Universität Zürich[UZH]

Institut für Informatik

Martin Glinz Thomas Fritz

Software Engineering

Kapitel 20

Software-Konfigurations-management

20.1 Grundlagen

20.2 Identifikation und Verwaltung

20.3 Version, Konfiguration, Release

20.4 Änderungswesen

20.5 Problemmeldewesen

Probleme

Ändern Sie noch eben schnell…

- Software ist scheinbar leicht änderbar
- Während der Entwicklung entstehen viele Artefakte in vielen Versionen
- Wird Software von mehreren Klienten eingesetzt, müssen Software-Produkte gebildet und unterhalten werden
- In der Pflege entstehen fortlaufend geänderte oder neue Artefakte
- Typische Probleme:
  - Paralleles, unkoordiniertes Ändern durch mehrere Personen
  - Verwendung nicht mehr aktueller Artefakte
  - Undokumentierte Schnellreparaturen an in Betrieb befindlicher Software

Probleme – 2

- Probleme wachsen überproportional mit der Anzahl der Komponenten
- ➪ Hohe Kosten

Das Gegenmittel heißt Software-Konfigurationsmanagement

Definitionen

Software-Konfigurationsmanagement (software configuration management) – Die Gesamtheit aller Verfahren zum Aufbau, zur Änderung und zur Überwachung von Konfigurationen eines Software-Systems

Software-Konfiguration (software configuration) – Eine konsistente Menge logisch zusammengehöriger Software-Einheiten.

Software-Einheit (software configuration item) – Der kleinste, im Rahmen des Konfigurationsmanagements als atomar behandelte Baustein einer Konfiguration.

- Als Ganzes identifiziert, registriert, freigegeben oder geändert
- Zum Beispiel Programm-Module und Dokumente

Aufgaben des Software-Konfigurationsmanagements

- Software-Einheiten registrieren, verwalten und versionieren
- Bilden und verwalten von Konfigurationen und Releases
- Änderungsmanagement
- Management von Problemmeldungen

- Software-Konfigurationsmanagements ist ein Teil des
  - Software-Projektmanagements in Entwicklungsprojekten
  - Software-Produktmanagements im Einsatz

20.1 Grundlagen

20.2 Identifikation und Verwaltung

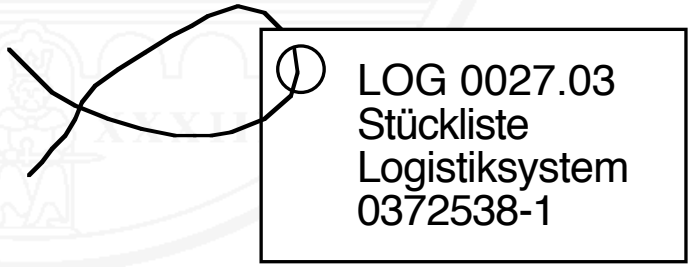
20.3 Version, Konfiguration, Release

20.4 Änderungswesen

20.5 Problemmeldewesen

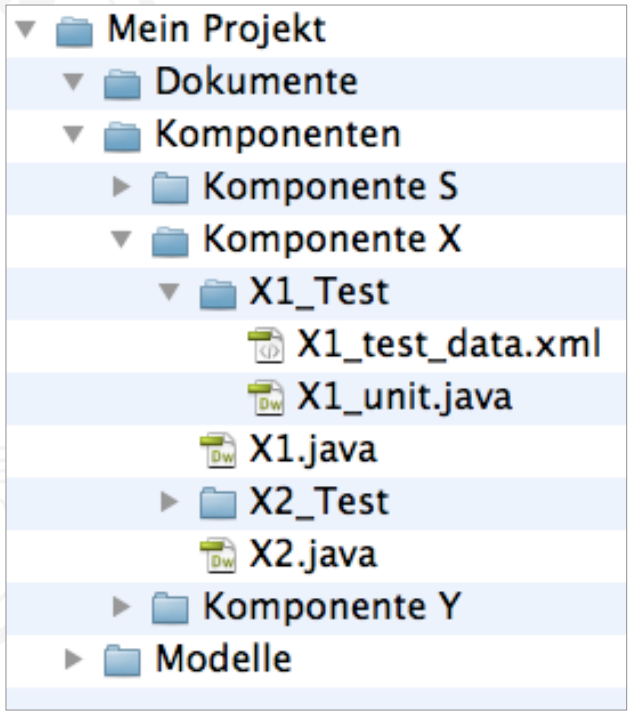
Kennzeichnung von Software-Einheiten

- Software-Einheiten haben eine eindeutige Kennzeichnung
- Besteht aus einem Namen und einer Versionsnummer
- Kann weitere Informationen enthalten, zum Beispiel Name des Systems oder Teilsystems
- Die Identität einer Software-Einheit ist feststellbar, z.B. mit Prüfsummen oder Hash-Codes



Registrierung und Verwaltung

- Software-Einheiten müssen registriert und verwaltet werden
- Typisch hierarchisch strukturiert als Verzeichnisbäume
- Bevorzugt mit Hilfe eines Konfigurations-managementsystems verwaltet
- Pro Einheit mehrere Versionen möglich



20.1 Grundlagen

20.2 Identifikation und Verwaltung

20.3 Version, Konfiguration, Release

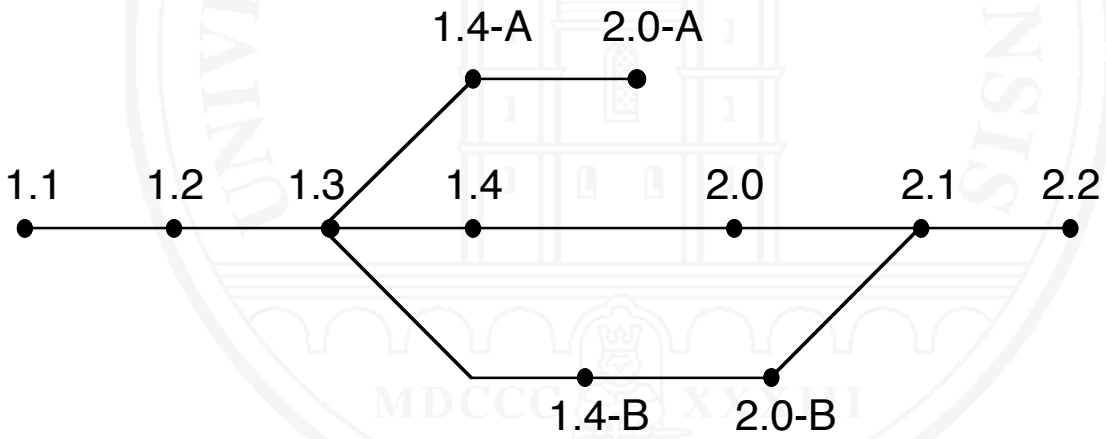
20.4 Änderungswesen

20.5 Problemmeldewesen

Versionierung

- Einfachste Art der Versionierung: aufsteigende Versionsnummern

- Im allgemeinen Fall: Revisionen (aufsteigend) und Varianten (parallel)



Konfiguration und Release

Konfiguration (configuration) – Eine konsistente Menge logisch zusammengehöriger Software-Einheiten

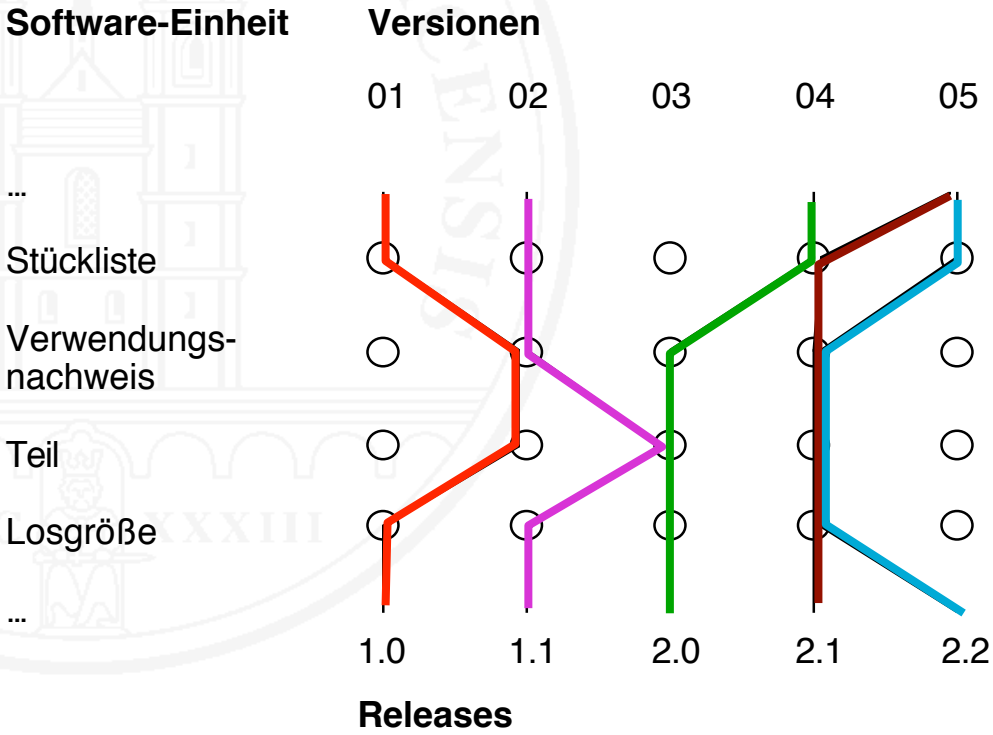
- Basis für lauffähige Software
  - Während der Entwicklung (Integrations- und Systemtest)
  - Zum Zweck der Auslieferung
- Beantwortet u.a. folgende Fragen:
  - Welche Software-Einheiten gehören dazu?
  - Wie wird ein lauffähiges System generiert?

Release – Eine zur Benutzung freigegebene Konfiguration

- Basis für die Auslieferung von Software an Kunden
  - Bildung von Software-Produkten
  - Periodische Lieferungen von Nachträgen und Verbesserungen

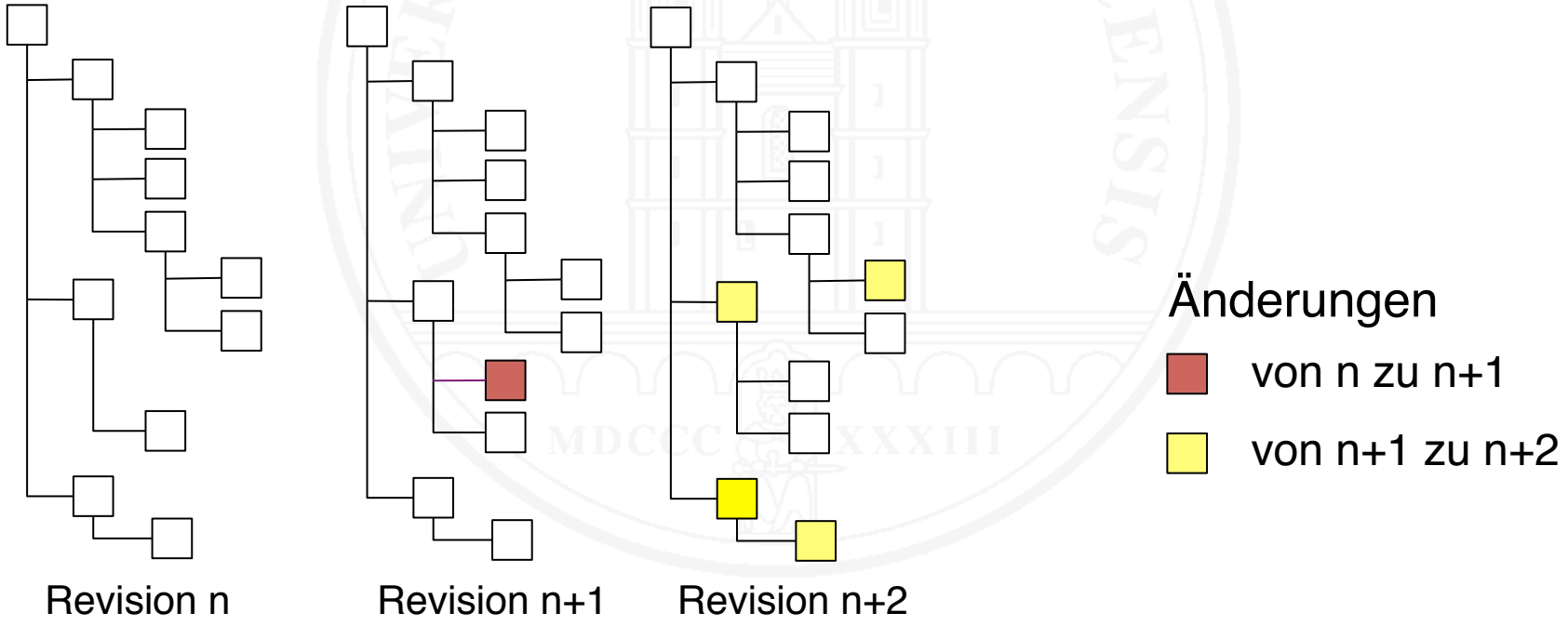
Klassische Versionierung von Konfigurationen

- Alle Software-Einheiten sind individuell versioniert
- Konfigurationen / Releases werden aus ausgewählten Software-Einheiten in bestimmten Versionen gebildet
- Konfiguration/Release wird ebenfalls versioniert



Vereinfachte Versionierung von Konfigurationen

- Keine individuelle Versionierung von Software-Einheiten
- Nur Konfigurationen werden versioniert
- Konfigurationen typisch als Verzeichnisbäume strukturiert



Zentral vs. verteilt

Zentrales Konfigurationsmanagement

- ein physisch zentralisiertes Repository als Referenz
- Benutzer checken einzelne Dateien aus, bearbeiten diese und checken die neue Version wieder ein
- Neue Konfigurationen nur im zentralen Repository gebildet

Verteiltes Konfigurationsmanagement

- n ko-existierende Repositories
- Benutzer erzeugt Klon eines vollständigen Repositories zwecks Bearbeitung
- Kann darauf individuell arbeiten und neue Konfigurationen bilden
- Verfahren zum Wiedervereinigen bearbeiteter Repositories nötig
- Typisch ist ein Repository als Referenz gekennzeichnet

Problem 1: zeitlich überlappende Änderungen

Zeit

Anna

Erstellt lokale Kopie der Klasse DemoCode.java aus Codebasis ✔

Modifiziert Methode blop ✔

Testet modifizierte Klasse ✔

Überschreibt DemoCode.java in Codebasis mit geänderter Klasse ✔

Bert

Erstellt lokale Kopie der Klasse DemoCode.java aus Codebasis ✔

Fügt Methode plup hinzu ✔

Testet modifizierte Klasse ✔

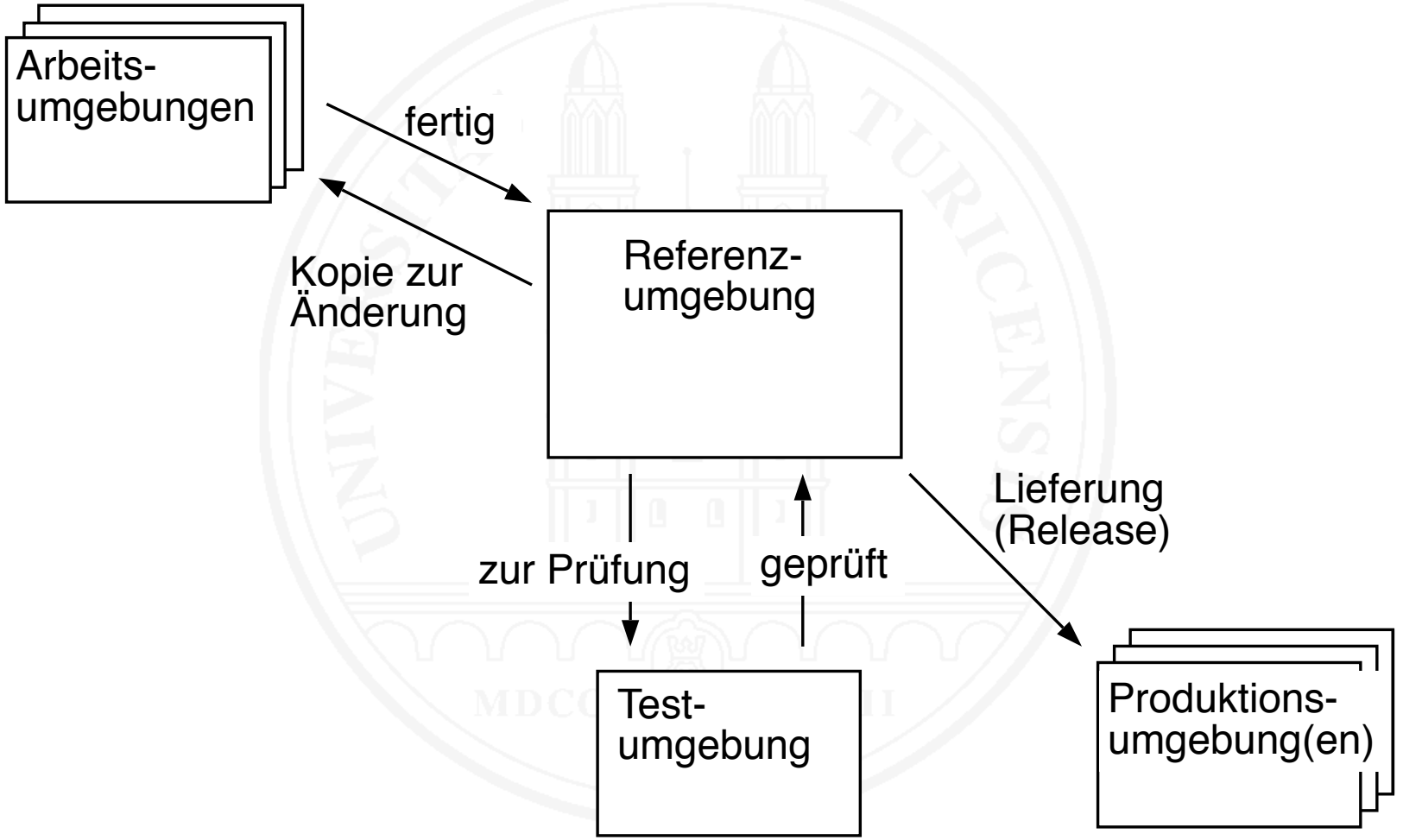
Überschreibt DemoCode.java in Codebasis mit geänderter Klasse ✘

Annas Änderungen sind verloren – und niemand merkt es sofort

Separate Umgebungen als Basis

- Getrennte Umgebungen für
  - Entwicklung (Arbeitsumgebung)
  - Verwaltung (Referenzumgebung, repository)
  - Test (Testumgebung)
  - Operativen Einsatz (Produktionsumgebung(en))
- Freie Änderungen nur lokal in Arbeitsumgebungen
- Reglementiertes Änderungsprozedere in der Referenzumgebung
  - „Pessimistisch“ durch Sperren
  - „Optimistisch“ durch Mischen (z.B. CVS, SVN)
  - „Kontrolliert optimistisch“ durch Einpflegen (z.B. Git/Github)
- Änderungen in Produktionsumgebungen nur durch Installation neuer Releases

Umgebungen und ihr Zusammenhang



Änderungsmanagement durch Sperren

Zeit

Anna

Bert

Sperrt Klasse DemoCode.java in RU ✔

Kopiert die Klasse in ihre AU ✔

Modifiziert und testet Methode blop ✔

Übergibt DemoCode.java an RU

RU publiziert neue Version von DemoCode.java und gibt Sperre frei ✔

Sperrt Klasse DemoCode.java in RU ✘

Wartet

Sperre gesetzt ✔

Kopiert die Klasse in seine AU ✔

Fügt Methode plup hinzu und testet ✔

Übergibt DemoCode.java an RU ✔

RU publiziert neue Version von DemoCode.java und gibt Sperre frei ✔

AU: Arbeitsumgebung
RU: Referenzumgebung

Änderungsmanagement durch Mischen

Zeit

Anna

Kopiert die Klasse DemoCode.java aus der RU in ihre AU ✔

Modifiziert und testet Methode blop ✔

Übergibt DemoCode.java an RU

RU publiziert neue Version von DemoCode.java ✔

Bert

Kopiert die Klasse DemoCode.java aus der RU in seine AU ✔

Fügt Methode plup hinzu und testet ✔

Übergibt DemoCode.java an RU

RU kennt Modifikation der Klasse durch Anna und weist Übergabe ab ✘

Mischt (gerne mit Werkzeughilfe) Annas Version von DemoCode.java mit seinen eigenen Änderungen ✔

Übergibt DemoCode.java erneut an RU

RU publiziert neue Version von DemoCode.java ✔

AU: Arbeitsumgebung
RU: Referenzumgebung

Mischen vs. Sperren

- Optimistisch: Mischen (merging)
  - Ermöglicht paralleles Arbeiten, erleichtert Zusammenarbeit
  - Potenziell unsicher
  - Nur möglich bei Artefakten mit mischbaren Änderungen (z.B. Code)
  - Kommunikation zwischen Beteiligten erforderlich
  - Einfügen in Referenzumgebung durch Änderungsprogrammierer
- Pessimistisch: Sperren (locking)
  - Behindert paralleles Arbeiten
  - Sicher
  - Einfügen in Referenzumgebung durch Verwalter der Referenz-umgebung
  - Für jeden Artefakttyp anwendbar

Änderungsmanagement durch Einpflegen

- Kombiniert die Vorteile von optimistischem und pessimistischem Änderungsmanagement:
  - Voll paralleles Arbeiten
  - Mischen parallel durchgeführter Änderungen
  - Änderungsprogrammierer fügt nicht selbst in Referenzumgebung ein, sondern stellt Übernahme-Antrag (pull request)
  - Produktverantwortlicher übernimmt „gute“ Änderungsanträge durch Mischen der Änderungen in die aktuelle Version der Referenzumgebung; weist „schlechte“ Änderungsanträge ab
- Produktverantwortlicher behält volle Kontrolle über Änderungen

Änderungsmanagement durch Einpflegen – 2

Zeit

Änderungs-
programmiererin

Produkt-
verantwortlicher

Erzeugt in ihrer AU einen Klon der RU

Führt Änderungen durch und testet

Beantragt Übernahme ihrer Änderungen in die RU

Produktverantwortlicher prüft Antrag und entscheidet

Positiv: Produktverantwortlicher übernimmt Änderungen durch Mischen in die RU; publiziert neue Version in RU ✔

Negativ: Produktverantwortlicher weist Antrag ab, ggf. mit Gründen bzw. Auflagen ✘

AU: Arbeitsumgebung
RU: Referenzumgebung

Problem 2: Änderung freigegebener Artefakte

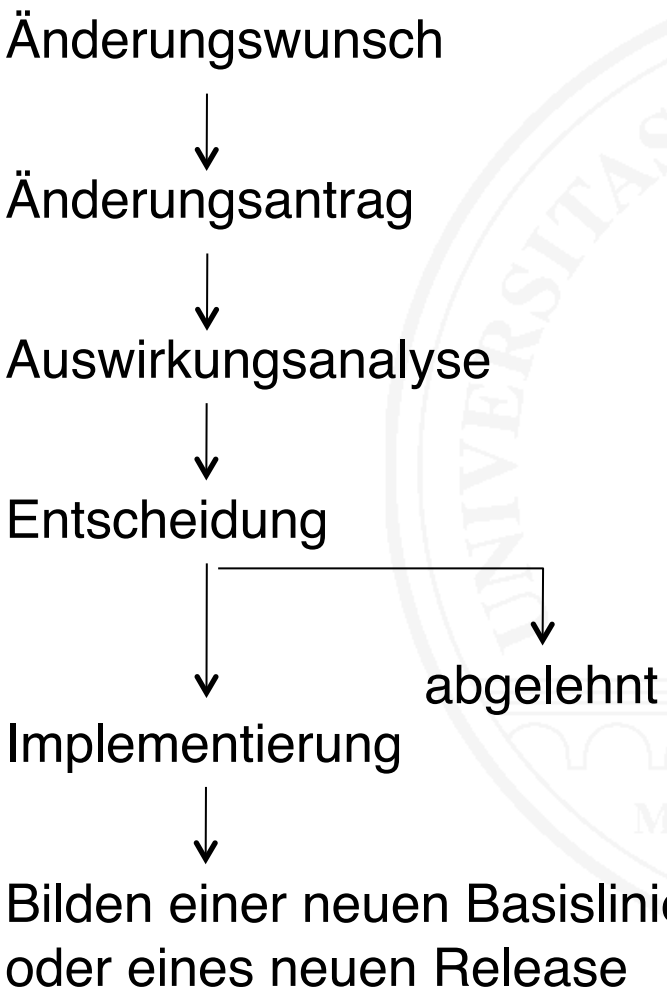
- Ist ein Artefakt freigegeben, zum Beispiel
  - ein Code-Modul, welcher Bestandteil eines Release ist
  - eine vom Auftraggeber formell gebilligte Version der Anforderungs-spezifikation

  so darf es nicht mehr unkontrolliert geändert werden
- Interne Freigaben werden mit Basislinien organisiert

Basislinie (baseline) – Eine freigegebene, nur kontrolliert änderbare Konfiguration von Artefakten.

- Die Änderung von Bestandteilen einer Basisline erfolgt nach einem strikt geregelten Änderungsprozess
- Notfallreparaturen müssen so rasch wie möglich durch ordentliche Änderungen ersetzt werden

Änderungsprozess für freigegebene Artefakte



- Beispiel: Kunde will eine Anforderung ändern
- Formular auszufüllen
- Machbar? Auswirkung auf vorhandene Artefakte? Auswirkung auf Kosten und Termine?
- Durch Change Control Board (besetzt mit Vertretern von Auftraggeber- und Auftragnehmerseite)
- Auftrag an Projektmitarbeiter; ggf. Änderung von Kosten- und Terminplan
- Formeller Abschluss der Änderung

20.1 Grundlagen

20.2 Identifikation und Verwaltung

20.3 Version, Konfiguration, Release

20.4 Änderungswesen

20.5 Problemmeldewesen

Das Problemmeldungswesen

- Systematische Behandlung von Kundenproblemen
- Kundenprobleme sind u.a.
  - Fehler
  - Anpassungsbedarf / -wünsche
  - Erweiterungsbedarf / -wünsche
  - Verbesserungsideen
  - ➔ reine Fehlerverfolgung (bug tracking) greift zu kurz!
- Grundlage: organisiertes Problemmeldungswesen
  - Problemmeldungsformular
  - Geordneter Bearbeitungsablauf (Problemmeldeprozess)

Problemmeldung – 1

Problemmeldung | **Nr.**

Verfasser

Name _____ Datum _____

Firma _____ Telefon / Fax / E-mail _____

Adresse _____

Betrifft

- ❑ Produkt _____
- ❑ Leistung _____
- ❑ anderes _____

Verwendete Hardware _____

Betriebssystem _____

Problem ist

	ja	nein
reproduzierbar	❑	❑
umgehbar	❑	❑

Problem betrifft

- ❑ Programme
- ❑ Unterlagen
- ❑ Leistungen

Antwort erwartet bis

_____

Problembeschreibung ❑ Problembeschreibung in Beilage

Problemmeldung – 2

Problembeschreibung ❑ Problembeschreibung in Beilage

Zu treffende Maßnahmen

Klassifizierung der Maßnahmen

Fehlerbehebung ❑
Anpassung ❑
Erweiterung ❑
Beratung/Info ❑
Schulung ❑

Verantwortlicher Sachbearbeiter

Name _____ Datum _____ Visum _____

Zwischenbescheid an Kunde

(erforderlich, wenn Meldung nicht bis zum vom Kunden erwarteten Termin erledigt werden kann)

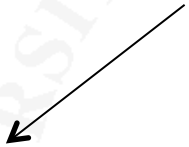
Datum _____ Visum _____

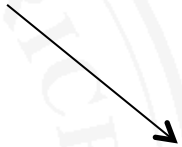
Problem erledigt und Kunde informiert _____

Name _____ Datum _____ Visum _____

Der Problemmeldeprozess

- Eingegangene Problemmeldung registrieren
- Problem analysieren und priorisieren
- Entscheidung: jetzt bearbeiten / später aufnehmen /nicht bearbeiten





- Problem zur Behebung zuweisen
- Problem beheben
- Gegebenenfalls neues Release bilden und ausliefern
- Problemmeldung abschließen und archivieren

- in Problemliste aufnehmen
- in der Releaseplanung Problemliste abarbeiten

- Problemmelder erhält Statusinformationen oder kann sie abfragen

Literatur

S. Chacon (2009) *Pro Git*. Online at http://http://git-scm.com/book

R. Conradi, B. Westfechtel (1998). Version Models for Software Configuration Management. ACM Computing Surveys 30(2):232–282.

K. Frühauf, J. Ludewig, H. Sandmayr (1999). Software-Projektmanagement und -Qualitätssicherung. Dritte, überarbeitete Auflage. Zürich: vdf.

J. Loeliger, M. McCullough (2012). *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*, 2nd edition. Sebastopol, Ca.: O'Reilly.

C.M. Pilato, B. Collins-Sussman, B.W. Fitzpatrick (2008). *Version Control with Subversion*, 2nd edition. Sebastopol, Ca.: O'Reilly. (auch erhältlich als online Buch: http://http://svnbook.red-bean.com)

A. Zeller, J. Krinke (2004). *Open-Source-Programmierwerkzeuge*. 2. Auflage. Heidelberg: dpunkt.