Softwaretechnik WS19/20

Allgemeine Informationen zum Ablauf der Vorlesung

DB ade, SWT juuccheeee

Naja fast …

Time Overruns	% of Responses
Under 20%	13.9%
21 - 50%	18.3%
51 - 100%	20.0%
101 - 200%	35.5%
201 - 400%	11.2%
Over 400%	1.1%

Cost Overruns	% of Responses
Under 20%	15.5%
21 - 50%	31.5%
51 - 100%	29.6%
101 - 200%	10.2%
201 - 400%	8.8%
Over 400%	4.4%

% of Features/Functions	% of Responses
Less Than 25%	4.6%
25 - 49%	27.2%
50 - 74%	21.8%
75 - 99%	39.1%
100%	7.3%

Chaos Report (1)

Flughafen Berlin Brandenburg

Baubeginn 2006

Eröffnungstermine:

- November 2011
- Juni 2012
- Herbst 2012
- August 2013
- Frühjahr 2014 – Teilweise
- Möglicherweise 2018
- Oktober 2020

Eigentlicher Plan: 2007

A2LL

- T-Systems und Herten (später ausgestiegen)
- Typische Webanwendung mit Applikationsservern
- Probleme:
  - Überlastung -> Schichtarbeit
  - Kontonummern wurden mit 0en aufgefüllt)
  - Straßennamen abgekürzt
  - Anmeldungen, Abmeldungen und Veränderungsmitteilungen zur Krankenversicherung aus unbekannten Gründen storniert
  - 25 Millionen Euro pro Monat zu viel an die Krankenkassen überwiesen
  - Berechnung des Zuschlags → Excel Tabelle

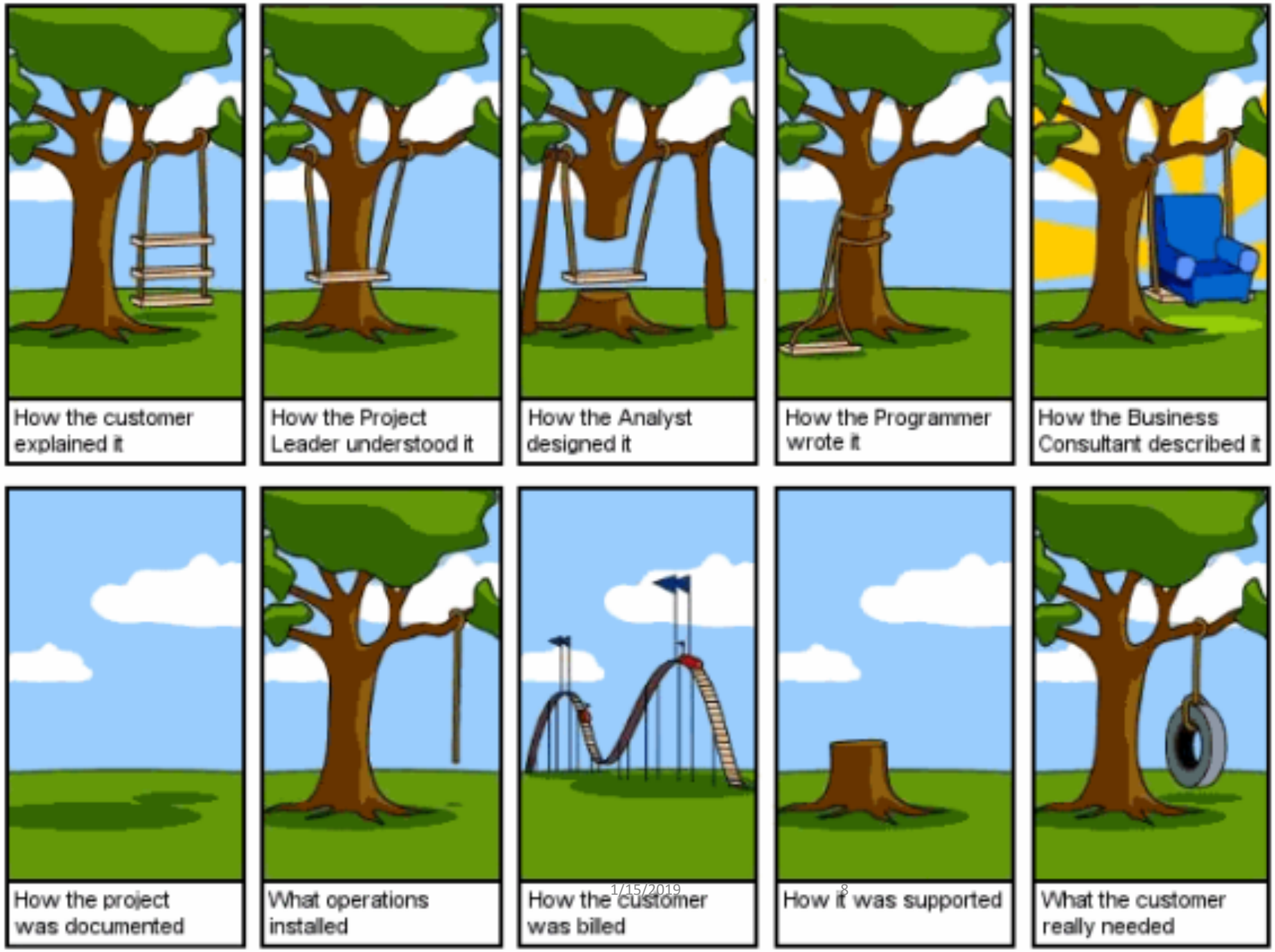
Chaos Report (2)

Project Impaired Factors	% of Responses
1. Incomplete Requirements	13.1%
2. Lack of User Involvement	12.4%
3. Lack of Resources	10.6%
4. Unrealistic Expectations	9.9%
5. Lack of Executive Support	9.3%
6. Changing Requirements & Specifications	8.7%
7. Lack of Planning	8.1%
8. Didn't Need It Any Longer	7.5%
9. Lack of IT Management	6.2%
10. Technology Illiteracy	4.3%
Other	9.9%

	Than 5 Years Ago	Than 10 Years Ago
Significantly More Failures	27%	17%
Somewhat More Failures	21%	29%
No Change	11%	23%
Somewhat Fewer Failures	19%	23%
Significantly Fewer Failures	22%	8%

Chaos Report (3)

Softwaretechnik



Ziele dieser Veranstaltung

- Vorbereitung auf die professionelle Software-Entwicklung
  - Entwicklung großer Softwaresysteme
    - Arbeitsteilig
    - Systematisch
  - Modellierung
  - Qualitätsmanagement
  - Vorgehensmodelle

Softwaretechnik

Die Kunst erfolgreiche Softwareprojekte durchzuführen

Best Practice lernen, um Zeit für eigene Innovationen zu haben

Vom Hacker zum Informatiker

Vorlesungsinhalte

Inhalte

- Software-Entwicklungsprozesse
  - Traditionell
  - Agil
- Analyse
  - Anforderungsanalyse
  - UML @ Anforderungsanalyse
  - Story Card
- Design
  - UML @ Design
  - Design Pattern
  - Architektur Pattern
- Entwicklung
  - Clean Code
  - Testbarer Code
- Test
  - Testverfahren
  - Teststrategien

Klausur

Klausur

- Notizen: 1 A4 Blatt, beidseitig per Hand beschrieben, keine Kopie
- Elektronischen Geräte sind nicht erlaubt
- Zettel werden gestellt
- Bitte keine Bleistifte, Holzstife und rote Stifte nutzen
- Probeklausuren werden im Laufe des Semester in studIP zur Verfügung gestellt.

Literatur

Allgemeine SWT Bücher

Software Engineering von Ian Sommerville
10th Edition

Lehrbuch der Softwaretechnik von Helmut Balzert

3. Auflage

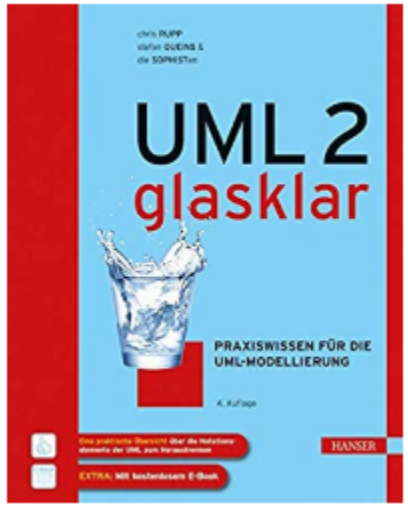
Bücher – Softwaretechnik Allgemein

S. Kleuker: Grundkurs Software-Engineering mit UML

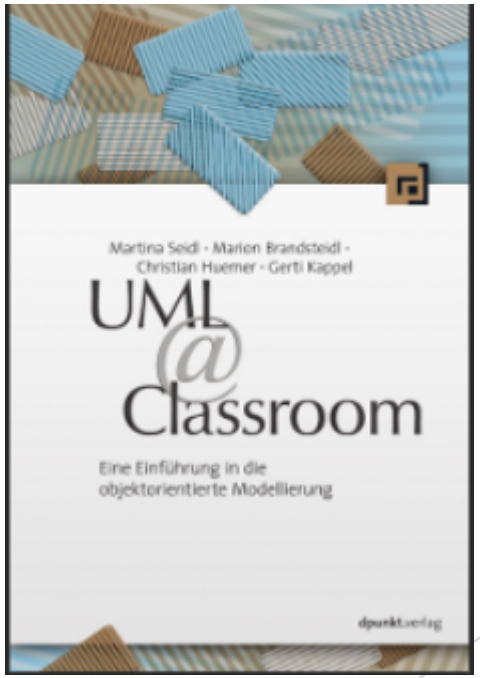
Download: http://dx.doi.org/10.1007/978-3-8348-9843-2

- van Vliet: Software Engineering: Principles and Practice.
- …

Bücher – Speziell UML



Viele Exemplare in der Bibliothek [BF 500 91]



Bücher – Speziell UML

- H. Störrle: UML 2 für Studenten.

  → Als Einführung geeignet – Leider teilw. nicht auf akt. Stand

- H.J. van Randen; et al.: Einführung in UML

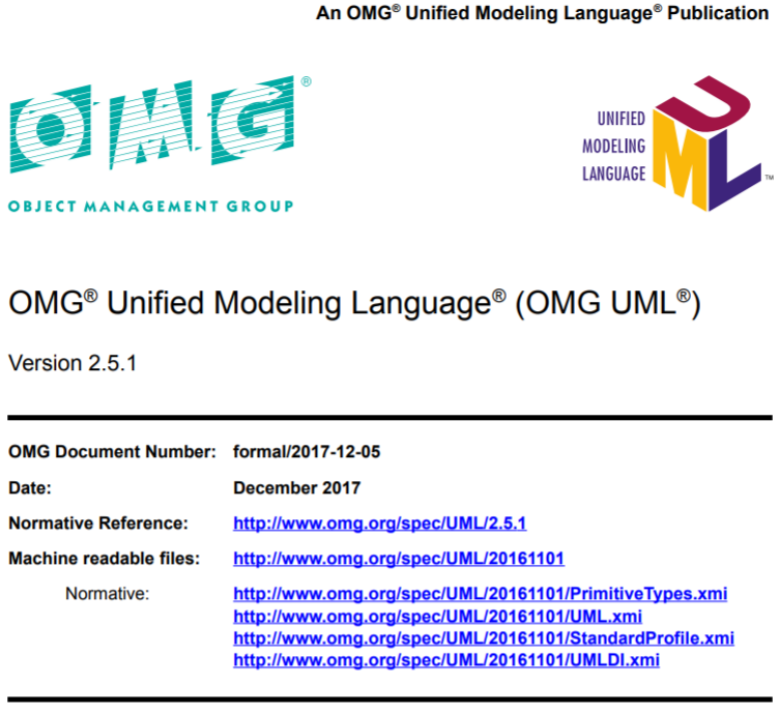
  Gratis in der Bib. verfügbar:

  https://hds.hebis.de/hsrm/Record/HEB386572232

- https://www.uml-diagrams.org

UML Spezifikation

- www.uml.org



An OMG® Unified Modeling Language® Publication

OBJECT MANAGEMENT GROUP

UNIFIED MODELING LANGUAGE

OMG® Unified Modeling Language® (OMG UML®)

Version 2.5.1

OMG Document Number:	**formal/2017-12-05**
Date:	**December 2017**
Normative Reference:	http://www.omg.org/spec/UML/2.5.1
Machine readable files:	http://www.omg.org/spec/UML/20161101
Normative:	http://www.omg.org/spec/UML/20161101/PrimitiveTypes.xmi http://www.omg.org/spec/UML/20161101/UML.xmi http://www.omg.org/spec/UML/20161101/StandardProfile.xmi http://www.omg.org/spec/UML/20161101/UMLDI.xmi

Elektronische Bücher in Unserer Bibliothek

- Zugriff auf eBooks
  - vom Hochschul-(W)LAN aus
  - von überall über Hochschul-VPN
- Lesen von eBooks
  - Online
  - Download PDF-Datei → offline

Der Anfang …

Die erste Entscheidung ist maßgeblich für den Erfolg des Projektes

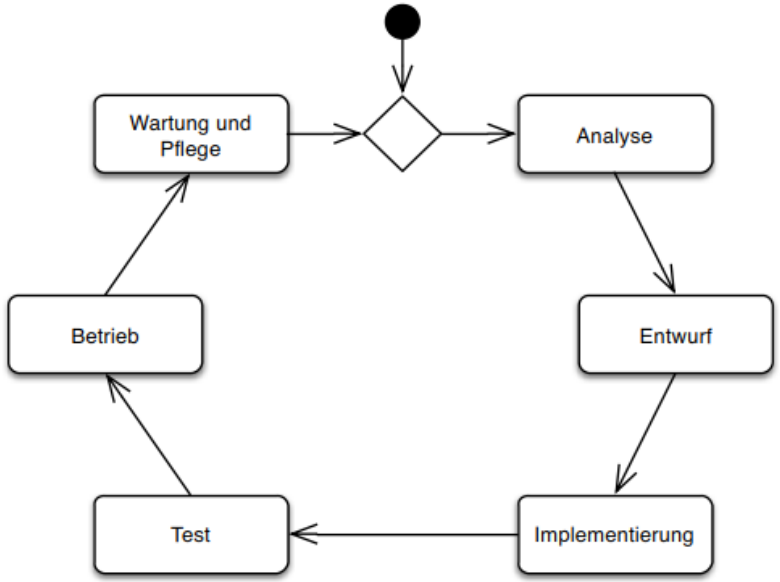
Typische Frage zu Anfang eines Projekts

- Wie gehe ich bei der Software-Entwicklung am besten vor?
  → Suche und nutze ein geeignetes Vorgehensmodell

- Muss ich bei der Software-Entwicklung alles mühsam von Hand machen?
  → Nein, es gibt eine Menge nützlicher Werkzeuge, die einen unterstützen

→ In dieser Vorlesungseinheit besprechen wir:
  - Vorgehensmodelle nochmals im Detail
  - Geeignete Werkzeuge zur Projektunterstützung

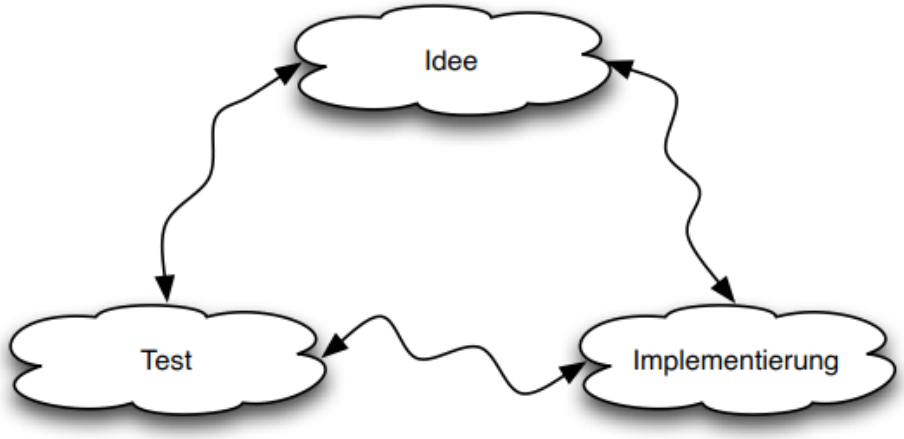
BSp für Vorgehensmodelle – Lebenszyklus von Software



→ Beschreibt die typischen Tätigkeiten bei der SW-Entwicklung

- einfache, klare Struktur
- sehr schlichtes Vorgehensmodell
- wichtigste Aussagen:
  - Software-Entwicklung umfasst 6 grundlegende Tätigkeiten
  - zyklisch
    - „Nach dem Spiel ist vor dem Spiel.“ [Sepp Herberger])

Bis jetzt arbeiten Sie so:



→ Sehr beliebt aber problematisch:

- Streng genommen kein Vorgehensmodell (zu allgemein)
- Verwandte Ansätze/Andere Namen:
  - Versuch und Irrtum (Trial and Error)
  - Hacken
- Sehr beliebt
- Manchmal sinnvoll: Neues ausprobieren, Prototypen, ...
- Sonst: problematisch
  → wächst einem schnell über den Kopf

Technische Infrastruktur

Ziel:
Geeignete Werkzeuge zur Unterstützung in Projekten kennen und nutzen

Technische Infrastruktur – Wofür brauchen wir das?

- Was meint technische Infrastruktur?
  - Sammlung an Werkzeugen (Programmen), die bei der SW-entwicklung helfen
  - Arbeiten hoffentlich so zusammen, dass Sie eine wirkliche Infrastruktur bilden

    → Möglichst keine Brüche

- Wofür ist das wichtig?
  - Effizientes Arbeiten & Strukturierung
  - Kollaboratives Arbeiten im Team

  → Bewältigung der aus den verschiedenen Artefakten und der daran arbeitenden Menschen entstehenden Komplexität

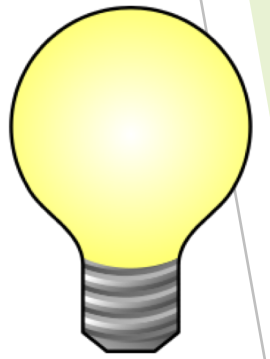
Technische Infrastruktur

- Die wichtigsten Werkzeuge in einem Entwicklungsprojekt:
  - Integrierte Entwicklungsumgebung (Eclipse IDE, NetBeans, …)
  - Testautomatisierung (JUnit, NUnit, PHPUnit, . . .)
  - Versionsverwaltung (Subversion (svn), git (derzeit sehr populär))

    → Wichtig für Konfigurationsmanagement (siehe folgendes Kap.)
  - Build-Automatisierung (make, ant, gradle, …)
  - Continuous Integration Server (Cruise Control, Trac)
  - Issue-Tracking (Bugzilla, Mantis, . . .)
  - Projektmanagement (Open Workbench, MS Project, . . .)
- Empfehlung:
  - Nicht zu viele **neue** Werkzeuge auf ein Mal
  - Lieber nach und nach einführen

Versionsverwaltung

Ziel:
Nutzen von Versionsverwaltungssystemen kennenlernen
→ Konfigurationsmanagement kennenlernen



VersionsVerwaltungswerkzeuge

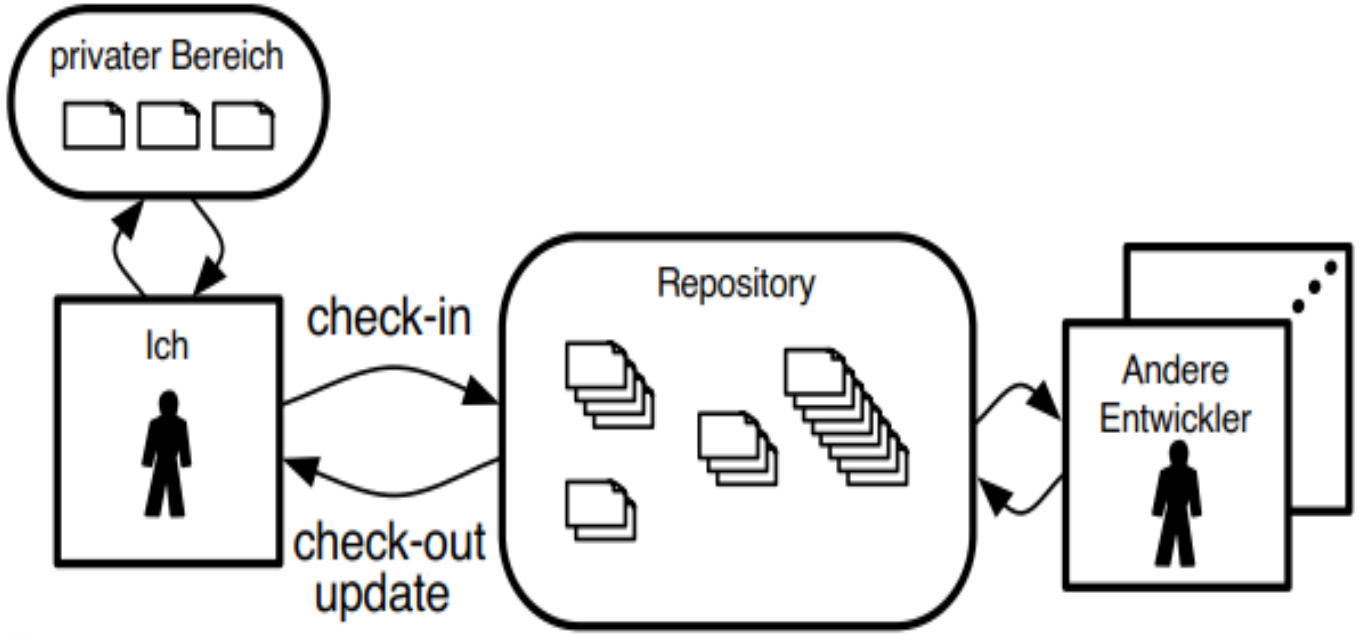
- Wofür brauchen wir eine Versionsverwaltung?
  - Backup-Restore
    - Wiederherstellen eines Standes, falls etwas kaputt gegangen ist
  - Nachvollziehbarkeit von Veränderungen
    - Stand der Datei X vor dem letzten Release
  - Parallele Entwicklung ermöglichen
  - Rekonstruktion vorheriger Konfigurationen
    - z.B. Quellcode-Dateien für Binaries, die mit Release XY geliefert wurden.

→ Versionsverwaltung ist **essentieller Bestandteil** einer **professionellen** Softwareentwicklung

- Nur Amateure & Bastler arbeiten ohne!!

VersionsVerwaltungswerkzeuge

- Bekannte Versionsverwaltungen:
  - Subversion (SVN) – Open Source, immer noch oft verwendet
  - Git – Open Source + neuere -revolutionäre- Konzepte
- Grober Aufbau einer Versionsverwaltung:



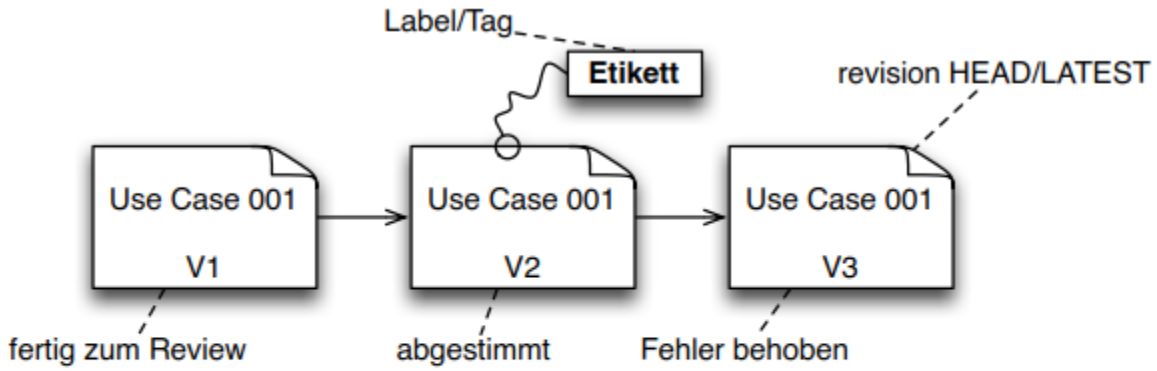
VersionsVerwaltungswerkzeuge

- Grundlegende Anforderungen:
  - Transparenz der Änderungen
    - Veränderung: Wer? Wann? Was?
  - Simultaner Zugriff + Konfliktlösung
  - Rekonstruktion
    - jederzeit beliebige Revision eines beliebigen Artefakts
    - jederzeit beliebige Konfiguration
  - Baselining / Labeling
    - Einfrieren von Ständen verschiedener Dateien zueinander (z.B. alle Stände der Dateien, die bei einem Release betroffen sind)

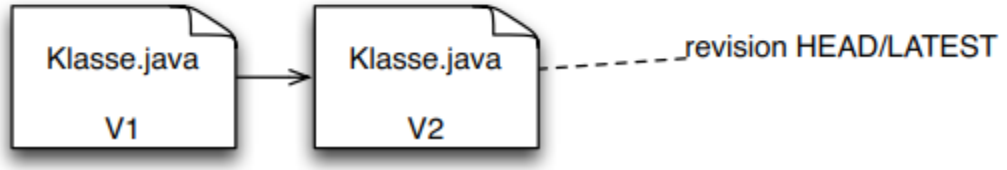
      → Bekommen ein sog. Label

Wie funktioniert die Versionierung von Dateien?

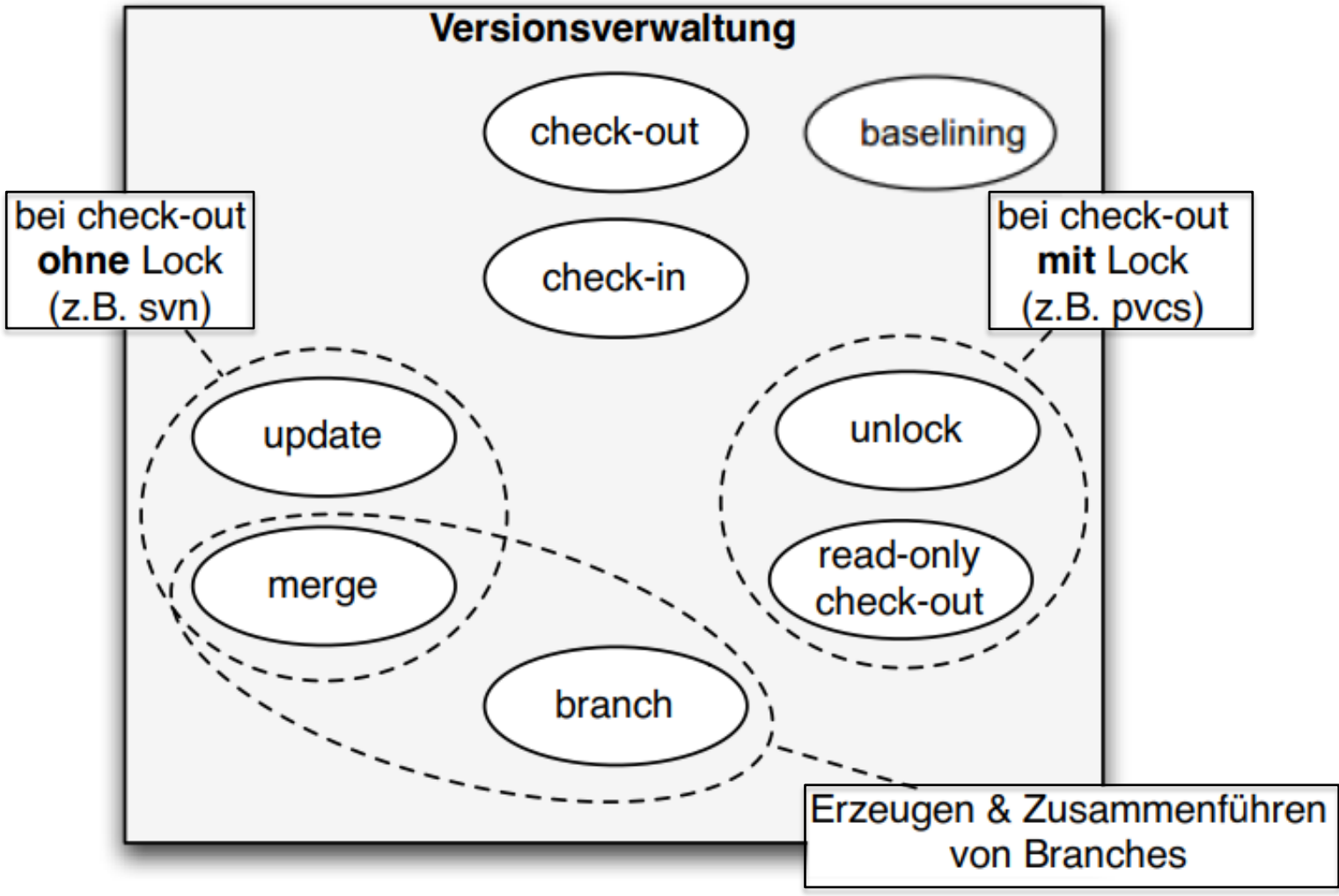
- Für jede Datei / Verz. wird ein Repräsentant (Head) angelegt
- Für jede Version der Datei /V wird eine sog. Revision erzeugt
  - Die Revision enthält die Datei in der jeweiligen Version
  - Daraus entsteht dann jeweils ein Historienbaum für jede Datei:



- Jede Re … s weiter gekennz …



VersionsVerwaltung – Grundlegende Use Cases



Konfigurationsmanagement

- Was sollte versioniert werden?

  → Alle Dateien, die mit einem Projekt zu tun haben

  - Code, Tests, Anforderungen, Designdokumente, Sonstige Texte, Möglichst alle Entwicklungswerkzeuge (haben auch Versionen!)

- Zu bestimmten Zeiten müssen alle Dateistände zueinander eingefroren werden → Baselines
  - Zu definierten Meilensteinen (meist vor und nach dem Review, …)
  - Zu bestimmten Meilensteinen müssen evtl. nur bestimmte Dateien zueinander versioniert werden (sog. Konfiguration)
- Zu freigegebenen Releases müssen evtl. noch Bugfixes, Patches, Service Packs nachgereicht werden

  → Sog. Branching erforderlich → Verzweigung der Entwicklung

  → Branches müssen später wieder in den Hauptzweig gemergt werden

Konfigurationsmanagement

→ Puh, das ist ja doch ganz schön kompliziert!

→ Deshalb gibt es in größeren Projekten meist mindestens einen Verantwortlichen, der hier den Überblick behält

→ Prozess wird als Konfigurationsmanagement bezeichnet

- Typische Tätigkeiten des Konfigurationsmanagements:
  - Konfigurationsmanagementplan entwickeln
    - Welche Dateien müssen versioniert werden?
    - Wann müssen welche Baselines gezogen werden?
    - Wie verläuft ein Branching Prozess?
    - …
  - Baselining durchführen, Probleme in der Versionsverwaltung lösen
  - Bei Branches den Rückmerge sicherstellen
  - …