

## 21. Konfigurationsmanagement

Prof. Dr. rer. nat. Uwe Aßmann  
Lehrstuhl Softwaretechnologie  
Fakultät Informatik  
Technische Universität Dresden  
Version 11-0.1, 08.06.11

- 1) Software-Wartung
- 2) Änderungsmanagement
- 3) Konfigurationsmanagement
  - 1) Dateibaumbasierte KM-Werkzeuge
  - 2) Subversion
  - 3) Long Runs in Ketten von Sichten

Softwaremanagement, © Prof. Uwe Aßmann

## 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/>

## 21.1 Software-Wartung

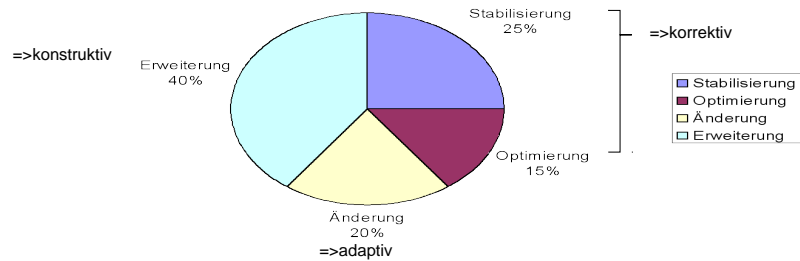
## Software-Wartung (1)

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

- ▶ **Wartungsmotivation**
  - **präventive Maßnahmen**
  - **korrektive Wartung**
    - Stabilisierung/Korrektur
    - Optimierung/Leistungsverbesserung
  - **progressive Wartung (Pflege)**
    - Anpassung/Änderung (adaptiv)
    - Erweiterung (konstruktiv)
- ▶ **Wartungsanforderungen bzgl. Dringlichkeit:**
  - **sofort (operativ):** schwerwiegende SW-Fehler, die weitere Nutzung in Frage stellen
  - **kurzfristig:** Code-Korrekturen
  - **mittelfristig:** Systemerweiterungen (upgrades)
  - **langfristig:** Restrukturierung (System Redesign)

## Software-Wartung (2)

Prozentualer Ausweis von Wartungsaufwand

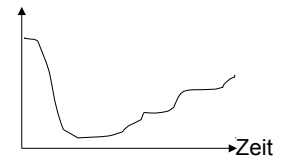


präventiver Wartungsaufwand nicht dargestellt

## Software-Wartung: Evolutionsgesetze

- ▶ Gesetz der kontinuierlichen Veränderung (Lehmans 1. Gesetz)
  - Kartoffeltheorem: Ein genutztes System verändert sich, bis die Neustrukturierung oder eine neue Version günstiger ist
- ▶ Gesetz der zunehmenden Komplexität (Lehmans 2. Gesetz)
  - Ohne Gegenmaßnahmen führen Veränderungen zu zunehmender Komplexität.
- ▶ Gesetz des Feedbacks (Lehmans 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:  
Fehlerkurve



[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):

- Planbar durch dokumentierten Inhalt
- Ausgangsbasis ist konsistentes Produkt, in das das gezielt - unter Beibehaltung der Konsistenz – Änderungen und Erweiterungen eingebracht werden
- Bandbreite der Änderungen/Erweiterungen kann von kleinen --> bis zu großen Modifikationen gehen
- 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

Def.: **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):

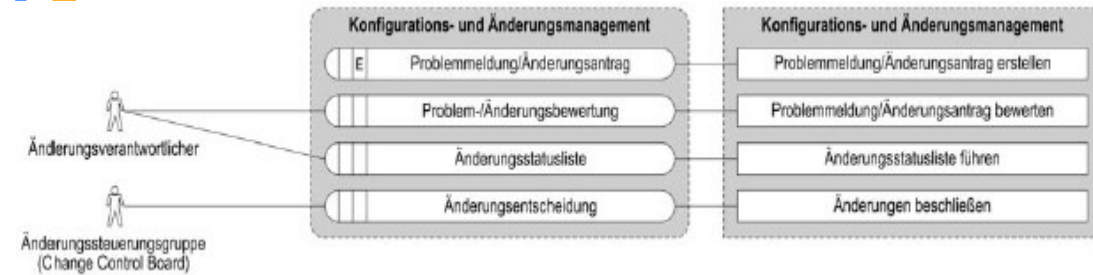
- Nicht planbar: Ereignisgesteuert, nicht vorhersehbar, schwer kontrollierbar
- Ausgangsbasis ist ein fehlerhaftes bzw. inkonsistentes Produkt
- Abweichungen zwischen Teilprodukten sind zu lokalisieren und zu beheben
- Die Korrektur einzelner Fehler hat nur begrenzte Auswirkungen auf das Gesamtprodukt, 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 Änderungsmanagement

## Vorgehensbaustein Problem- und Änderungsmanagement



Notwendige Produkte (Belege) sind:

- Problemmeldung und Änderungsantrag
- Problem- und Änderungsbewertung
- Änderungsentscheidung, -mitteilung
- Änderungsstatusliste

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

## Änderungsmanagement

- ▶ ist nötig für Firmen, die sukzessive neue Versionen ihrer Produkte erzeugen
  - im Produktgeschäft tätig sind
  - im Produktlinien-Geschäft tätig sind
- ▶ weniger nötig für eine Anwendungslandschaft in einer Firma

## Inhalt des Änderungsmanagement laut V-Modell XT

### **Aufgaben** der Aktivitäten des Problem- und Änderungsmanagements:

- Zustandserfassung von Problemmeldungen/Änderungsanträgen
- Dokumentieren und Verwaltung aller Problemmeldungen und Änderungsanträge über eine Statusliste
- Änderungen bewerten (Ursachen, Auswirkungen, ...)
- Entscheidung, Freigabe und Veranlassung der Bearbeitung
- Abschluss der Änderung, Information der Betroffenen
- Erfassung von Problemmeldungen, Fehlermeldungen, Verbesserungsvorschlägen und Änderungswünschen

## Aktivität Problemmeldung/Änderungsantrag erstellen

- Jede Rolle kann aus den verschiedensten Gründen eine Problemmeldung/Änderungsantrag auslösen
- Er sollte grundsätzlich folgende Informationen enthalten:
  - Beschreibung des Problems bzw. der gewünschten Änderung
  - Identifikation Antragsteller, Projekt, betroffenen Konfiguration
  - Begründung des Antrages bzgl. Nutzen bzw. Schaden bei Nichtdurchführung
  - Lösungsvorschlag aus Sicht des Antragstellers
  - Nummer Änderungsantrag/Problemmeldung
  - Vergabe einer Registriernummer pro Problemmeldung/Änderungsantrag
- Gründe für Änderungen können sein:
  - neue Entwicklungserfordernisse
  - Zeitprobleme
  - Kosteneinhaltung
  - Änderungen gesetzlicher Vorschriften
  - Verbesserung von Marktchancen
  - Nutzerwünsche
- Änderungen können ‚direkt‘ oder ‚indirekt‘ durch Dritte motiviert sein.

13

## Aktivität Problemmeldung/Änderungsantrag bewerten

- Problemmeldung/Änderungsantrag analysieren wie dringend Lösung des Problems bzw. der beantragten Änderung ist
- Lösungsvorschläge erarbeiten mit vollständiger bzw. auch erst nur teilweiser Lösung. Folgende Informationen sollte er enthalten:
  - Teile des Projektes, die von der Änderung betroffen sind
  - Phase des Entwicklungsprozesses, in der Änderung anfällt
  - Lösungsbeschreibung und -vorgehen
  - erforderliche Aufwendungen
  - Auswirkungen der Änderung auf das Projekt
- Empfehlung aussprechen:
  - auf Basis der erarbeiteten alternativen Lösungsvorschläge
  - alle Lösungsvorschläge sind anhand ihrer Auswirkungen auf das Projekt zu bewerten
  - aus dieser Basis ist eine Entscheidung zu fällen und zu begründen
- Alle Bewertungsaktivitäten werden im Produkt Problem-/Änderungsbewertung niedergelegt

14

## Aktivität Änderungen beschließen

- Vorbereitung des Entscheidungsmeetings durch Sammeln alle Anträge und Bewertungen, **Erstellen der Agenda** für das Meeting
- Einladungen an beteiligte Rollen oder Stakeholder **verschicken**
- Anträge vorstellen und **präsentieren** mit:
  - entstehenden Kosten
  - Verfügbarkeit von Mitteln und Personal
  - zeitliche Projektverzögerung
  - technische Eignung der vorgeschlagenen
- Änderungsentscheidung **beschließen** und Dringlichkeit der Umsetzung festlegen
  - Festlegung der Kategorie (Fehler [in Spezifikation, Entwurf, Codierung, im Verfahren], Problem, Modifikation, Erweiterung, Verbesserung, usw.)
  - gewünschter Fertigstellungszeitpunkt
- Auswirkungen der Änderung ermitteln
- Änderungsentscheidung – im Änderungsbescheid - **protokollieren**
- Änderungsentscheidung **verteilen** bzw. kommunizieren
- Alle beschlossenen Änderungen werden im Produkt Änderungsentscheidung/-mitteilung niedergelegt

15

## Aktivität Änderungsstatusliste führen

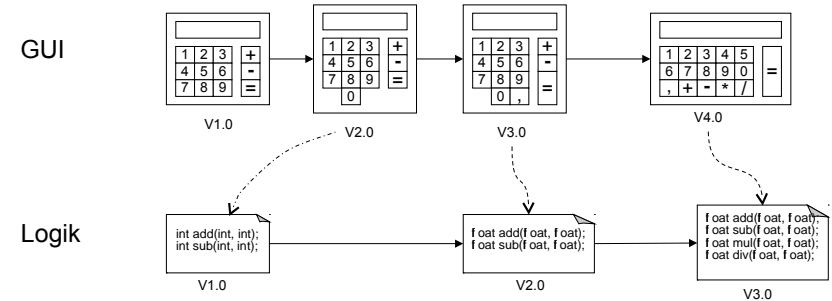
- Änderungsstatusliste dient dem Ziel, alle wichtigen Informationen zum Projekt hinsichtlich Änderungsanforderungen und –auswirkungen zu aktualisieren und dokumentieren
- Ablauf und Dokumentation ist für jede Änderungsanforderung gleich:
  - **Änderungsanforderungen registrieren** mit Prüfung der benötigten Daten auf Vollständigkeit
  - **Änderungsanforderungen prüfen** auf Realisierbarkeit und Festlegung der erforderlichen Mittel, Termine und Verantwortlichkeiten
  - **Änderungsstatusliste aktualisieren** nach bereits bestehenden Änderungsanforderungen bzw. durch Hinzufügen neuer Anforderungen
- Änderungsstati sind z. B.: ‚beantragt‘, ‚beabsichtigt‘, ‚abgelehnt‘, ‚genehmigt‘, ‚zurückgestellt‘, ‚beauftragt‘, ‚erledigt‘
- Bemerkungen bei Beziehungen zu bereits gestellten Änderungsanträgen
- Referenzen auf die Änderungsbewertung oder die Änderungsentscheidung sind in der Änderungsstatusliste ebenfalls festzuhalten
- Wird oft in einer Datenbank geführt
  - z.B. MANTIS-System

16

# 21.3 Konfigurationsmanagement

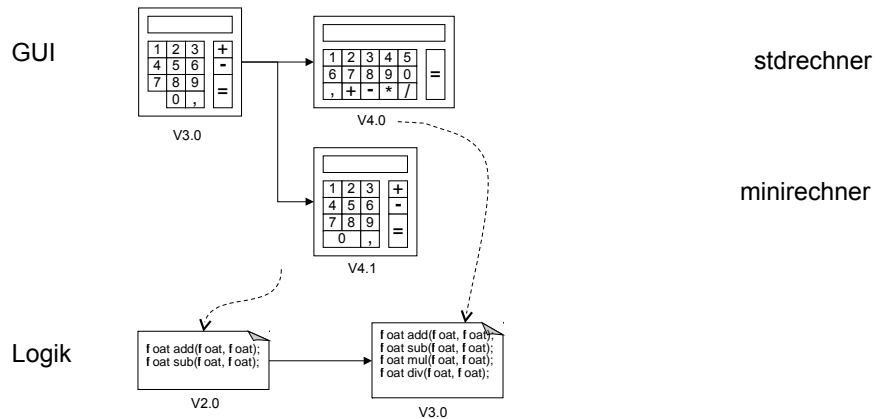
## Einführungsbeispiel (1)

- 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)

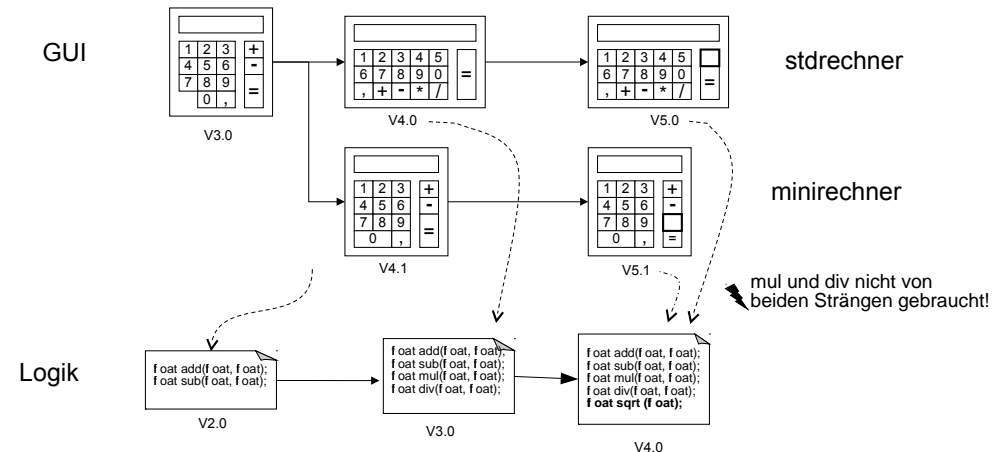
Der Taschenrechners soll in zwei Varianten auf den Markt kommen. Komponenten können mehrere Zweige besitzen, die parallel weiterentwickelt werden.



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

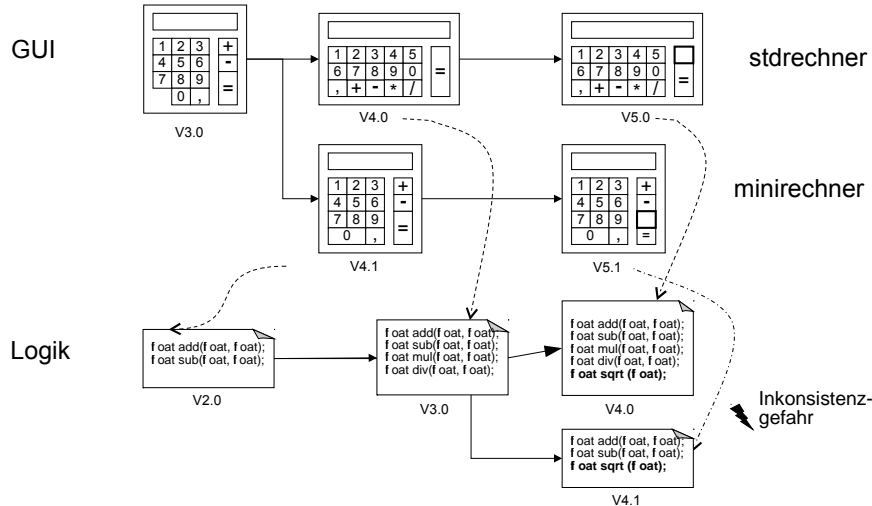
## Einführungsbeispiel (3)

Mögliche Lösung: Einführung in den jeweils aktuellsten Stand des Komponentenzweigs



## Einführungsbeispiel (4)

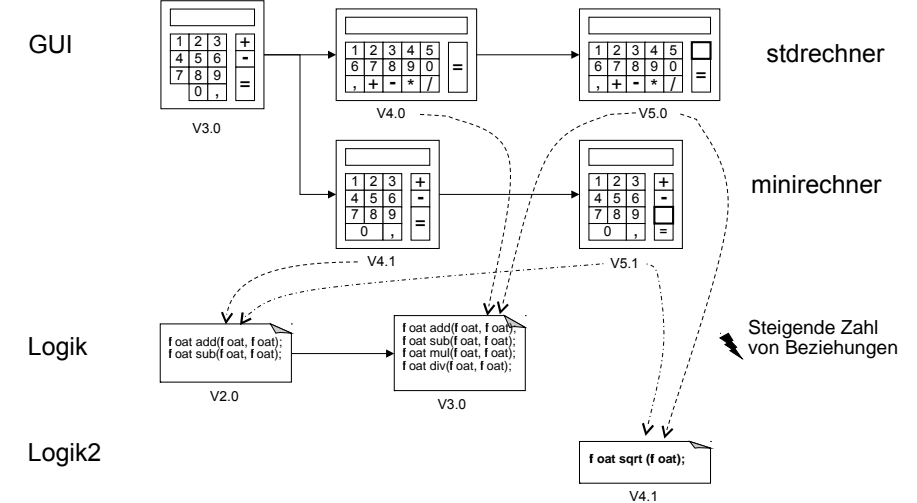
Alternative Lösung: Aufspalten in mehrere Versionszweige



21

## Einführungsbeispiel (5)

Alternative Lösung: Aufspalten in mehrere Komponenten



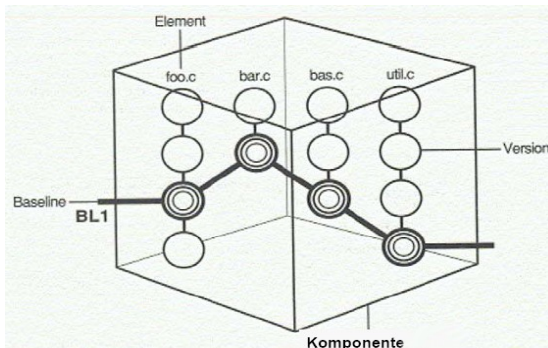
22

## Schlussfolgerung

- Die Verwaltung der Produkte und Teilergebnisse ist nicht trivial
- Daraus ergeben sich folgende grundlegende Aufgaben des Konfigurationsmanagements:
  - Notieren von lauffähigen/fehlerhaften Kombinationen (=Konfigurationen)
  - Aufbewahrung aller Versionen um alte Konfigurationen wiederherstellen zu können

### Begriffe:

- Element: atomare Softwareinheit (im weitesten Sinne eine Datei)
- Version: Zustand eines Elements (Anm: Branches fehlen noch)
- Komponente: Menge von Elementen (z.B ein Package)
- Baseline: Menge von Versionen (gesichertes Zwischenergebnis aus Menge freigegebener Versionen)



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

23

## Konfigurationsmanagement

**Def.:** Eine **SW-Konfiguration** ist die **Gesamtheit der Artefakte** (Menge von [Produkt-]Versionen), die zu einem **bestimmten Zeitpunkt des Life Cycle** in ihrer Wirkungsweise und ihren Schnittstellen aufeinander abgestimmt sind.

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

### Ziele:

- Bestimmung der **Artefakte** (Moduln, Packages, Dateien, Datenbanken), die eine Konfiguration bilden; Sichtbarkeit, Verfolgbarkeit, Kontrollierbarkeit von Produkten
- Erfassung und Überwachung der Änderungsanstöße (Meldungen) ==> Prüfung ==> Mitwirkung bei der Umsetzung in **Aufträge**
- Prüfung, ob Konsistenz der SW-Elemente erhalten bleibt; Sicherstellung, dass jederzeit auf vorherige Versionen /Konfigurationen zurückgegriffen werden kann
- Erfassung und Nachweis aller Änderungen; Überwachung der (Produkt-)Konfigurationen während des Lebenszyklus
- (5. Auslieferungskontrolle)

24

## Gegenstände des KM

25

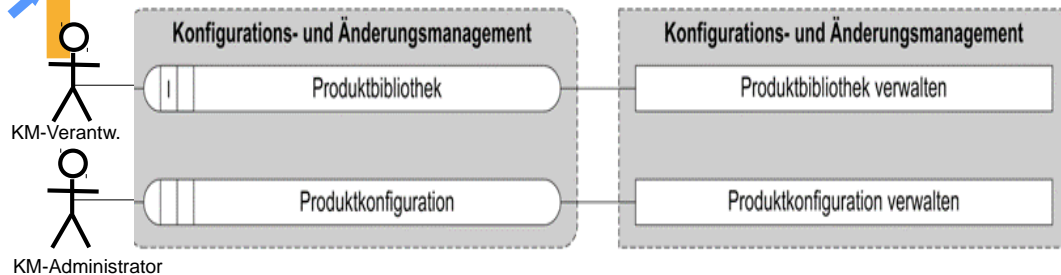
- Gegenstand des KM:** gesamtes SW-System mit seinen Komponenten
- **Spezifikation:** Daten und Requirements
  - **Entwurf:** formale und informale Dokumente (Datenorg., Programmstrukturen, SS-Entwürfe, ...)
  - **Programme:** Code-Teile, Datenbeschreibungen, Prozeduren
  - **Testkonzept:** Dokumente für Testdaten, Testumgebung
  - **Integr.-konzept:** alle Dokumente für Integration und Einführung (auch Benutzerdokumentation)

### Verwaltung der Komponenten in der Produktbibliothek

- enthält Menge aller Produkte, Komponenten und deren Versionen
- Ein Produkt enthält viele Artefakte (**Artefaktbibliothek**) zur Verarbeitung existieren bereits zahlreiche Tools

## Vorgehensbaustein Konfigurationsmanagement im V-Modell XT

27



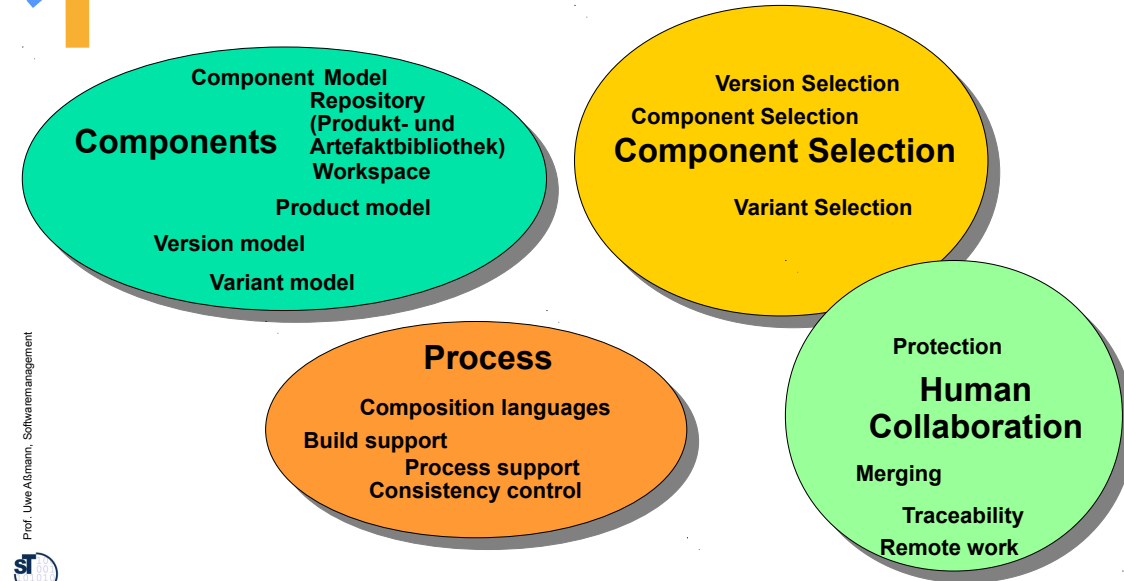
Notwendige Produkte (Belege) sind:

- **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.

## Weitere Aspekte des KM

26



## Inhalt des Konfigurationsmanagements laut V-Modell XT

28

### Aufgaben der Aktivitäten des Konfigurationsmanagements:

- 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 Produktkonfigurationen während des Entwicklungsprozesse als auch während der Nutzung
- Vorgabe definierter Aufsetzpunkte für weitere Prozessschritte

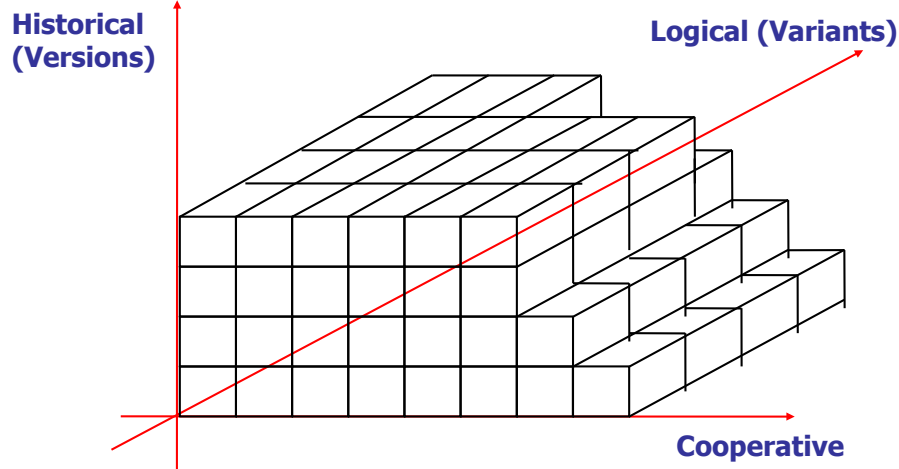
## Aktivität Produktbibliothek verwalten

- 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

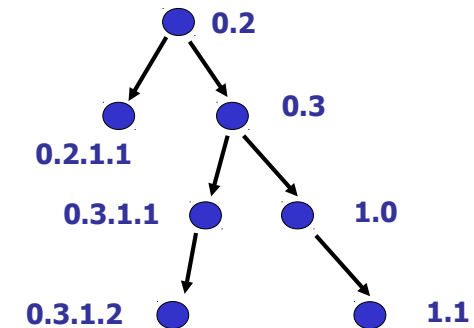
- Produktkonfigurationen dienen der Identifikation inhaltlich zusammengehöriger Produkte, also von Produkten, die über Produktabhängigkeiten miteinander in Beziehung stehen
- Konfiguration initialisieren und Fortschreiben mit folgenden Inhalten
  - Identifikatoren zur Namensgebung, Bearbeitungszustand 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

## Version/Variant/Zusammenarbeits-Universum



## Die Versionsdimension

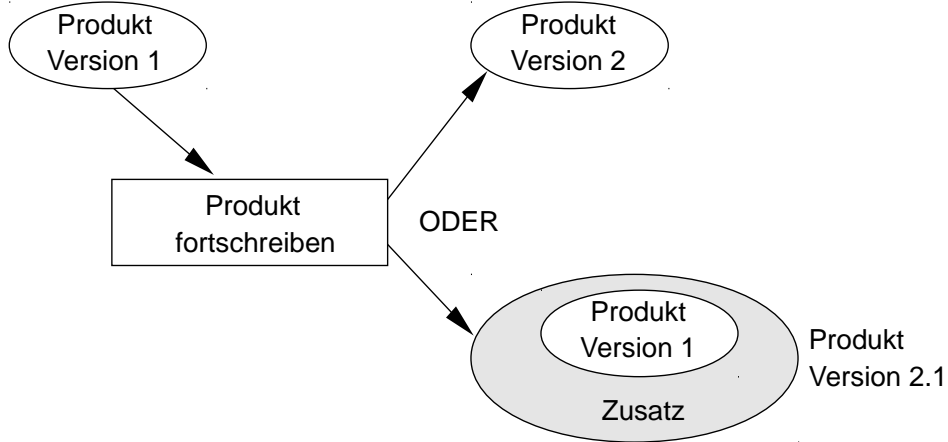
- ▶ Baum unbestimmter Tiefe und Breite
- ▶ Jede Modifikation generiert ein Kind oder einen Zweig (branch)





## Produktkonfigurationen aus Versionen

33

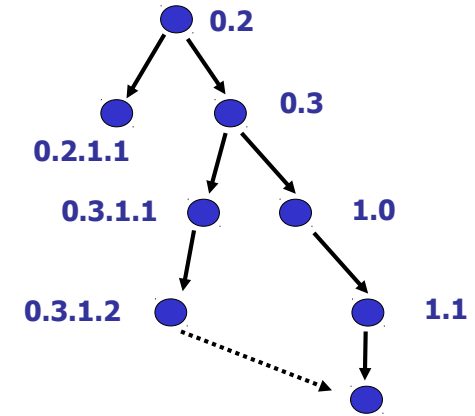


Vorgehensweisen bei der Produktfortschreibung

## Die Versionsdimension

34

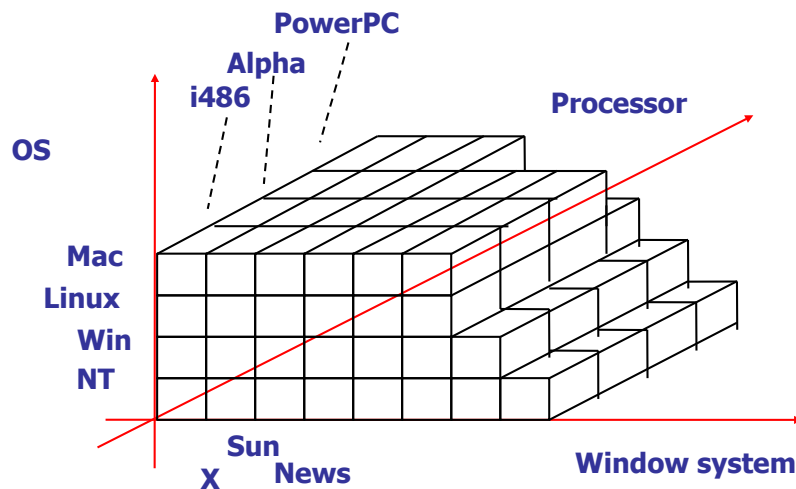
- Verschmelzen ist möglich (branch merge)



## Varietendimension: Ein Variantenuniversum

35

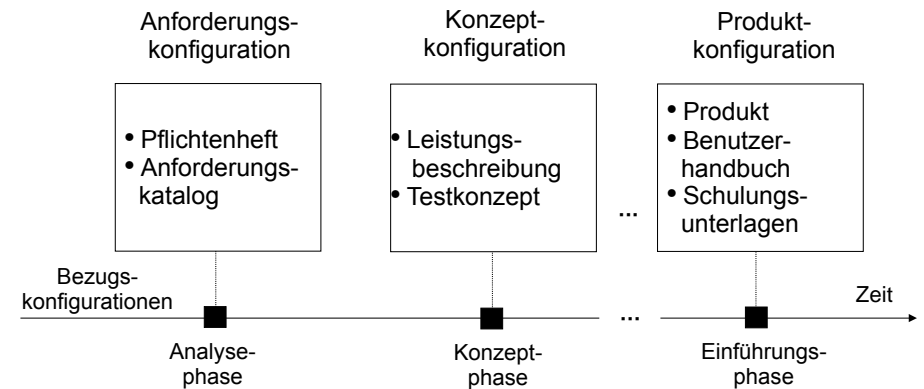
- n-dimensionaler Raum mit k Werten pro Parameter



## Bezugskonfigurationen (Baselines)

36

- Besondere Schnitte im Versionen und Variantenraum sind Bezugskonfigurationen (baselines)



Quelle: Bersoff, E., Henderson, V., Siegel, S.: Software Configuration Management; Prentice Hall 1980

# KM im Lasten-/Pflichtenheft

(Gliederung nach Entwurf VDI/VDE 3694)

## 8. Anforderungen an die Projektabwicklung

### 8.1. Projektorganisation

### 8.2. Projektdurchführung

### 8.3. Konfigurationsmanagement

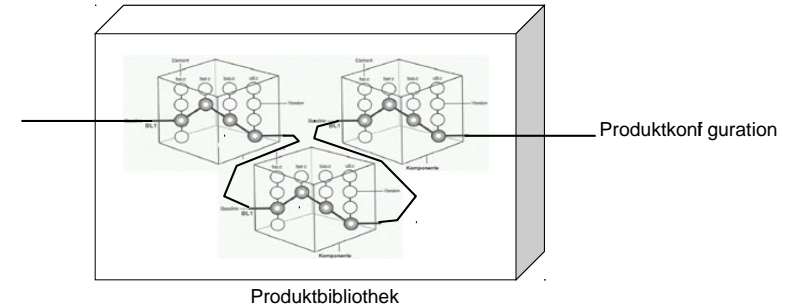
#### - Vorgaben für die Gliederung

- der Dokumentation
- der Software
- der Hardware

#### - Konfigurationsmaßnahmen

- Änderungsdienst
- Fehlerverfolgung
- Versionsverwaltung
- periodische Datensicherung
- Katastrophenschutzmaßnahmen
- Verwaltung sensibler Daten
- Führung der Projekthistorie

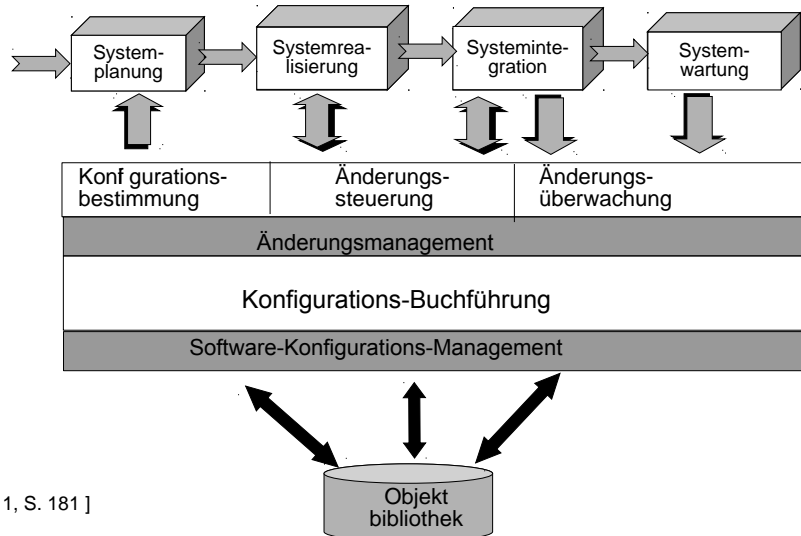
# Zusammenhang Produktbibliothek/Produktkonfiguration



Produktbibliothek:  
Produktkonfiguration:

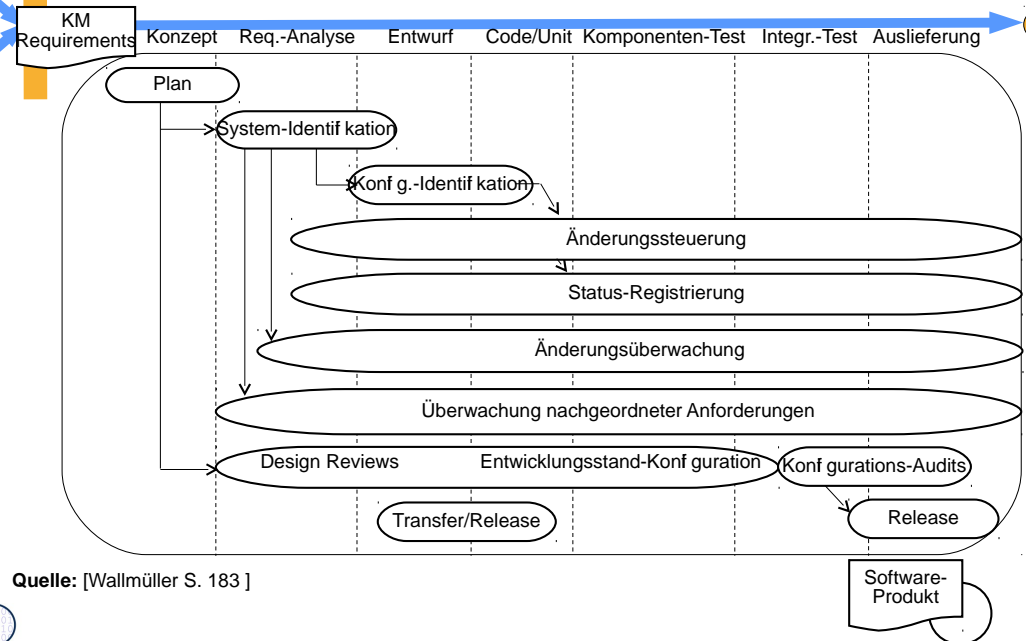
Menge aller Komponenten und deren Versionen etc  
Menge von (Produkt-)versionen

# Hauptfunktionen des KM



Quelle: [ 1, S. 181 ]

# Prozess des Konfigurationsmanagements



Quelle: [Wallmüller S. 183 ]

# Werkzeuge zum Konfigurationsmanagement

41

Mit Dateibaum-basierter Produktbibliothek:  
CVS

Open Source Project, Per Cederquist  
<http://www.cvshome.org>

Mit Datenbank-basierter Produktbibliothek:  
ClearCase

IBM/Rational  
<http://www.rational.com/products>

Visual SourceSafe - Microsoft

<http://www.eu.microsoft.com/germany/produkte>

Mit beidem:  
subversion

Subversion portal <http://subversion.tigris.org>

Andere:  
Telelogic Synergy

IBM  
<http://www.telelogic.com/product/synergy>

in-Step

microTOOL GmbH, Berlin  
<http://www.microTOOL.de>

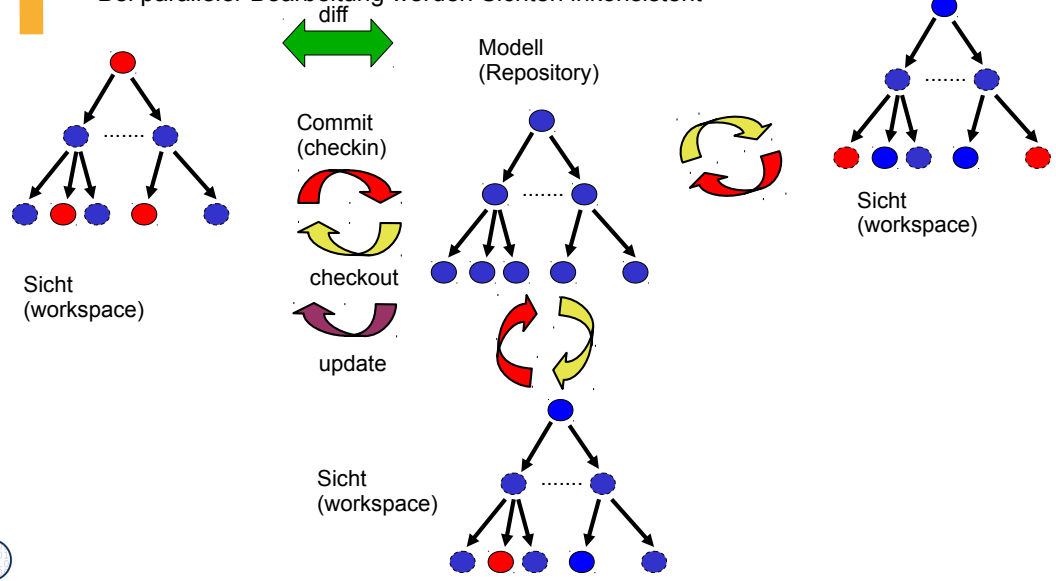
# 21.3.1 Dateibaumbasierte Konfigurationsmanagement-Werkzeuge

42

# Zusammenarbeitdimension: Views and Models

43

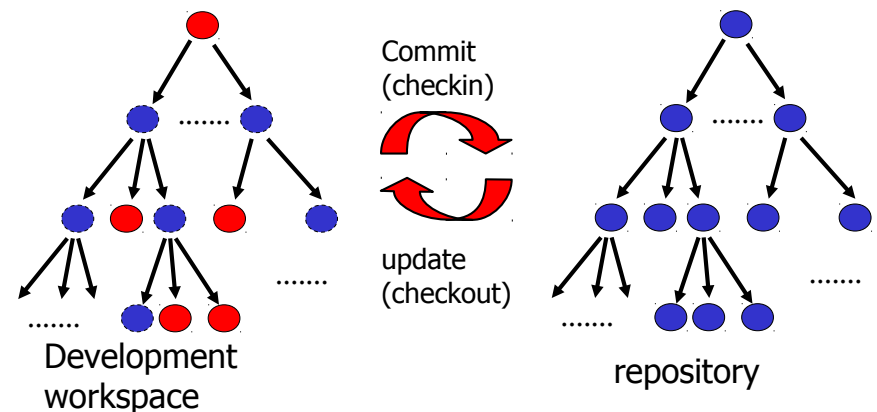
Bei paralleler Bearbeitung werden Sichten inkonsistent



# Einfache Sichten in cvs

44

Nur Tiefe 1 möglich



## Beschränkungen 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

45

## 21.3.2 Subversion – Ein verteiltes Konfigurationsmanagementsystem

46

## Hilfe

- ▶ `svn --help`
- ▶ `svn <command> --help`
- ▶ `svnadmin --help`
- ▶ `svnadmin create --help`

47

## 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 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/

48

Softwaremanagement, © Prof. Uwe Aßmann

## Commit und Rollback

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

49

## 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 .`

50

## Move in Sicht und im Modell

- ▶ `svn copy <subtree>`
- ▶ `svn move <subtree>`
- ▶ `svn delete <subtree>`

51

## Info über Sichten

- ▶ `svn log`
- ▶ `svn diff`: vergleicht mit jungfräulichen Kopien in `.svn`
  - arbeitet inkrementell auch auf binären Dateien
- ▶ `svn cat`
- ▶ `svn list (svn ls)`
- ▶ `svn status`

52

## Jenseits von Transaktionen

- ▶ Falls andere in 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)

53

## Meldungen des Aktualisierungsalgorithmusses

- alles gut gegangen
  - 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
  - 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

54

## Konfliktauflösung

- ▶ wie bei cvs oder
- ▶ mit 3 speziellen files
  - 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

55

## 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://

56

## Der svn Cache

- ▶ Der svn Cache erlaubt es,
  - die BASE Version, die in .svn gespeichert wird, zu vergleichen
  - "spät" zurückzuschreiben
  - Vermeidet direkte commits über das Netzwerk (was nicht vorhanden sein könnte)
    - Commits werden in das Metadatendirectory .svn geschrieben
    - alle Kommandos funktionieren trotzdem

57

## 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

58

## Repositories (Produkt- und Artefaktbibliotheken)

- ▶ Datenbankbasiert mit Berkeley-DB
  - nicht portable 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>`

59

## Kritik an Subversion

- ▶ Noch immer
  - kein Variantenmanagement
  - keine Komponentenselektion
  - keine Unterstützung von automatischen Builds

60

## 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.”

- ▶ na ja, man kann schon wesentlich mehr!
  - Branchmanagement (Ketten von Sichten)
  - long runs
  - ACID
  - Web

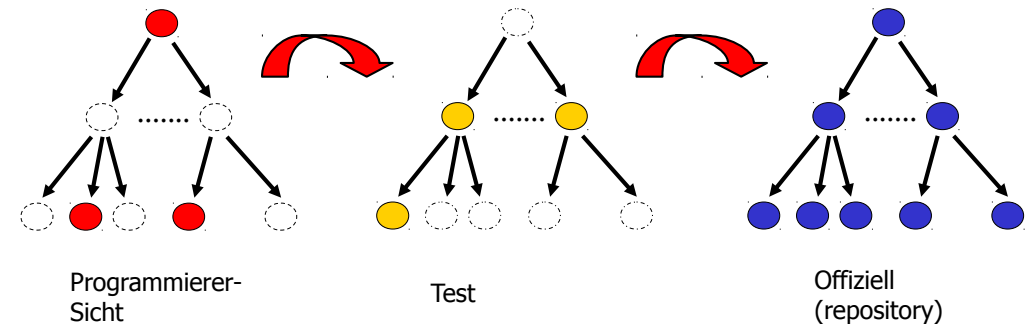
## Tools

- ▶ 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
  - Raptidsvn
  - esvn
  - supervision
  - subclipse, Svn4Eclipse for Eclipse
- ▶ Wie man zu Subversion wechselt
  - Cvs2svn
  - Prcs2svn
  - vss2svn

## 21.3.3 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" macht

65

- ▶ Lege eine Sicht an

- ▶ Lege immer neue Versionen in der Sicht an

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

Wiederhole das rekursiv

Ketten von Sichten entstehen

## Das svn Entwicklungsmuster "Ketten von Sichten" (Branches)

66

- ▶ 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

67

- ▶ 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`

## Wie man mit Subversion auf einem Laptop offline arbeitet

68

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

The End

