Vorlesung Software Engineering

Foliensatz Nr. 6 (23.11.11)

Arbeitsgruppe Software Engineering
Prof. Elke Pulvermüller

Universität Osnabrück
Institut für Informatik, Fachbereich Mathematik / Informatik
Raum 31/318, Albrechtstr. 28, D-49069 Osnabrück

elke.pulvermueller@informatik.uni-osnabrueck.de

http://www.inf.uos.de/se

Sprechstunde: mittwochs 14 – 15 und n.V.

Inhalt

4 Konfigurationsmanagement

Inhalt

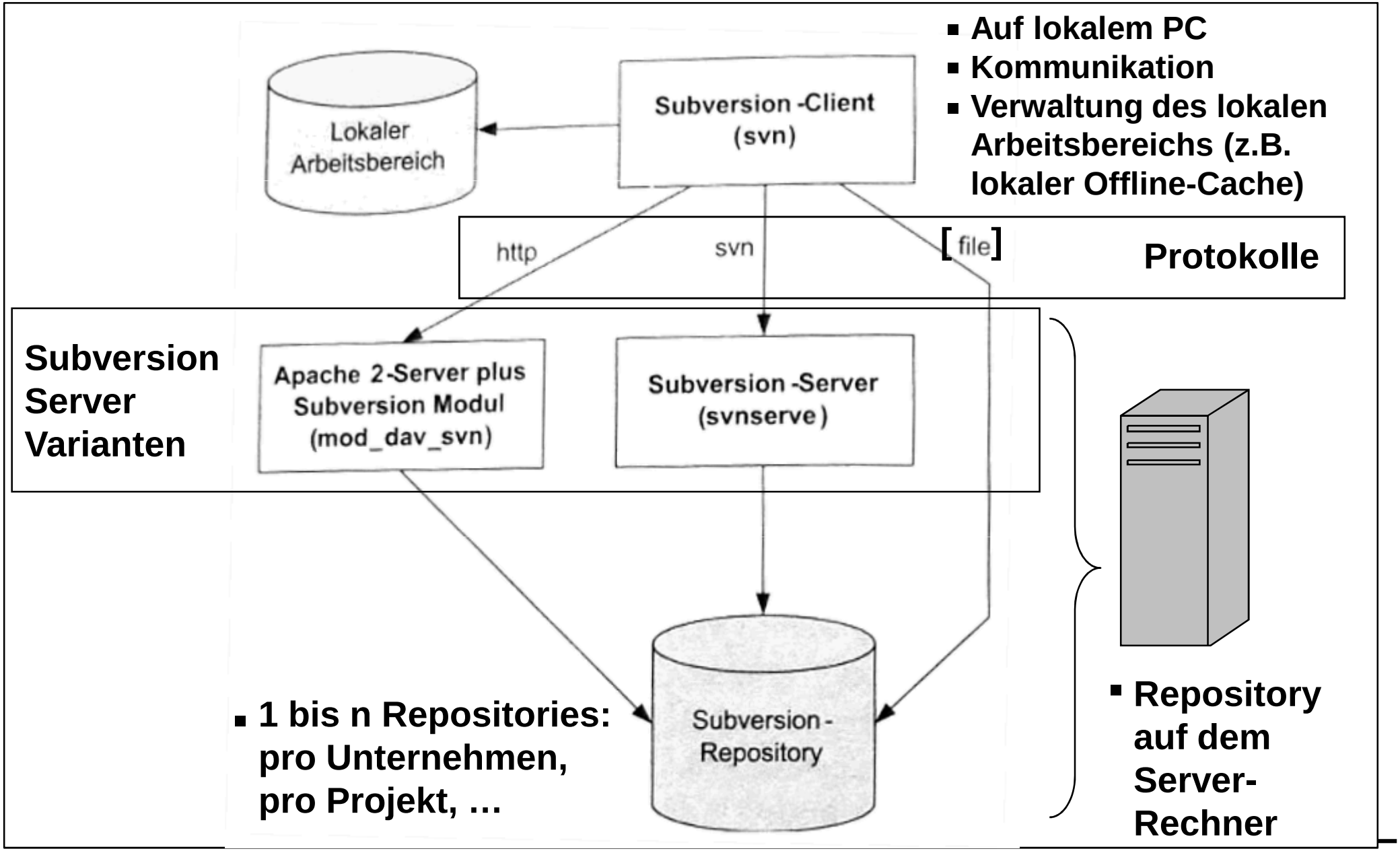
4.8 Werkzeug zur Versionskontrolle: Subversion

- **Versionskontrollsystem**
- **Frei verfügbar (http://subversion.tigris.org)**
- **Entwickler: CollabNet Inc., 2000 (erste Version 1.0: Februar 2004)**
- **Nachfolger von CVS (Concurrent Versions System), Open-Source**
- **Ziel: Beseitigung von Schwächen in CVS**

  - Versionierung von Dateien und auch Verzeichnissen
  - Atomarer Check-in (Transaktionen)
  - Versionierte Metadaten (Properties) für Elemente im Repository
  - Effiziente Deltabildung (auch für Binärdateien)
  - Entwicklung in C und mit der Apache Portable Runtime APR Bibliothek
    $\Rightarrow$ Plattformunabhängigkeit, für viele Betriebssysteme verfügbar
  - GUI Zusätze, Integration in Entwicklungsumgebungen,
    z.B. Subclipse http://subclipse.tigris.org
  - Client-Server-Architektur

4.8 Werkzeug zur Versionskontrolle: Subversion

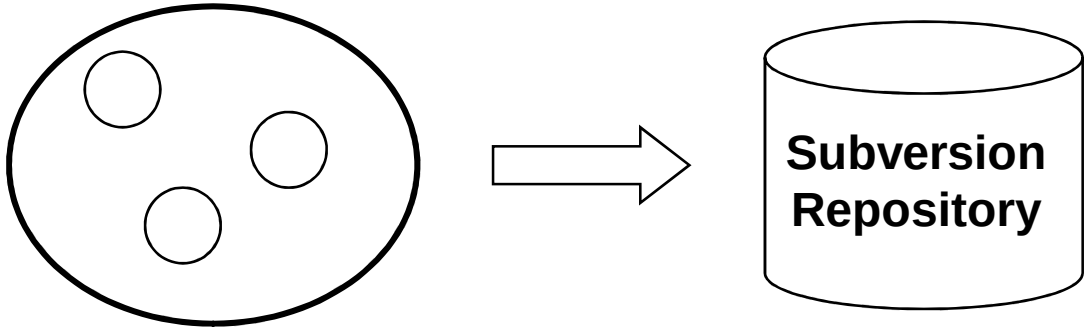
Subversion Client-Server-Architektur



Repository: Revision, ChangeSet

- **Nutzung eines Repositories, Prinzip der Changesets:**

Transaktion: Schreiben ins Repository
Automatisch vergebene, kennzeichnende Revisionsnummer X

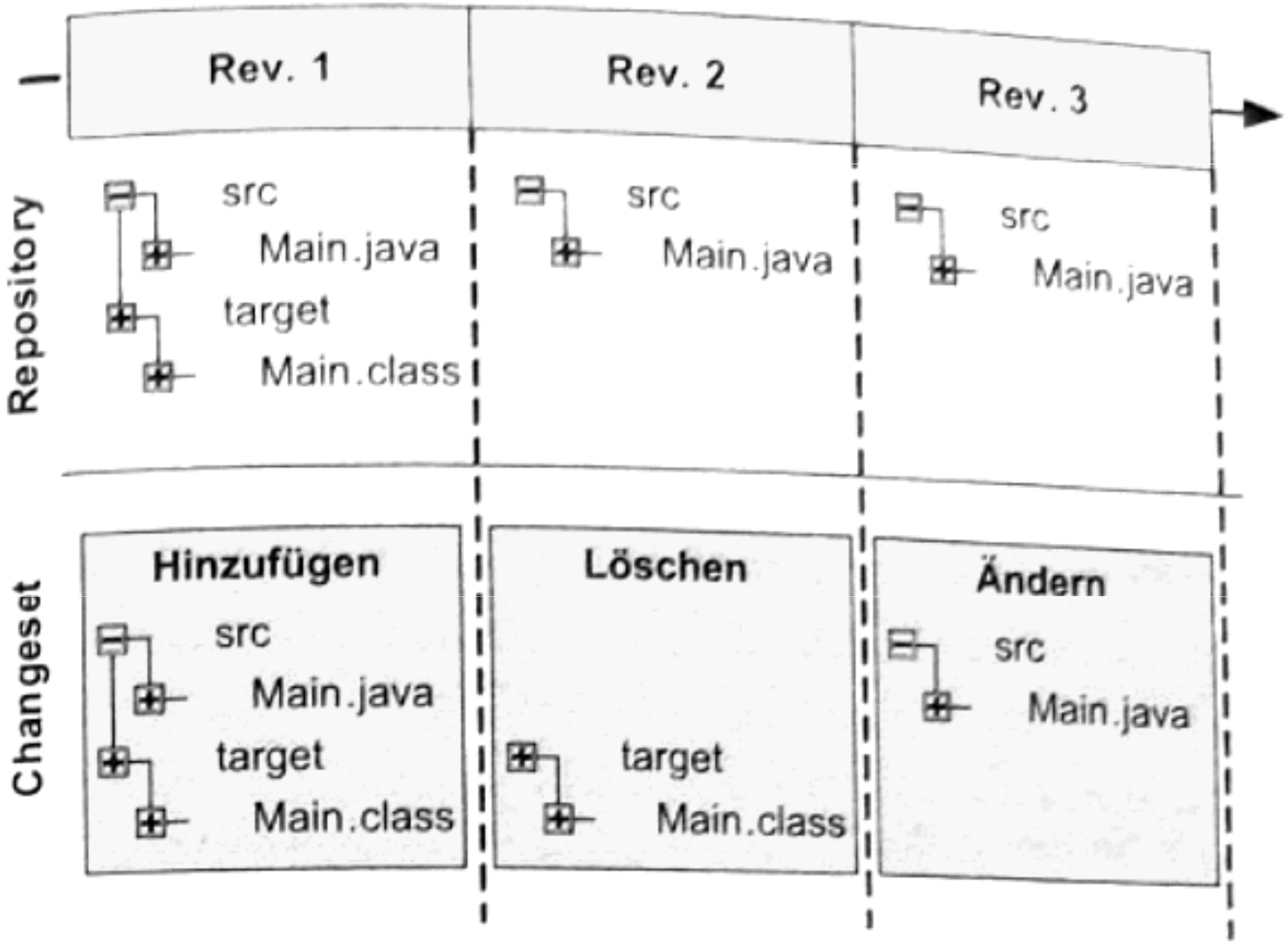


Changeset der Revision X:
Dateien und Verzeichnisse, die in der Transaktion geändert werden

Nach der Transaktion:
Repository in Revision X

- **Keine Versionsnummern auf Datei- und Verzeichnisebene, sondern nur Stand des gesamten Repositories (Revision)**

Repository: Revision, ChangeSet - Beispiel



4.8 Werkzeug zur Versionskontrolle: Subversion

UNIVERSITÄT OSNABRÜCK

Schritte zum Anlegen

- Leeres Repository anlegen, Revision 0 (svnadmin)
- Benutzer und Zugriffsrechte definieren (svnserve.conf, users.conf, access.conf)
- Starten eines Subversion Server svnserve
  Alternative: Direktzugriff durch den Client
- Client-Zugriff auf das Repository (über den Server):
  svn <Kommando> [<Optionen>][<Ziel>]
- Festlegung der Projektstruktur und der Konfigurationselemente (Konfigurationselemente, Release-Plan, Namens-Templates für Branches und Tags)
  Problem: Subversion kennt keine “echten” Tags und Branches
- Festlegung von Properties (einzeln oder via Konfigurationsdatei), Client-Konfiguration

Schritte zum Anlegen

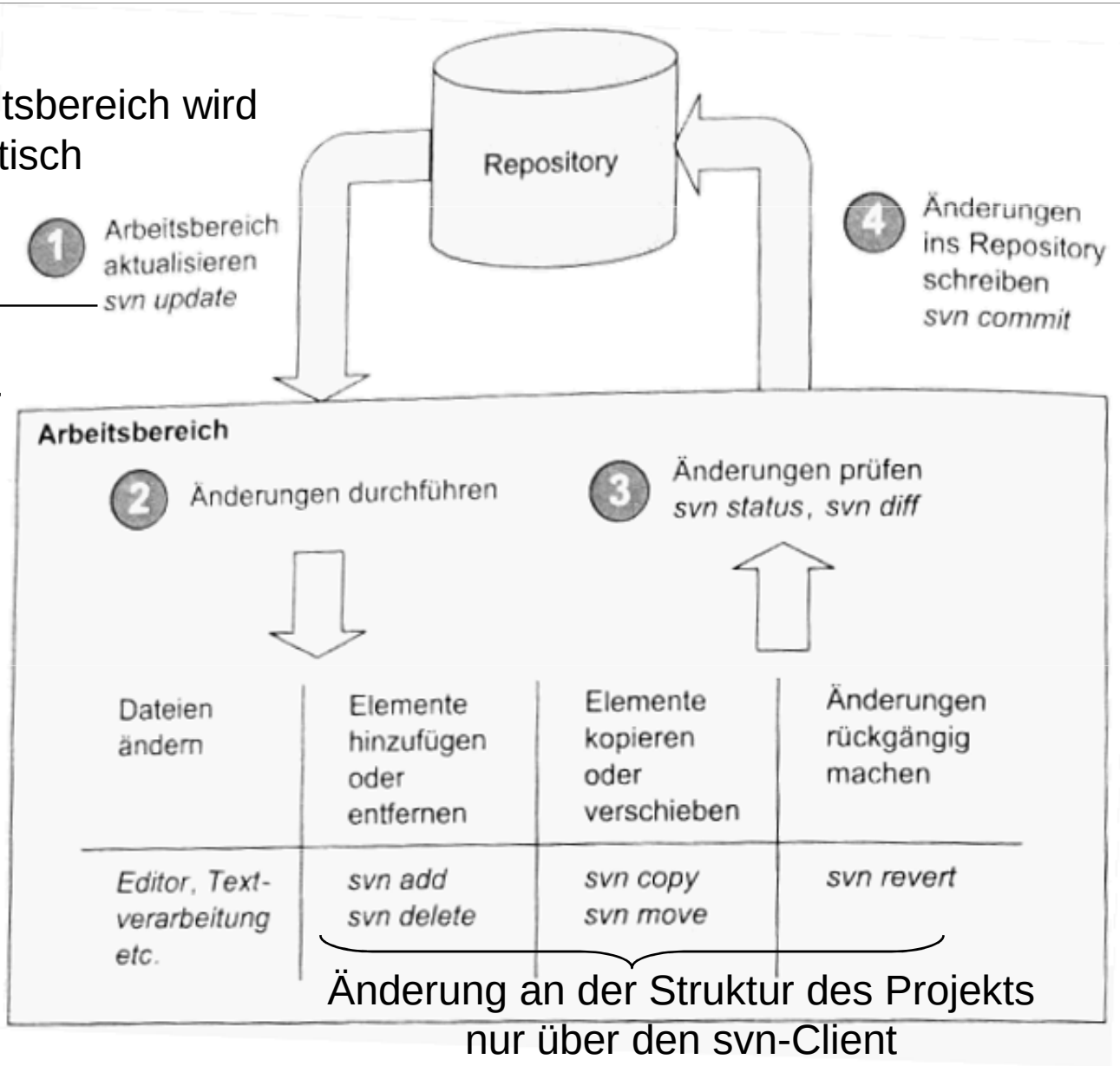
- Anlegen der Struktur im Repository: lokale, temporäre Erstellung der Projektstruktur und Import (→ Revision 1)
- Lokalen Arbeitsbereich beim Subversion-Client anlegen: Check-out

4.8 Werkzeug zur Versionskontrolle: Subversion

Arbeiten mit dem Subversion Repository:

Lokaler Arbeitsbereich wird nicht automatisch aktualisiert

Rückgabe des Statuscode (z.B. A: Datei ist gegenüber dem Arbeitsbereich neu)



Abfrage der Änderungen im lokalen Arbeitsbereich

Änderung an der Struktur des Projekts nur über den svn-Client

move = copy + delete

4.8 Werkzeug zur Versionskontrolle: Subversion

Arbeiten mit dem Subversion Repository:

- **Wiederherstellen durch Kopieren:**

svn copy -<Revisionsnummer> Repository-Pfad

Gewünschte Revision URL des Quellelements

- **Vorgehen zum Einspielen ins Repository (commit):**

1) Prüfung des Changesets gegenüber dem aktuellen Stand des Repositorys:
   (svn list +) svn status
2) Detailprüfung von Dateien:svn diff
3) Changeset ins Repository schreiben: svn commit

Mögliche Commit-Fehler:
- Technische Probleme (z.B. Netzwerkfehler)
- Fehlende Rechte
- Konflikt mit anderen Teammitgliedern und gesperrte Elemente

Repository-Konsistenz ist auch im Fehlerfall sichergestellt (→ Transaktion)

4.8 Werkzeug zur Versionskontrolle: Subversion

UNIVERSITÄT OSNABRÜCK

Weitere Aktivitäten:

- **Arbeiten mit der Versionshistorie: Logbuch und Archiv**
- **Umgang mit Konflikten beim Copy-Modify-Merge**
- **Umgang mit Sperrung beim Lock-Modify-Unlock**
- **Einsatz von Properties**
- **Anlegen und Arbeiten mit Tags und Branches**

4.9 Automatisierung des Build-Prozesses

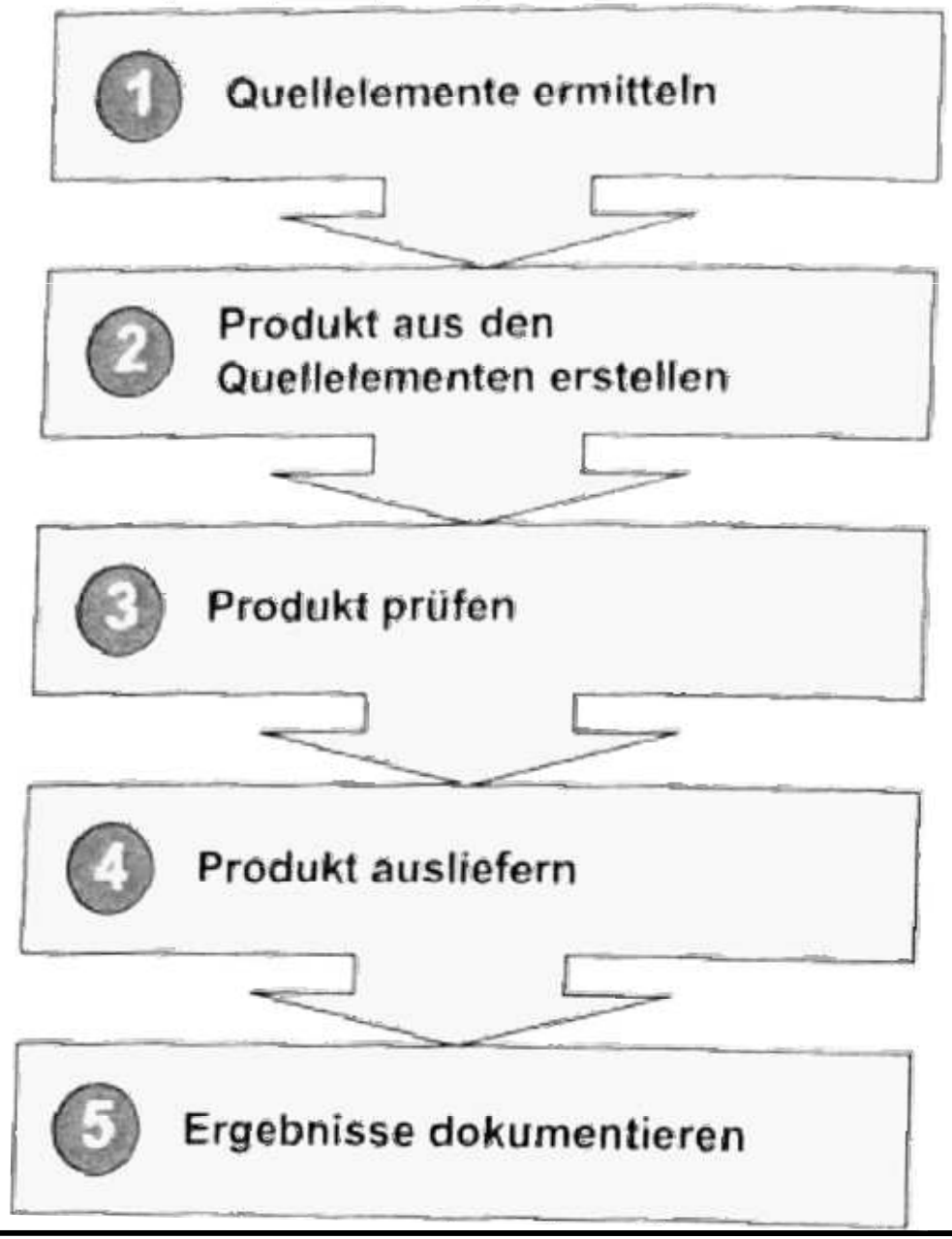
Alle Tätigkeiten, die man bereits zweimal manuell durchgeführt hat, sollten automatisiert werden.

[Clark, M.: Pragmatic Project Automation, How to Build, Deploy and Monitor Java Applications, The Pragmatic Starter Kit – Volume III, Pragmatic Bookshelf, 2004]

- **Projektautomatisierung:** **Automatisierung der Schritte zur Erstellung eines Produkts.**
- **Hilfsmittel:** **Build-Prozess**
- **Nutzen:**
  - ⇒ **Produktivität**
  - ⇒ **Wiederholbarkeit**
  - ⇒ **Reproduzierbarkeit**

4.9 Automatisierung des Build-Prozesses

Schritte:



Varianten:

- **Entwickler-Build**
- **Integrations-Build**
- **Produktions-Build**

4.9 Automatisierung des Build-Prozesses

Umsetzung – Werkzeuge zur Automatisierung:

- Entwicklungsumgebungen genügen nicht
- Skripte zur Beschreibung der Schritte (imperativ vs. deklarativ)
- Interpreter (z.B. Ant, Maven, Shell-Skripte)

4.9 Automatisierung des Build-Prozesses: Werkzeuge

- **Werkzeug zur Entwicklung von getrennten Modulen und deren Kombination bzw. zur Automatisierung des Build-Prozesses**

- **Ausgangsziel: Zusammenhänge zwischen verschiedenen Quellcodedateien erkennen und die benötigten neu übersetzen**

- **Automatisierungsaufgaben:**

- Effizientes Neuübersetzen aller Quellen
z.B. in C: Neuübersetzung der C-Datei, falls sich die inkludierte Header-Datei ändert

- Archiverstellung

- Erzeugen von Dokumentationen

- Aufräumarbeiten …

4.9 Automatisierung des Build-Prozesses: make

Werkzeug Make – der Klassiker

- **Beispiel für `makefile`:**

```
clean:
    rm *.o
    rm SampleSystem

all: Main.o F1.o F2.o F11.o F12.o
    g++ -o SampleSystem Main.o F1.o F2.o F11.o F12.o

Main.o: Main.cc Main.h F1.h F2.h F11.h F12.h
    g++ -c Main.cc

F1.o:  F1.cc F1.h F11.h F12.h
    g++ -c F1.cc
```

Pseudoziel (es soll nichts erzeugt werden)

Strophe (Stanza)

Primary Target

Ziel (Target)

Abhängigkeiten (Dependants): Quelle

Aktion/Befehl zur Erstellung des Ziels

Tabulatorzeichen!

Zum Vergleich ein Ziel mit Java:

```
Test.class : Test.java
    javac Test.java
```

Tausch des Compilers (z.B. cpp)
Besser: Einsatz von Variablen
Definition: CC = /user/bin/gcc
Nutzung: $(CC) –c F1.cc

Befehl wird nur ausgeführt, wenn eine Quelle neuer ist als das Ziel!
Topologisch sortierte Liste der Dateiabhängigkeiten

4.9 Automatisierung des Build-Prozesses: Ant

- Werkzeug zur Automatisierung des Build-Prozesses
- Entstehung im Open-Source-Projekt Tomcat (zunächst nur projektintern): Another Neat Tool
- 2000: erste öffentliche, frei verfügbar Version 1.1 (http://ant.apache.org)
- Verwendung: häufig in Java-Entwicklungsprojekten
  Aber: Keine Einschränkung auf eine bestimmte Progr.sprache oder Technologie
- Alternative v.a. für .NET-Entwickler: NAnt (http://nant.sourceforge.net)
- Besonderheit von Ant: Plattformunabhängigkeit
  Ant-Implementierung in Java (Java Bibliotheken), Skript-Sprache in XML
- Offene Architektur: Erweiterbarkeit um eigene Befehle mittels Java-API
- Nachfolge zu make, Vorgänger zu maven

Ein Projekt pro Ant-Skript (Wurzelelement)

Pro Projekt ein Default-Target und evtl. weitere Targets

Alle relative Pfadangaben starten in Verzeichnis basedir

```
<project name="MyProjectName" default="run" basedir=".">
   <property name="src.dir" value="${basedir}/src"/>
   <property name="bin.dir" value="${basedir}/bin"/>

   <target name="clean">
      <delete dir="${bin.dir}"/>
   </target>

   <target name="run"  ...
        </target>

   <target name="help"
       description="Hilfeausgabe" >
      <echo>
         z.B. Ausgabe der Tasksliste ...
      </echo>
   </target>
</project>
```

Property setzen

Property als spezielle Task (außerhalb von Targets notiert und gesetzte Variablen stehen allen Targets zur Verfügung)

Zugriff auf den Wert der Property

Target: beschreibt einen Teil des Build-Prozesses (Ausführung von Tasks) Target = Regel in make

Einzeilige Beschreibung der Target

4.9 Automatisierung des Build-Prozesses: Ant

```
<project name  …

   <target name="compile" depends="init">
      <javac srcdir="${src.dir}"
         destdir="${bin.dir}"
               classpath="${bin.dir}"
               debug="${compile.debug}" />
   </target>

   <target name="init">
      <delete dir="${bin.dir}"/>
      <mkdir dir="${bin.dir}"/>
   </target>

   <target name="run" depends="compile">
      <java classname="HelloWorld" >
               <arg value="keines"/>
      </java>
</project>
```

Depends: zunächst wird Target compile und init ausgeführt und anschließend run (falls compile und init erfolgreich waren)

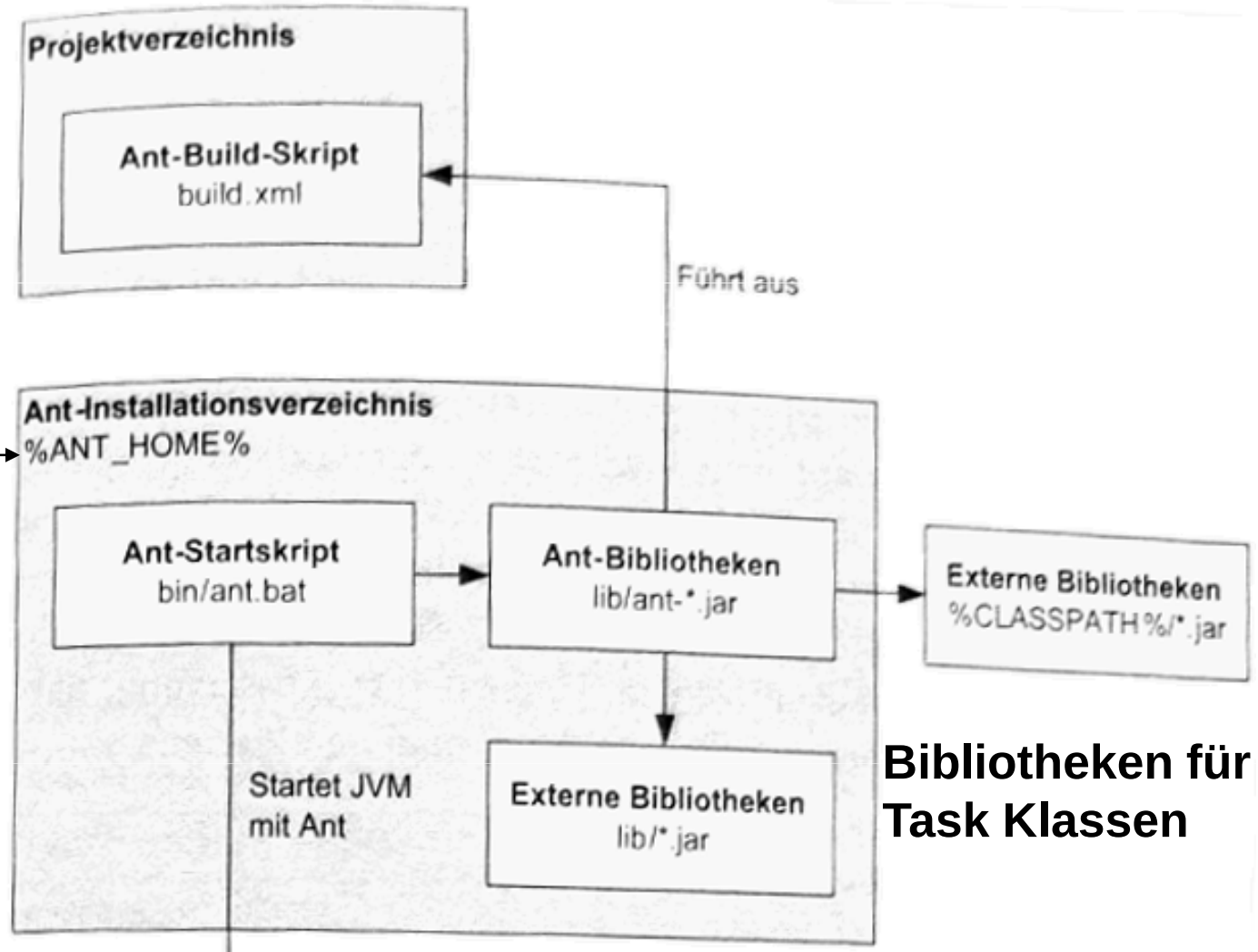
Mehrere Abhängigkeiten möglich: Targets werden durch Kommata getrennt z.B. depends=“clean,sth”

Abhängige Targets werden in der richtigen Reihenfolge und nur einmal ausgeführt

Target Namenskonventionen: (prinzipiell aber frei wählbar)

init	**Erstellt Verzeichnisse und Initialisierungsdateien**
build	**Inkrementeller Aufbau**
test	**Ablaufen der Tests mit JUnit**
clean	**Ausgabeverzeichnisse, -dateien löschen**
deploy	**Archive erstellen (z.B. jar)**
publish	**Veröffentlichen der Ergebnisse**
fetch	**Bezieht die letzten Quellcodedateien vom Cvs-Server**
docs, javadocs	**Erstellt die Dokumentation**
all	**Abfolge von clean, fetch, build, test, docs, deploy**
main	**Erstellt das Projekt, in der Regel build oder build, test**

Ant-Architektur



Umgebungsvariable, die auf das Ant-Installationverzeichnis zeigt

Bibliotheken für Task Klassen

Umgebungsvariable, die auf das JDK Installations-verzeichnis zeigt (mind. JDK 1.2)

Einlesen des Build-Skripts mit Namen build.xml im aktuellen Arbeitsverzeichnis

4.9 Automatisierung des Build-Prozesses: Ant

UNIVERSITÄT OSNABRÜCK

Tasks:

- **Übersicht über die in einem Build.xml enthaltenen Tasks:**

```
ant -projecthelp
```

- **Was ist eine Task?**

```
<target name="build">
    <javac srcdir="." />
</target>
```

Taskname → javac

Attribut → srcdir

Attributwert → "."

Task = Klasse bzw. ausführbares Codestück (vordefiniert in Ant, importieren oder selbst geschrieben)

In Ant vordefinierte(r) Task
http://ant.apache.org/manual/tasksoverview.html

Andere z.B. jar, javadoc, exec, copy, echo

- **Eigene Tasks schreiben (Taskname und Java Klasse und Package, die die Task implementieren)**

Bemerkung: eigene Tasks sind besser als der Aufruf externer, spezifischer Programme (⇒ Plattformunabhängigkeit)

Eigene Tasks schreiben:

```
import org.apache.tools.ant.Task;
import org.apache.tools.ant.BuildException;
public class HelloWorld extends Task {
    String message;
    public void setMessage(String msg) {
        message = msg;
    }

    public void execute() {
        if (message==null) {
            throw new BuildException("No message set.");
        }
        log(message);
    }
}
```

Klasse in Ant integriert (z.B. Unterstützung des Logging u.v.m.)

Ant Namenskonvention

Build-Failes-Message, falls kein Parameter vorhanden ist

Einbindung/Benutzung:

Task-Klasse liegt nicht im Ant /lib-Verzeichnis (built-in Property in Ant)

```
  <target name="use" description="Use the Task">
      <taskdef name="helloworld"
               classname="HelloWorld„
               classpath="${ant.project.name}.jar"/>
      <helloworld message="Hello World"/>
  </target>
```

Eigene Task definieren

Verwendung der Task

4 Konfigurationsmanagement

Zusammenfassung und Ausblick

1 Software-Krise und Software Engineering
2 Grundlagen des Software Engineering
3 Projektmanagement
4 Konfigurationsmanagement
5 Software-Modelle
6 Software-Entwicklungsphasen, -prozesse, -vorgehensmodelle
7 Qualität
8 … Fortgeschrittene Techniken

- 4.1 Motivation und Begriffe
- 4.2 Aufgaben und Verfahren
- 4.3 Konfigurationselemente
- 4.4 KM Plan
- 4.5 Projektstruktur
- 4.6 Verwaltung der Konfigurationselemente
- 4.7 Release-Management
- 4.8 Werkzeug Subversion
- 4.9 Build-Prozess

→ Softwareentwicklung in Projekten benötigt **Modelle und Modellierungssprachen zur Gestaltung und Dokumentation von Systemen**