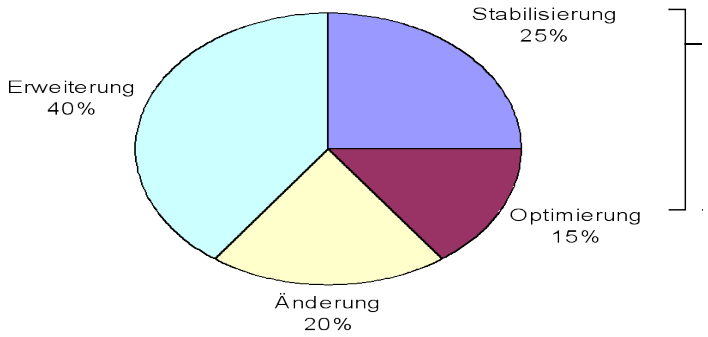
In der Wartungs- & Pflegephase lassen sich durchzuführende Aktivitäten in folgende Gruppen einteilen:



- Wartungsanforderungen bzgl. **Dringlichkeit**:
  - sofort (operativ): schwerwiegende SW-Fehler, die weitere Nutzung in Frage stellen
  - kurzfristig: Code-Korrekturen
  - mittelfristig: Systemerweiterungen (upgrades)
  - langfristig: Restrukturierung (System Redesign)

Anteile der Software-Wartungsarten

=>konstruktiv



=>korrektiv

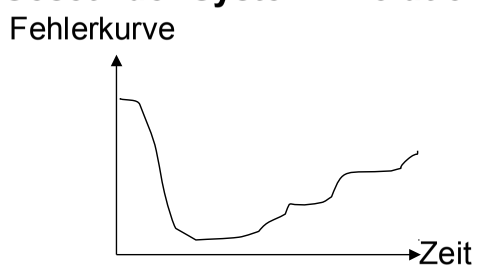
=>adaptiv

präventiver Wartungsaufwand
nicht dargestellt

Software-Wartung: Evolutionsgesetze

- Gesetz der kontinuierlichen Veränderung (Lehmanns 1. Gesetz)
  - Kartoffeltheorem: Ein genutzes System verändert sich, bis die Neustrukturierung oder eine neue Version günstiger ist
- Gesetz der zunehmenden Komplexität (Lehmanns 2. Gesetz)
  - Ohne Gegenmaßnahmen führen Veränderungen zu zunehmender Komplexität.
- Gesetz des Feedbacks (Lehmanns 3. Gesetz)
  - Die Evolution eines Systems wird immer durch einen Rückkopplungsprozess (feedback process) gesteuert.
- Gesetz der System-Evolution
  - Die Wachstumsrate globaler Systemattribute (z. B. LOC, Anzahl Module) im Laufe der Zeit erscheint stochastisch, sie folgt jedoch einem statistisch bestimmten Trend.
- Gesetz der Grenze des Wachstums
  - Ab einem gewissen Grad an Veränderungen eines Systems treten Probleme bzgl. Qualität und Verwendbarkeit auf.

Gesetz der System-Evolution:



[Rombach/Endres]

Verbesserung der Pflege

Def.: **Pflege** beschäftigt sich mit der Lokalisierung und Durchführung von in Betrieb befindlichen Softwareprodukten, wenn **die Art** der gewünschten Änderungen/Erweiterungen **festliegt**.

Charakteristika von Pflegeaktivitäten (adaptiv, konstruktive Wartung):

- Ausgangsbasis ist konsistentes Produkt, in das gezielt - unter Beibehaltung der Konsistenz – Änderungen, Refaktorierungen (refactorings) und Erweiterungen eingebracht werden
- Änderungen/Erweiterungen sind in allen Teilprodukten (Produkt-Definition, -Entwurf, -Implementierung, Dokumentationsbausteine) durchzuführen.
- Pflege bzw. Weiterentwicklung werden erleichtert, wenn das Software-Produkt die Qualitätsmerkmale nach DIN ISO 9126 **Änderbarkeit** und **Übertragbarkeit** besitzt.

Quelle: [Balzert2: S. 1094]

Verbesserung der Wartung

Wartung beschäftigt sich mit der Lokalisierung und Behebung von Fehlerursachen bei in Betrieb befindlichen Softwareprodukten, wenn die **Fehlerwirkung bekannt** ist.

Charakteristika von Wartungsaktivitäten (korrektive Wartung):

- Nicht planbar: Ereignisgesteuert, nicht vorhersehbar, schwer kontrollierbar
- Ausgangsbasis ist ein fehlerhaftes bzw. inkonsistentes Produkt
- Abweichungen zwischen Teilprodukten sind zu lokalisieren und zu beheben
- Die Korrektur einzelner Fehler hat nur begrenzte Auswirkungen auf das Gesamtprodukt, d.h. Bandbreite der Änderungen/Erweiterungen ist relativ gering
- Fehlerkorrekturen konzentrieren sich im Allgemeinen auf die Implementierung.

Wartungsaktivitäten werden erleichtert, wenn das Software-Produkt die Qualitätsmerkmale nach DIN ISO 9126 **Zuverlässigkeit** und **Effizienz** besitzt.

Quelle: [Balzert2]

21.2 Konfigurationsmanagement

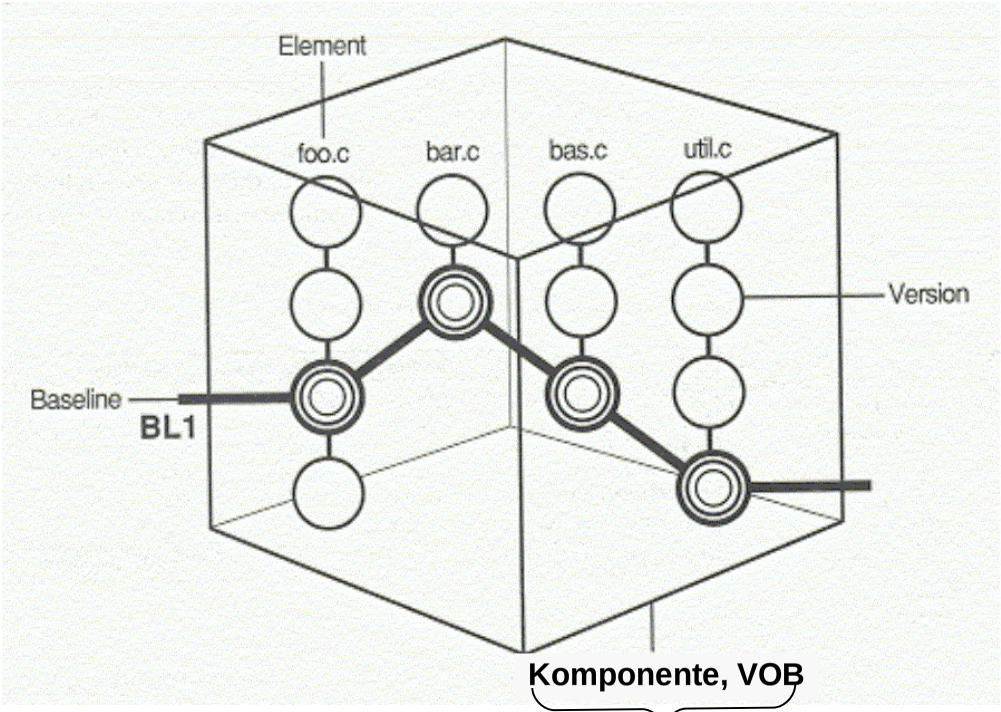
21.2.1 Konfigurationsmanagement

Eine **SW-Konfiguration** ist die **Gesamtheit der Artefakte** (Menge von Komponenten in zeitlichen Versionen und funktionalen Varianten), die zu einem **bestimmten Zeitpunkt des Life Cycle** in ihrer Wirkungsweise und ihren Schnittstellen aufeinander abgestimmt sind.

Unter **SW-Konfigurationsmanagement** versteht man die Gesamtheit der Methoden, Werkzeuge und Hilfsmittel, die die Entwicklung, Wartung und Pflege eines Softwareprodukts durch die Konfiguration geeigneter Varianten in passenden Revisionen unterstützt.

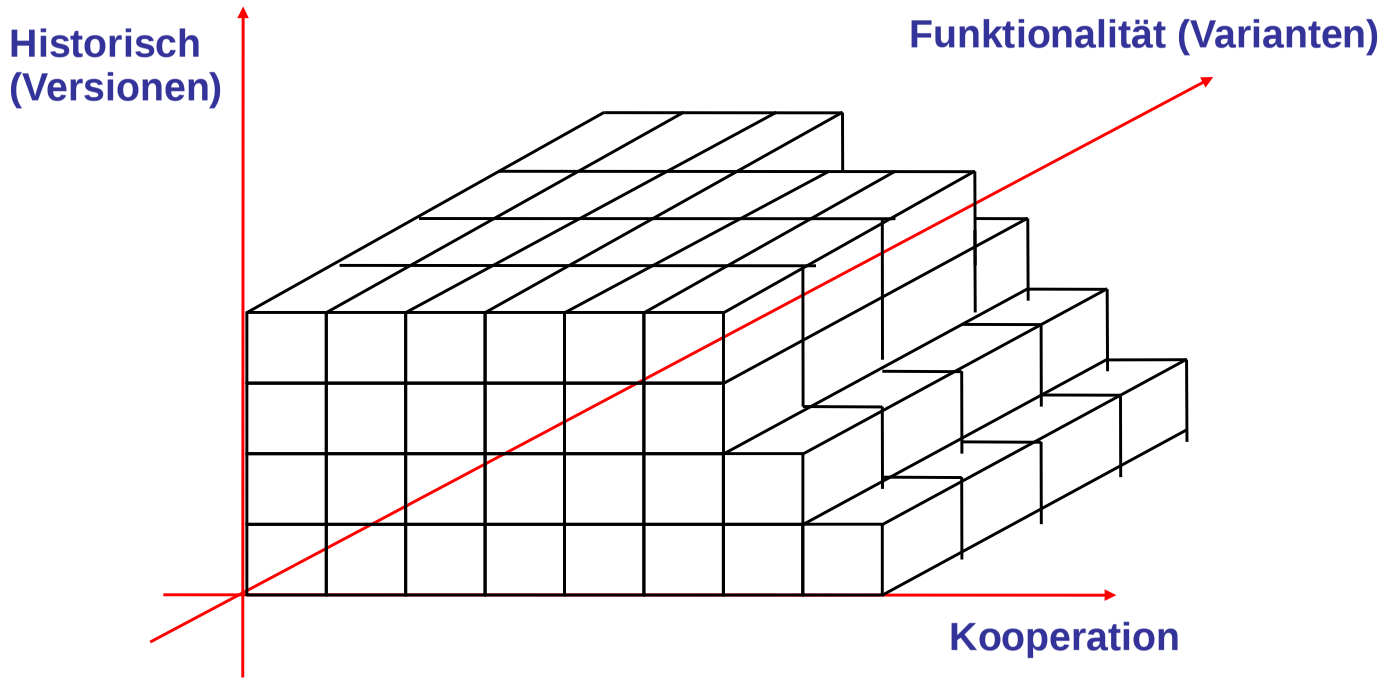
- **Ziele**
  - Bestimmung der **Artefakte** (Komponenten, Module, Pakete, Dateien, Datenbanken), die eine Konfiguration bilden; Ableitung aus der PBS
  - Konsistenzsicherung, Sichtbarkeit, Verfolgbarkeit, Kontrollierbarkeit von Produkten
- **Langfristige Ziele über Jahre hinweg**
  - Prüfung, ob Konsistenz der SW-Konfiguration erhalten bleibt
  - Sicherstellung, dass jederzeit, jahrzehntelang auf vorherige Konfigurationen zurückgegriffen werden kann
  - Erfassung und Nachweis aller Änderungen; Überwachung der (Produkt-)Konfigurationen während des Lebenszyklus
  - Auslieferungskontrolle

Darstellung einer SW-Konfiguration



Quelle: Thomas, D., Hunt, A.: Versionsverwaltung mit CVS (Reihe Pragmatisch Programmieren); Hanser 2004

Konfigurationsuniversum mit Versions-, Varianten- und Zusammenarbeitsdimensionen



Prof. Uwe Aßmann, Softwaremanagement

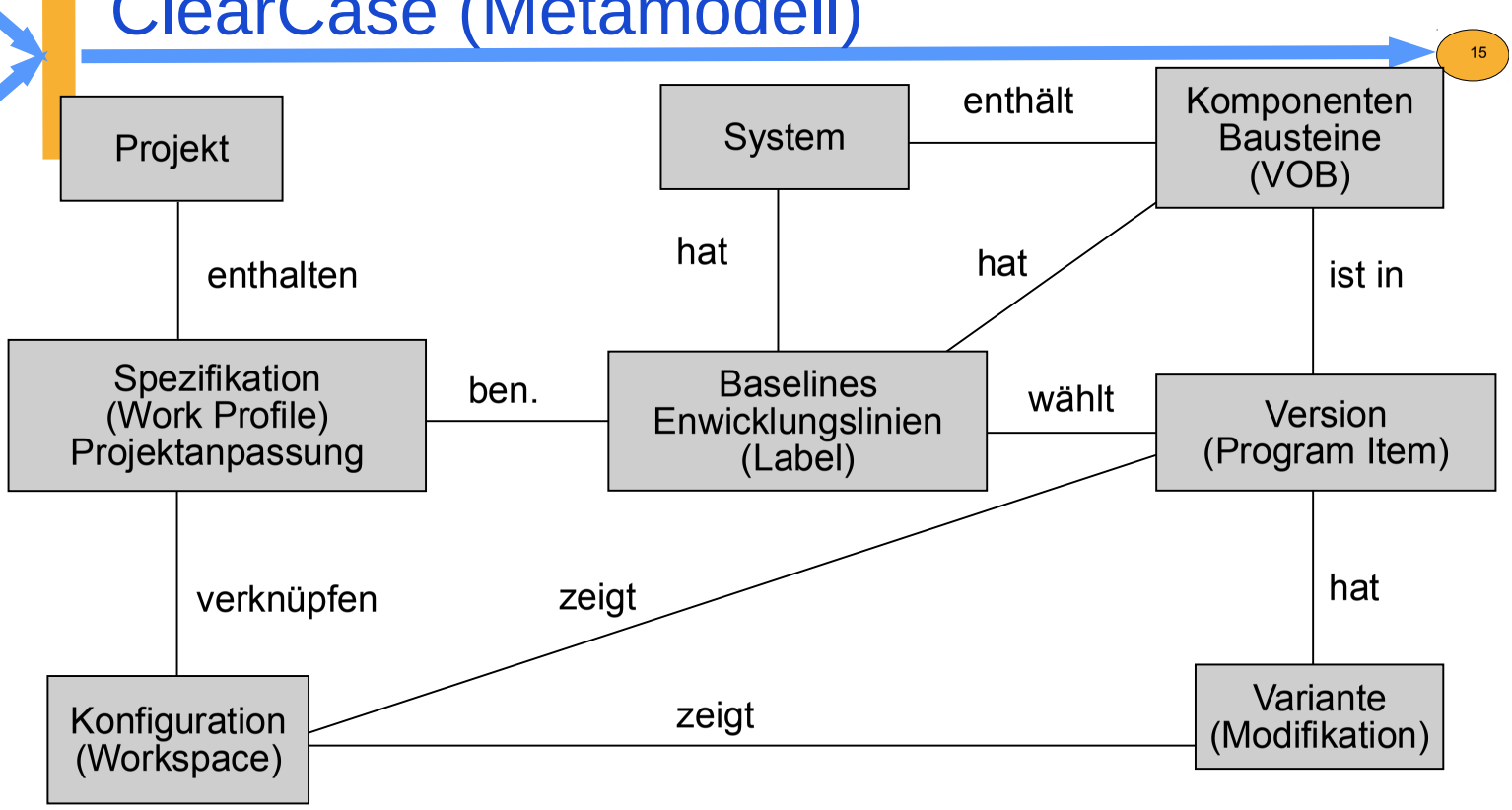
Gegenstände des KM (Komponenten, Artefakte)

- Der Gegenstand des KM ist das gesamte Softwaresystem mit seinen Komponenten und Artefakten (Produkt- und Artefaktstruktur)
  - **Spezifikation:** Daten und Requirements
  - **Entwurf:** formale und informale Dokumente
  - **Programme:** Code-Teile, Datenbeschreibungen, Prozeduren
  - **Testfälle und -umgebung:** Dokumente für Testdaten, Testumgebung
  - **Integrationskonzept:** alle Dokumente für Integration und Einführung (auch Benutzerdokumentation)
- Verwaltung der Komponenten in der **Produkt-(Artefakt-)bibliothek**

Begriffe des Konfigurationsmanagement

- **Konfiguration:** benannte und formal freigegebene Menge von Komponenten für eine Abarbeitung (Editierung, Compilation, Link oder GO-Lauf)
- **Version (Revision)**: Instanz bzw. Entwicklungsstand einer Komponente; je Editierung erhöht sich die Versions-Nr. um eins;
  - Identifikation: (Komponenten-Name, Versions-Nr., Variantenkennzeichnung)
- **Komponente: =** Software-Artefakt, Baustein
- **VOB** *Versioned Object Base* sind die Repräsentation von Entitäten (Objekte mit Funktionalität) des Systems in der Mini-Welt des Anwendungsprojektes
- **Variante:** weitere Untergliederungskoordinate für Änderungsstände, Modifikationen, Revisionen innerhalb einer Version der Komponente
- **Release:** eine für die Nutzung freigegebene Version (konsistente Menge von VOB)
- **Baseline (Stückliste):** gesichertes Zwischenergebnis (Entwicklungsstand), das aus Menge freigegebener Versionen einer Komponetenmenge besteht.
- **Spezifikation:** angepasste Konfigurationen für ein Projekt, freigegeben für ein spezifi- sches Anwendungsprofil oder einen Auftraggeber
- **Checkout:** Lesen eines Quelltextes der Version n für Editierung
- **Checkin:** Schreiben eines Quelltextes der Version n+1 nach Editierung
- **Re-Konf gurierung:** Aktualisierung einer existierenden Konfiguration bei neuen Versionen oder Varianten mit dem Ziel der Konsistenz

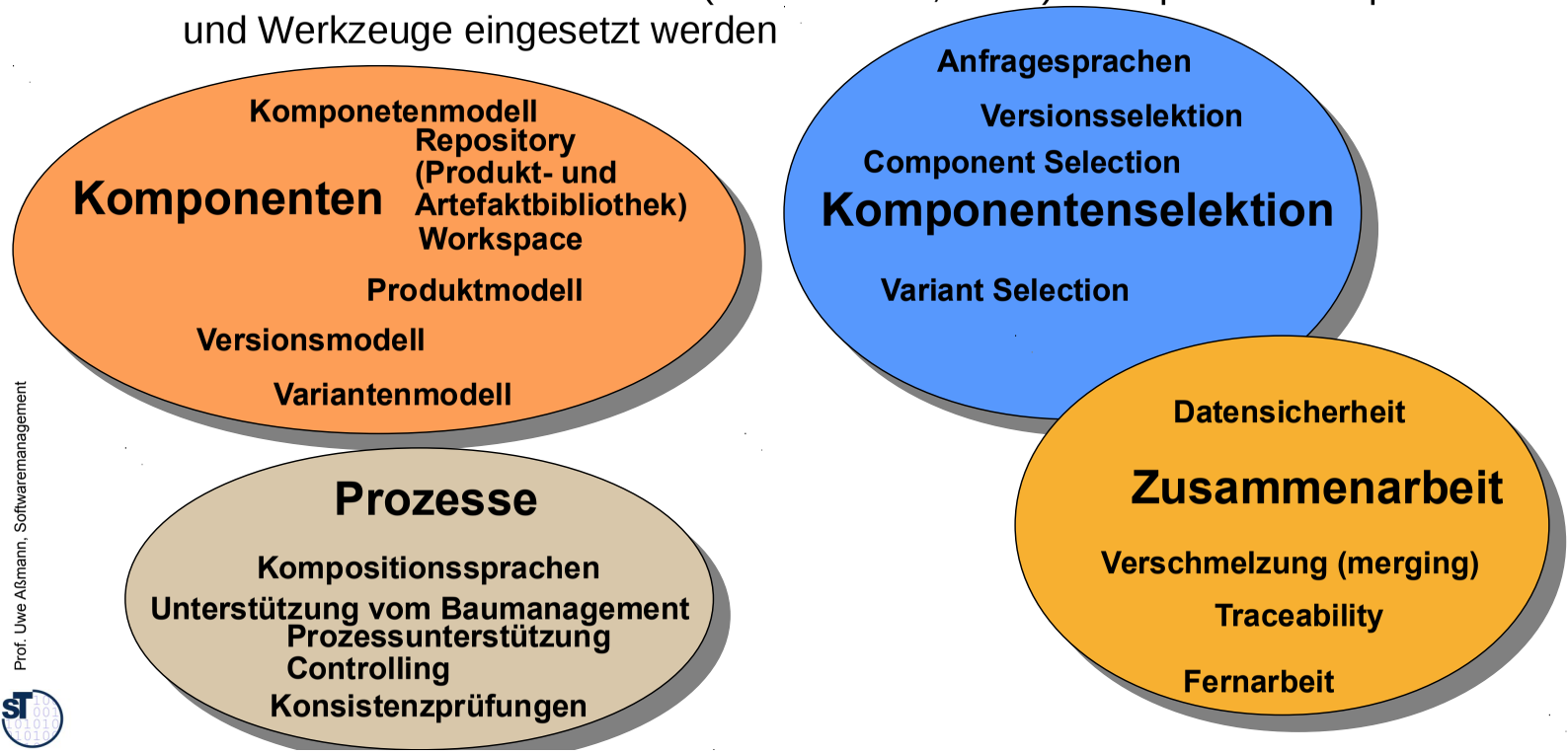
Bsp.: Begriffe des KM im Werkzeug ClearCase (Metamodell)



Quelle: nach ClearCase; http://www.rational.com/f eld/se/products/clearcase/implement_docs/

KM unterstützt Entwicklungs und QS-prozesse

- Für viele der Begriffe des KM müssen firmen- und projektspezifische Metamodelle definiert werden (Kurse CBSE, SEW) und spezifische Sprachen und Werkzeuge eingesetzt werden



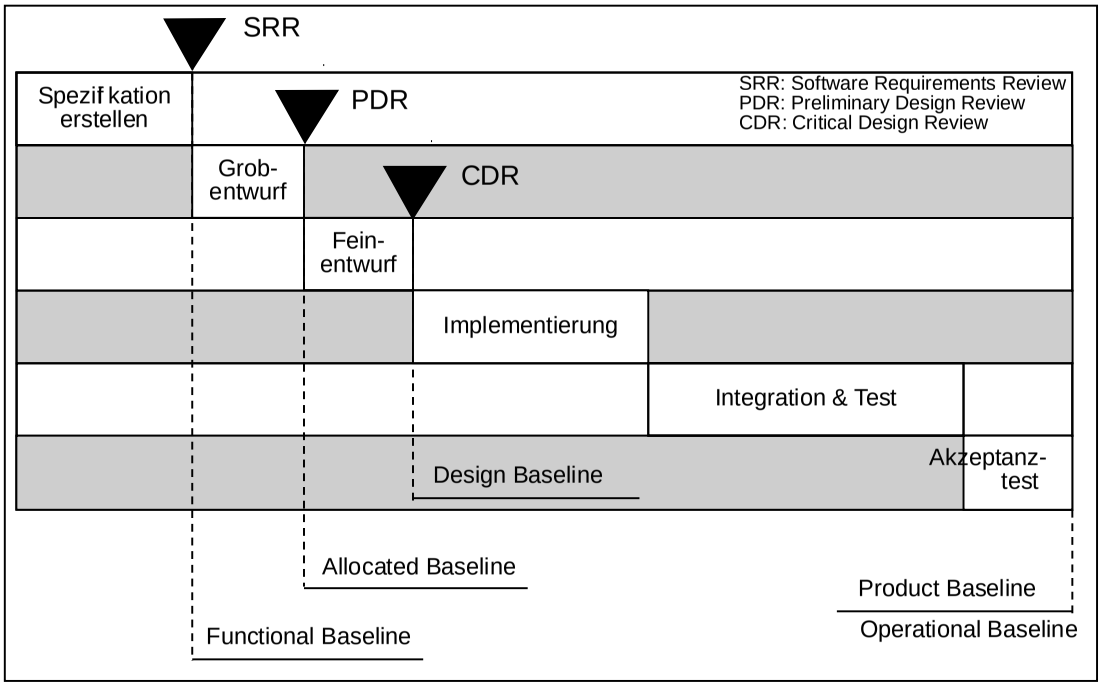
21.2.2 Versionsverwaltung

Aufgaben der Versionsverwaltung

Der Versionsverwaltung obliegt die **persistente** Verwaltung der Komponenten.

- Identifizierung von Versionen: Komponentenname, Versions-Nr., Variante
  - Komponentennamen können Pfadnamen sein, oder Surrogates, oder URL
- Festlegung aller zu verwaltender Bestandteile, die zu einer Komponente gehören, wie
  - Analyse-Spezifikation,
  - Entwurfsspezifikation,
  - Code der Komponente,
  - Testspezifikation, Dokumentation, ...
- Bestimmen der Namenskonventionen und Bezeichnung der Relationen zwischen den Komponenten
- Abspeichern verschiedener Versionen einer Komponente einschließlich ihrer Wiederauffindungspfade
- Bereitstellen beliebiger abgelegter Versionen
- Dokumentieren von Änderungen
- Festlegen und Kontrollieren von Zugriffsrechten
- Verwalten des Komponenten-Repositories

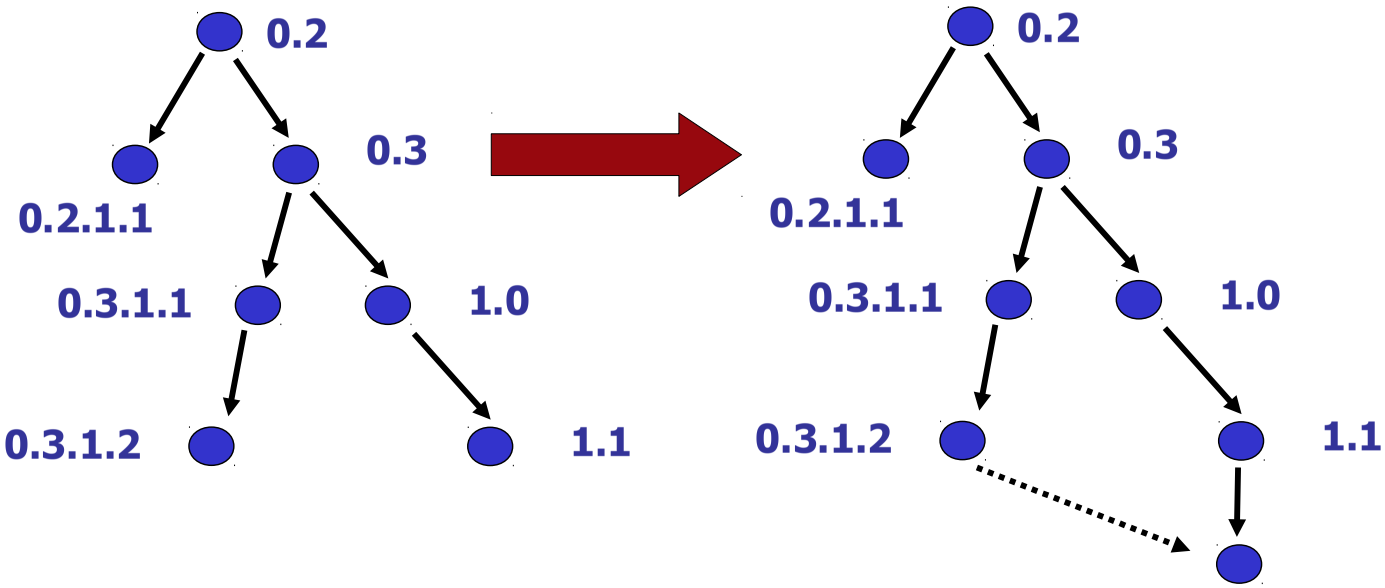
Baselines in der Artefaktstruktur



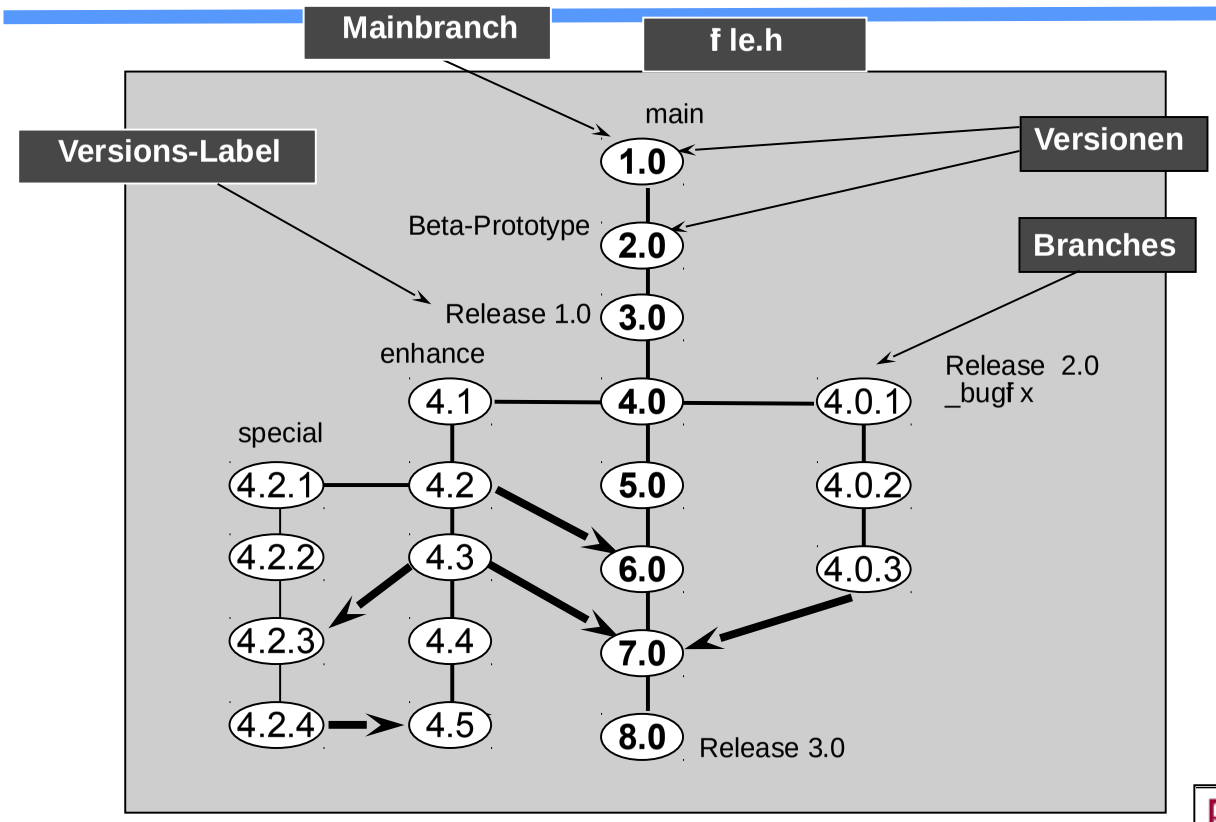
Quelle: Thaller, G., E.: ISO 9001 – Software-Entwicklung in der Praxis(2. aktual. Auf age); Heinz Heise Verlag 2000, S. 157

Die Versionsdimension

- Verzweigungsbaum unbestimmter Tiefe und Breite
- Jede Modifikation generiert ein Kind oder einen Zweig (branch)
- Verschmelzen ist möglich (branch merge)



Bsp: Versions-Baum einer Komponente in ClearCase

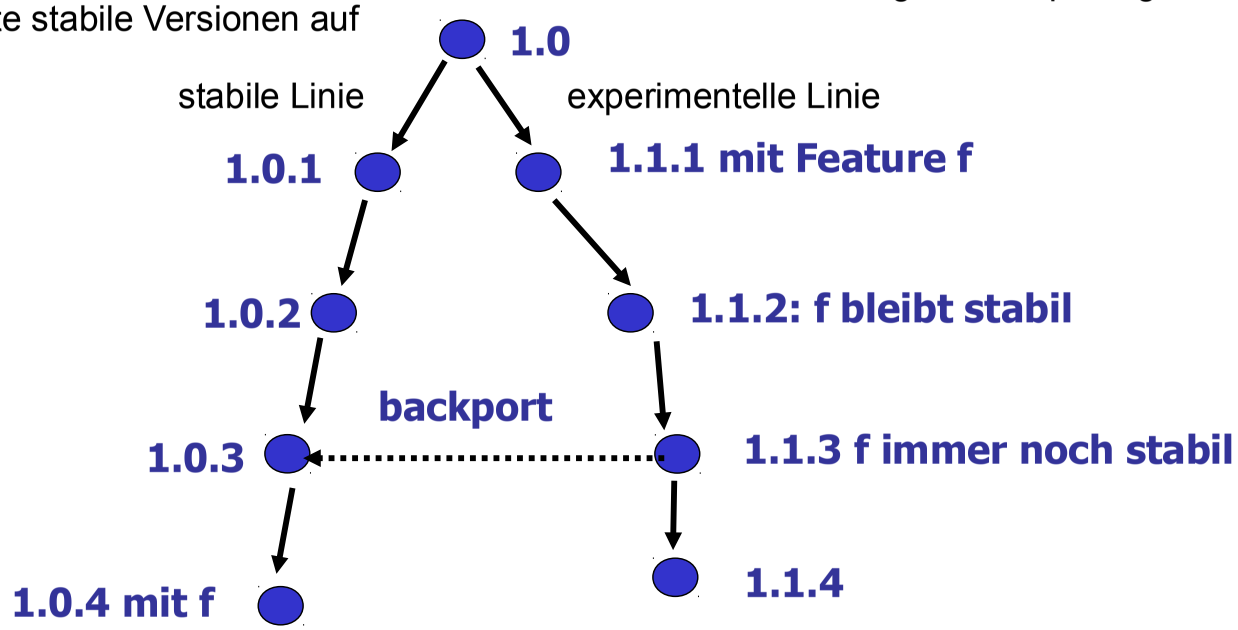


RATIONAL SOFTWARE CORPORATION

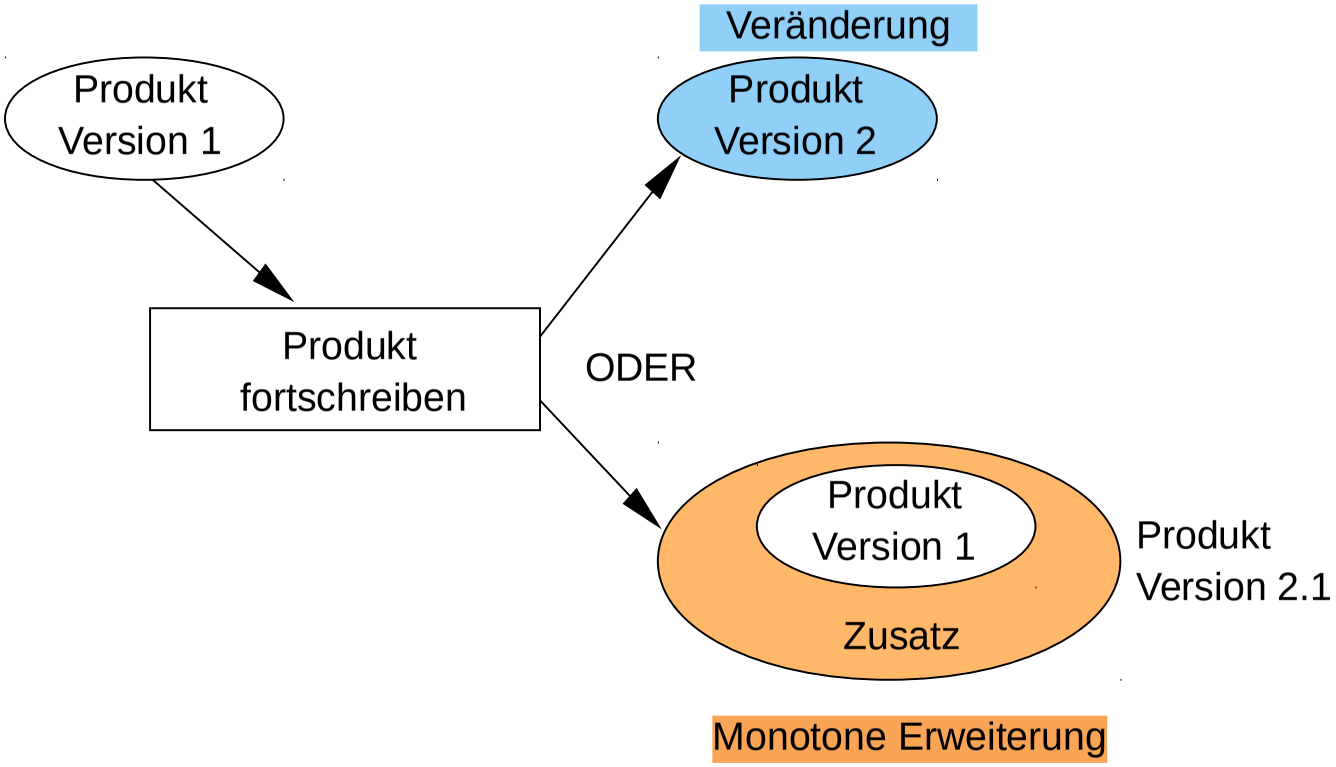
Quelle: nach ClearCase; http://www.rational.com/field/se/products/clearcase/implement_docs/

Produktfortschreibung: Backporting (Rückintegration von neuem Branch in stabilen Branch)

- Stabile Versionen erhalten oft eine „gerade Linie“ von Versionsnummern, experimentelle eine ungerade Linie
- Erweist sich ein Feature auf einer experimentellen Linie als stabil, dann es in die stabile Linie **rückintegriert werden (backporting)**
- Vorteil: stabile Linien erlauben es, kommerzielle Nutzer zu befriedigen, Backporting frischt alte stabile Versionen auf

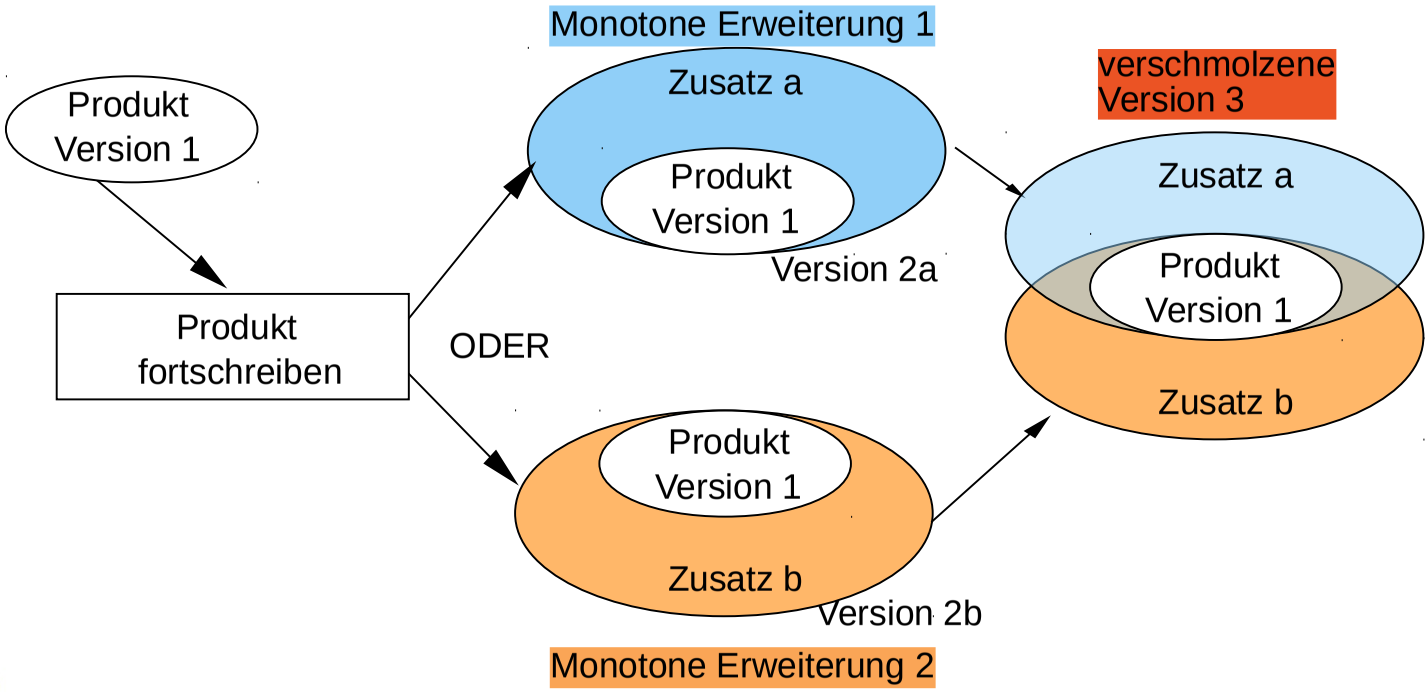


Produktfortschreibung: Monotone Erweiterung



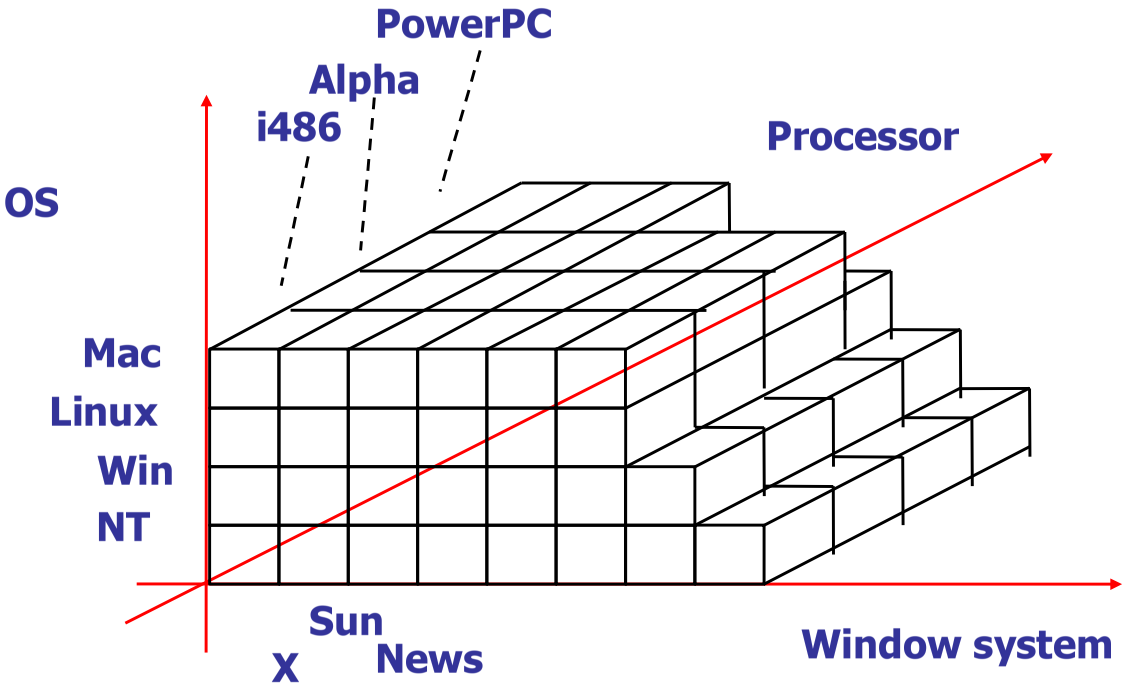
Produktfortschreibung: Verschmelzung von disjunkten monotonen Erweiterungen

- Disjunkte Erweiterungen ergänzen sich modular und können einfach verschmolzen werden



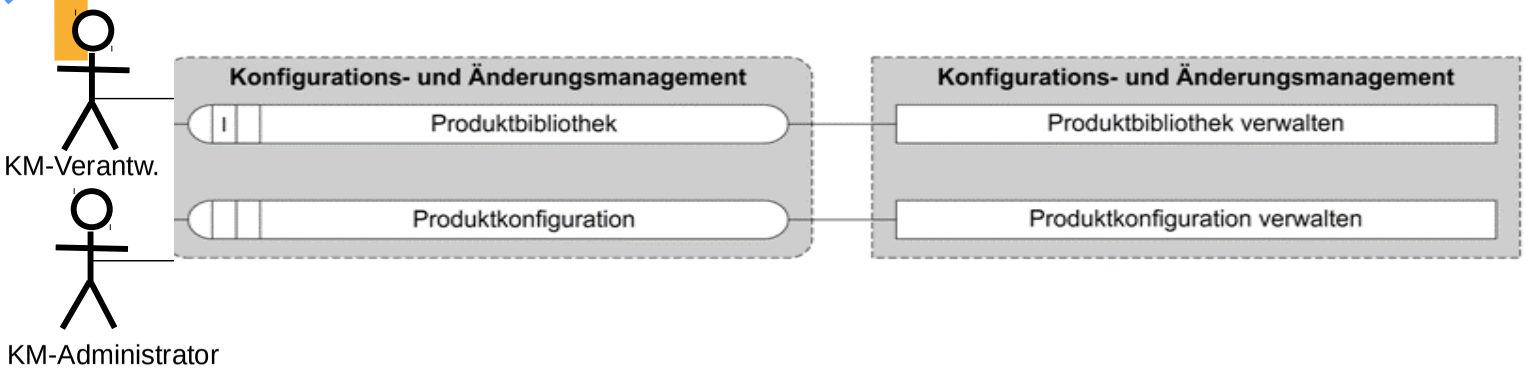
Variantendimension: Ein Variantenuniversum

- n-dimensionaler Raum mit k Werten pro Parameter



21.2.3 Vorgehensmodelle des KM

Vorgehensbaustein Konfigurationsmanagement im V-Modell XT



Notwendige Produkte für das KM sind:

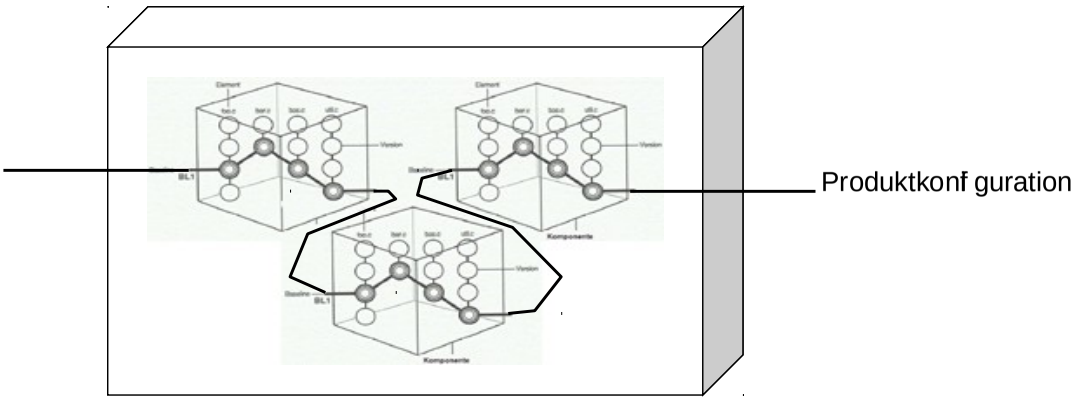
- **Produktbibliothek** (und Artefaktbibliothek) zur Aufbewahrung aller Produkte und ihrer Bestandteile
- **Produktkonfiguration** zur Verwaltung zusammengehöriger Produkte und Hilfsmittel, wie HW-Testumgebung, Software-Entwicklungs-Umgebung in einem bestimmten Bearbeitungszustand bzw. in einer bestimmten Version

Sie werden in den zugehörigen Aktivitäten des V-Modells XT bearbeitet.

Aufgaben des Konfigurationsmanagements laut V-Modell XT, Vorgehensbaustein KM

- Korrekte Ermittlung, Verwaltung und Sicherstellung des Konfigurationsstandes
- Aufzeichnen des Änderungszustandes der physikalischen und funktionellen Charakteristika der Produkte
- Initialisieren, Verwalten und Archivieren aller Produkte und Produktkonfigurationen, so dass alle Änderungen an Produkten nachvollziehbar sind
- Jederzeit eindeutige Identifizierung aller Produkte
  - Sicherstellung einer nachvollziehbaren Fortschreibung von Produkt-konfigurationen während des Entwicklungsprozesse als auch während der Nutzung
- Vorgabe definierter Aufsetzpunkte für weitere Prozessschritte

Zusammenhang Produktbibliothek/Produktkonfiguration



Produktbibliothek: Menge aller Komponenten und deren Versionen etc
Produktkonfiguration: Menge von (Produkt-)versionen

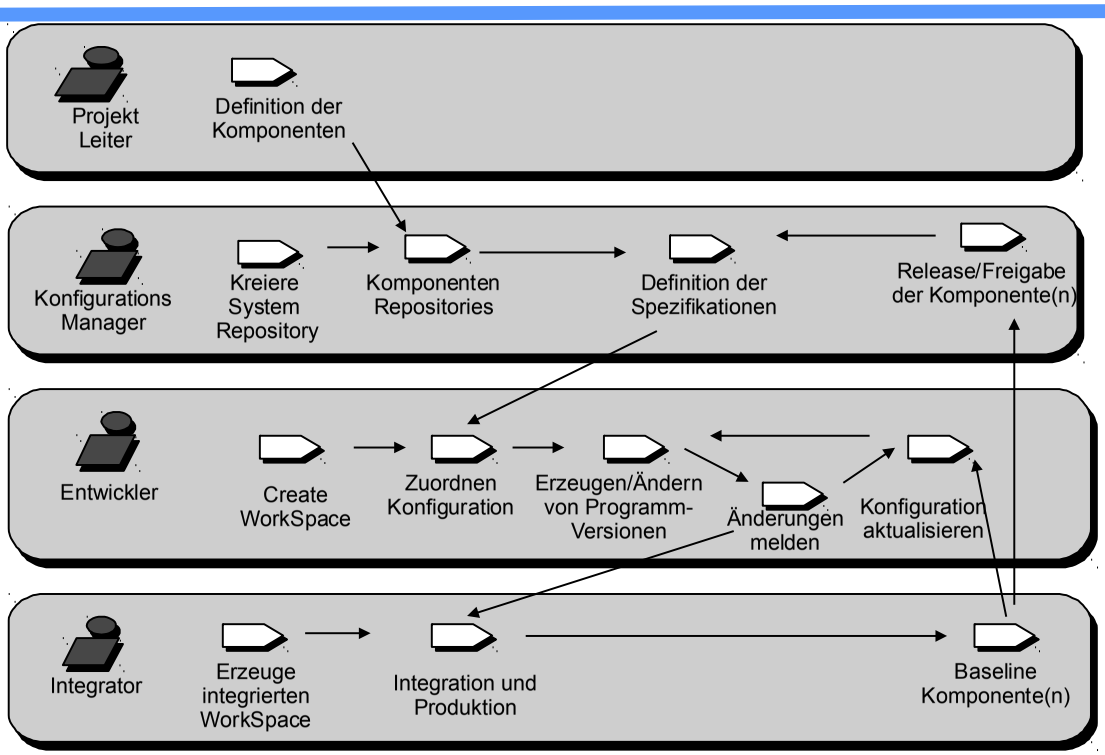
Aktivität Produktbibliothek verwalten

- Einrichten des KM in folgenden Schritten (= KM-Planung):
  - Initialisierung und Verwaltung der zugehörigen Produkte in der Produktbibliothek
  - Einrichten festgelegter Konventionen nach Projekthandbuch (oder PH)
  - Beachtung des Sicherheitskonzeptes wie Datenschutz, Kontrollmechanismen usw.
  - Zugriffsrechte bearbeitungszustands- und rollenbezogen einrichten und verwalten
- Produkte initialisieren und einrichten z. B. durch
  - Aufnahme neue Produkte einschließlich Bearbeitungszustand
  - Aufnahme bereits existierender Produkte durch Versionsfortschreibung
  - Sicherstellung der Rückverfolgung von Produkten
  - Pflege der Identifikatoren als wichtigste Metadaten zur Produktkennzeichnung
- Produkte sichern und archivieren zu regelmässigen oder durch Meilensteine festgelegten Ereignissen
  - KM-Auswertungen erstellen
  - Statusliste der Produkte
  - Statusliste mit Aussagen zur Bestimmung der Konfiguration

Aktivität Produktkonfiguration verwalten

- Produktkonfigurationen dienen der Identifikation inhaltlich zusammengehöri-ger Produkte, also von Produkten, die über Produktabhängigkeiten miteinan-der in Beziehung stehen
  - Konfiguration initialisieren und Fortschreiben mit folgenden Inhalten
  - Identifikatoren zur Namensgebung, Bearbeitungszustand oder Version
  - Aufbau von Referenzhierarchien
- Regeln zur Fortschreibung von Produktkonfigurationen
- Pflege von Prozeduren zur automatisierten Zusammenstellung gewünschter Konfigurationen
  - Auslieferungsinformation dokumentieren mit folgenden typischen Fragestellungen
  - Welche Konfiguration wurde ausgeliefert
  - Wann und an wen wurde ausgeliefert
  - Über welches Speicher- beziehungsweise Übertragungsmedium ist dies erfolgt
  - Zu welchem Zweck erfolgte die Auslieferung

Der KM-Prozeß als RUP-Workflow



Quelle: nach ClearCase; http://www.rational.com/f eld/se/products/clearcase/implement_docs/

Prof. Uwe Aßmann, Softwaremanagement

KM bereits im Lasten-/Pflichtenheft planen

(Gliederung nach Entwurf VDI/VDE 3694)

8. Anforderungen an die Projektabwicklung

8.1.Projektorganisation

8.2.Projektdurchführung

8.3.Konfigurationsmanagement

- Vorgaben für die Gliederung
  - der Dokumentation
  - der Software
  - der Hardware
- Konfigurationsmaßnahmen
  - Änderungsdienst
  - Fehlerverfolgung
  - Versionsverwaltung
  - periodische Datensicherung
  - Katastrophenschutzmaßnahmen
  - Verwaltung sensitiver Daten
  - Führung der Projekthistorie

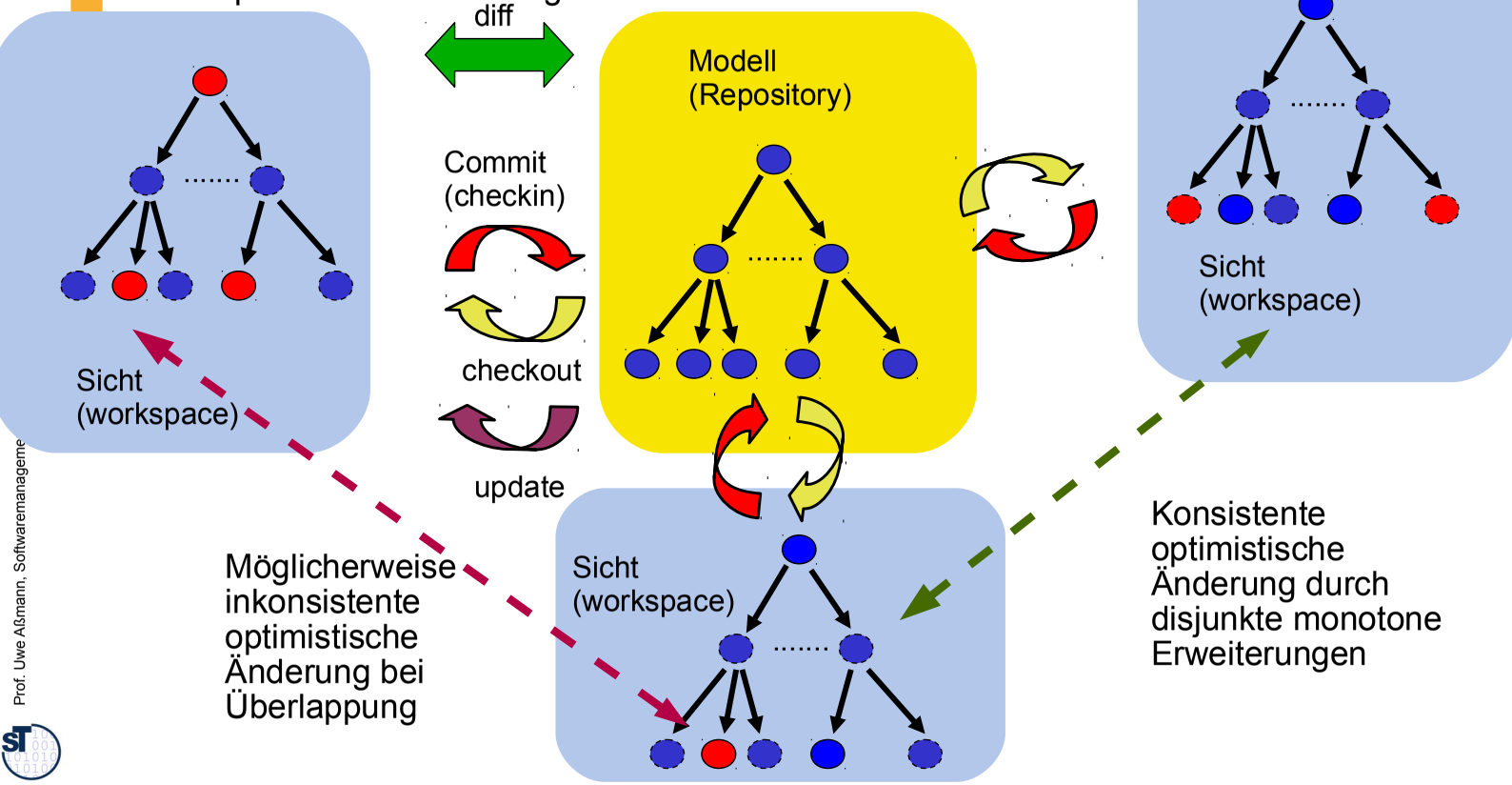
21.3 Dateibaumbasierte Konfigurationsmanagement-Werkzeuge

KM-Werkzeuge

Produktbibliothek			URL
Dateibaum-basiert	cvs	OSS	http://www.cvshome.org
	git	OSS	Linus Thorvalds, www.git-scm.com
Datenbank-basiert	ClearCase	IBM/Rational	http://www.rational.com/products
	Visual SourceSafe	Microsoft	http://www.eu.microsoft.com/germany/produkte
beides	subversion	OSS	http://subversion.tigris.org
andere	Synergy	IBM Telelogic	http://www.telelogic.com/product/synergy
	mercurial	Selenic, OSS	http://mercurial.selenic.com/
	InStep	microTOOL GmbH, Berlin	http://www.microTOOL.de

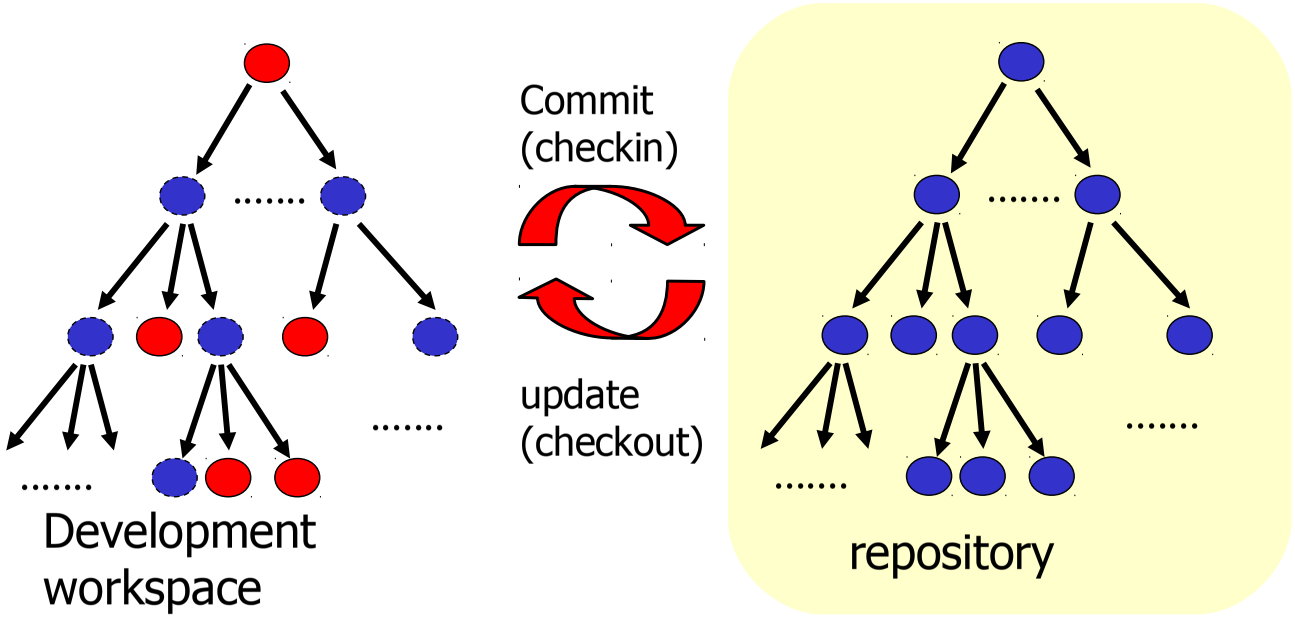
Zusammenarbeitsdimension: Views and Models

- Bei paralleler Bearbeitung werden Sichten inkonsistent



Einfache Sichten in cvs und svn

- Nur Tiefe 1 möglich



21.3.1 Subversion (svn) – Ein verteiltes Konfigurationsmanagementsystem

Arbeitsweise von Subversion

(1) **Einrichten der Repositories** zusammen mit den Server- und Clientkomponenten
(2) **Festlegung der Benutzer** und der Zugriffsrechte
(3) **Konf guration des Client** mit vordef nierten Properties
(4) **Start eines Subversions-Servers** zum Aufruf vom Subversions-Client *svn* aus

(5) **Festlegung von Projektstruktur** und **Konf guration** nach den Konf gurationselementen
  – Directories, Quellkomponenten, abgeleitete Komponenten
(6) **Check-out aus dem Repository** in lokalen Arbeitsbereich und Durchführung von Änderungen an Dateien
(7) **Änderungen überprüfen** und mit Teammitgliedern abstimmen vor *Changeset* Rückspeichern ins Repository
(8) **Rückschreiben der Änderungen** ins Repository mit *commit*-Befehl

Quelle: Popp, G.: Konf gurationsmanagement mit *Subversion*, *Ant* und *Maven*; dpunkt.verlag 2006

Hilfe

- svn –help
- svn <command> --help
- svnadmin –help
- svnadmin create --help

Prof. Uwe Aßmann, Softwaremanagement

Initialisierung eines Repository aus einem Dateibaum heraus

- `svnadmin create /home/ua1/svn/Ontologies1`
- `ls /home/ua1/svn/Ontologies1`
  - `README.txt  conf  dav  db  format  hooks  locks`
- `cat /home/ua1/svn/Ontologies1 format`
  - `3`
- Auschecken eines Repositories:
- `svn checkout . file:///home/ua1/svn/Ontologies1`
  - `creates a metadata subdirectory .svn`
- `ls .svn/`
  - `README.txt  entries  prop-base/  text-base/  wcprops/ empty-file format  props/  tmp/`

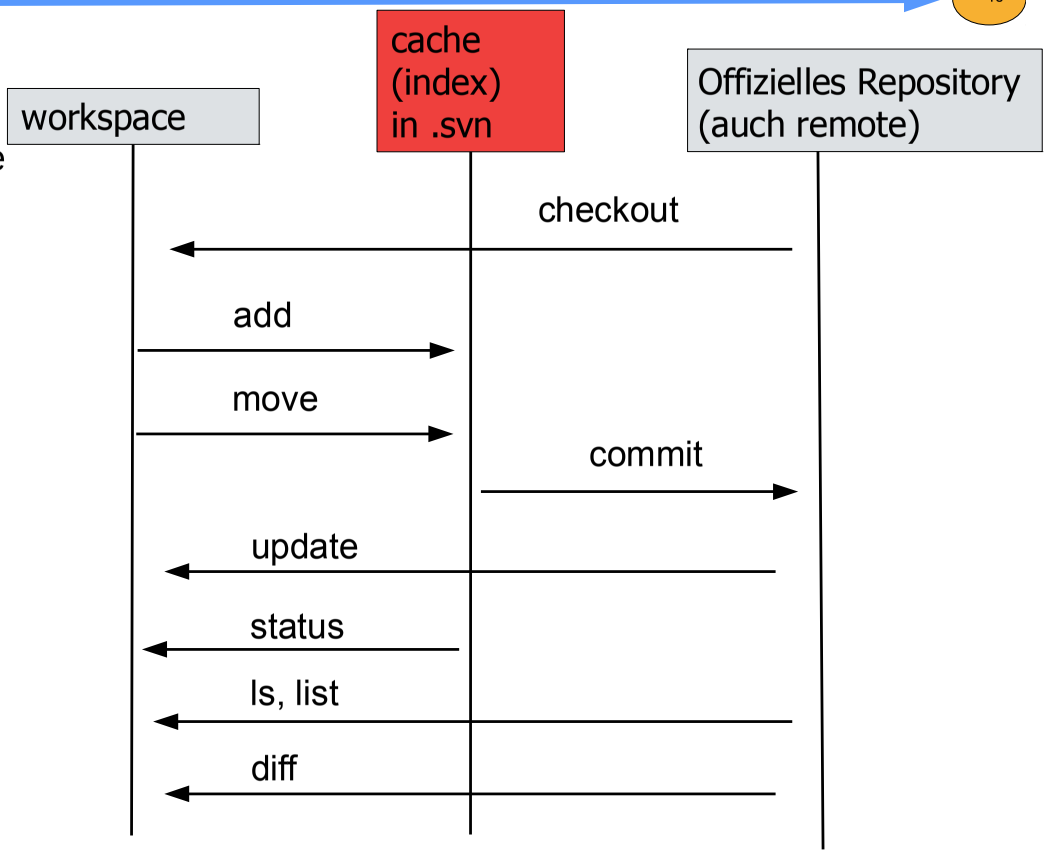
Arbeiten; Commit und Rollback

- svn log: Historie eines Files
- svn diff: vergleicht mit jungfräulichen Kopien in .svn
- arbeitet inkrementell auch auf binären Dateien
- svn cat: Ausgeben eines Files auf Terminal
- svn list (svn ls): Listing aller versionierten Files
- svn status: lokale Veränderungen

- Zum sauberen Arbeiten in parallelen Kontexten braucht man ein Transaktionskonzept mit ACID Merkmalen (Atomizität, Consistency, Integrität, Durability)
- svn commit
- svn revert

Ketten von Sichten in svn

- Der svn Cache erlaubt es,
  - die BASE Version, die in .svn gespeichert wird, zu vergleichen
  - “spät” zurück zu schreiben
  - direkte commits über das Netzwerk zu vermeiden
    - erlaubt vom Netz abgetrenntes Arbeiten
    - Commits werden in den Cache geschrieben
    - alle Kommandos funktionieren trotzdem



Addieren von Dateien zum Repository

- `svn add models-mda.lyx brainstorm.lyx`
- `svn import /home/ua1/tmp/Ontologies-1 file:///home/ua1/svn/Ontologies1 -m "importing all files"`
- Danach Commit
- Danach neues Auschecken möglich
  - `cd /home/ua1/tmp/NewDir`
  - `svn checkout file:///home/ua1/svn/Ontologies1 .`
- svn copy <subtree>
- svn move <subtree>
- svn delete <subtree>

Konflikte bei optimistischen Transaktionen

- Falls andere parallel zurückschreiben (committen), kann man die eigene, nun inkonsistent gewordene Arbeitskopie (Sicht) auf den neuesten Stand bringen
  - svn update
- Das ist mehr als was Datenbanken tun!
- Aktualisierungen können automatisch erfolgen
- oder schiefgehen (Konflikte)

Meldungen des Aktualisierungsalgorithmusses

- alles gut gegangen (disjunkte monotone Erweiterungen von anderen Entwicklern)
  - U foo (updated)
    - file wieder konsistent
  - A file (added)
    - alles gutgegangen, file ist neu ins Repository aufgenommen worden
  - D file (deleted)
    - file wurde gelöscht.
  - R file (replaced)
    - file wurde ersetzt
  - G file (managed)
    - Es gab zwar eigene Modifikationen von file, aber die waren harmlos
- fehlgeschlagen: (überlappende Erweiterungen)
  - C file (conflict)
    - Konflikt konnte nicht automatisch gelöst werden (überlappende Änderungen). Manueller Eingriff nötig.
- svn status meldet den Zustand für alle Files und Dirs in der Sicht

Konfliktauflösung

- Aufruf eines Editors, mit dem der Konflikt interaktiv beseitigt wird (interaktives Merge)
- mit 3 speziellen Dateien
  - file.mine
  - file.<old-revision>: the BASE revision from which file.mine was copied
  - file.<new-revision>: the HEAD revision which was committed in parallel
  - Kopiere eines der Files auf file und sage dann
    - svn resolve file

Benutzung übers Web

- Gesteuert durch die URL
  - file://
  - http:// (webdav)
    - apache module mod_dav_svn
  - https:// (encrypted webdav)
  - svn://
    - svnserver läuft auf dem Server, lauscht an Port 3690
  - svn+ssh://

Merge über mehrere Views

- svn merge ist eine Kombination von diff und patch
  - es führt zunächst ein svn diff durch und wendet dann die patches an
  - Da svn eine lineare Versionsnummerung über Sichten und Sichtenkopien durchführt, können direkt Sichten miteinander verglichen und verschmolzen werden
- Beispiele:
- `svn merge -r 4:7 file:///home/ua1/svn/Ontologies-1`
  - Ermittelt parallele Änderungen in Hauptsicht und Sichtenkopie
- `svn merge –dry-run`
  - Zeigt, was getan werden soll

Repositories (Produkt- und Artefaktbibliotheken)

- Datenbankbasiert mit Berkeley-DB
  - nicht portabel von Maschine zu Maschine
  - nicht auf AFS, NFS
  - svn create –fs-type bdb <path>
- Dateibaumbasiert (FSFS im Filesystem)
  - portabel
  - svn create –fs-type fsfs <path>

Kritik und Lob

> **“The original design team settled on some simple goals. They didn't want to break new ground in version control methodology,**
>
> **they just wanted to fix CVS. “**

- svn hat noch immer
  - kein Variantenmanagement
  - keine Komponentenselektion
  - keine Unterstützung von automatischen Builds
- na ja, es kann schon wesentlich mehr als cvs
  - Branchmanagement (Ketten von Sichten)
  - long runs
  - ACID
  - Web

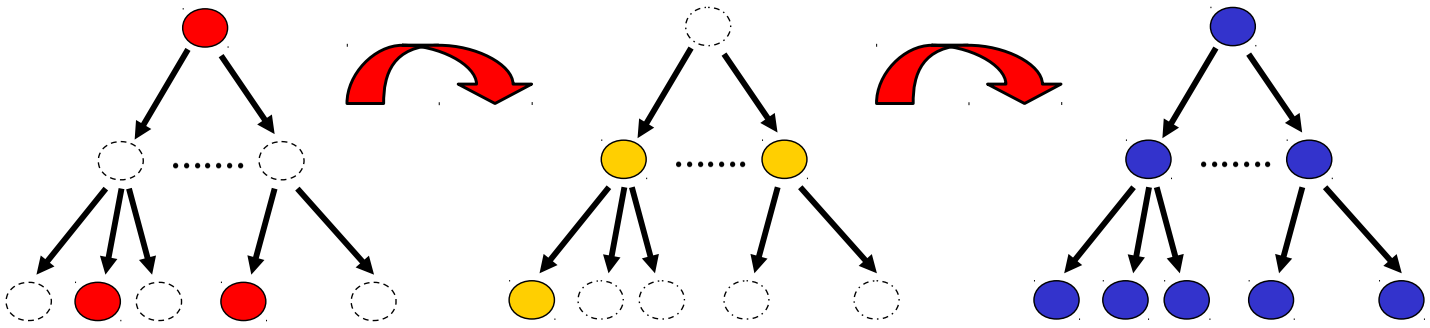
svn Tools

- Tools
  - Kwiki with subversion backend
  - svk – decentralized version system
  - subissue subversion issue tracking
  - scmbug bug tracking
- Clients
  - ankhSVN (Visual Studio plugin)
  - psvn.el for emacs
  - Rapidsvn
  - esvn
  - supervision
  - subclipse, Svn4Eclipse for Eclipse
- Wie man zu Subversion wechselt
  - Cvs2svn
  - Prcs2svn
  - vss2svn

21.3.2 Long Runs in Ketten von Sichten

Ketten von Sichten auf Repositories (svn)

- Programmierer haben eine Sicht von der Testversion, die eine Sicht von der offiziellen Version ist
- Sichten werden *Zweige (branches) genannt, wenn sie über mehrere Versionen hinweg parallel entwickelt werden*



Wie man einen “long-run” mit svn macht

- Lege eine Sicht an
- Lege immer neue Versionen in der Sicht an

Wiederhole das rekursiv

- Verschmelze die HEADs der Sichten mit svn merge
  - Die lineare Nummerung schafft einfache Vergleichbarkeit

Ketten von Sichten entstehen

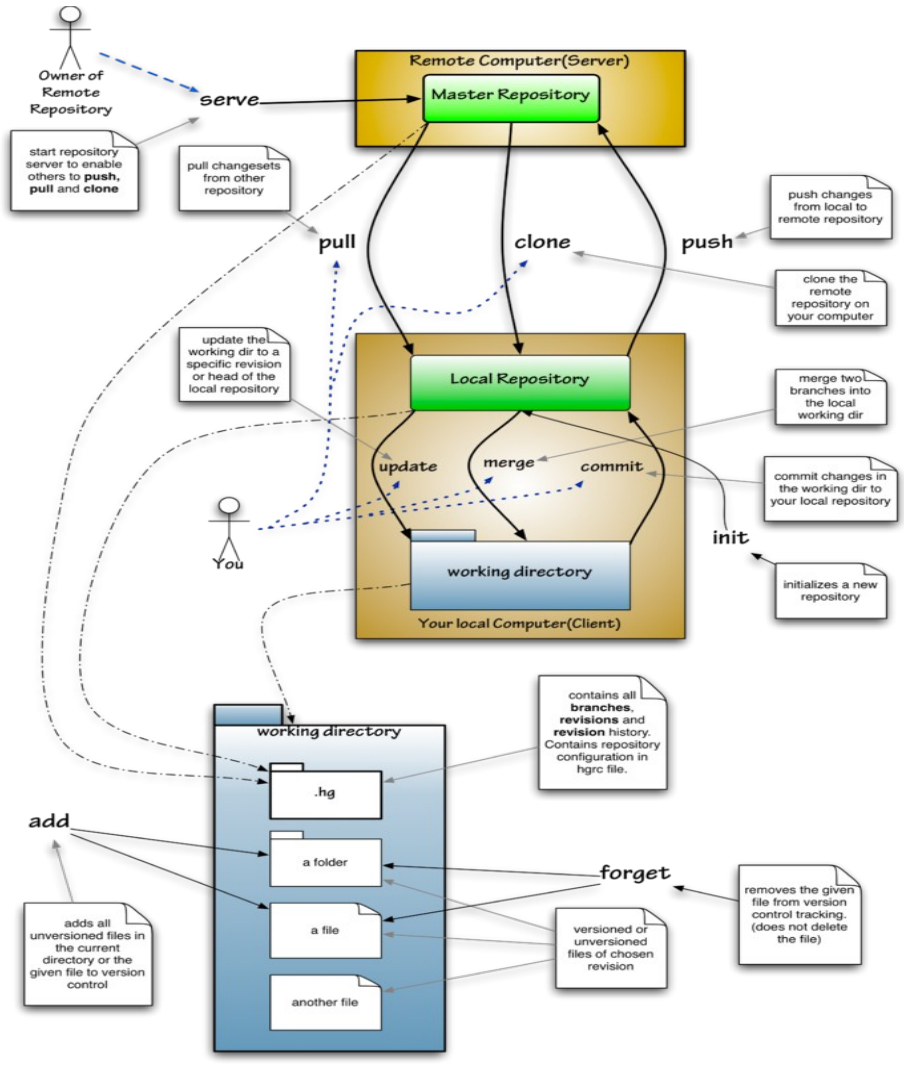
Das svn Entwicklungsmuster “Ketten von Sichten” (Branches)

- Das svn copy Kommando ist so mächtig, dass nun Sichten und Zweige alle im view und im Repository dargestellt werden können
  - **Typische Aufteilung des Projektbaumes:**
  - **Nebensichten** in /home/ua1/tmp/Ontologies1/branches
    - /home/ua1/tmp/Ontologies1/branches/testversion
    - /home/ua1/tmp/Ontologies1/branches/testversion-mary
    - /home/ua1/tmp/Ontologies1/branches/testversion-frank
  - **Snapshots** (tagged branches) in /home/ua1/tmp/Ontologies1/tags
    - /home/ua1/tmp/Ontologies1/tags/alpha-release
    - /home/ua1/tmp/Ontologies1/tags/beta-release
  - **Hauptsicht** in /home/ua1/tmp/Ontologies1/trunk

Ketten von Sichten

- Kopiere die Hauptsicht als neue Nebensicht
  - svn copy /home/ua1/tmp/Ontologies1/trunk /home/ua1/tmp/Ontologies1/branches/new-view
- Friere eine Nebensicht als Snapshot ein
  - svn copy /home/ua1/tmp/Ontologies1/branches/view1 /home/ua1/tmp/Ontologies1/tags/alpha-release
- Verschmelze Sicht Programmierer 1 und 2
  - svn merge /home/ua1/tmp/Ontologies1/branches/view-horst /home/ua1/tmp/Ontologies1/branches/view-maria
- Geht auch auf Repositories
  - svn copy http:///home/ua1/svn/trunk/Ontologies1 http://home/ua1/svn/Ontologies1/branches/new-view

Ketten von Sichten in Mercurial



http://commons.wikimedia.org/wiki/File:Mercurial_commandd_and_their_relations.png

Prof. Uwe Aßmann, Softwaremanagement

Wie man mit Subversion auf einem Laptop offline arbeitet

Lege eine Sicht für den Laptop an, in <project>/branches
Arbeite auf ihr, auch offline
Änderungen werden für die Sicht in den Cache gelegt

Sobald online, schreibe in das globale Repository

Falls Verschmelzung gewünscht, verschmelze die Server-Sicht
in <project>/branches/my-view mit <project>/trunk

Das bedeutet: trunk und branches werden wie “Teilprojekte” behandelt
und können sowohl auf Server wie auch auf Laptop liegen

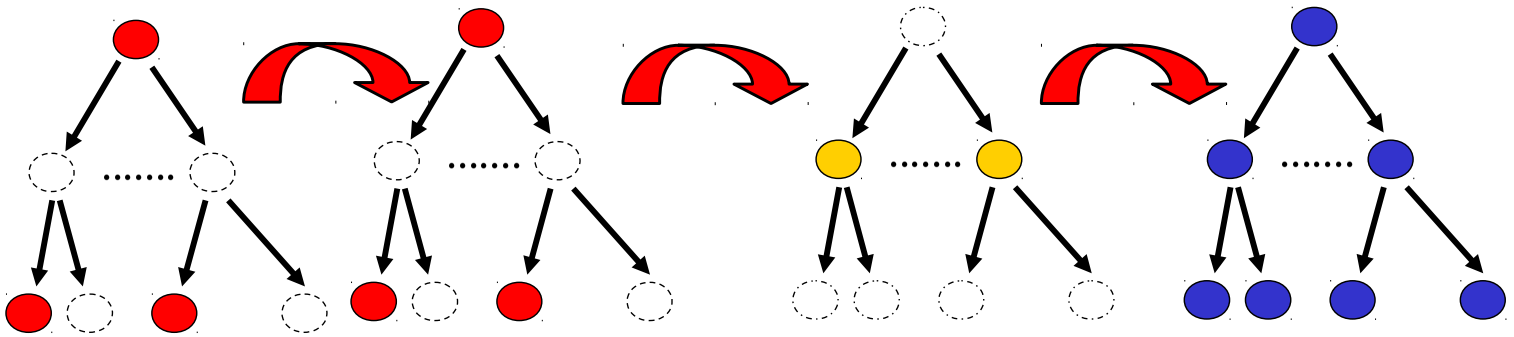
21.3.3 GIT

Neuigkeiten

- Ein zentrales Repository ist keine Pflicht, sondern beliebig “transportierbar”
- Statt dessen **GIT workflows**, verschiedene Arten, wie Sichten zusammenarbeiten
- GIT speichert keine Deltas, sondern Vollversionen
- GIT kennt **features,** die benamten Branches entsprechen

GIT Standard Workflow
Ketten von Sichten auf Repositories

- Programmierer haben eine Sicht vom workspace, die eine Sicht von der Sicht von der Sicht der offiziellen Version ist



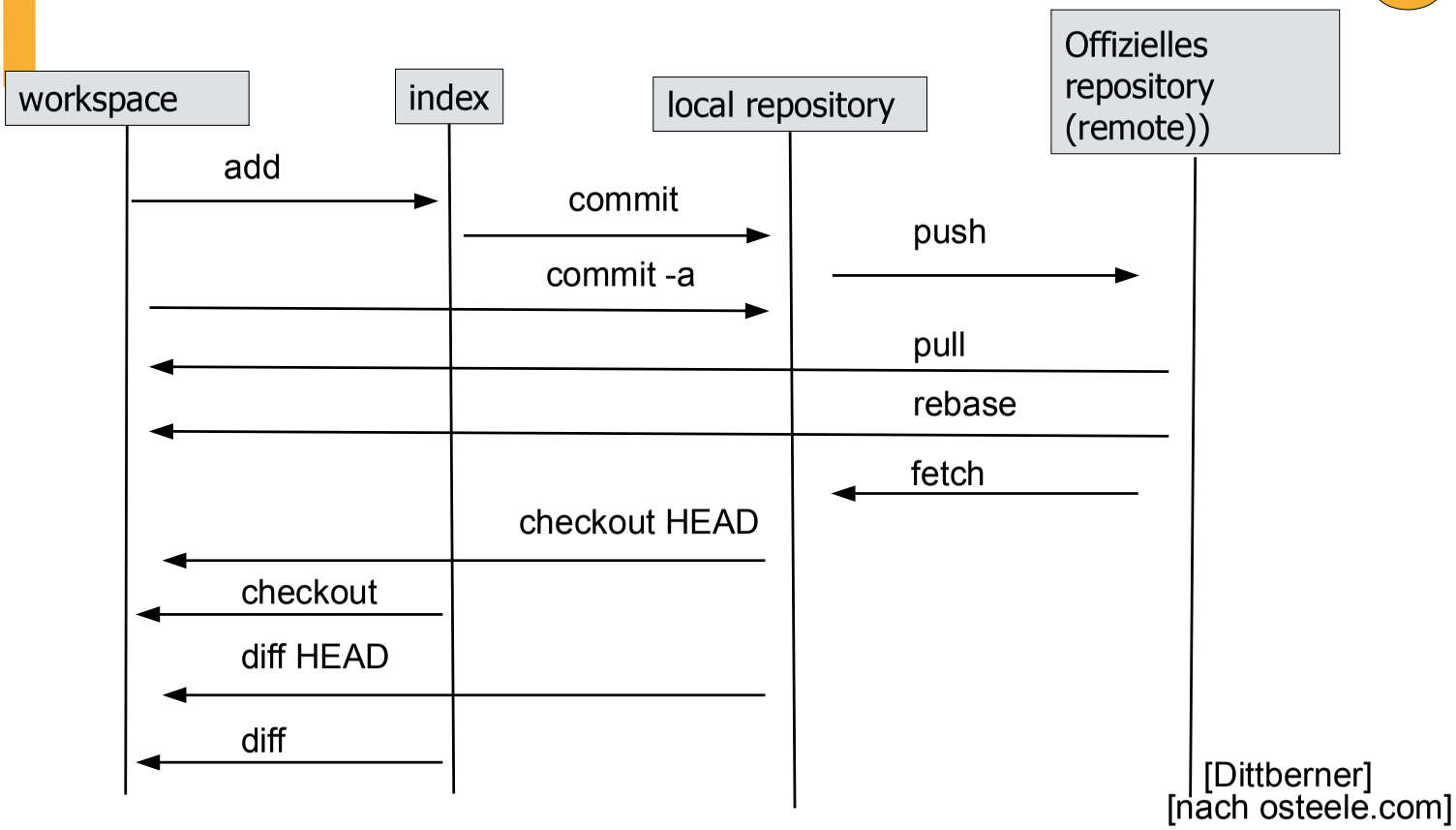
Programmierer-Sicht (workspace)

index (Vorbereitung für den nächsten commit)

local repository (commit base HEAD)

Offizielles repository (remote))

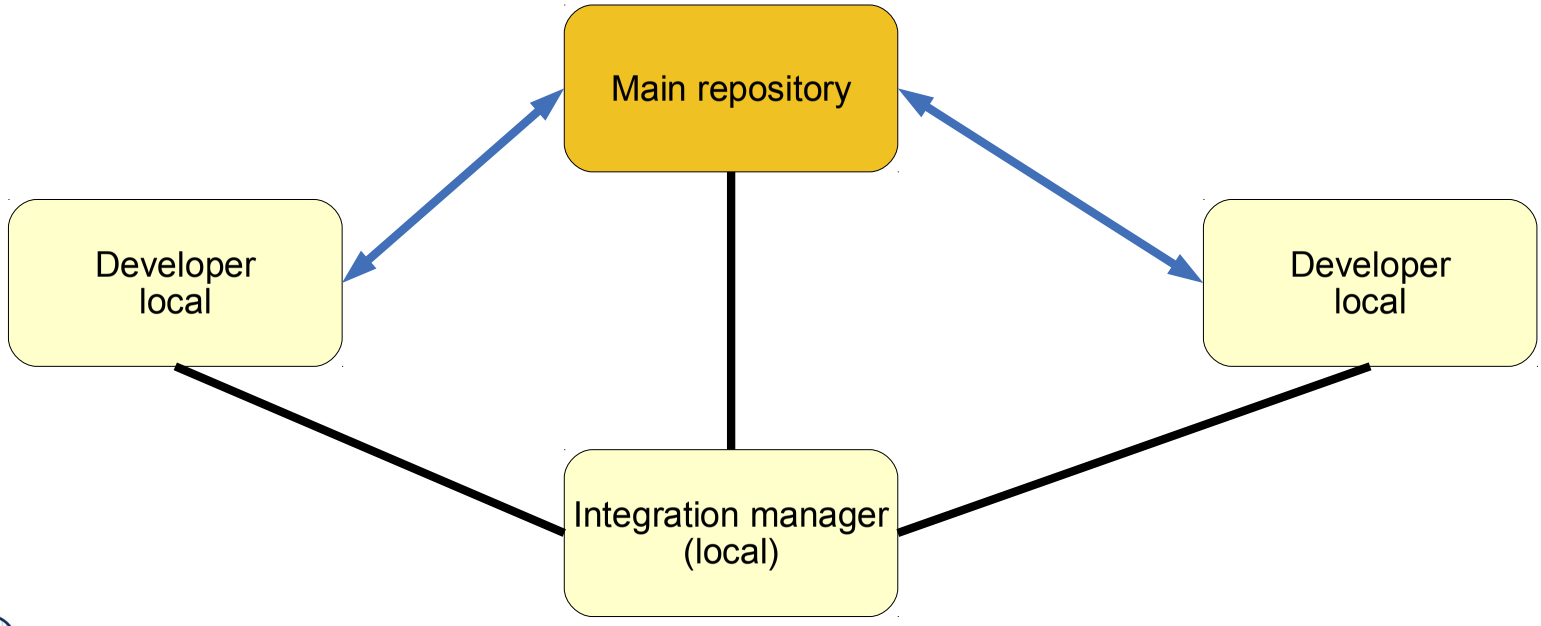
Ketten von Sichten in GIT



[Dittberner]
[nach osteele.com]

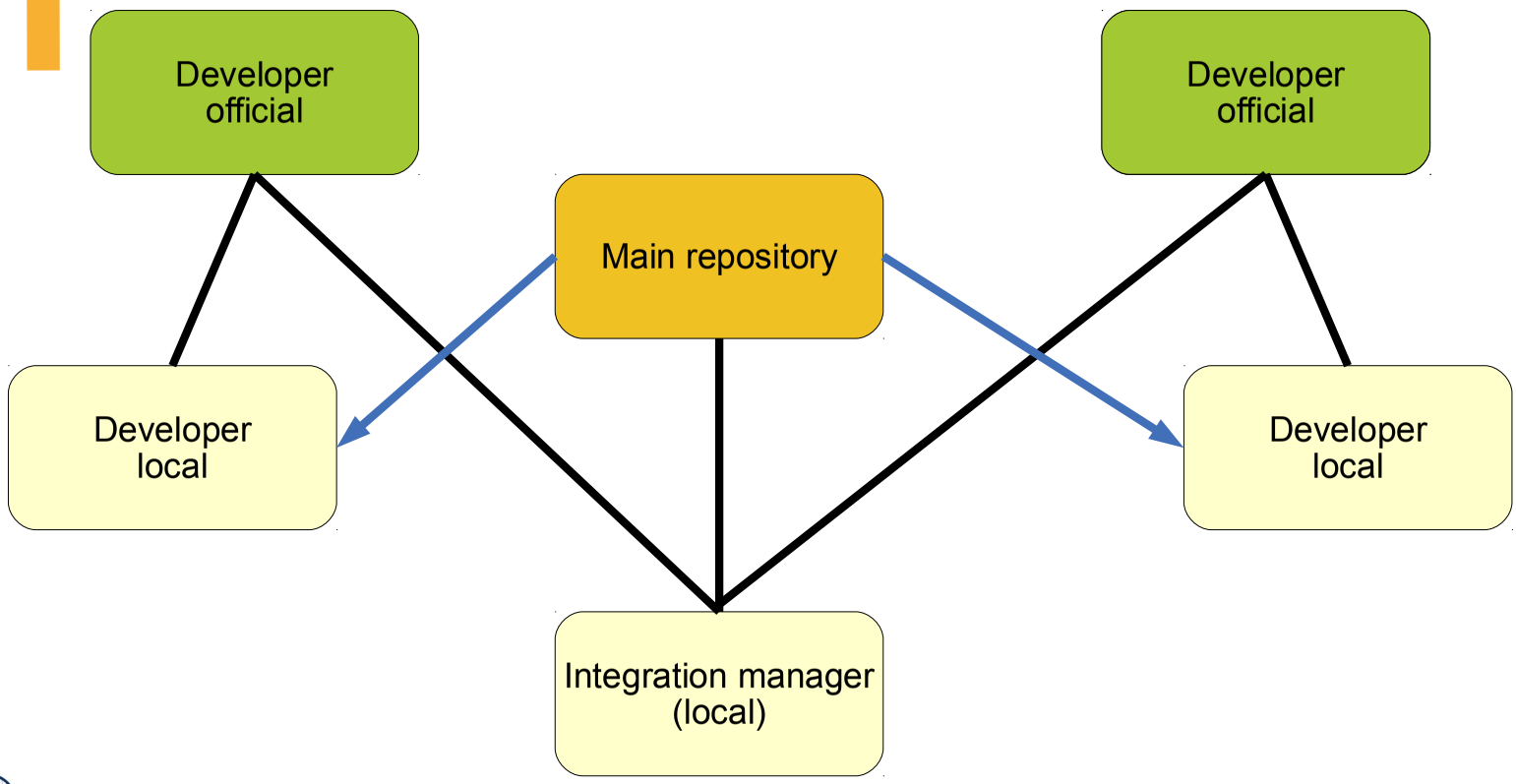
Git Net with Interaction between Developers

- Merge between the local repositories of the developers



[Dittberner]

Git Net with Integration Manager



[Dittberner]

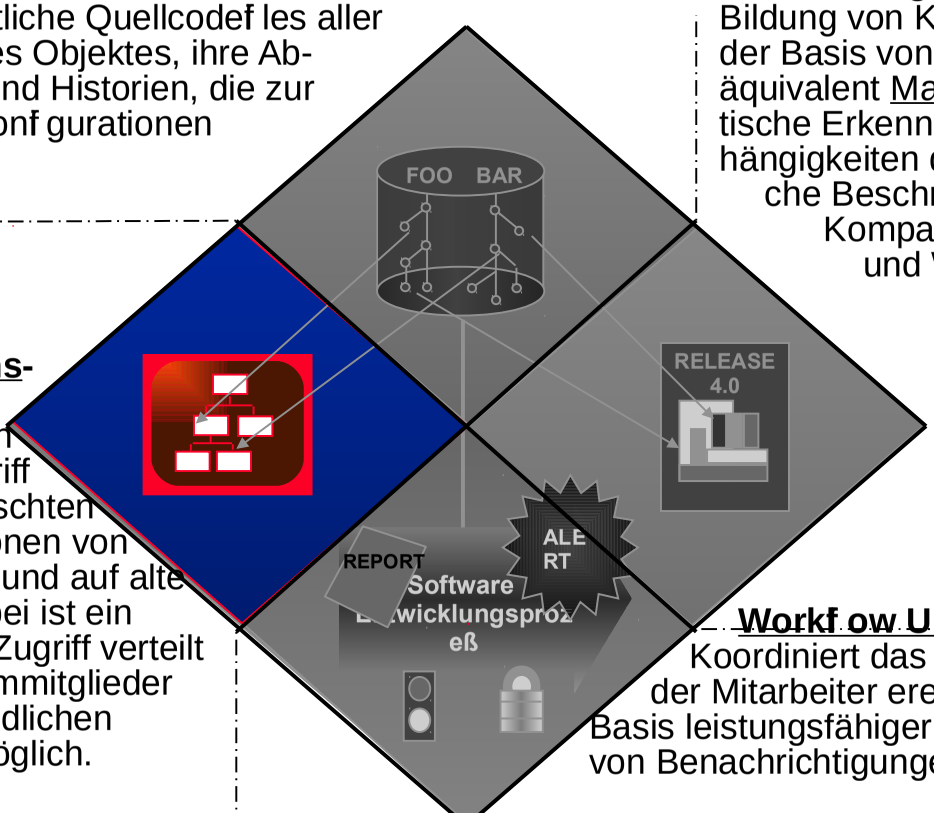
21.4 Datenbankgestützte KM-Werkzeuge am Beispiel ClearCase

- von IBM Rational
- unterstützt zusätzlich Views mit einem Multi-View-Dateisystem auf der Datenbank obenauf

ClearCase-Funktionsumfang

Versionskontrolle:
Verwaltet sämtliche Quellcodef les aller Versionen eines Objektes, ihre Abhängigkeiten und Historien, die zur Bildung von Konf gurationen nötig sind.

Build Management:
Bildung von Komponenten auf der Basis von „build scripts“ äquivalent Makef les. Automatische Erkennung von Fileabhängigkeiten durch eine einfache Beschreibungssprache. Kompatibel zu den UNIX und Windows Varianten von *make*.



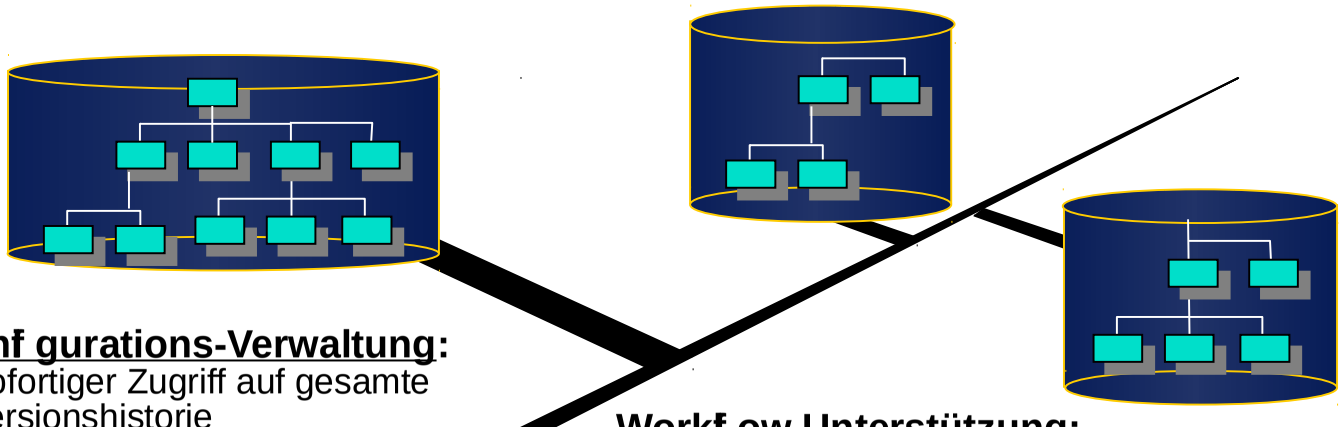
Konf gurations-Verwaltung:
Organisiert den schnellen Zugriff auf die gewünschten exakten Versionen von Komponenten und auf alte Releases. Dabei ist ein transparenter Zugriff verteilt durch alle Teammitglieder von unterschiedlichen Plattformen möglich.

Workf ow Unterstützung:
Koordiniert das Zusammenwirken der Mitarbeiter ereignisgesteuert auf Basis leistungsfähiger Werkzeuge sowie von Benachrichtigungen und Berichten.

ClearCase Teilgebiete

Versions-Kontrolle:
- zentrale Datenhaltung im Netzwerk eff zient verteilt
- Dateien und Verzeichnisse können versioniert werden
- VOBs sind in Größe und Anzahl skalierbar

Build Management:
- Erzeugt automatisch "Stückliste" aller zum Build verwendeten Sourcen
- Erkennt automatisch Abhängigkeiten
- Garantiert 100% Reproduzierbarkeit
- Überwachte Builds



Konf gurations-Verwaltung:
- Sofortiger Zugriff auf gesamte Versionshistorie
- Integriert alle isolierten Entwicklungsbereiche
- Mergemanager unterstützt automatisches Mergen

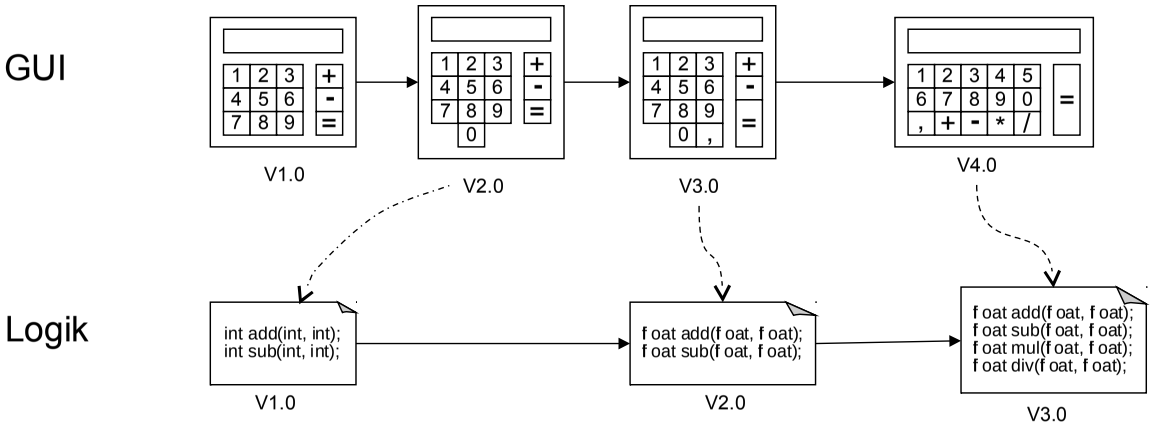
Workf ow Unterstützung:
- Erhöhte Eff zienz durch Automatisierung von Routinetätigkeiten
- Anpassbar an spezif schen Prozess
- Flexible Wahl der Vorgehensweise

21.A Anhang: Beispiel einer Evolution

zum Selbststudium

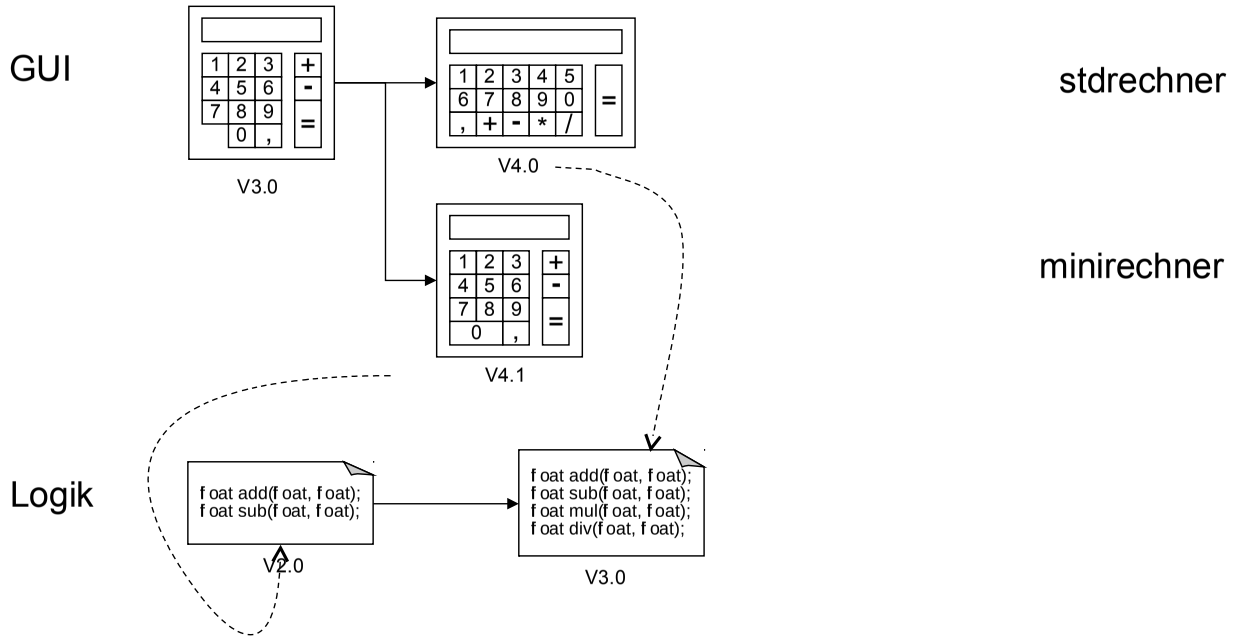
Einführungsbeispiel (1)

Entwicklung eines Taschenrechners, bestehend aus GUI- und Logikkomponente:
- Die Entwicklung erfolgt schrittweise (Komponenten besitzen Versionen)
- Komponenten(-versionen) werden idR zu verschiedenen Zeitpunkten fertig
- nicht jede GUI-Version passt zu jeder Logikversion (V4.0,V2.0)
- es gibt Kombinationen die zwar passen,
  aber nicht (mehr) ausgeliefert werden dürfen (V1.0,V1.0) - fehlende 0-Taste
  oder nicht (mehr) ausgeliefert werden sollen (V2.0,V2.0) - langsamer als mit Logik V1.0



Einführungsbeispiel (2)

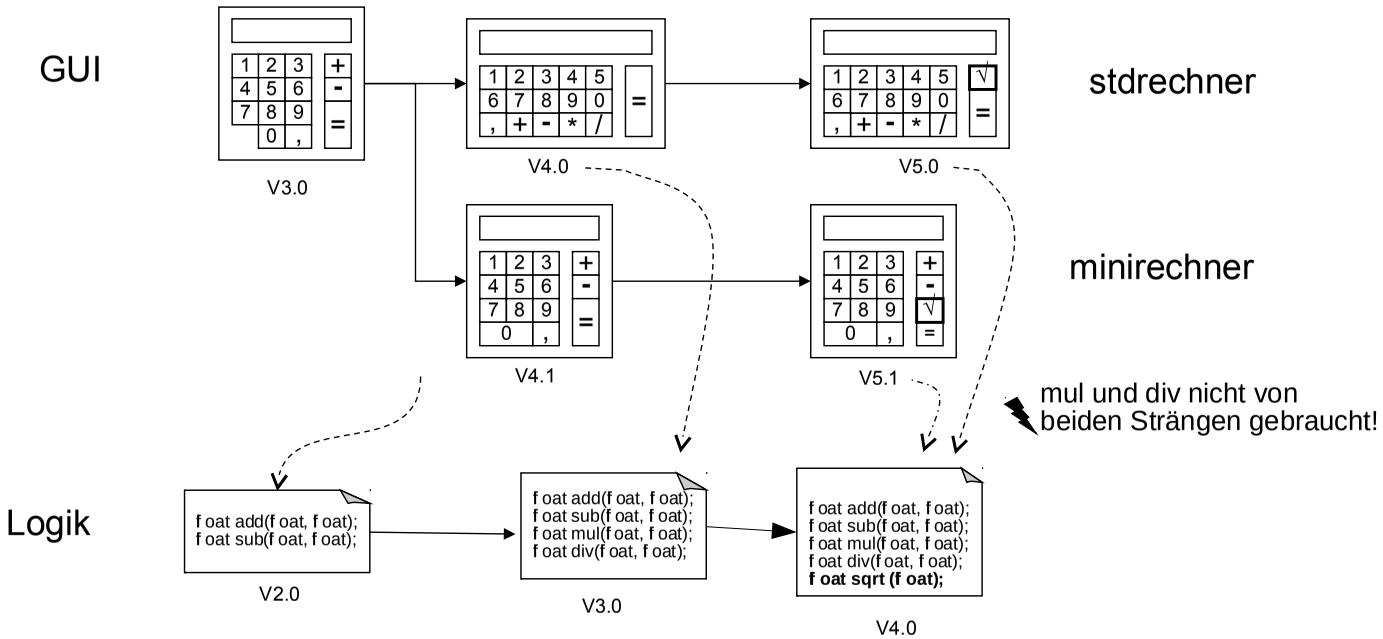
Der Taschenrechners soll in zwei Varianten auf den Markt kommen.
Komponenten können mehrere Zweige besitzen, die parallel weiterentwickelt werden.



Problem: eine neue Funktionalität (z.B. Wurzelziehen) soll in beide Zweige eingeführt werden.

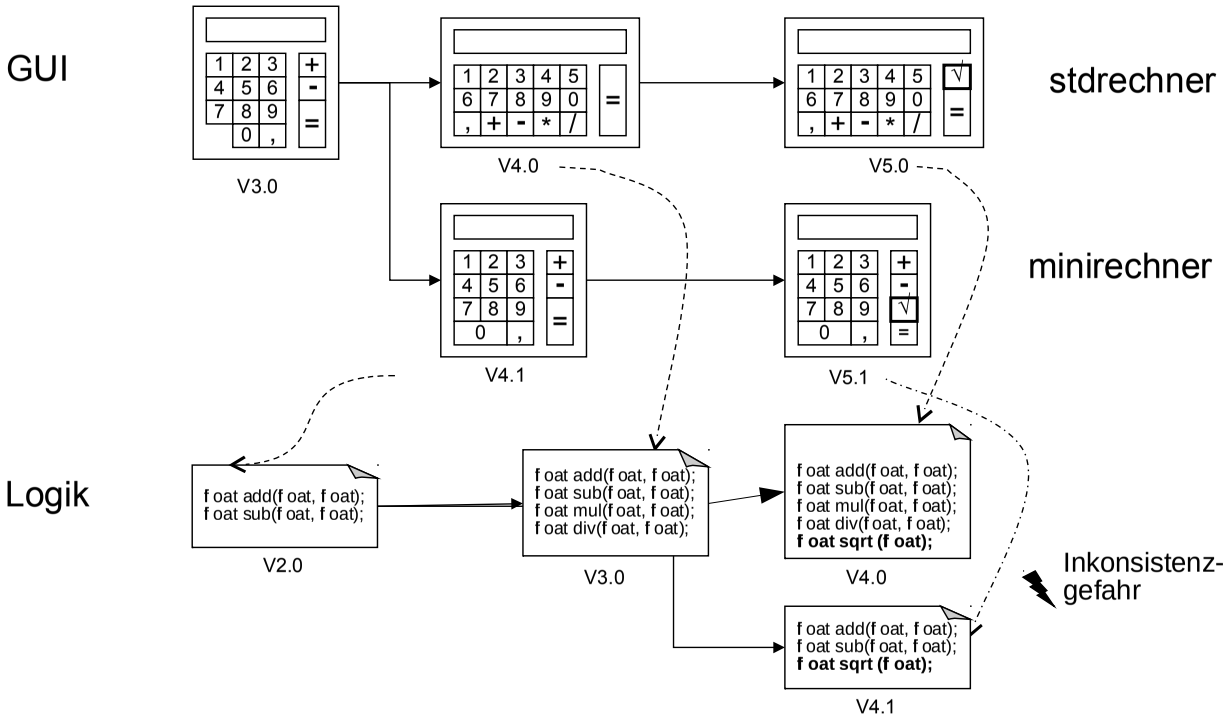
Einführungsbeispiel (3)

Mögliche Lösung: Einführung in den jeweils aktuellsten Stand des Komponentenzweigs



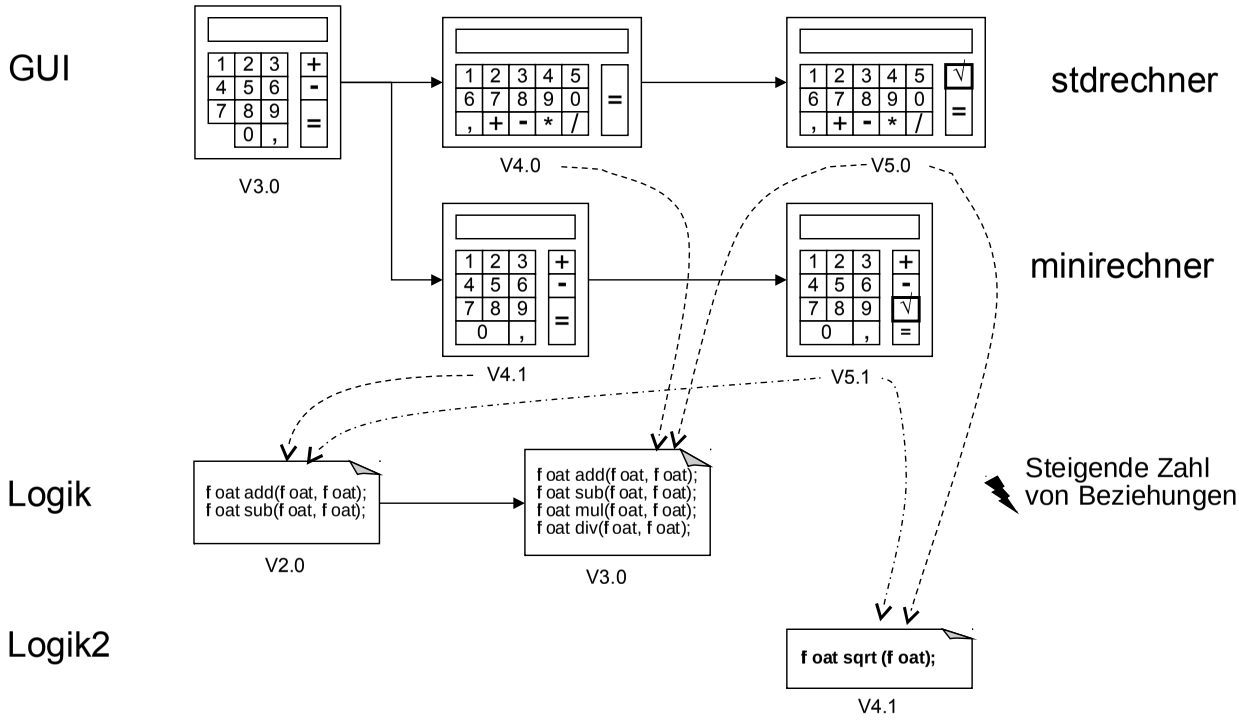
Einführungsbeispiel (4)

Alternative Lösung: Aufspalten in mehrere Versionszweige



Einführungsbeispiel (5)

Alternative Lösung: Aufspalten in mehrere Komponenten



Einschränkungen von cvs

- Behandlung von Teilbäumen schwierig
  - kein atomares Commit von Teilbäumen
  - kein move
    - move == remove oldfile; add newfile
    - Versionsgeschichte geht immer verloren
  - kein copy
  - Repräsentation von Zweigen *nur* im Repository
- Kein Verschmelzen von Zweigen im gleichen Repository
  - noch von verschiedenen Repositories
  - Keine Unterstützung für “long-running changes” (Ketten von Sichten)
- Schwer, das Repository zu bewegen
  - Alle Sichten werden inkonsistent

The End

Prof. Uwe Aßmann, Softwaremanagement