

Kapitel 2. **Software Configuration Management mit CVS und SVN**

Motivation und Grundbegriffe

Arbeiten mit SVN und CVS

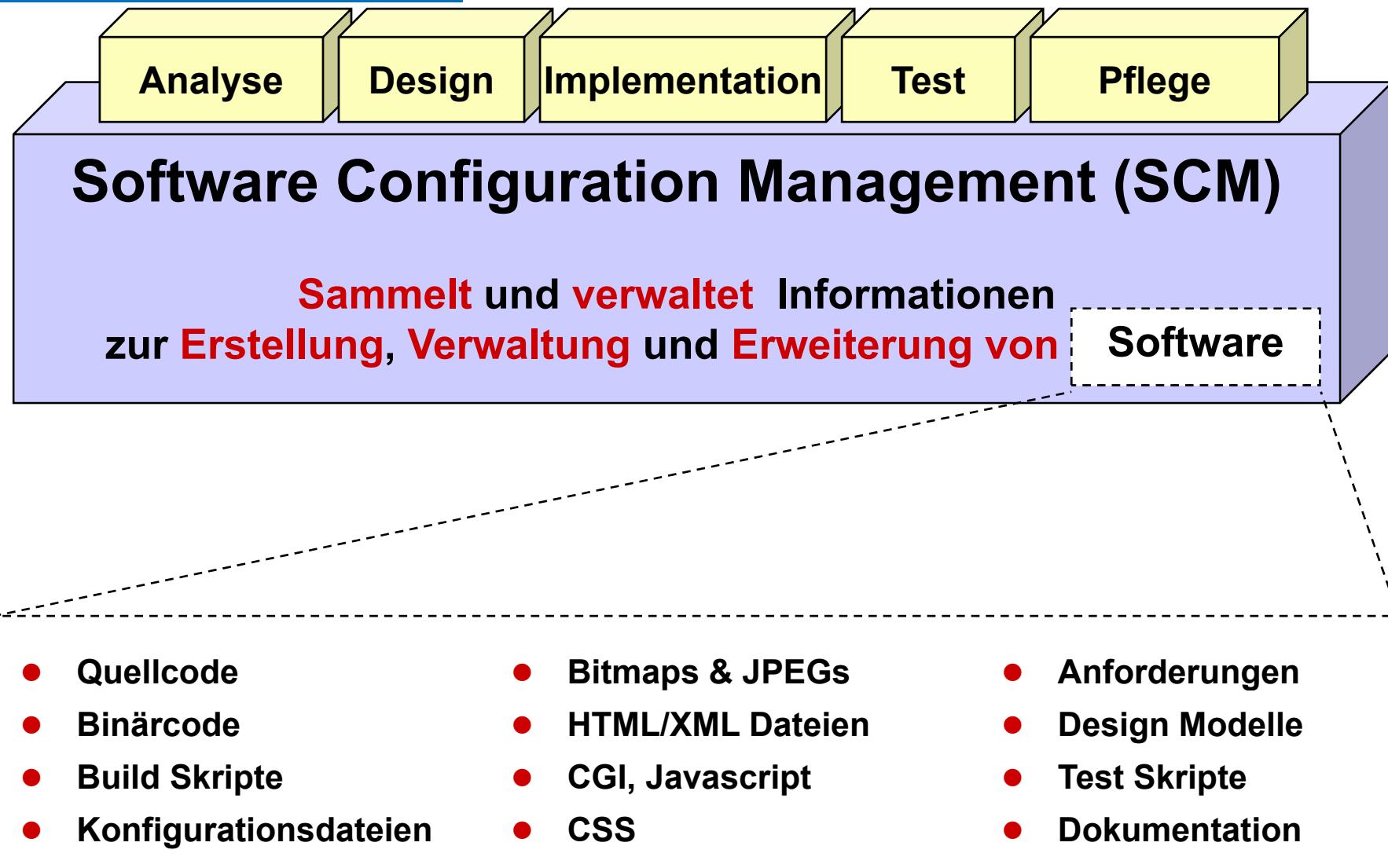
Vergleich von SVN und CVS

Einrichten des „Subversive“-Plugins für Eclipse

Verbinden mit dem Repository via „Subversive“

Struktur von Subversion-Repositories

SCM: Grundlage der Softwareentwicklung



Wozu braucht man Software Configuration Management?

Wir möchten

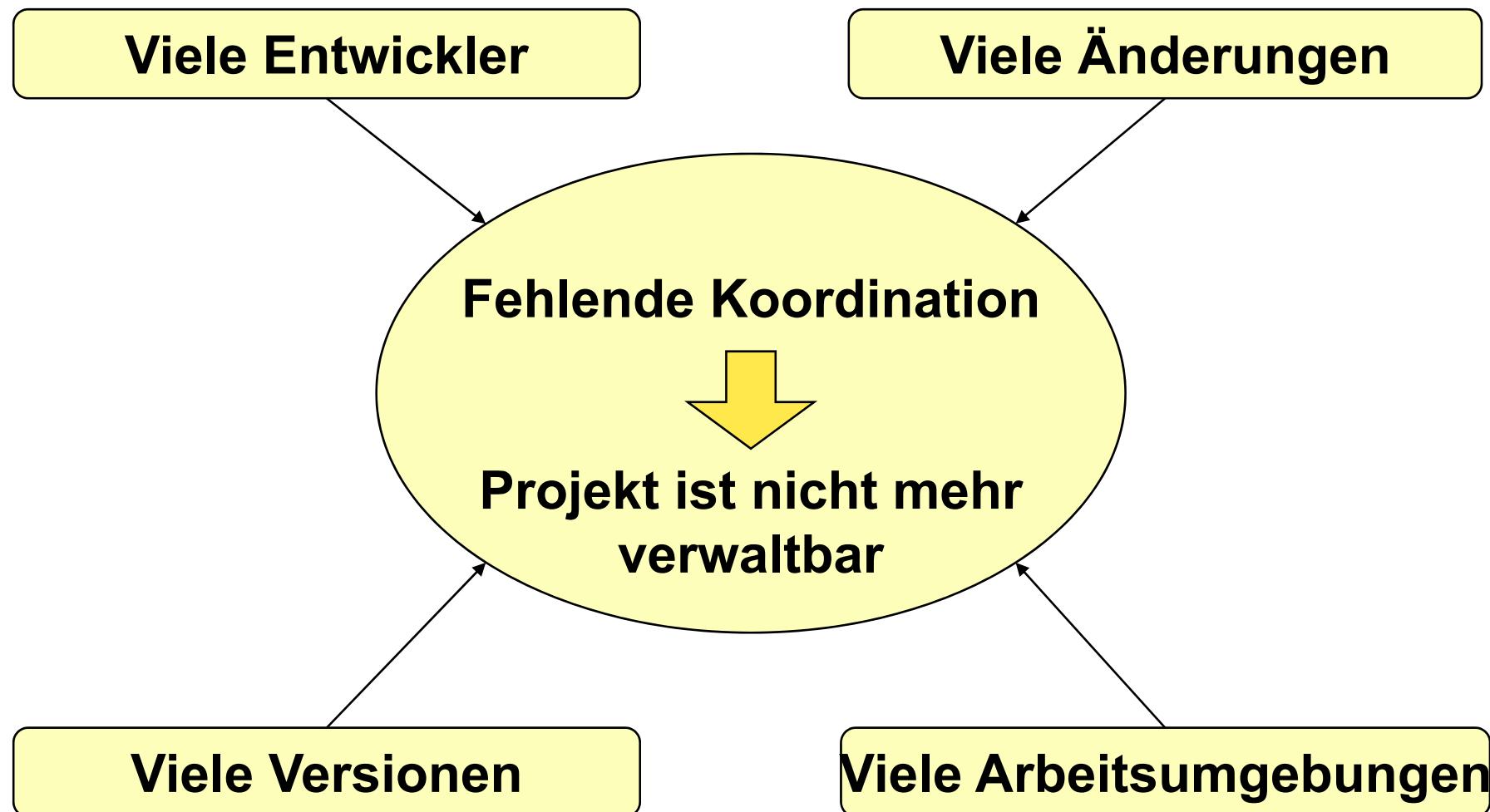
- Softwareentwicklung ist nicht linear
 - ◆ Man macht Programmierfehler
 - ◆ Man trifft die falschen Designentscheidungen
- Softwareentwicklung ist Teamwork
 - ◆ Man selbst und andere macht gleichzeitig Änderungen
 - ◆ ... in vielen verschiedenen Dateien
- Verwaltung verschiedener Versionen
 - ◆ Kunden bekommen stabile Releases während die Entwicklung der nächsten Version weitergeht
 - ◆ Bugfixes müssen in alle Versionen integriert werden
 - ◆ Verschiedene Kunden bekommen verschiedene Varianten des Systems

→ wissen was, wann, warum gemacht wurde.
→ zu einer älteren Version zurückkehren können.

→ wissen wer was, wann, warum gemacht hat
→ Änderungen teilen.
→ Abhängigkeiten verwalten.

→ Gleichzeitige Entwicklung und Integration verwalten.
→ Variationen verwalten.

Probleme während der Softwareentwicklung



Wo ist der Unterschied?

Aktuelle Version

```
public void myTestMethod()
{
    log("Test method entered.");

    boolean keepOnRunning = true;
    int i=0;

    while( keepOnRunning )
    {
        int r1 = doSomething1();
        int r2 = doSomething1();

        if( r1+r2 > MAX_THRESHOLD )
        {
            log("Threshold exceeded.");
            log("Iterations: " + i);
            keepOnRunning = false;
        }
        else
        {
            printStatus(r1, r2);
            i++;
        }
    }
}
```

Vorherige Version

```
public void myTestMethod()
{
    log("Test method entered.");

    boolean keepOnRunning = true;
    int i=0;

    while( keepOnRunning )
    {
        int r1 = doSomething1();
        int r2 = doSomething2();

        if( r1+r2 > MAX_THRESHOLD )
        {
            log("Threshold exceeded.");
            log("Iterations: " + i);
            keepOnRunning = false;
        }
        else
        {
            printStatus(r1, r2);
            i++;
        }
    }
}
```

Wo ist der Unterschied? Hier!

Aktuelle Version

```
public void myTestMethod()
{
    log("Test method entered.");

    boolean keepOnRunning = true;
    int i=0;

    while( keepOnRunning )
    {
        int r1 = doSomething1();
        int r2 = doSomething1(); -----|

        if( r1+r2 > MAX_THRESHOLD )
        {
            log("Threshold exceeded.");
            log("Iterations: " + i);
            keepOnRunning = false;
        }
        else
        {
            printStatus(r1, r2);
            i++;
        }
    }
}
```

Vorherige Version

```
public void myTestMethod()
{
    log("Test method entered.");

    boolean keepOnRunning = true;
    int i=0;

    while( keepOnRunning )
    {
        int r1 = doSomething1();
        int r2 = doSomething2(); -----|

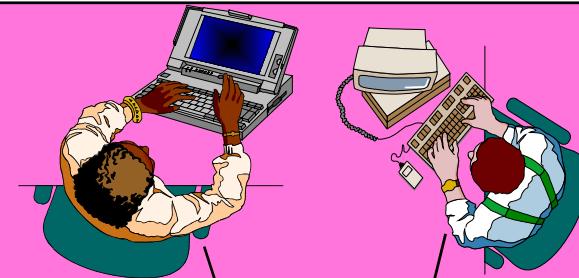
        if( r1+r2 > MAX_THRESHOLD )
        {
            log("Threshold exceeded.");
            log("Iterations: " + i);
            keepOnRunning = false;
        }
        else
        {
            printStatus(r1, r2);
            i++;
        }
    }
}
```

SCM Features

- Managt alle Komponenten des Projekts in einem „Repository“
 - ◆ Keine redundanten Kopien
 - ◆ Sicher
- Macht Unterschiede zwischen Versionen sichtbar
 - ◆ „Was hat sich verglichen mit der gestrigen Version verändert?“
- Verwaltet Metadaten: Wer hat was, wann, warum und wo getan?
 - ◆ „Wer hat was in Klasse X geändert?“
 - ◆ „Warum hat er es verändert?“
- Ermöglicht die Wiederherstellung voriger Zustände
 - ◆ Vollständig oder selektiv
- Erlaubt die Definition von Referenzversionen
 - ◆ Letzte konsistente Version
 - ◆ Milestones
 - ◆ Releases
- Erlaubt Änderungen zu identifizieren, auszuwerten, zu diskutieren (!) und sie schließlich anzunehmen oder zu verwerfen.

Software Configuration Management

- Arbeitskopie
 - ◆ Unter Kontrolle eines Programmierers
 - ◆ Nur für ihn sichtbar



Erstellung von Versionen für andere Programmierer → **Promotion**

- Repository
 - ◆ Zentrales Verzeichnis aller promoteten Versionen
 - ◆ Von allen Teammitgliedern benutzt



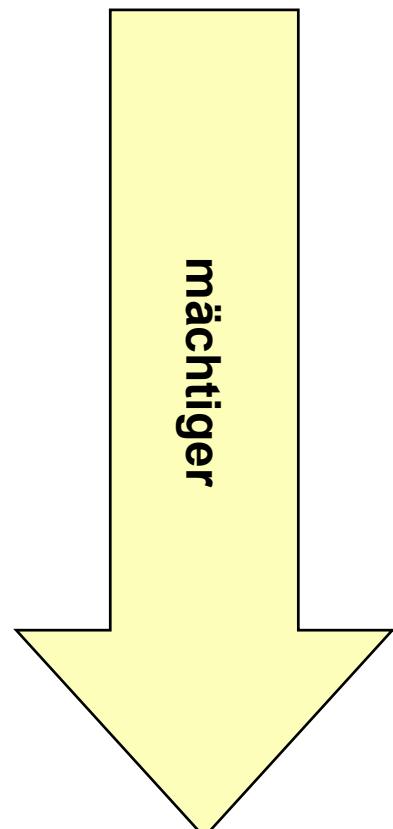
Erstellung von Versionen für Kunden / Benutzer → **Release**

- Produkt
 - ◆ Externe, veröffentlichte Releaseversion



Tools for Software Configuration Management

- Software Configuration Management wird von Tools mit unterschiedlicher Funktionalität unterstützt. Beispiele:
 - ◆ CVS (basic, kostenlos)
 - ⇒ Repository im Dateisystem
 - ⇒ Dateiversionierung
 - ⇒ Hauptsächlich Textdateien, Binärdateien nur eingeschränkt
 - ◆ SVN (besser, kostenlos)
 - ⇒ Repository in Datenbank
 - ⇒ Versionierung von Dateien und Ordnern
 - ⇒ Verfolgt die Aktivitäten der Entwickler und zeichnet sie auf
 - ◆ ClearCase (high-end, kommerziell)
 - ⇒ Virtuelles Dateisystem
 - ⇒ Konfigurationsregeln
 - ⇒ Multiple Server, Replikation
 - ⇒ Prozessmodellierung, “Policy check” Mechanismen

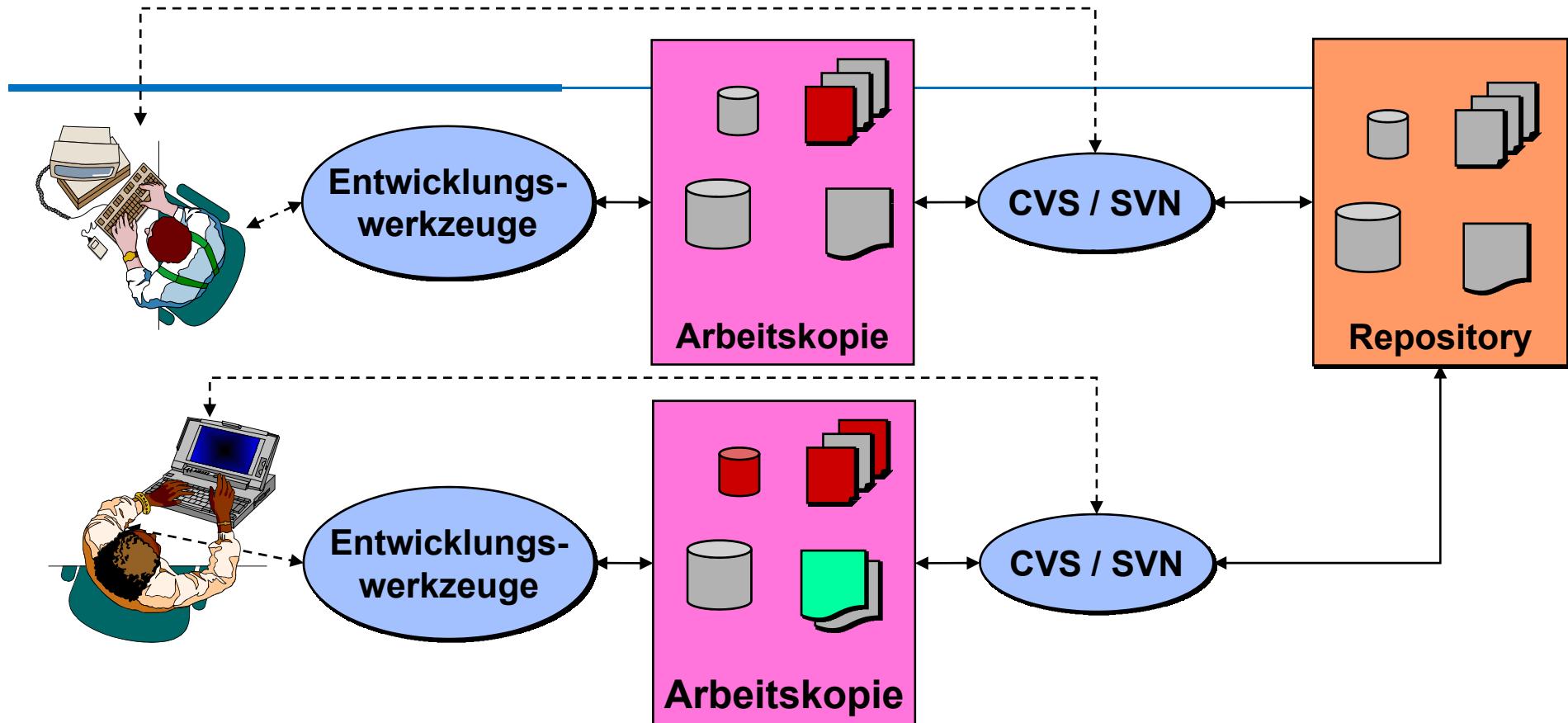


mächtiger

Arbeiten mit SVN (und CVS)

- Check-in und Check-out
- Commit und Update
- Synchronisierung
- Konfliktbeurteilung durch Dateivergleich
- Manuelle Konfliktauflösung

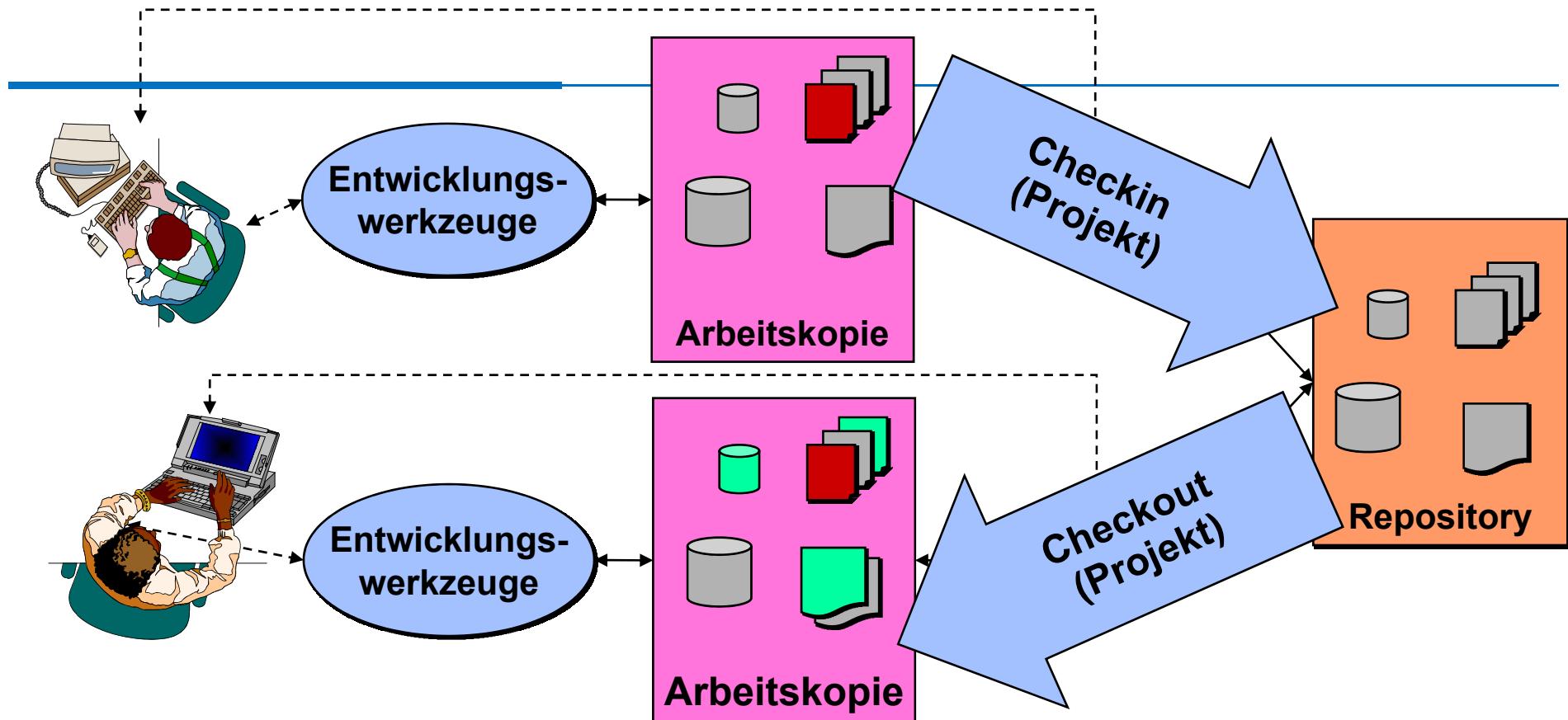
CVS und SVN: Grundprinzipien



- Prinzip

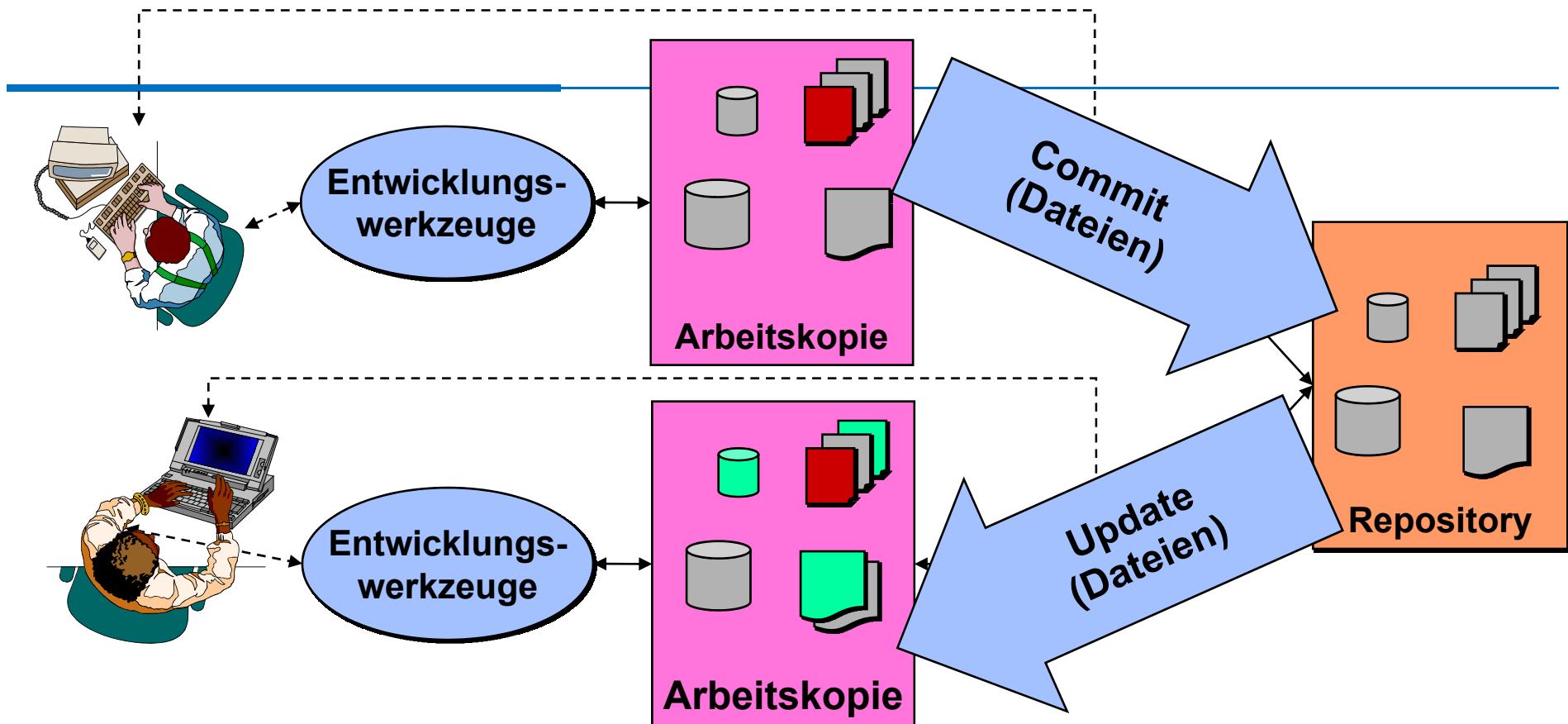
- ◆ Ein zentrales Sammelbecken („Repository“) aller relevanter Dateien
 - ⇒ Nur „offizielle“ Versionen
- ◆ Viele private Arbeitsumgebungen („Sandbox“) mit Kopien von Dateien
 - ⇒ Auch temporäre, inkonsistente, unfertige, ... Versionen

CVS und SVN: Checkin und Checkout



- Checkin
 - ◆ Fügt **Projekt** dem **Repository** hinzu
- Checkout
 - ◆ Erstellt eine **Arbeitskopie des Projekts** vom **Repository**

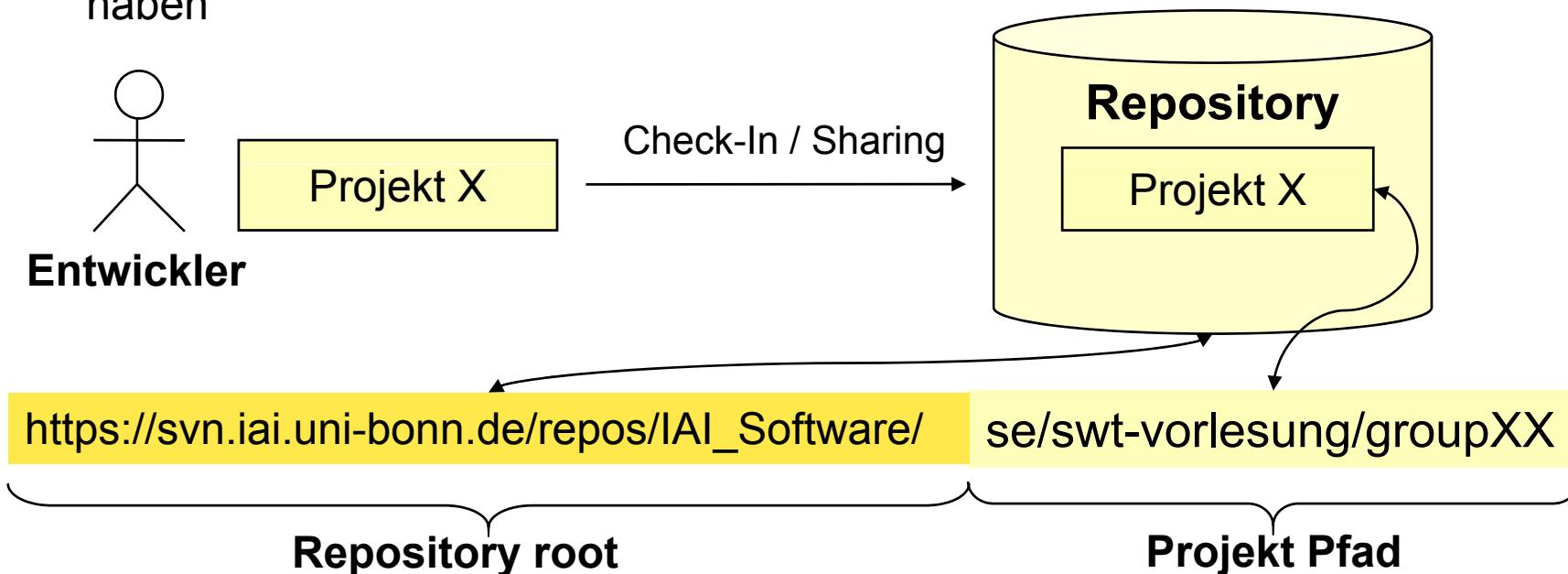
CVS und SVN: Commit und Update



- Commit
 - ◆ Transferiert vom Programmierer geänderte **Dateien** in das **Repository**
- Update
 - ◆ Transferiert geänderte **Dateien** vom **Repository** in die **Arbeitskopie**

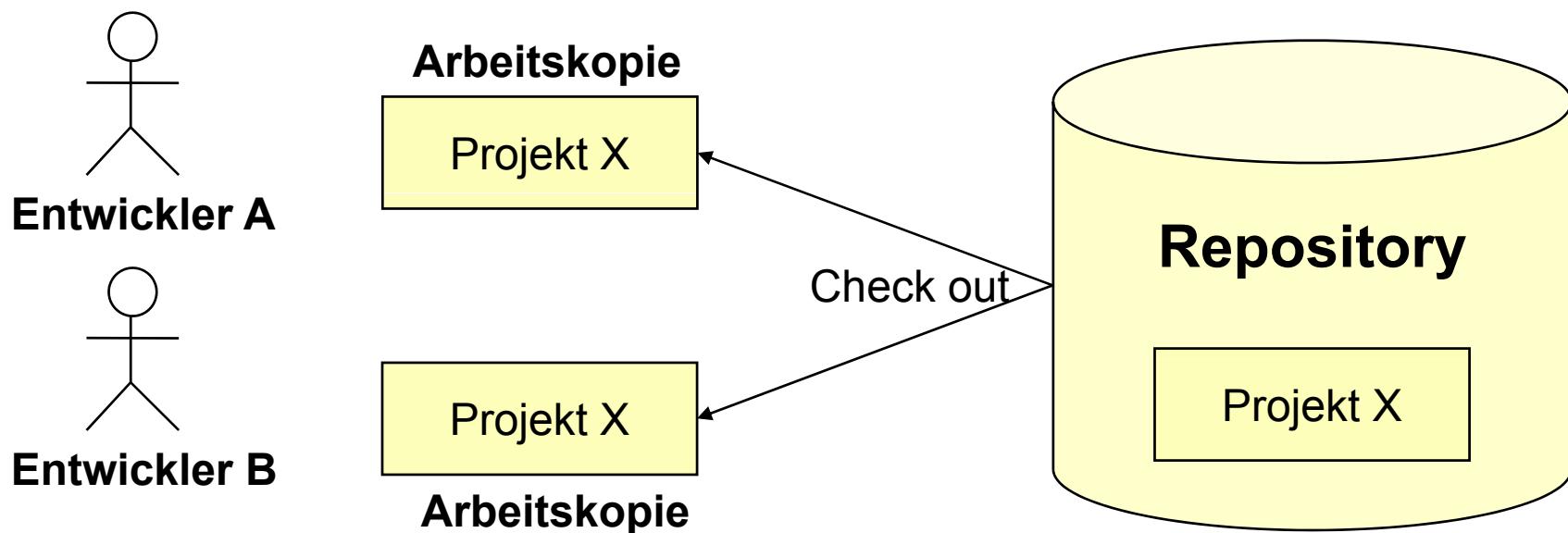
Check-In: Projekt unter Versionskontrolle stellen

- Ausgangssituation
 - ◆ Ein Repository existiert
 - ◆ Ein Projekt existiert, wird aber noch nicht gemeinsam genutzt.
- Check-In / Sharing / Promotion des Projektes
 - ◆ Das Projekt wird dem Repository hinzugefügt
 - ◆ Es ist nun für alle Entwickler verfügbar die Zugriff auf das Repository haben



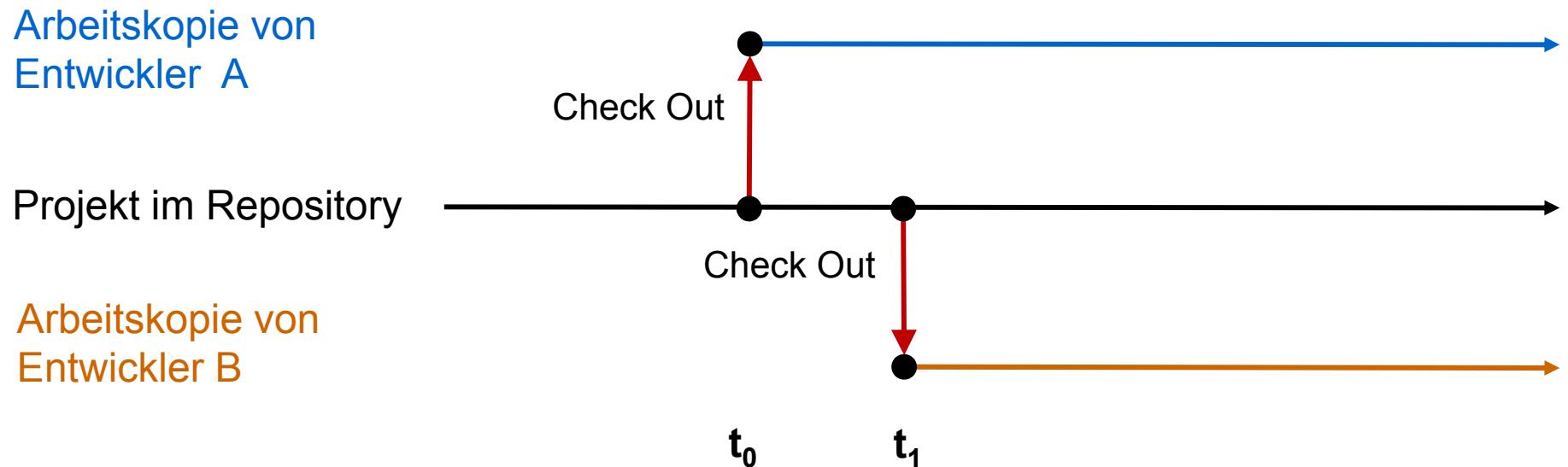
Check-Out: Initialer Projektdownload

- Check-out eines Projektes
 - ◆ Entwickler bekommen eine lokale Arbeitskopie
 - ◆ Von jetzt an können sie an dem Projekt arbeiten
- Auschecken wird nur einmal pro Projekt gemacht!
 - ◆ Neue Version aus Repository holen → siehe „Update“



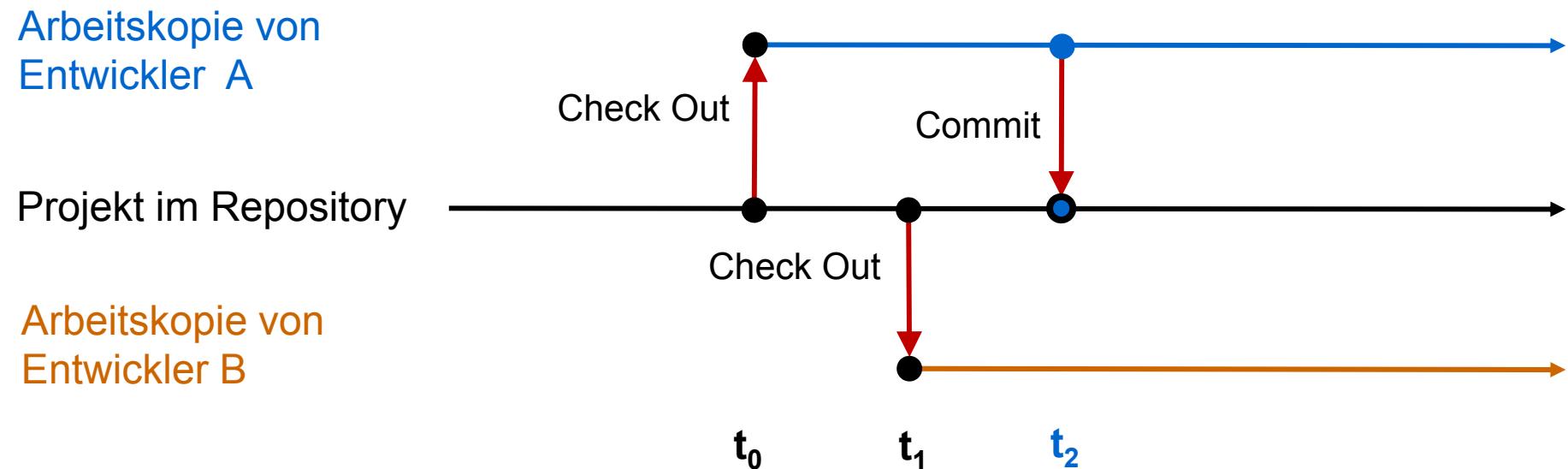
Check-Out: Initialer Projektdownload

- Check-out eines Projektes
 - ◆ Entwickler bekommen eine lokale Arbeitskopie
 - ◆ Von jetzt an können sie an dem Projekt arbeiten
- Auschecken wird nur einmal pro Projekt gemacht!
 - ◆ Neue Version aus Repository holen → siehe „Update“



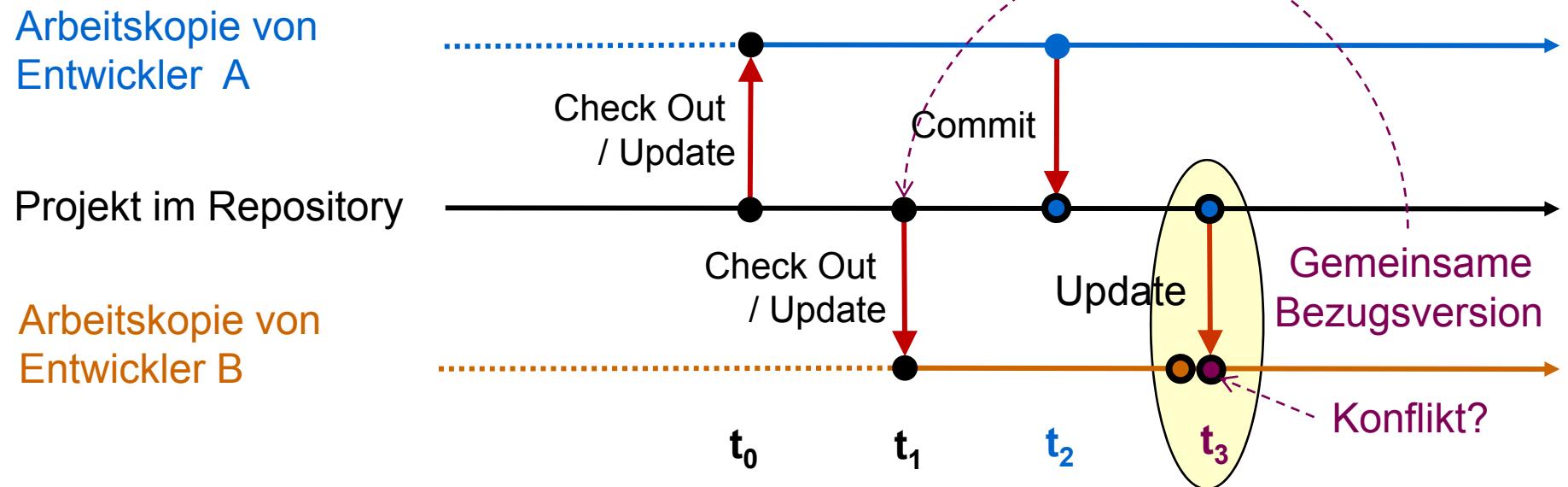
Commit: Änderungen in das Repository übertragen

- Entwickler A ändert seine Arbeitskopie
- Er fügt seine Änderungen dem Repository hinzu → Commit
- Mit einem „Commit Kommentar“ teilt er dem Team mit was er warum geändert hat
 - ◆ „NullPointerException behoben, die auftrat wenn ...“



Update: Neueste Änderungen vom Repository übernehmen

- Ausgangslage: Repository hat sich geändert
 - ◆ Entwickler B ist sich **sicher(!)**, dass er die Änderungen übernehmen will
- Update
 - ◆ Aktualisiert die Arbeitskopie mit dem aktuellen Zustand des Repository
- Problem
 - ◆ Woher weiß der Entwickler, ob er die Änderungen übernehmen soll?
 - ◆ Besser: Zuerst „Synchronize“ benutzen, danach erst entscheiden wofür ein „Update“ und wofür ein „Commit“ in Frage kommt!



Das Projekt synchronisieren

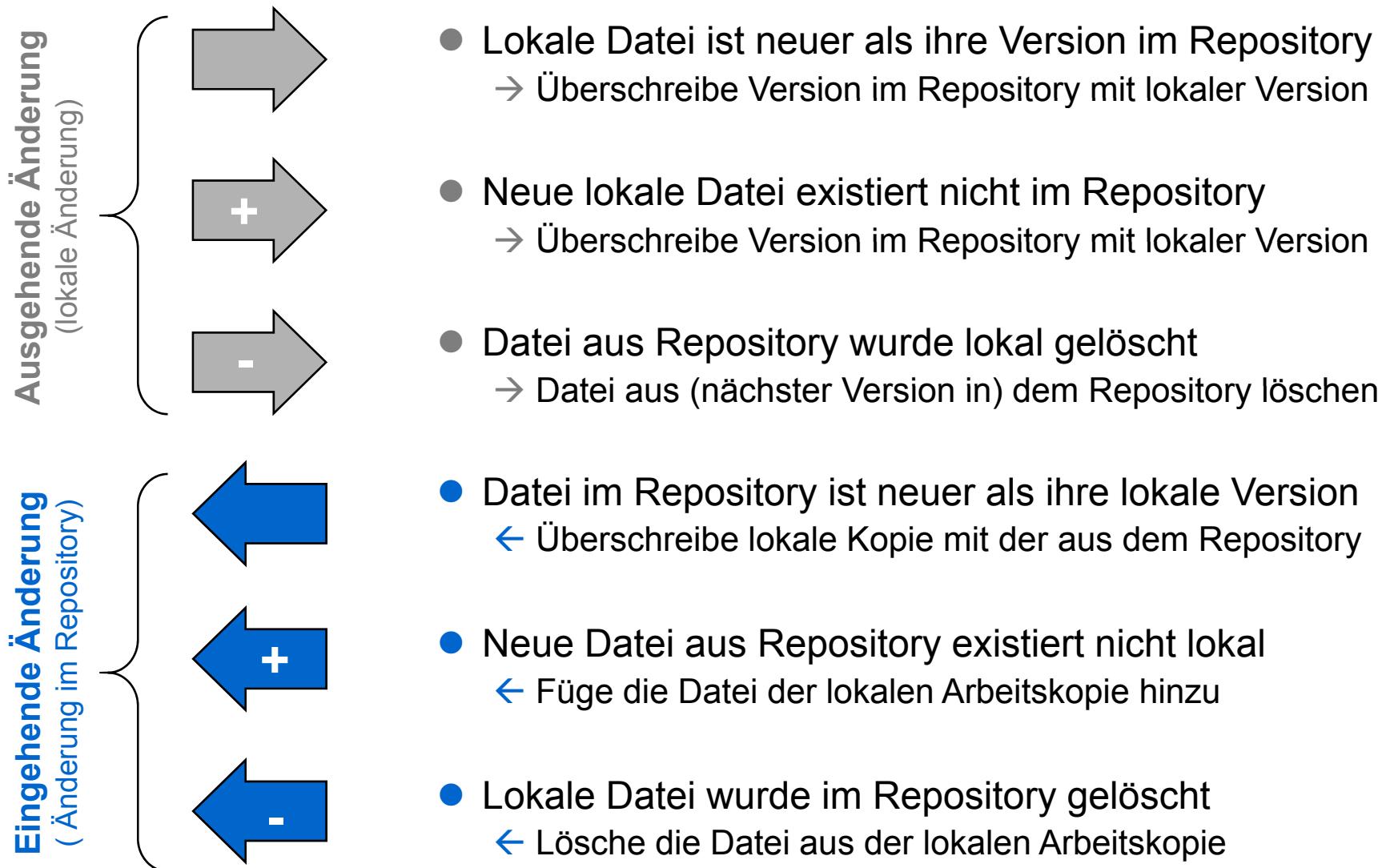
- Synchronisation: Vergleich der Arbeitskopie mit Repository bezogen auf Stand beim letzten Abgleich (check-in / commit / check-out / update)
 - ◆ Automatisierter Vergleich **aller Dateien** im Projekt
 - ◆ Automatisierter Vergleich **einzelner Dateiinhalte**
 - ◆ Der Entwickler entscheidet selbst was aktuell ist
 - ◆ ... und führt Updates oder Commits durch (eventuell selektiv)

“Synchronize View”: Gesamtübersicht

- Automatisierte Gesamtübersicht
 - ◆ Vergleich auf Projektebene: Alle Dateien des Projektes (oder des selektierten Ordners) werden verglichen
 - ◆ Zeigt pro Datei ein- und ausgehende Änderungen sowie Konflikte durch entsprechende Symbole



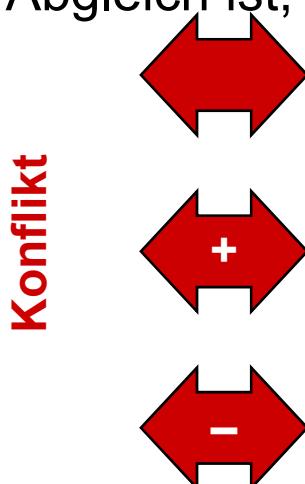
Bedeutung der Symbole im „Synchronize View“



Bedeutung der Symbole im „Synchronize View“ (Fortsetzung)

- Die vorherige Folie stellt die Fälle dar, wenn eine Datei gegenüber dem Stand des letzten Abgleichs mit dem Repository nur auf einer Seite verändert wurde
 - Änderung nur lokal → Übernahme ins Repository (ausgehende Änderung)
 - Änderung nur in Repository → Übernahme in lokale Kopie (eingehende Änderung)

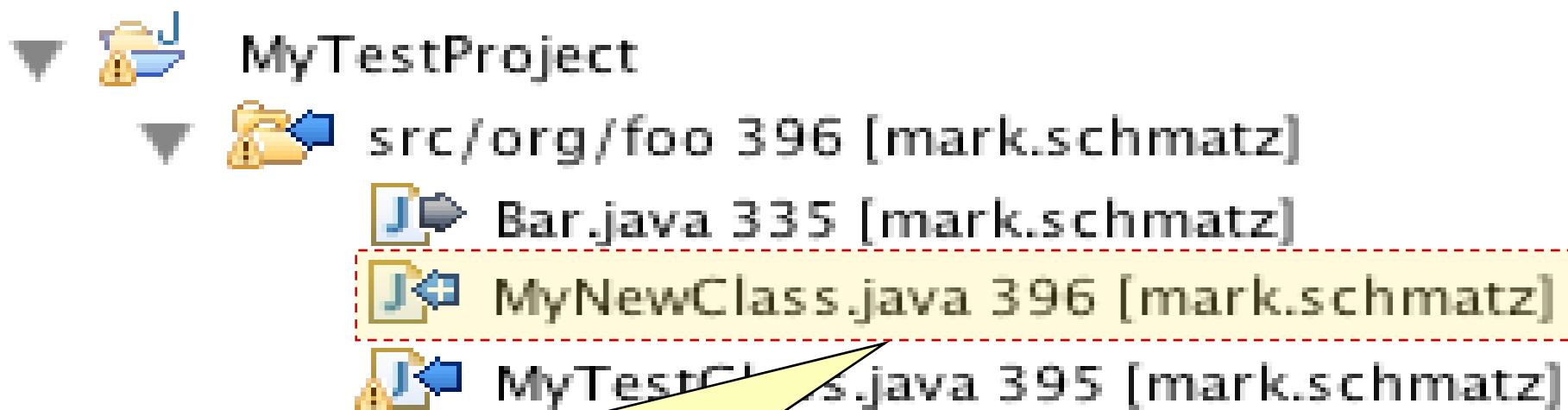
- Wenn eine Datei auf beiden Seiten neuer als der Stand beim letzten Abgleich ist, wird ein Konflikt gemeldet:



- Datei wurde lokal und im Repository verändert
↔ Manuelle Konfliktlösung notwendig
- Lokale veränderte Datei wurde im Repository gelöscht
↔ Manuelle Konfliktlösung notwendig
- Lokale gelöschte Datei wurde im Repository verändert
↔ Manuelle Konfliktlösung notwendig

Synchronisieren: Konfliktauflösung

- Konfliktauflösung erfordert Inhaltsvergleich der Dateien
 - ◆ Angezeigt in “Side-by-side View” / “Compare Editor” von Eclipse
 - ◆ Zeigt Versionsvergleich von Dateien („diff“) übersichtlich an



Synchronisieren: Inhaltsvergleich im „Compare“ Fenster

- Die lokale Arbeitskopie wird mit der Repository Version verglichen
 - Hier ein Fall ohne Konflikte: nur ausgehende Änderung

Local File	Remote File (335 [mark.schmatz])
<pre>package org.foo; /** * * @author schmatz */ public class Bar { }</pre>	<pre>package org.foo; public class Bar { }</pre>

- SVN- / CVS-Plugin für Eclipse hilft beim Versionsvergleich
 - „Compare View“
- Programmierer entscheidet was getan wird
 - „Overwrite and Commit“, „Overwrite and Update“, „Merge“

Synchronisieren: Inhaltsvergleich im „Compare“ Fenster

- Hier die Anzeige eines Konfliktes

Local File	Remote File (394 [mark.schmatz])
<pre>public void myTestMethod() { log("Test method entered."); boolean keepOnRunning = true; int i=0; while(keepOnRunning) { int r1 = doSomething1(); int r2 = doSomething3(); if(r1+r2 > MAX_THRESHOLD) { log("Threshold exceeded."); log("Iterations: " + i); keepOnRunning = false; } } }</pre>	<pre>public void myTestMethod() { log("Test method entered."); boolean keepOnRunning = true; int i=0; while(keepOnRunning) { int r1 = doSomething1(); int r2 = doSomethingElse(); if(r1+r2 > MAX_THRESHOLD) { log("Threshold exceeded."); log("Iterations: " + i); keepOnRunning = false; } } }</pre>

- Bei Bedarf kann auch die gemeinsame Bezugsversion angezeigt werden (“Show Ancestor Pane”).
 - So kann man selbst entscheiden, welches die relevante Änderung ist
 - Siehe Folie 18 („Update: Neueste Änderungen vom Repository ...“)

Synchronisieren: Aktionen zur Konfliktauflösung

- „Overwrite and Commit“ (Menupunkt)
 - ◆ Überschreibt Repository-Version mit lokaler Version
 - ◆ Nie ohne vorherige Kommunikation mit dem Autor der Repository-Version!
- „Overwrite and Update“ (Menupunkt)
 - ◆ Überschreibt lokale Version mit Repository-Inhalt
- „Merging“ (Manueller Vorgang)
 - ◆ Selektive Übernahme einzelner Änderungen der Datei im „Compare Editor“
 - ◆ Nur Übernahme von Repository-Stand in lokale Version!
 - ◆ Anschließend „Mark as Merged“ (Menupunkt) auf lokaler Version
 - ⇒ Lokale Version wird nun als aktueller als die aus dem Repository angesehen
 - ⇒ Bei einem „Commit“ würden nun die nicht übernommenen Repository-Inhalte überschrieben!

Vergleich von Subversion (SVN) und CVS

Versionierung von Verzeichnissen
Globale Commit-Nummerierung
Atomare Commit-Aktionen

Subversion kann mehr als CVS:

1. Versionierung von Verzeichnissen

- Es ist möglich Dateien und Verzeichnisse zu löschen, umzubenennen und zu verschieben
 - ◆ Das entspricht einer neuen Version des umgebenden Verzeichnisses

Warnung

- Man muss diese Operationen mit einem „Subversion Client“ durchführen!
 - ◆ Siehe Folie 34, „Wichtige Links“
- Führt man es selbst auf Systemebene durch wird Subversion nichts vom Umbenennen oder Verschieben erfahren.
 - ◆ Er wird annehmen eine Datei wurde gelöscht und eine andere hinzugefügt!

Subversion kann mehr als CVS:

2. Globale Versions-Nummerierung

- CVS weist jeder Datei **eigene** Versions-Nummern zu
 - ◆ Problem: Es ist nicht einfach zusehen welche Versionen verschiedener Dateien zusammengehören
 - ◆ „Gehört Version 1.1 von *Datei A* wirklich zu Version 1.10 von *Datei B*?“
- SVN weist jeder Datei **die bei einem Commit verändert wurde** die **selbe** Versions-Nummer zu.
 - ◆ Vorteil: Dateiversionen die zusammengehören sind auf einen Blick zu erkennen
 - ⇒ Zu einer Projektversion V gehören alle Datei-/Ordnerversionen mit Versionsnummer V oder kleiner
 - ◆ Ein „Roll back“ auf eine alte Version ist einfach, selbst ohne „Tags“
 - ⇒ „Gehe zurück zu Version 1073“
 - ⇒ Tagging sollte für wichtige Releases, Backups, etc. trotzdem genutzt werden
 - „Release 1.0 alpha“ ist leichter zu merken als Version 1073

Subversion kann mehr als CVS:

3. Atomic commits

- Netzwerkfehler oder andere Probleme können zu unvollständigen Commits führen (nicht alle Dateien wurden „committed“)
- Wenn dies passiert, hinterlässt CVS das Repository in einem inkonsistenten Zustand.
- SVN führt bei unvollständigen Commits einen „Roll back“ durch.
 - ◆ SVN benutzt ein Datenbanksystem um das Repository zu speichern und kann daher Commits als atomare Transaktionen implementieren!

Wichtige Links

- Subversion Buch
 - ◆ <http://svnbook.red-bean.com/nightly/en/svn-book.pdf>
- Subversion Server
 - ◆ subversion.tigris.org
- Subversion Clients als Erweiterung für den Windows Explorer
 - ◆ TortoiseSVN
 - ◆ Nur für Windows!
- Subversion Clients als Plugins für Eclipse
 - ◆ Subclipse
 - ⇒ www.eclipse.org
 - ◆ Subversive
 - ⇒ Homepage: www.polarion.org
 - ⇒ Plugin Download Seite:
<http://www.polarion.org/projects/subversive/download/1.1/update-site/>
- ◆ **Wir empfehlen „Subversive“ zu benutzen**

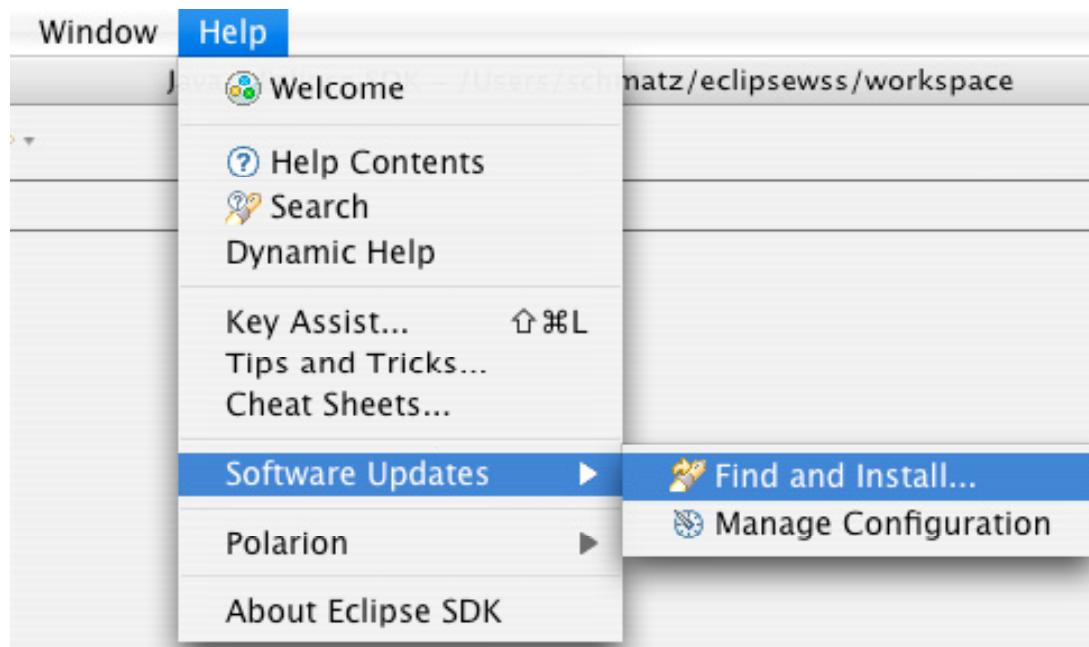
Installation des „Subversive“ Plugins

Bitte nicht die im Folgenden gezeigten URLs wörtlich nehmen, die ändern sich bei jeder neuen Version (X.Y). Sie sind nur innerhalb von Bugfixversionen (X.Y.Z) stabil!

Schauen Sie bitte auf die Vorlesungswebsite, dort steht ein aktueller Überblick, incl. aktueller URLs!!!

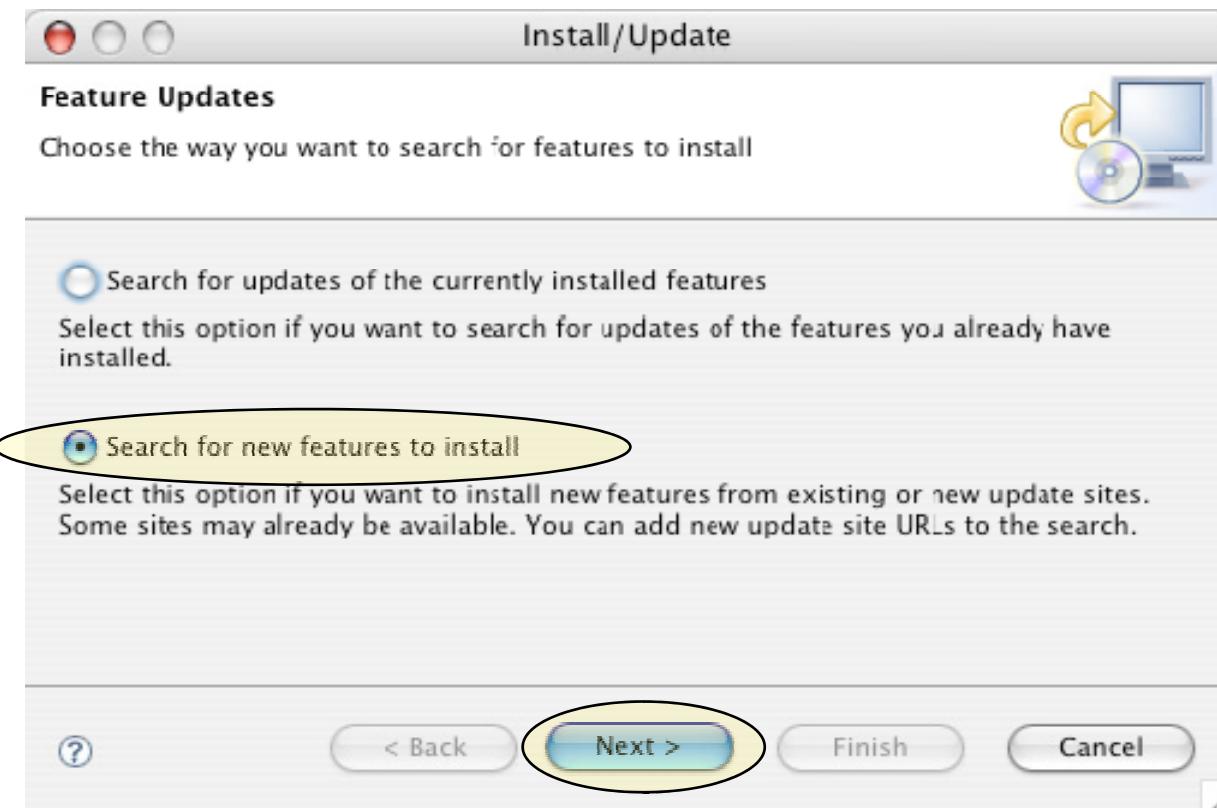
Subversive installieren

- Nutze das Help Menü von Eclipse
 - ◆ Help → Software Updates → Find and Install...



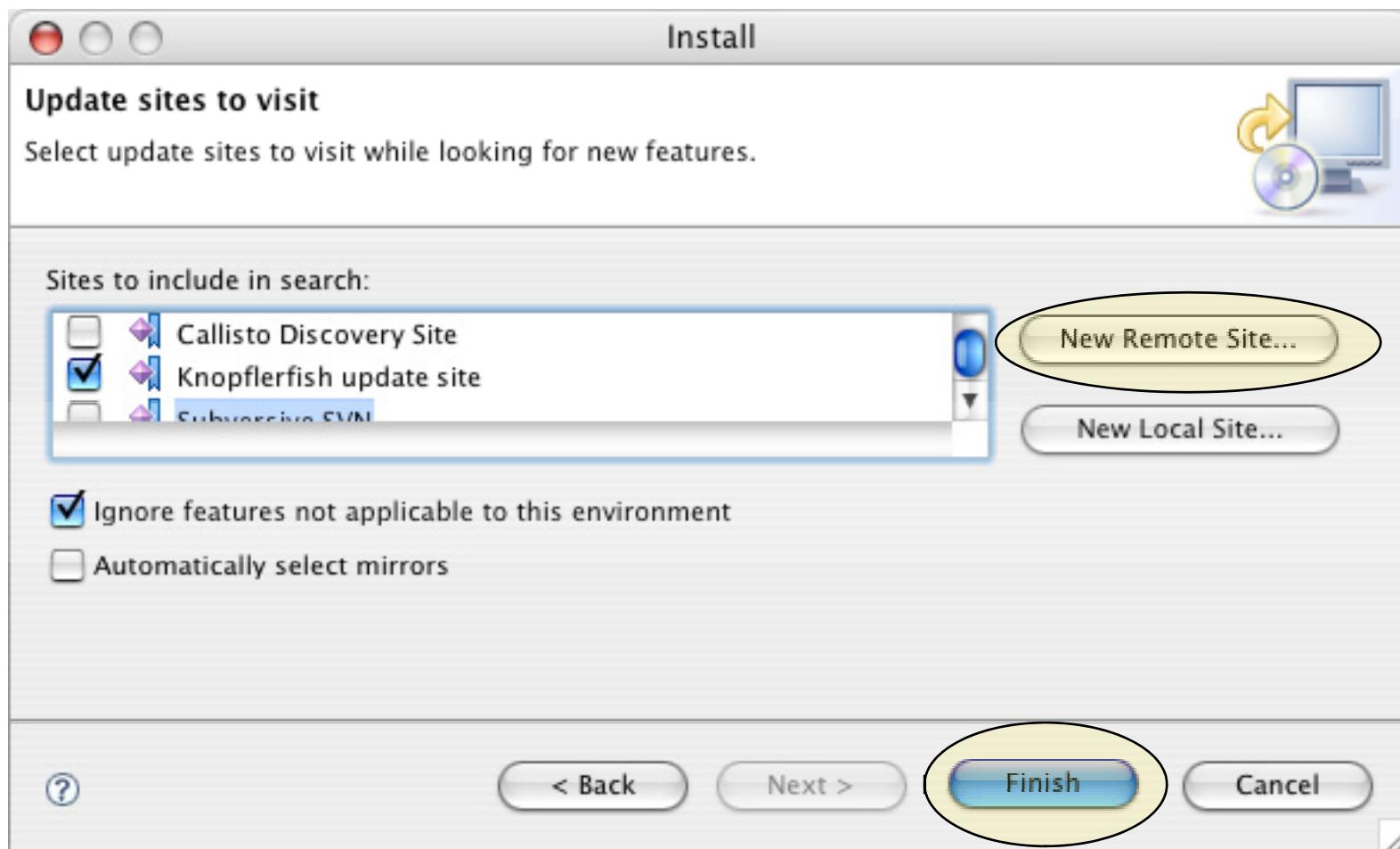
Subversive installieren

- Search for new features to install → Next



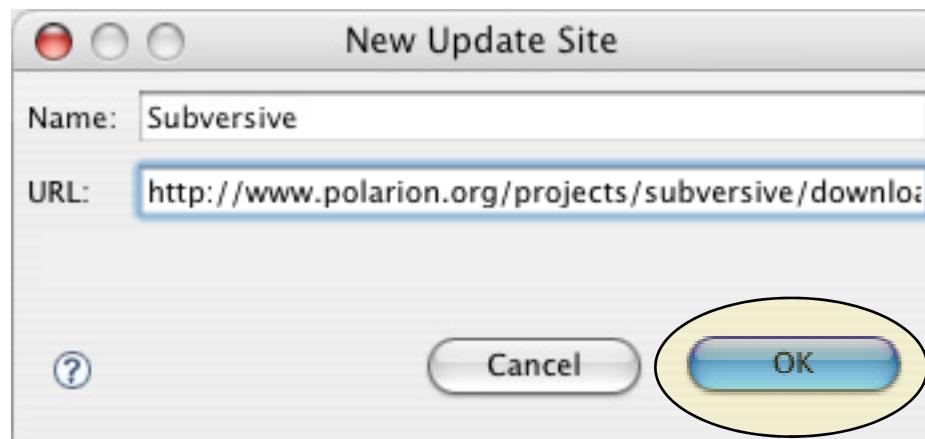
Subversive installieren

- New Remote Site → Finish
 - ◆ Empfehlung: “Automatically select mirrors” spart weitere Rückfragen



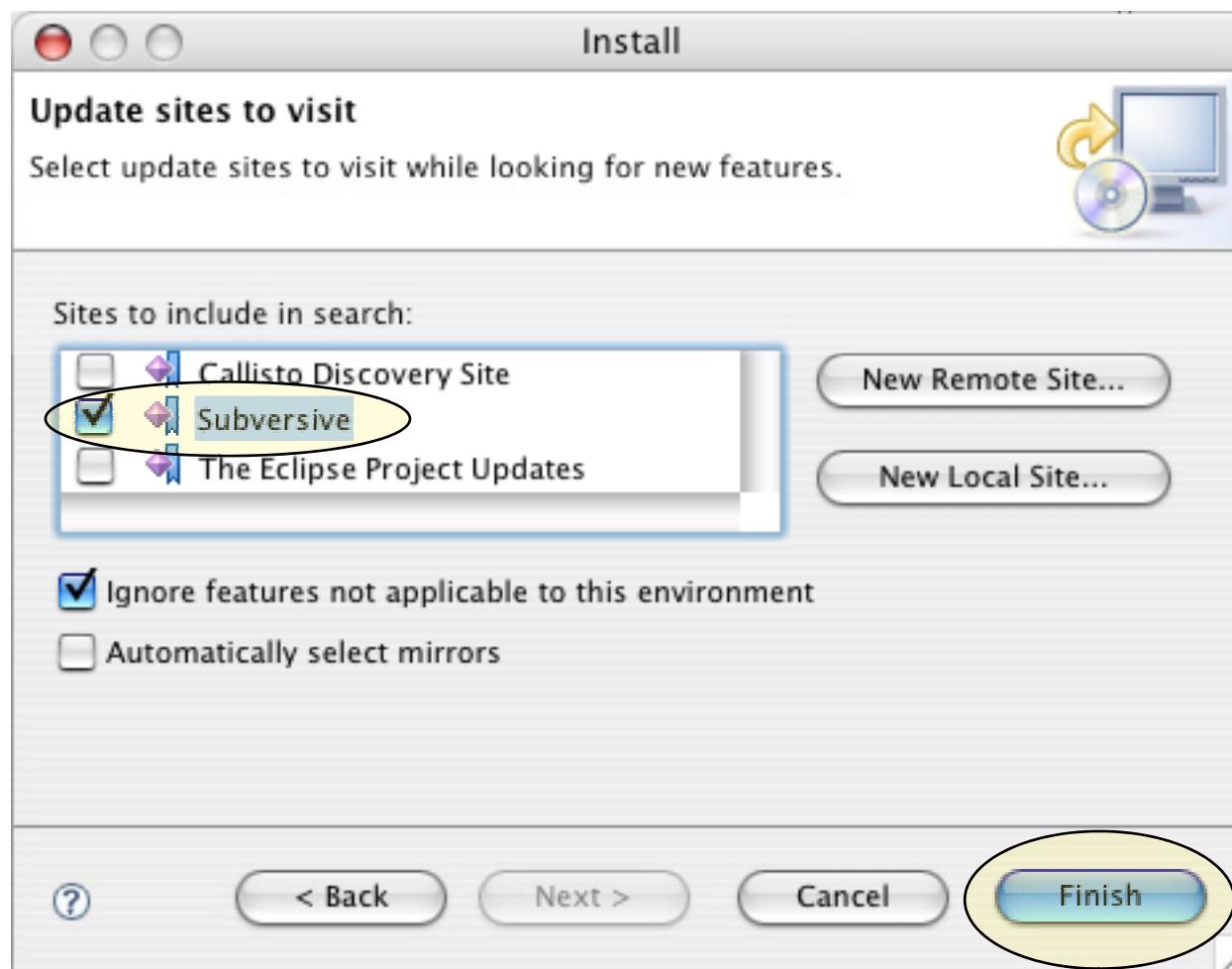
Subversive installieren

- Neue “Update Site” Anlegen
 - ◆ Name: z.B.: Subversive
 - ◆ URL: siehe Vorlesungswebsite → “Subversive”

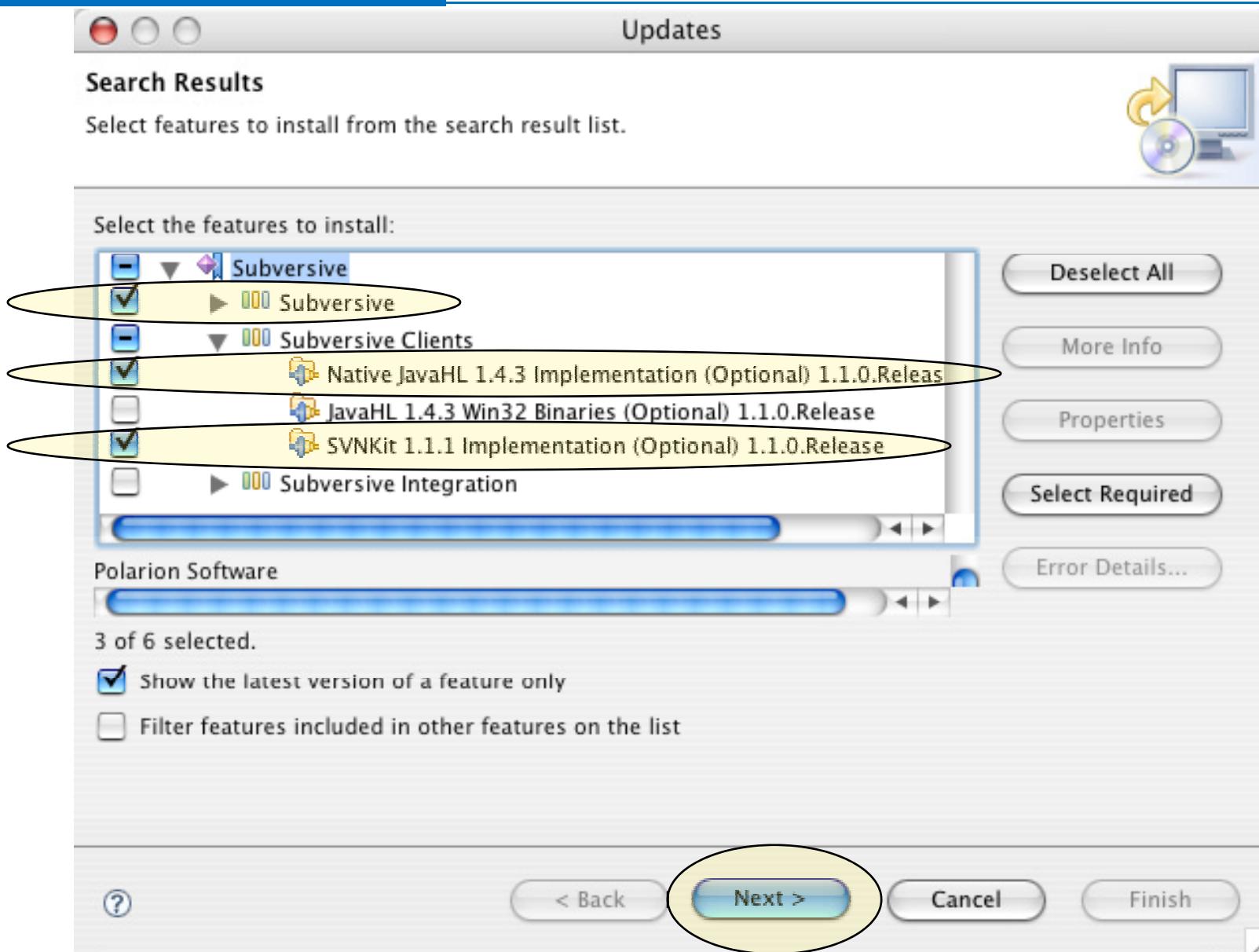


Subversive installieren

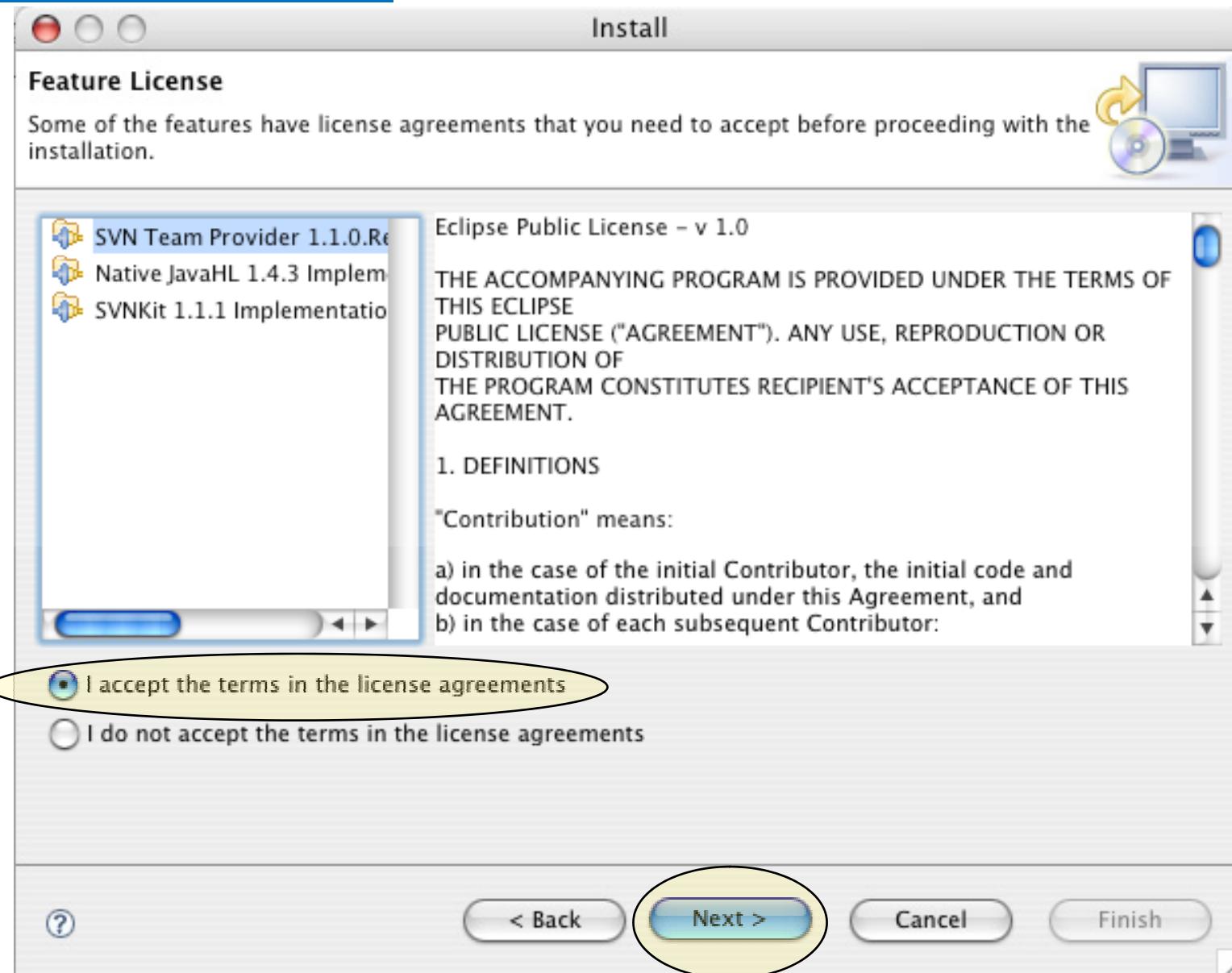
- Subversive Update Site selektieren → Finish



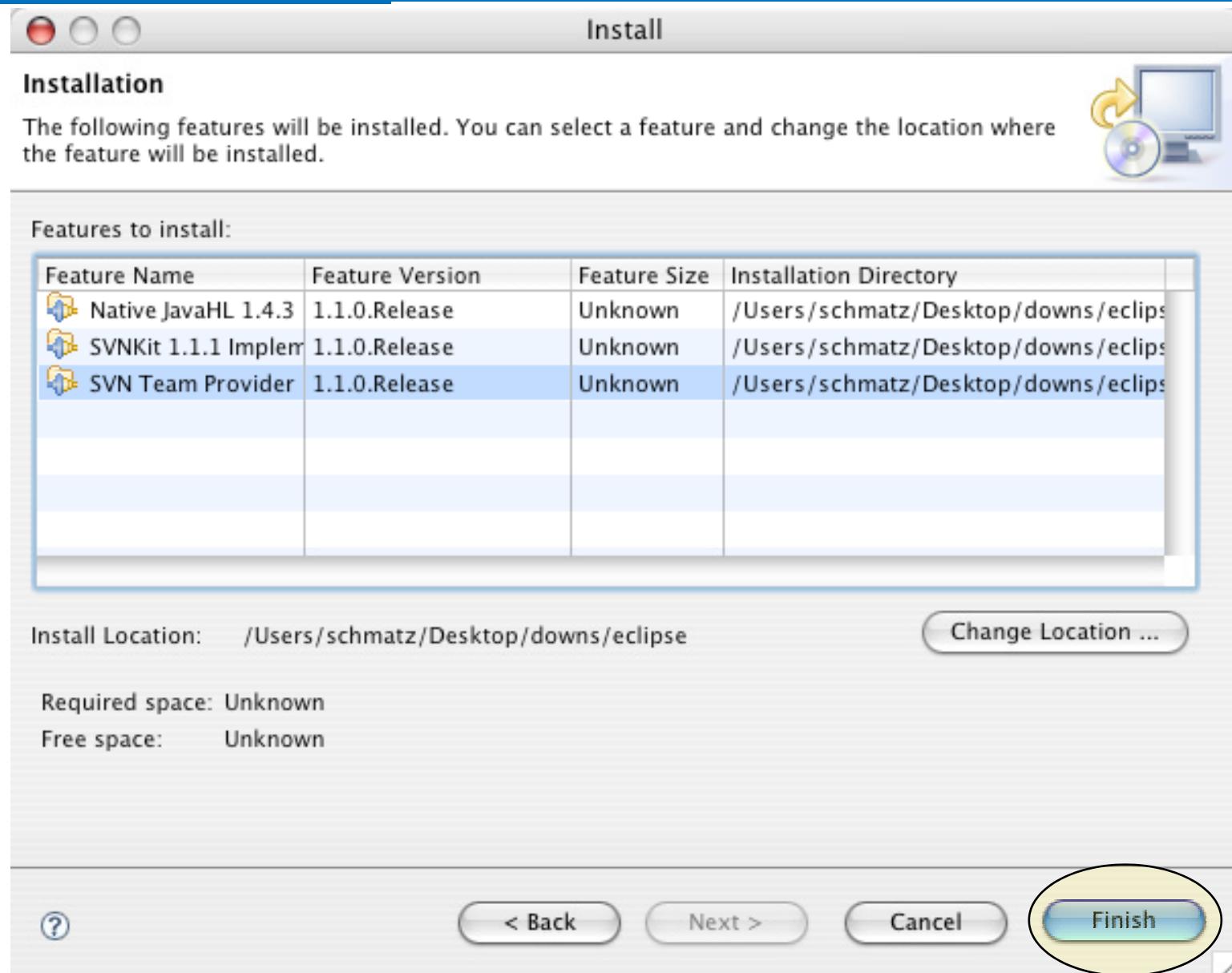
Subversive installieren



Subversive installieren



Subversive installieren

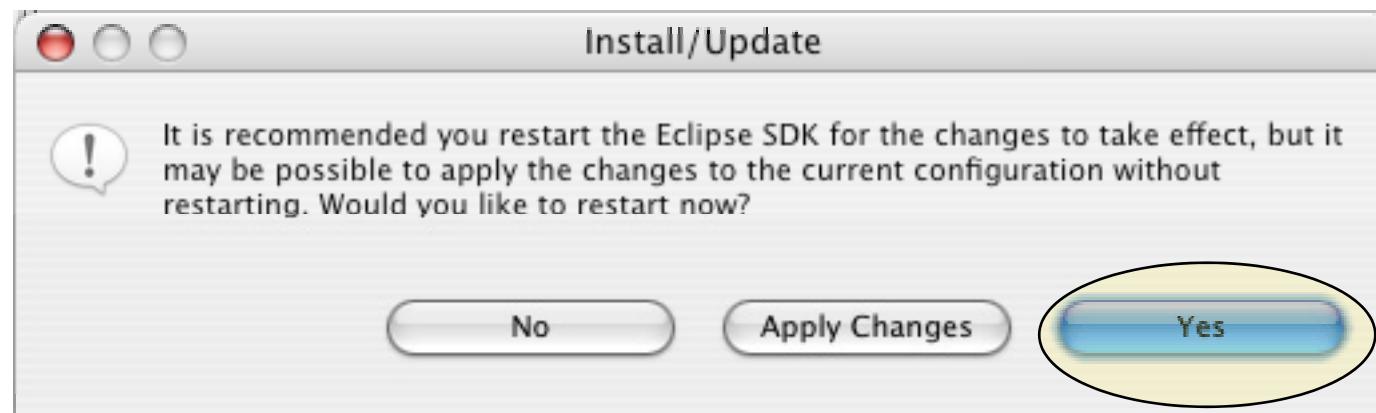


Subversive installieren



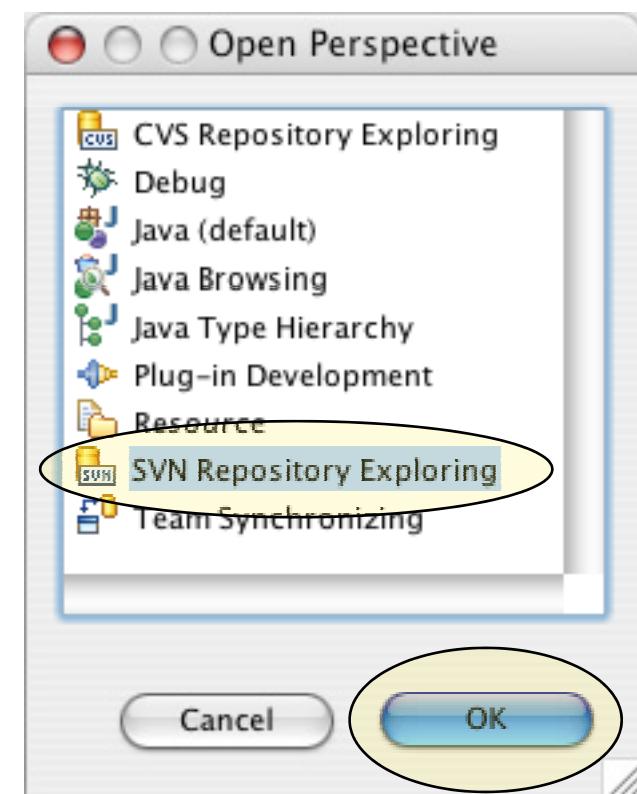
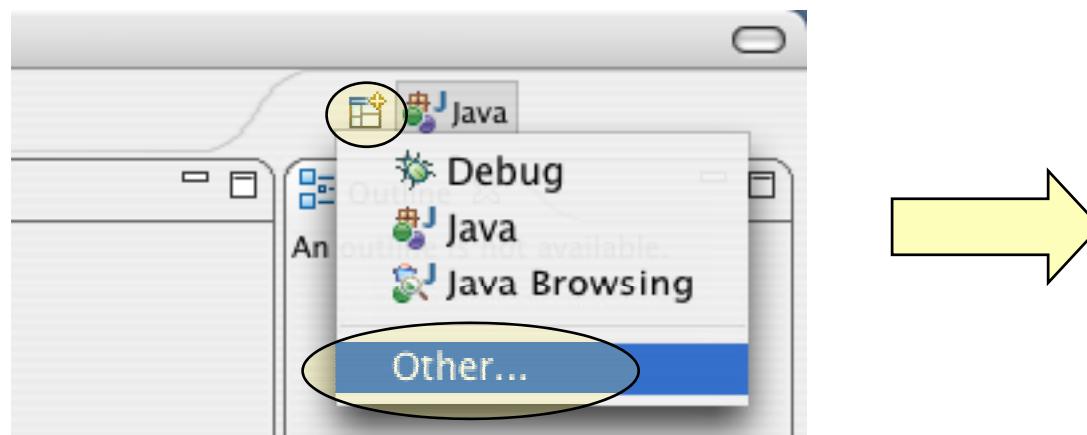
Subversive installieren

- Eclipse neu starten



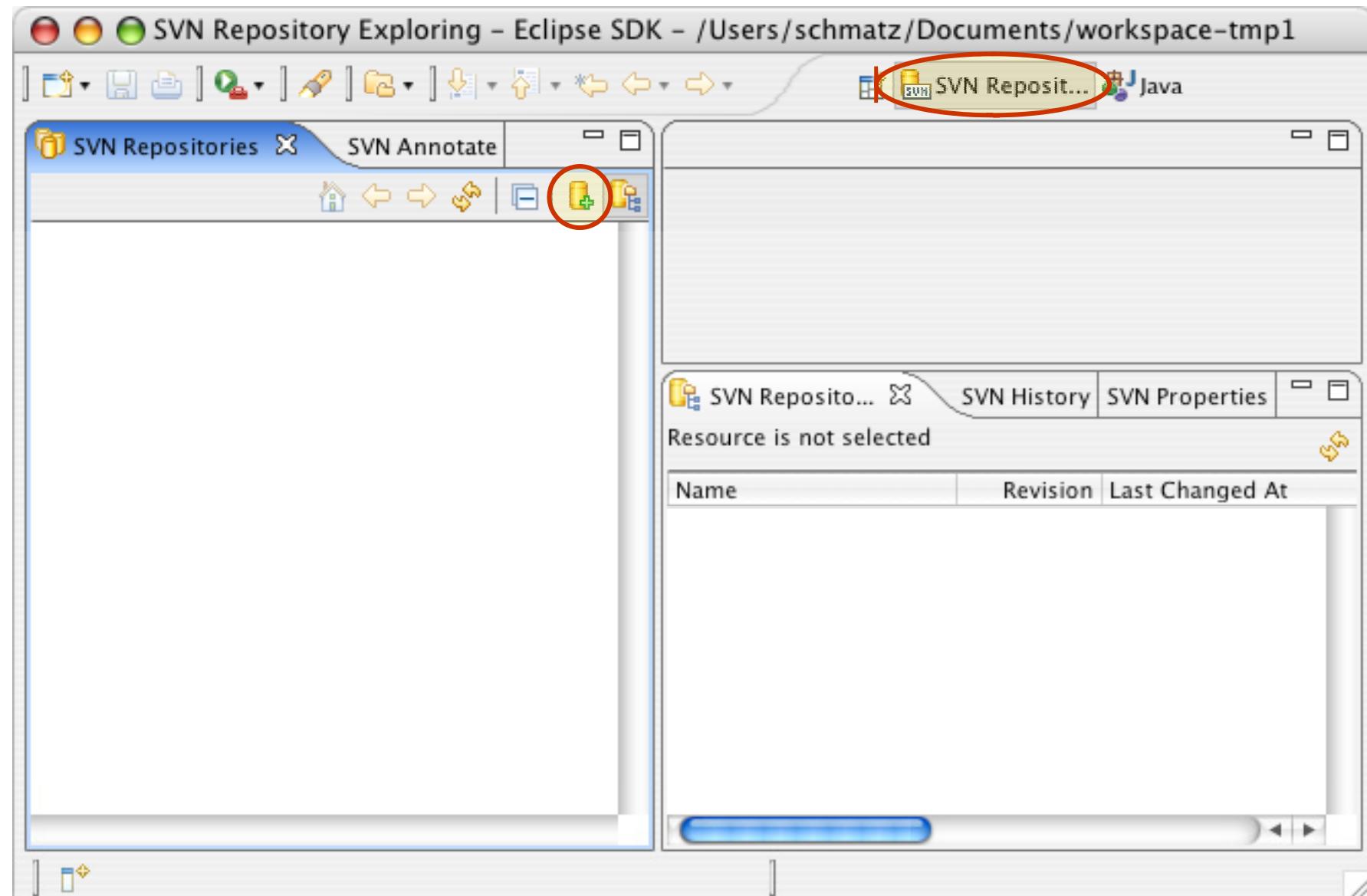
Subversive installieren

- SVN Perspektive anzeigen
 - ◆ In Eclipse ist eine “Perspektiven” eine auf eine bestimmte Aufgabe zugeschnittene Anordnung von Fenstern

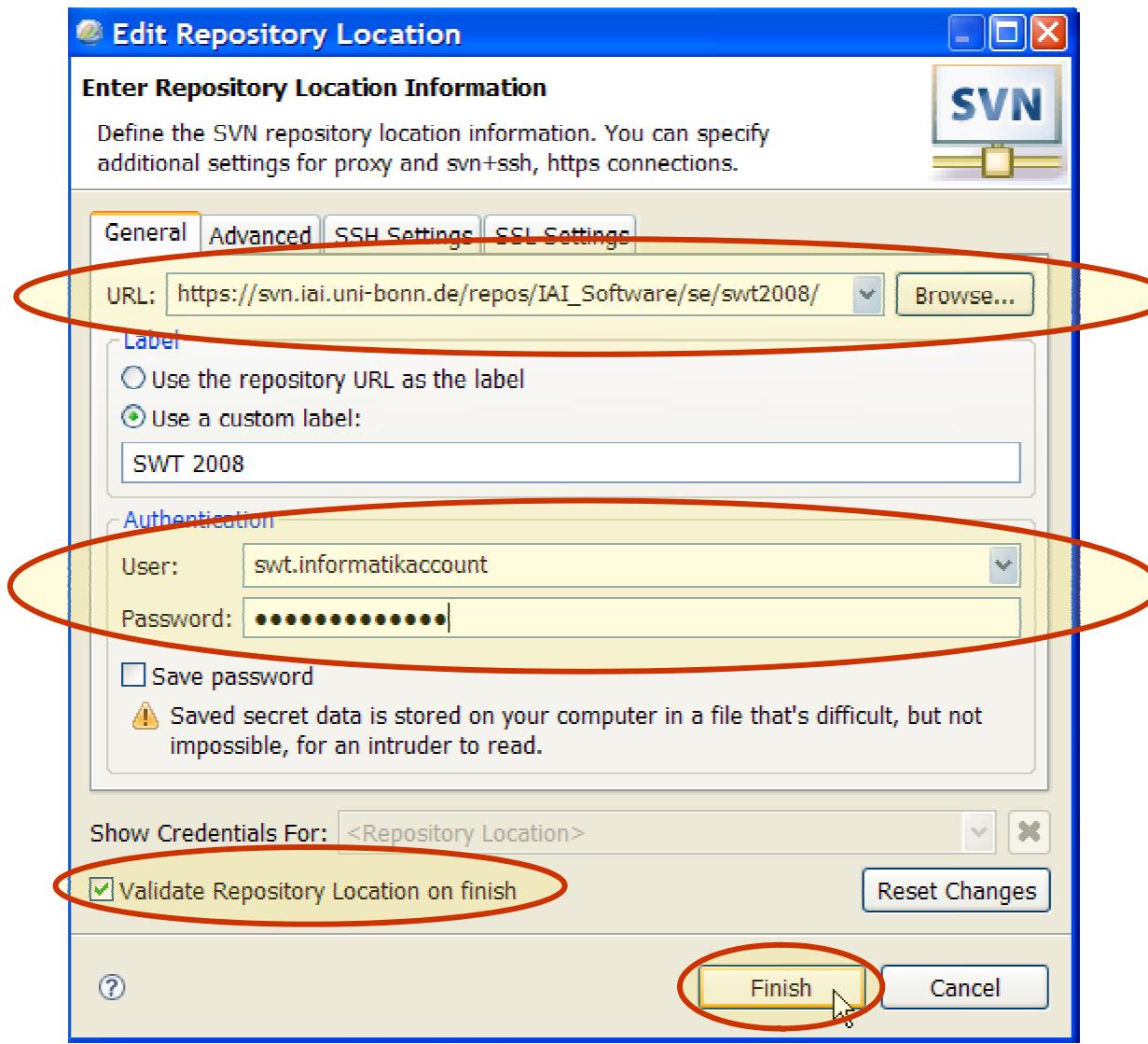


Verbinden mit dem „Subversion“ Repository via „Subversive“

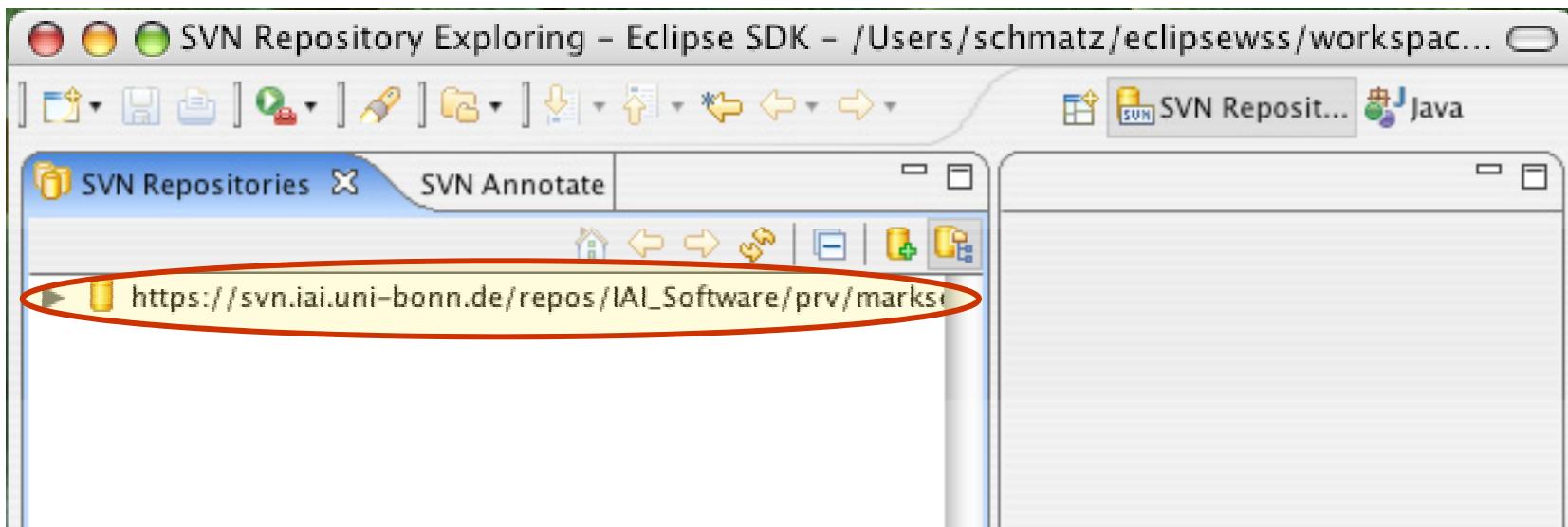
„SVN Repository Exploring“ Perspektive auswählen



Repository Information eingeben



Ergebnis



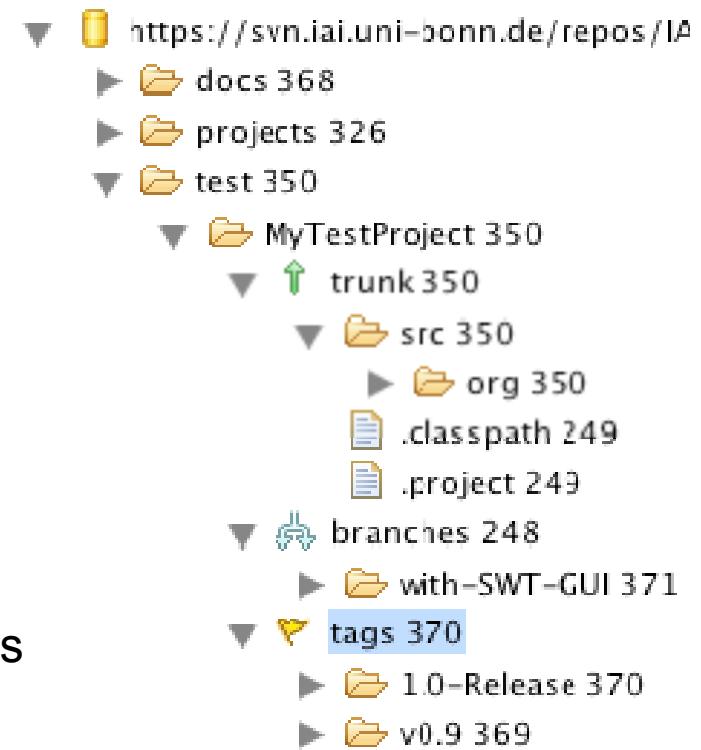
Struktur von Subversion-Repositories

Spezielle Ordner

Optionen

Repository Layout: Spezielle Ordner

- trunk
 - ◆ Enthält die aktuelle Entwicklungslinie (wie HEAD in CVS)
 - ◆ Also den trunk „auschecken“!
- tags
 - ◆ Enthält unveränderliche Versionen des Projekts
 - ⇒ Für Releases, Milestones, Backups
- branches
 - ◆ Enthält alternative Entwicklungszweige des Projekts
 - ◆ Für Untergruppen eines Teams oder parallele Entwicklung verschiedener Varianten



Repository Layout: Alles nur Konvention!

- Vorherige Folie ist nur Konvention!
 - ◆ trunk, branches, tags sind für Subversion normale Ordner
 - ◆ Es ist bloß der Subversive-Client, der sie besonders behandelt
 - ◆ Ob das geschehen soll, kann als Option angegeben werden
- Man darf die Ordner des Projekts organisieren wie man möchte!

