Softwaretechnologie
Dr. Günter Kniesel
Julia Kuck, Malte Appeltauer, Mark Schmatz
SS 2007

Subversion (SVN)

Einführung in die Versionskontrolle mit SVN

Überblick

1. Probleme bei der Softwareentwicklung
2. Lösung: Versionskontrolle (Features, Systeme)
3. Erste Schritte im Umgang mit SVN
4. Fortgeschrittene SVN Techniken
5. Installation, Einrichtung und Demo des *Subversive* Plugins

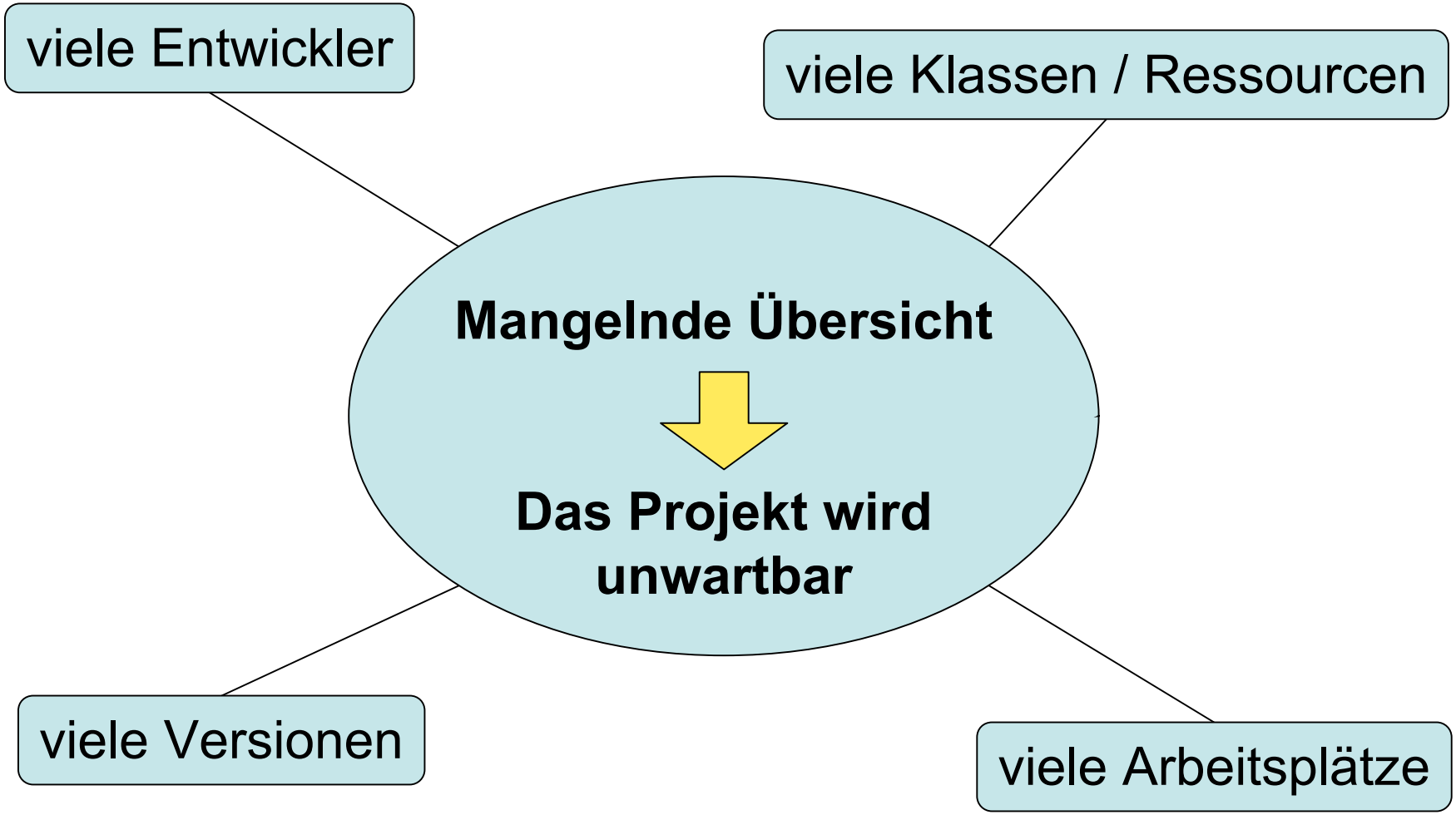
Probleme bei der Entwicklung (1/3)

- Softwareentwicklung ist kein linearer Prozess
  - man macht **Fehler** und möchte ein paar „Schritte“ zurück gehen
  - man trifft nicht immer nur beste **Entscheidungen**
- Softwareentwicklung ist (meist) keine „Ein-Mann-Show“
  - es arbeiten **mehrere Entwickler** gemeinsam an einem Projekt
  - man möchte wissen **wer**, **was**, **wann**, **wieso** gemacht hat
  - **ich** möchte wissen was, wann, wieso von mir gemacht wurde
- Pflege und Wartung von verschiedenen Versionen
  - Kunden erhalten stabile Versionen (Releases)
  - während die interne Entwicklung an der aktuellen Version voranschreitet
  - gefundene Fehler müssen aber in die stabilen Versionen zurückfließen (Backporting)

Probleme bei der Entwicklung (2/3)

- Backups sind notwendig, aber
  - oft verliert man den Überblick
  - weiß nicht mehr welche Version was tut
  - oder in welcher Version der Fehler X behoben wurde
  - komplette (nicht-iterative) Backups sind platz- und zeitraubend
- 80/20-Regel: 20% Entwicklung, 80% Debugging
  - „Ich finde den Fehler nicht…“ / „Gestern ging es doch noch…“
- Arbeitsplatzwechsel erfordert „Mitnahme“ / Kopieren des Projekts
  - Inkonsistenzen sind vorprogrammiert
- Kommunikation wird vergessen
  - „In der Klasse X habe ich Y implementiert weil Z…“
  - „Der Fehler X in Y ist gefixt…“
  - „Halt! Diese Version hat einen Fehler! Ich gebe dir meine…“

Probleme bei der Entwicklung (3/3)



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++;
        }
    }
}
```

Aha! (Warum auch selber suchen?!)

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++;
        }
    }
}
```

Versionskontrolle: einige Features (1/2)

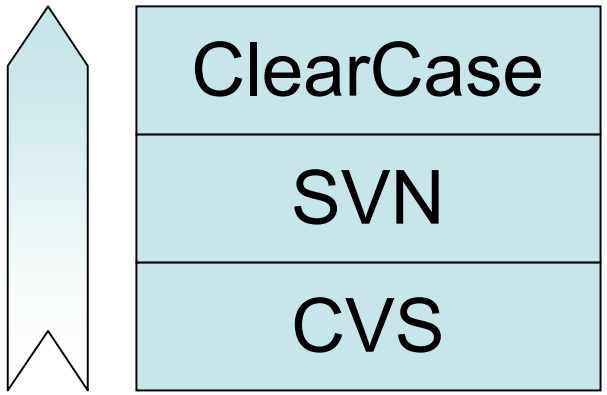
- Zentrale (und sichere) Verwaltung des Projekts in einem Repository
  - keine redundanten Kopien
- alle Unterschiede zwischen zwei oder mehr Versionen erkennbar
  - „Welche Änderungen gab es im Vergleich zu gestrigen Version?“
  - „Was haben verschiedene Entwickler in der Klasse X verändert?“
- ermöglicht Änderungen (eigene und fremde) zu
  - begutachten
  - übernehmen
  - verwerfen (kommt auch vor)
  - diskutieren (!)

Versionskontrolle: einige Features (2/2)

- vorherige Zustände können wiederhergestellt werden
  - komplett oder partiell
- Fixierung des Projekts in unveränderliche Versionen
  - Releases, Milestones oder einfach nur Backups
- Metadaten
  - wer hat wann was warum wo gemacht?

- **SVN**: Verzeichnisse und Dateien können umbenannt / verschoben werden
  - Metainformationen bleiben erhalten! (nicht wie bei CVS)

Versionskontrolle: Systeme



High-End

gute Mittelklasse

das Minimum

- **Jetzt**: Einführung in SVN
  - für die Übungsarbeit in Gruppen
- **Später**: eigene Vorlesung zu *Konfigurationsmanagement*
  - inklusive CVS und ClearCase

Subversion (SVN)

SVN: erste Schritte

- **Sharing**: Projekt unter Versionskontrolle bringen
- **Check Out**: Projekt initial herunterladen
- **Commit**: Änderungen in das Repository schreiben
- **Update**: Änderungen aus dem Repository übernehmen

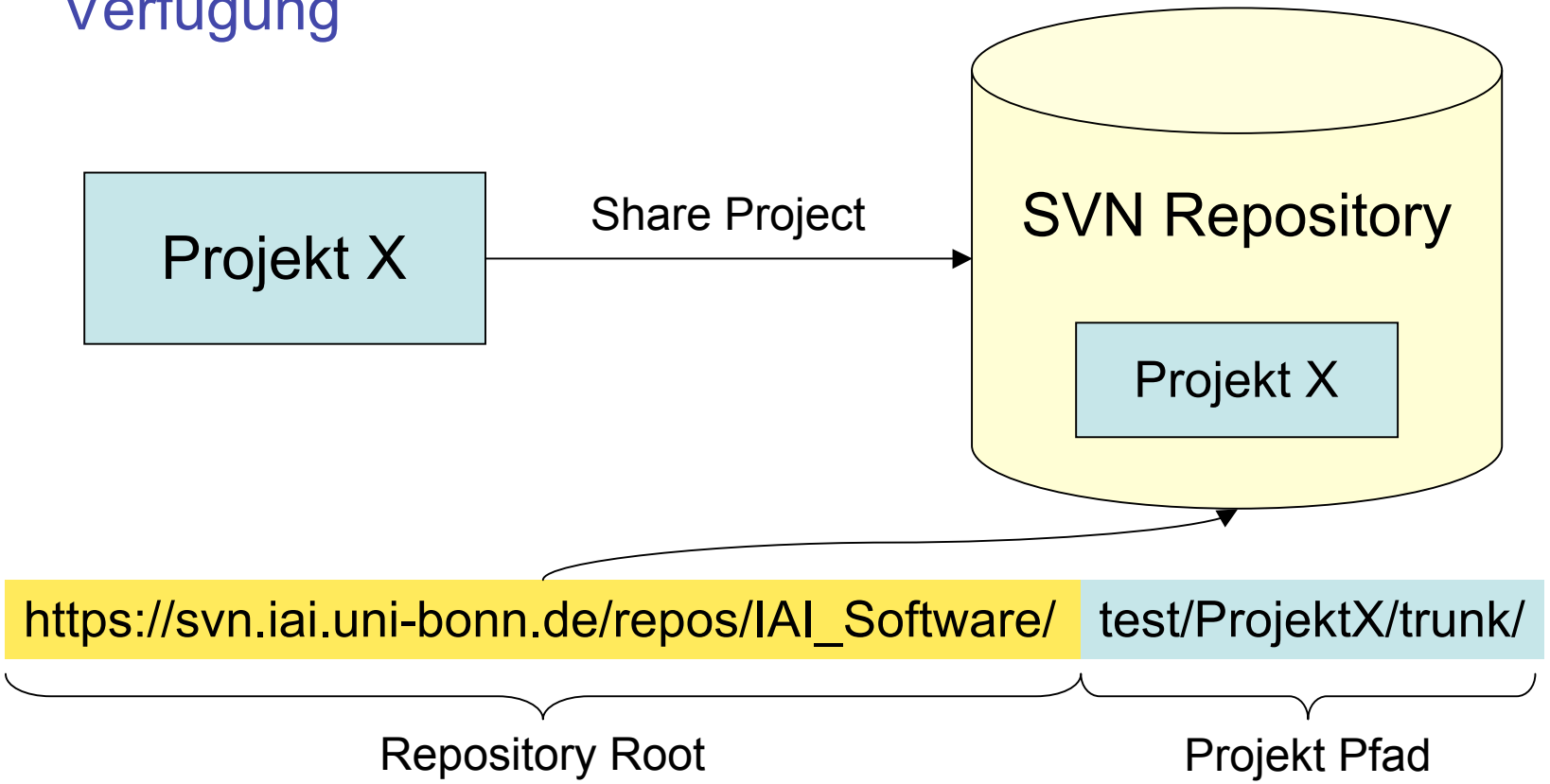
- Layout des Repositorys (**trunk**, **tags, branches**)

Das SVN Plugin integriert all diese Funktionen in die Eclipse IDE.

→ **Demo im Anschluss**

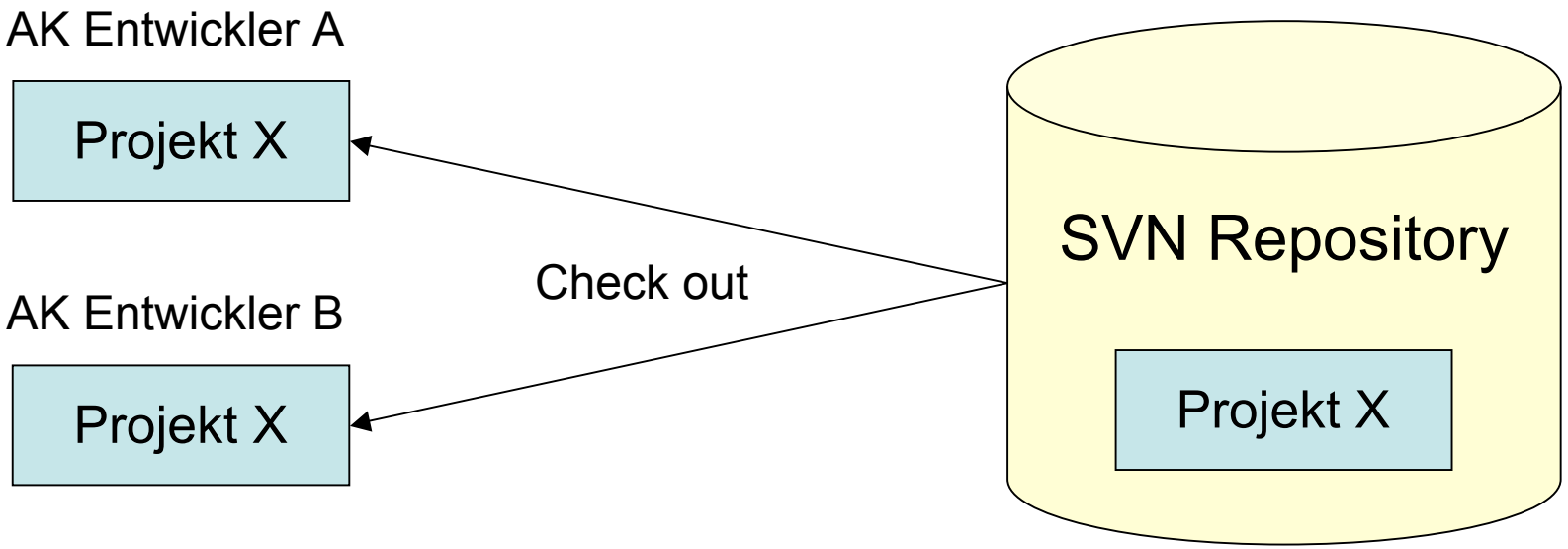
Sharing: Projekt unter Versionskontrolle bringen

- das Projekt war vorher nur lokal vorhanden,
- wird durch **Sharing** dem Repository hinzugefügt und
- steht nun anderen (autorisierten) Entwicklern zur Verfügung



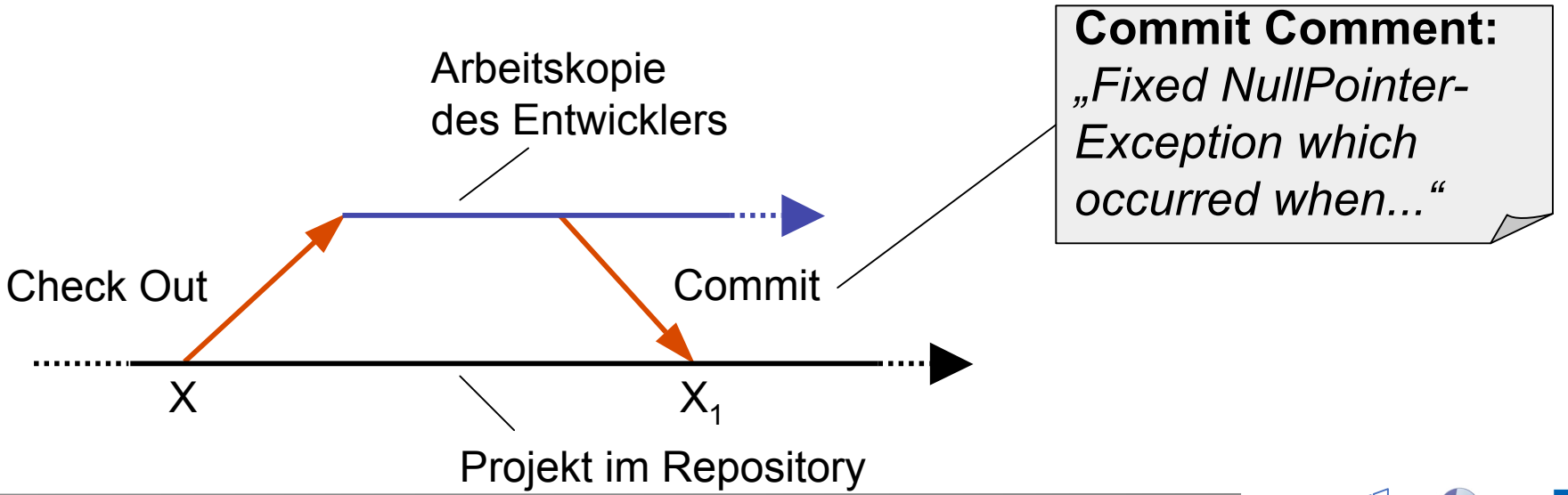
Check Out: Projekt initial herunterladen

- die Entwickler checken das Projekt aus
- und erhalten eine aktuelle, **lokale** Kopie (**Arbeitskopie**)
- von nun an können sie ihre Arbeit an dem Projekt beginnen
- Check Out nur beim ersten Mal! Danach → **Update**



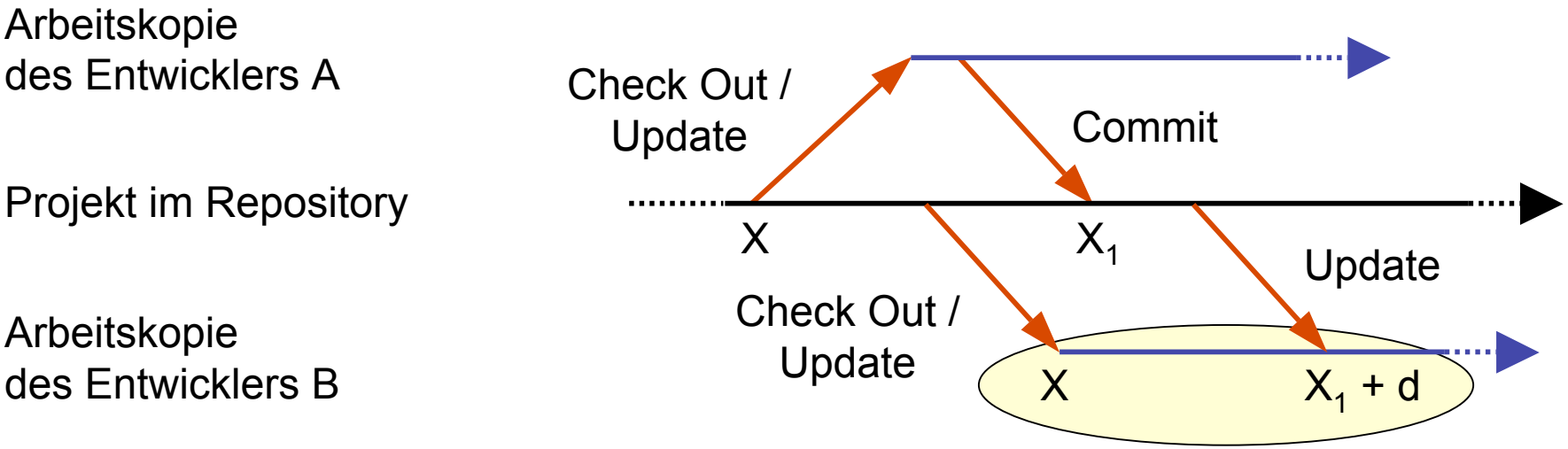
Commit: Änderungen in das Repository schreiben

- nach dem Auschecken ändert der Entwickler das Projekt **lokal** und
- bringt anschließend seine Änderungen in das Repository ein (Commit)
- Commits sind **atomar** (alle Änderungen oder keine)
- eine Notiz (**Commit Comment**) beschreibt **warum** die Änderung durchgeführt wurde



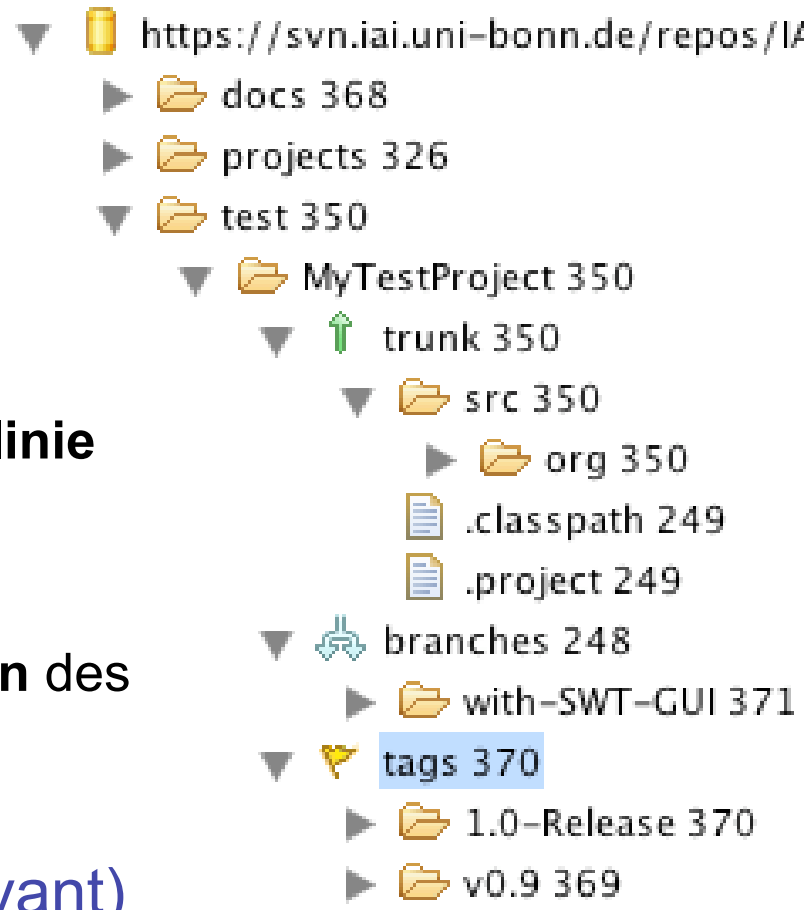
Update: Änderungen aus dem Repository übernehmen

- übernimmt den **aktuellen Stand** des Repositorys in die lokale Arbeitskopie
- nach einer Pause immer zuerst die Arbeitskopie aktualisieren, bevor man weiter arbeitet!
- Nachteil: Änderungen werden **ungesehen** übernommen besser: → **Synchronisieren**



Layout des Repositorys (1/2)

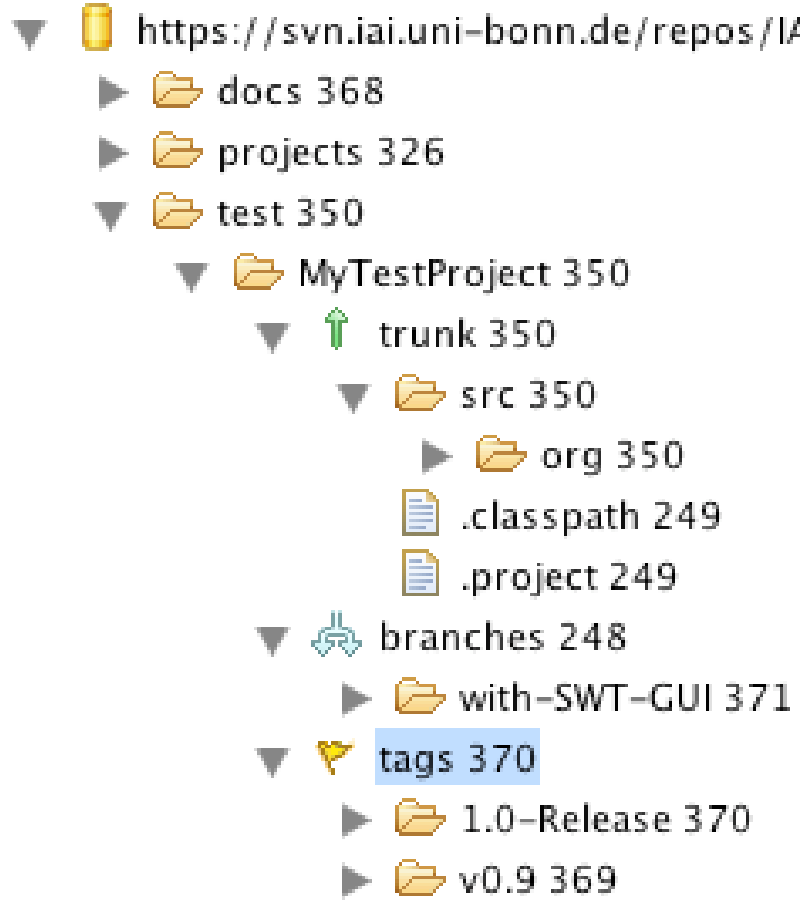
- SVN organisiert Projekte in drei spezielle Verzeichnisse:



- **trunk**
  - enthält die **aktuelle Entwicklungslinie** (war HEAD in CVS)
- **tags**
  - enthält **unveränderliche Versionen** des Projekts
    - für Releases, Milestones, Backups
- **branches** (hier nicht weiter relevant)
  - enthält **Verzweigungen** des Projekts
    - für Untergruppen eines großen Teams oder alternative Projektentwicklungen

Layout des Repositorys (2/2)

- Konvention:
  - trunk, branches, tags befinden sich **unterhalb** des Projektnamens
  - In diesem Fall den **Trunk** auschecken!
- das ist nur eine Konvention
  - Abweichungen sind legitim
- trunk, tags, branches sind normale Verzeichnisse!
  - werden von SVN aufgrund ihres Namens gesondert behandelt

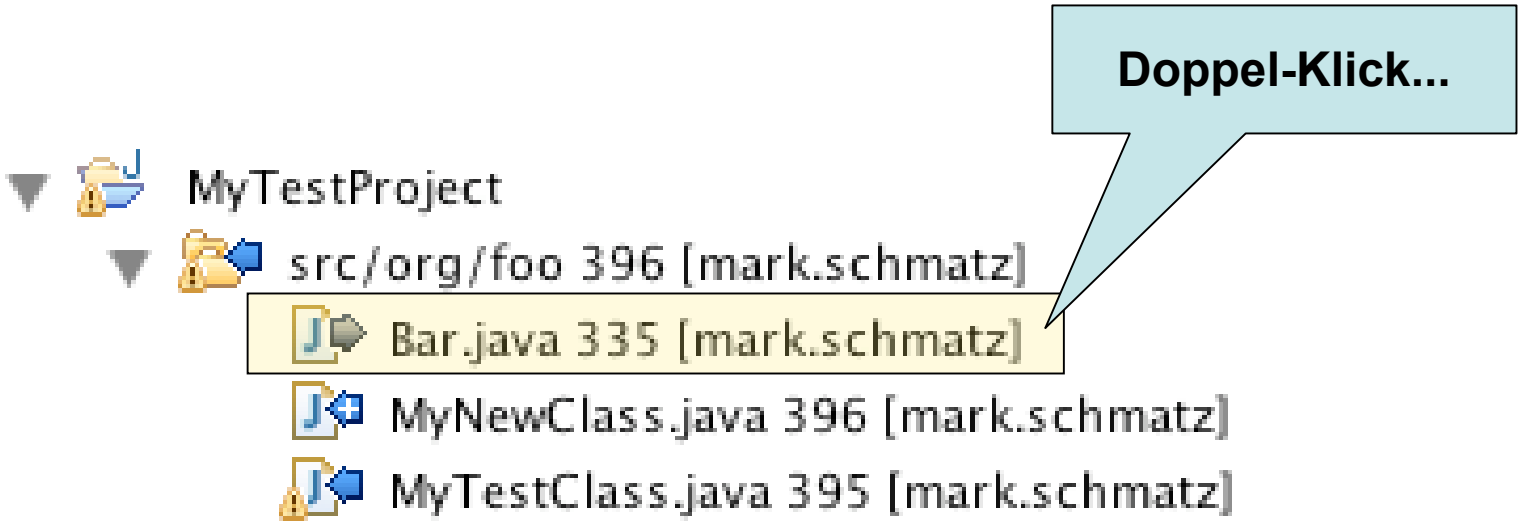


Fortgeschrittene Techniken

- **Synchronize**: Projekt synchronisieren
- Konflikte
- Wiederherstellung gelöschter Dateien

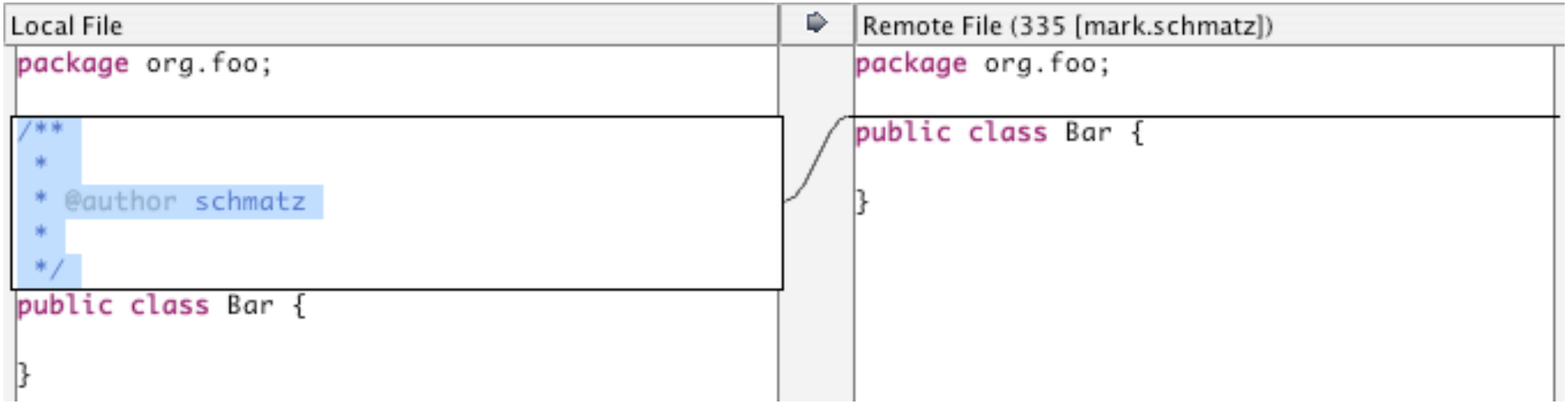
Synchronize: Projekt synchronisieren

- Änderungen werden **nicht** ungesehen übernommen
- **selektive Updates** möglich
- **ausgehende** und **einkommende** Änderungen werden übersichtlich dargestellt

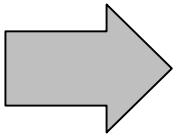


Synchronize: Projekt synchronisieren

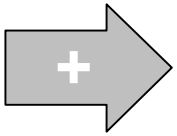
- ... öffnet das **Compare** Fenster
  - die Versionen der lokalen Arbeitskopie und des Repositorys werden gegenübergestellt
  - partielle Updates nun möglich



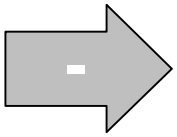
Synchronize: Projekt synchronisieren



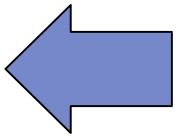
Datei wurde in der AK verändert und existiert im Repository
→ Änderungen in das Repository übernehmen



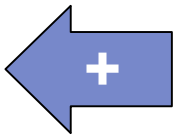
Datei wurde der AK hinzugefügt und existiert nicht im Repository
→ Datei dem Repository hinzufügen



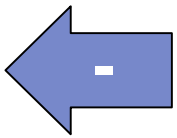
Datei wurde aus der AK gelöscht und existiert im Repository
→ Datei aus dem Repository entfernen



Datei wurde im Repository verändert und existiert in der AK
→ Änderungen in die AK übernehmen

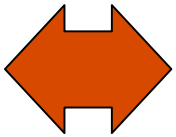


Datei wurde dem Repository hinzugefügt und existiert nicht in der AK
→ Datei der AK hinzufügen



Datei wurde aus dem Repository gelöscht und existiert in der AK
→ Datei aus der AK entfernen

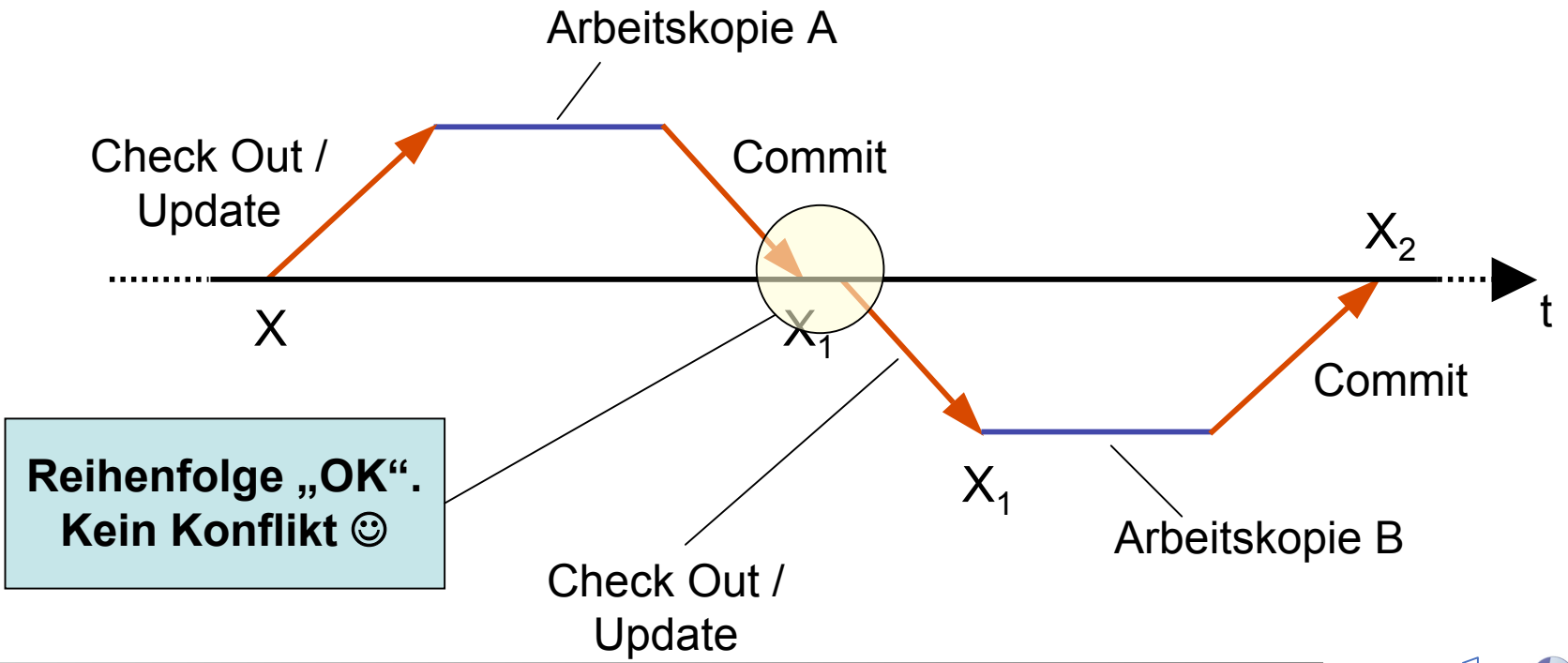
Synchronize: Projekt synchronisieren



Datei wurde in der AK **und** im Repository verändert → **Konflikt!**
→ Manuelle Lösung des Konflikts notwendig

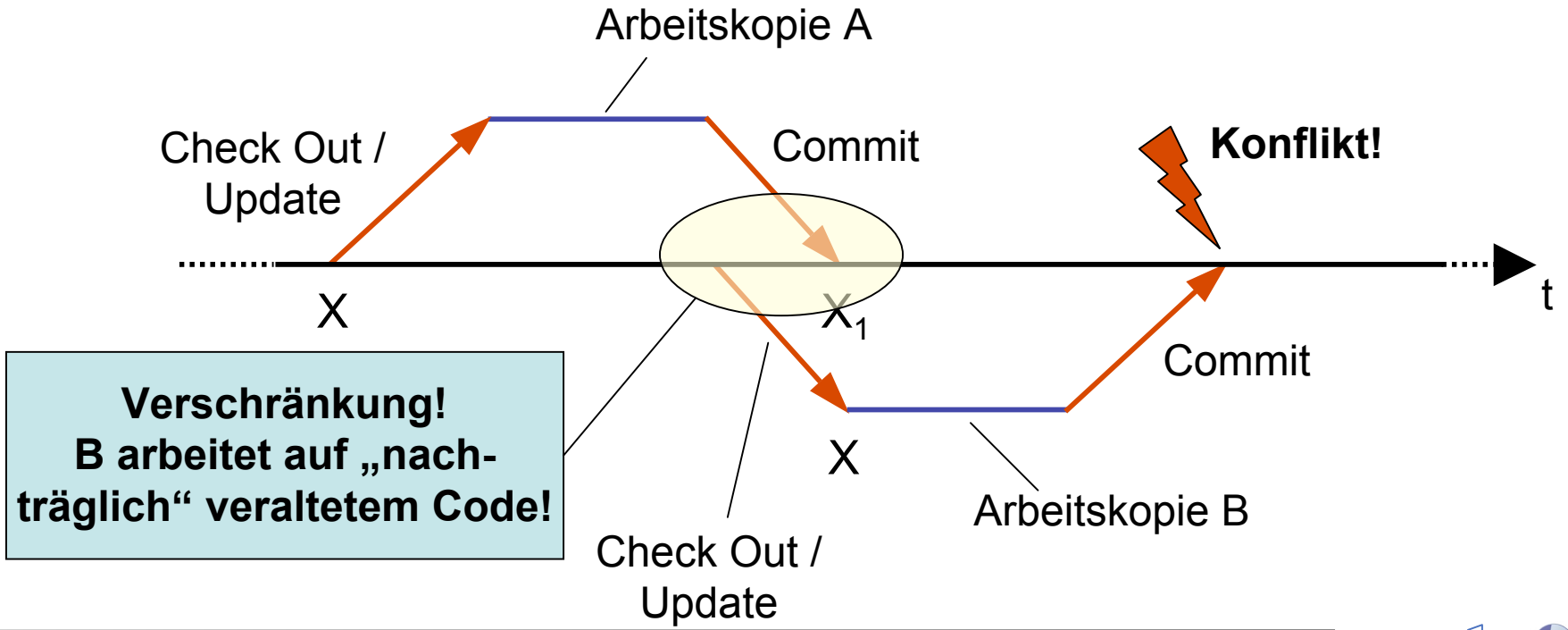
Konflikte

- keine Probleme, solange nur „**seriell**“ gearbeitet wird
  - d.h. keine **Verschränkung** von Check Out / Update und Commit



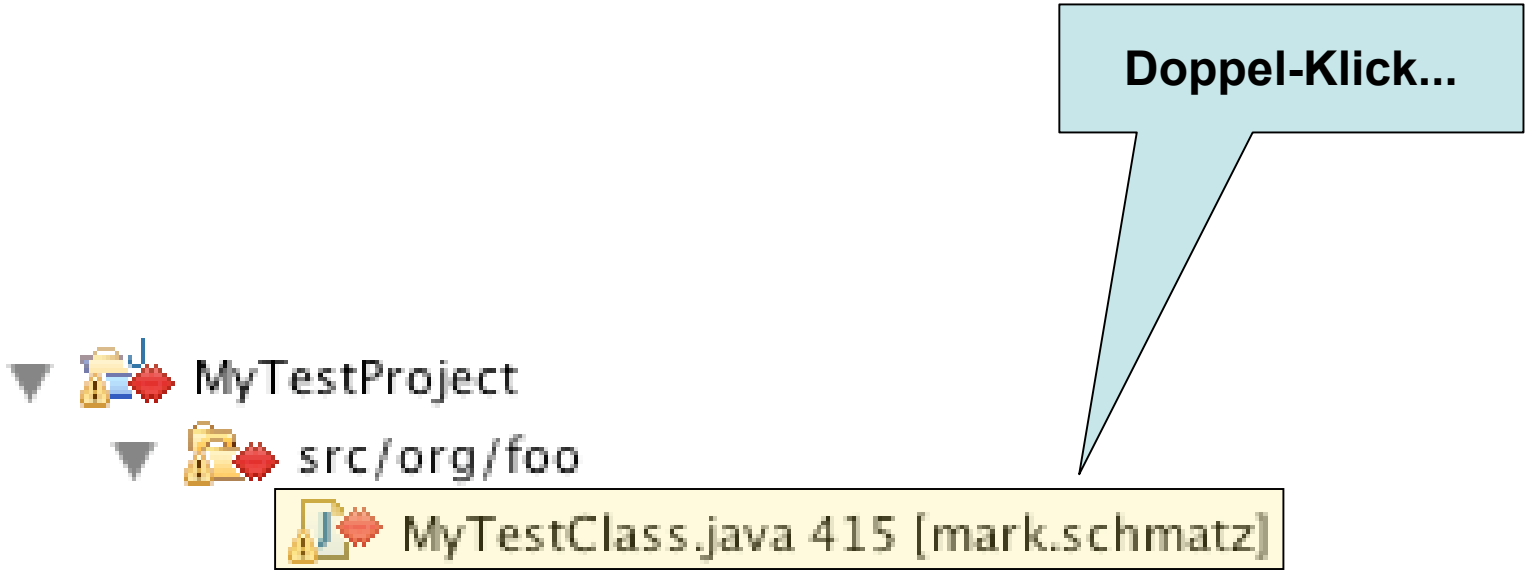
Konflikte

- keine Probleme, solange nur „**seriell**“ gearbeitet wird
  - d.h. keine **Verschränkung** von Check Out / Update und Commit
- doch dies ist im Allgemeinen nicht der Fall
- manuelle Lösung des Konflikts erforderlich



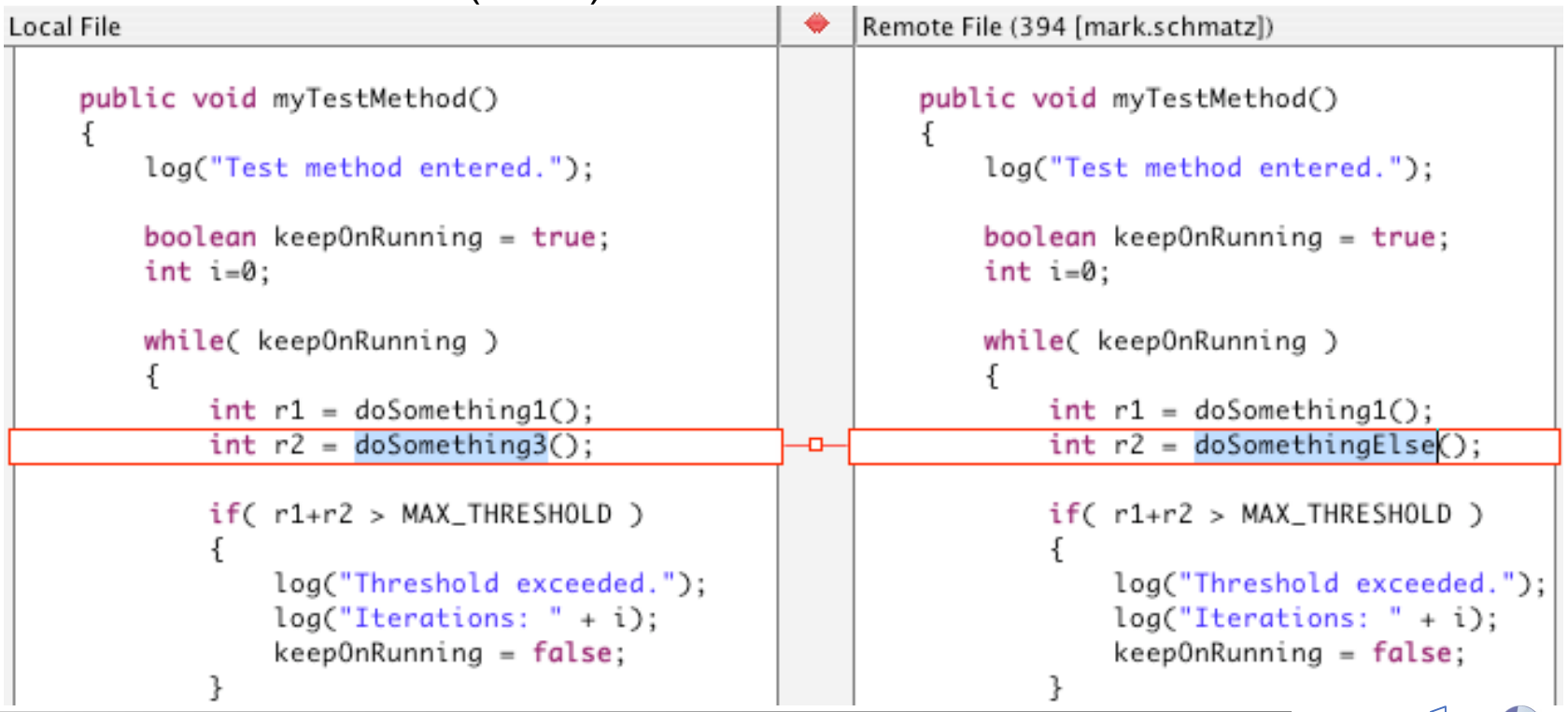
Konflikte lösen

- Konflikte können nur vom Menschen gelöst werden
- SVN unterstützt, der Entwickler entscheidet
- in 90 Prozent aller Fälle einfach, da unterschiedliche Codeblöcke betroffen sind



Konflikte lösen

- SVN stellt fremde und eigene Version gegenüber
- Konflikte lösen heißt Änderungen...
  - in die Arbeitskopie übernehmen (**Overwrite and Update**) oder
  - eigene Version ins Repository schreiben (**Overwrite and Commit**)
    - OaC bedarf (meist) der Kommunikation!



Wiederherstellung gelöschter Dateien

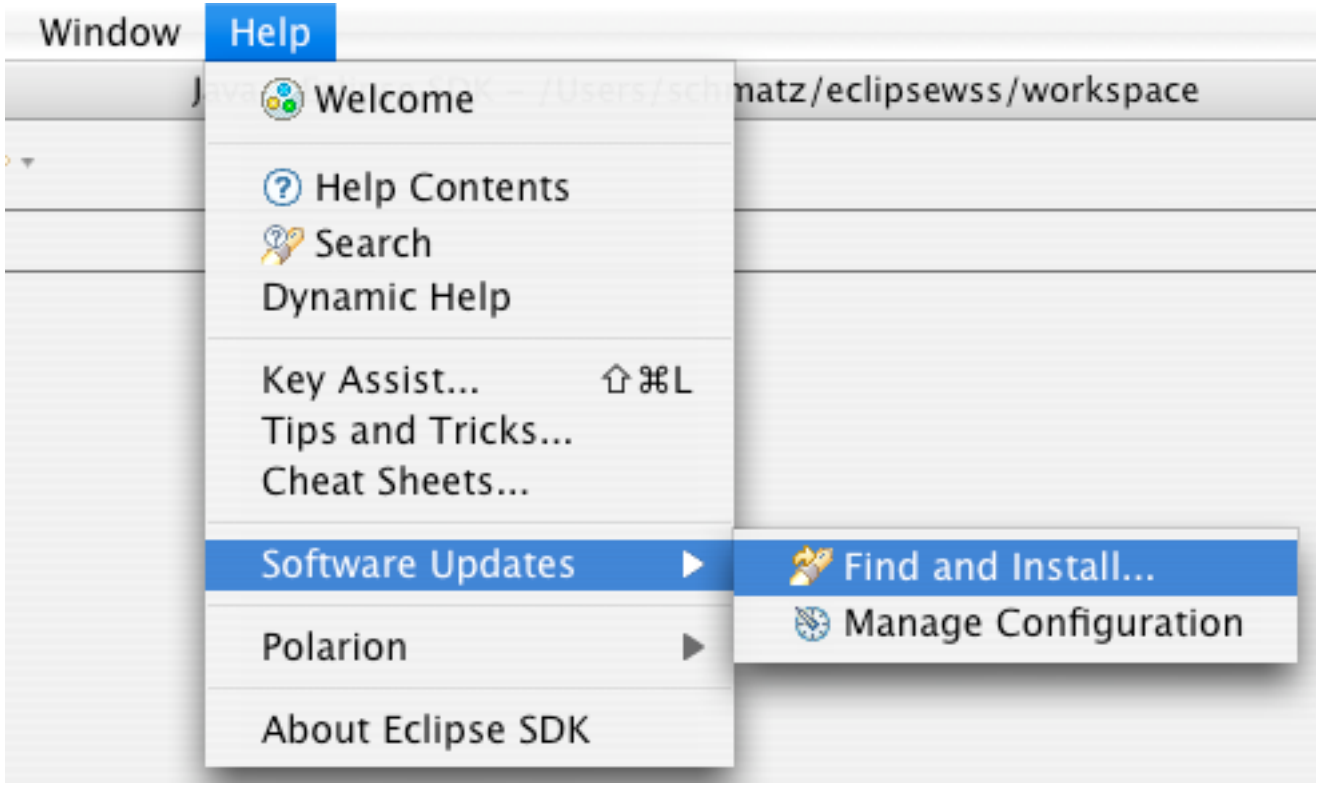
- das Plugin kennt diese Funktionalität leider nicht!
  - obwohl Grundsätzlich in SVN vorhanden: **SVN copy** (als Kommandozeilen-Befehl)
  - siehe hierzu SVN Buch: Abschnitt „*Resurrecting Deleted Items*“
- Work-around 1: **Merging**
  - zu grob-granular, denn **alle** Änderungen, welche mit dem jeweiligen Commit in das Repository eingebracht wurden, werden wieder rückgängig gemacht. Daraus folgt:
  - **Best Practise**:
    1. Synchronisieren: alles „committen“ und „updaten“
    2. Datei löschen
    3. Löschung „committen“
- Work-around 2: aus **History** kopieren und neu anlegen
  - bei Binärdateien unpraktikabel
  - alle bisherigen Metainformationen gehen verloren!

Subversive Plugin

Installation

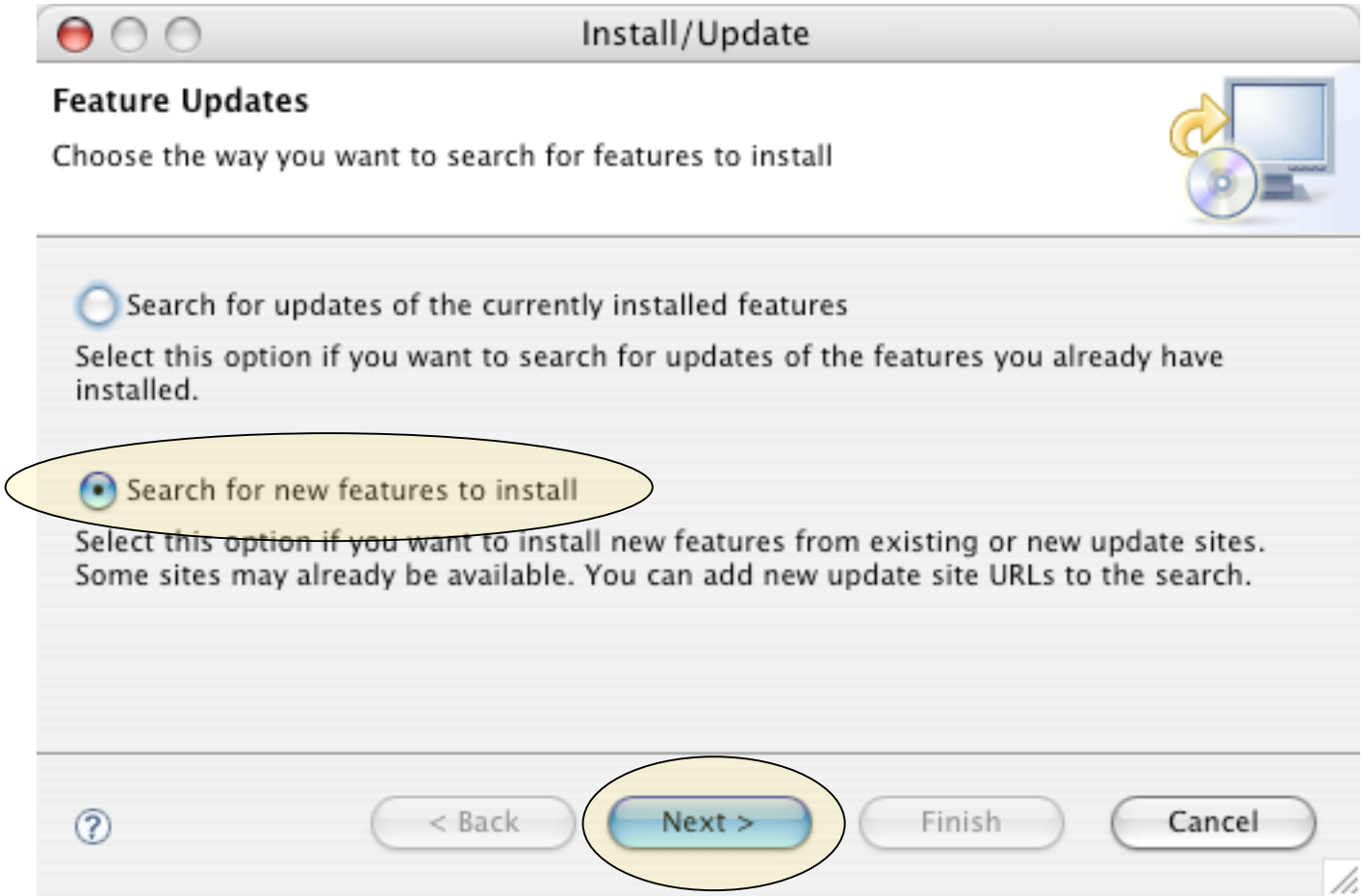
Subversive installieren

- über das **Help** Menu von Eclipse
  - Help → Software Updates → Find and Install...



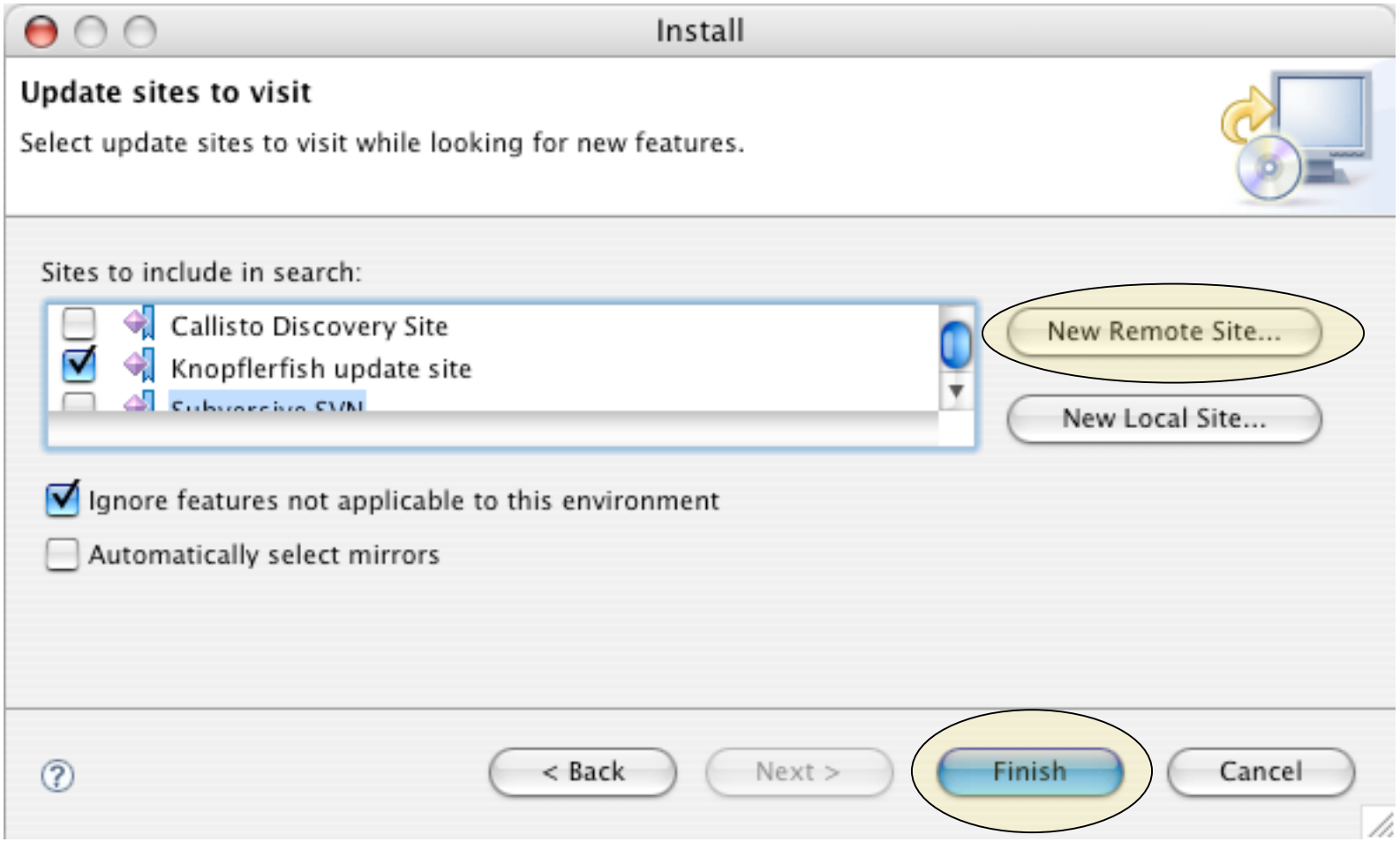
Subversive installieren

- Search for new features to install → Next



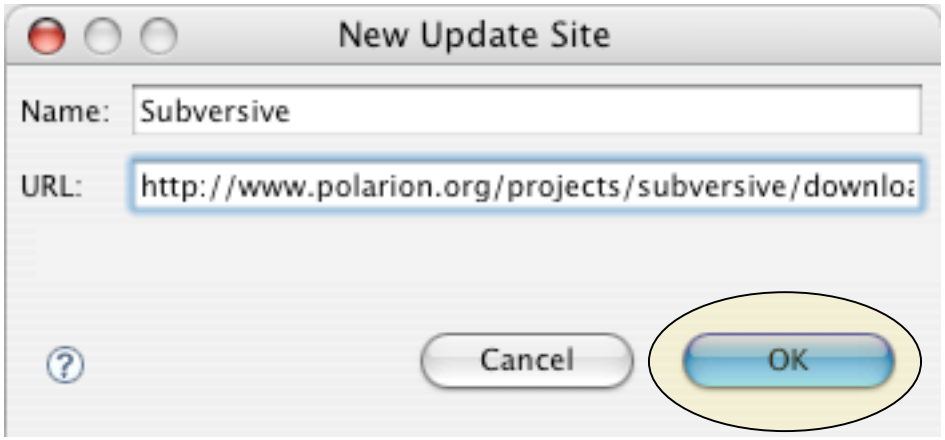
Subversive installieren

- New Remote Site → Finish



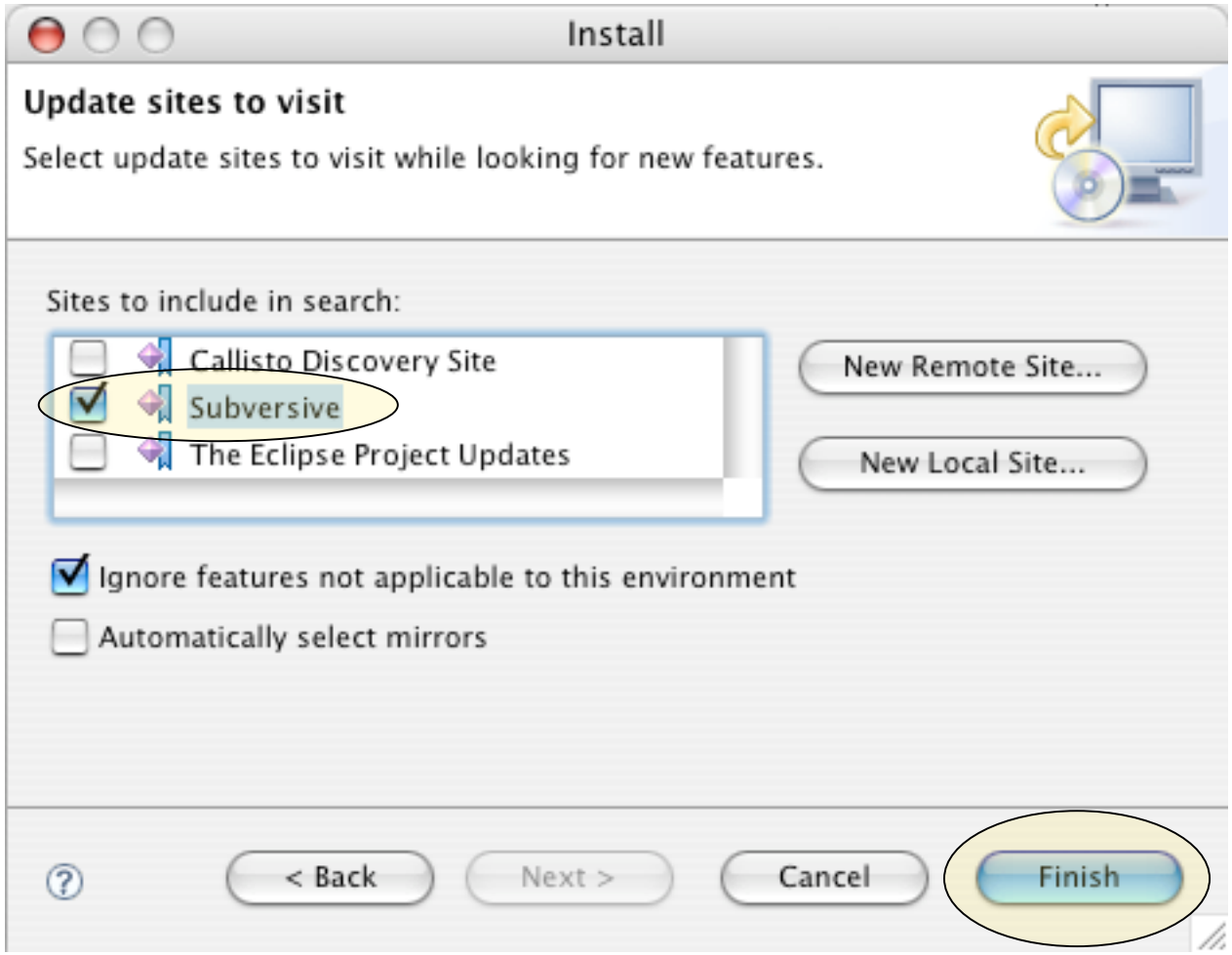
Subversive installieren

- Name:
  z.B.: Subversive
- URL:
  http://www.polarion.org/projects/subversive/download/1.1/update-site/

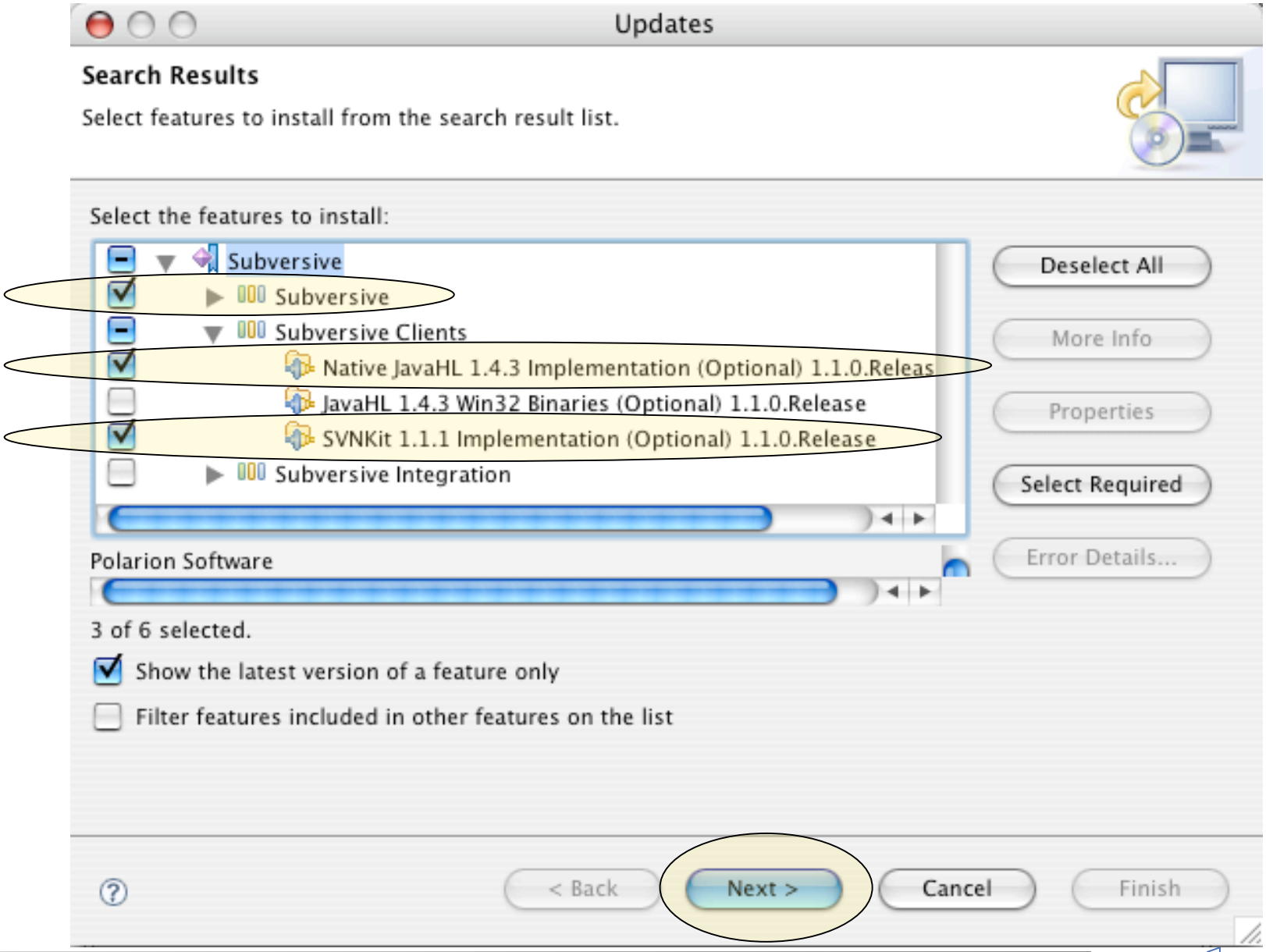


Subversive installieren

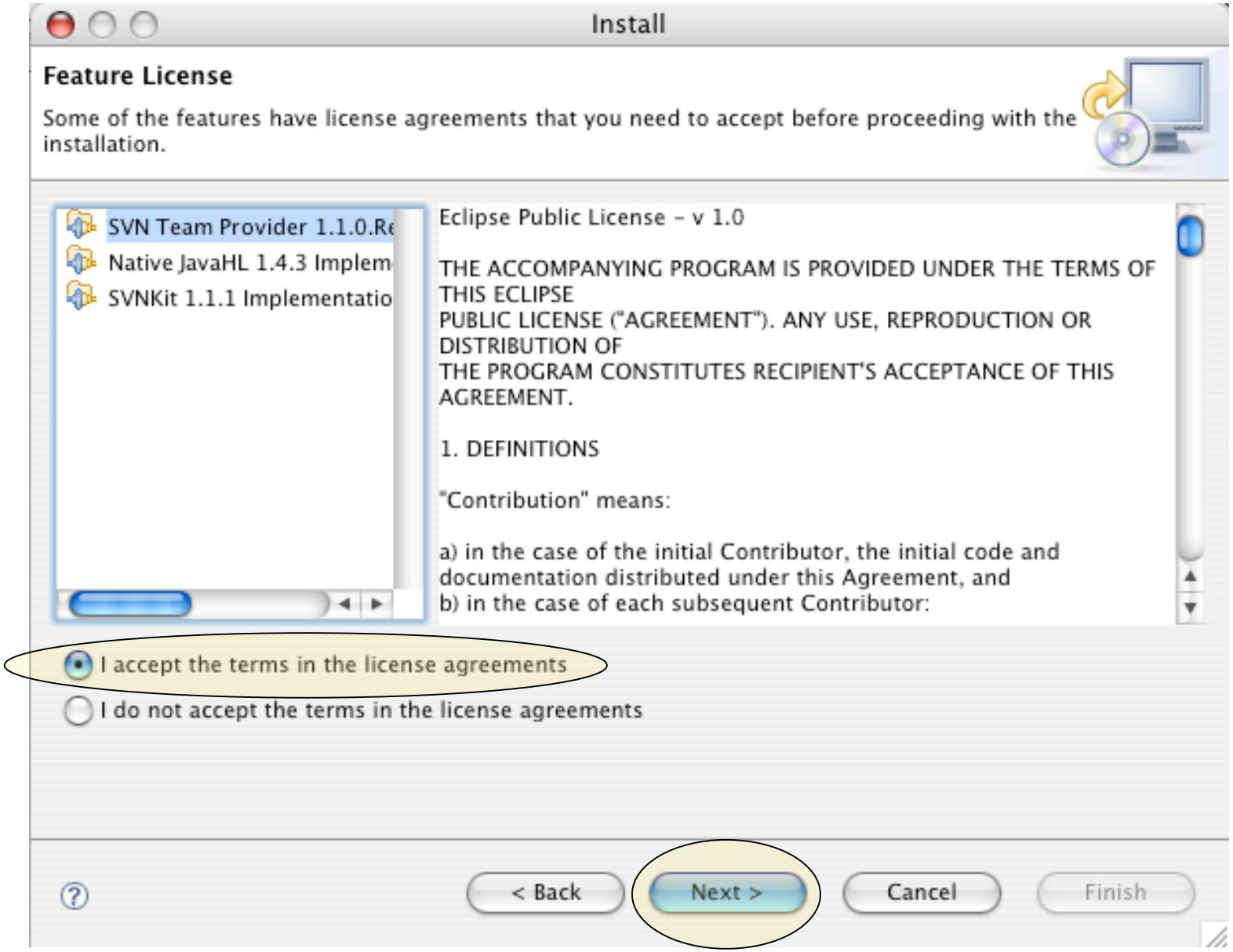
- Subversive Site selektieren → Finish



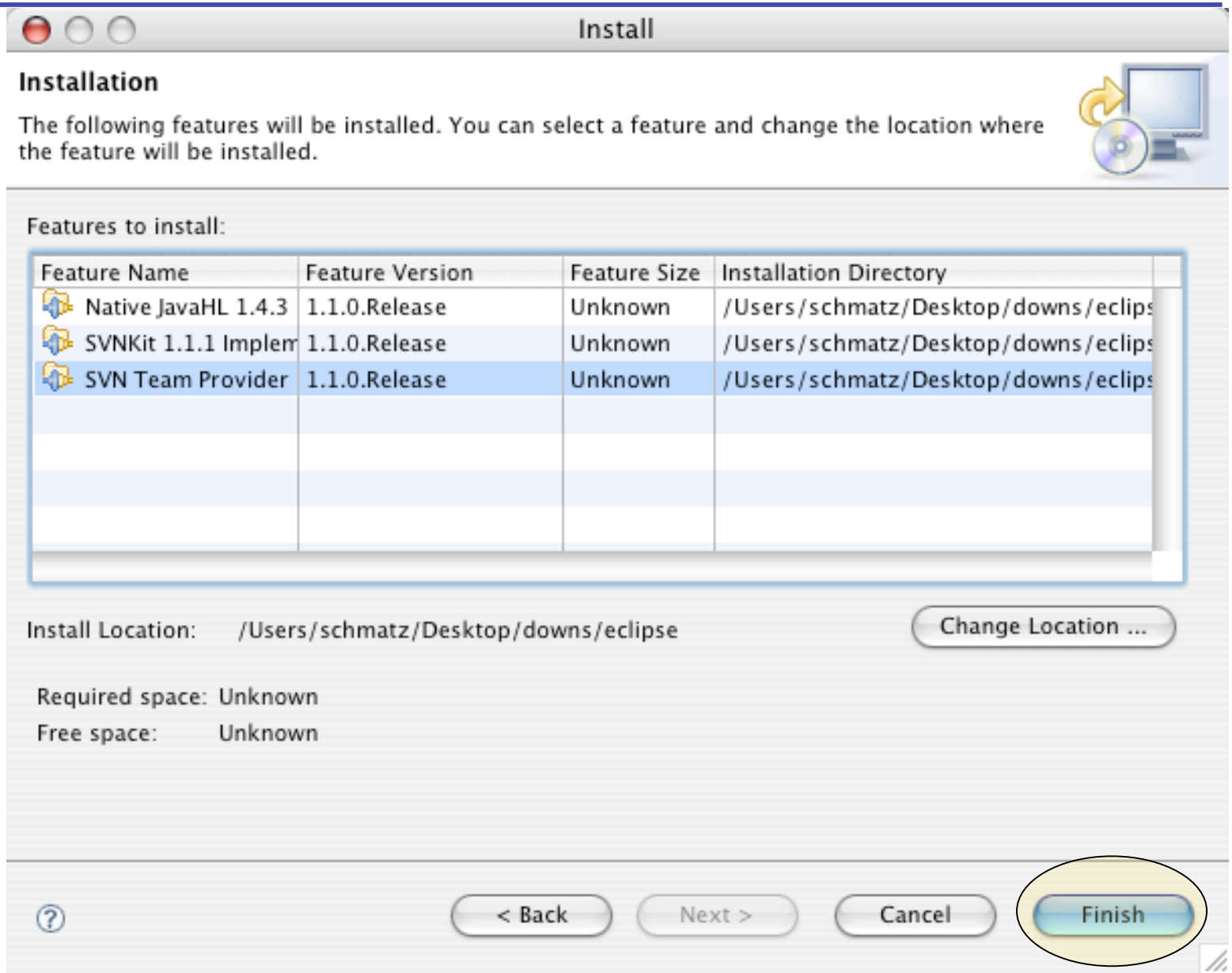
Subversive installieren



Subversive installieren



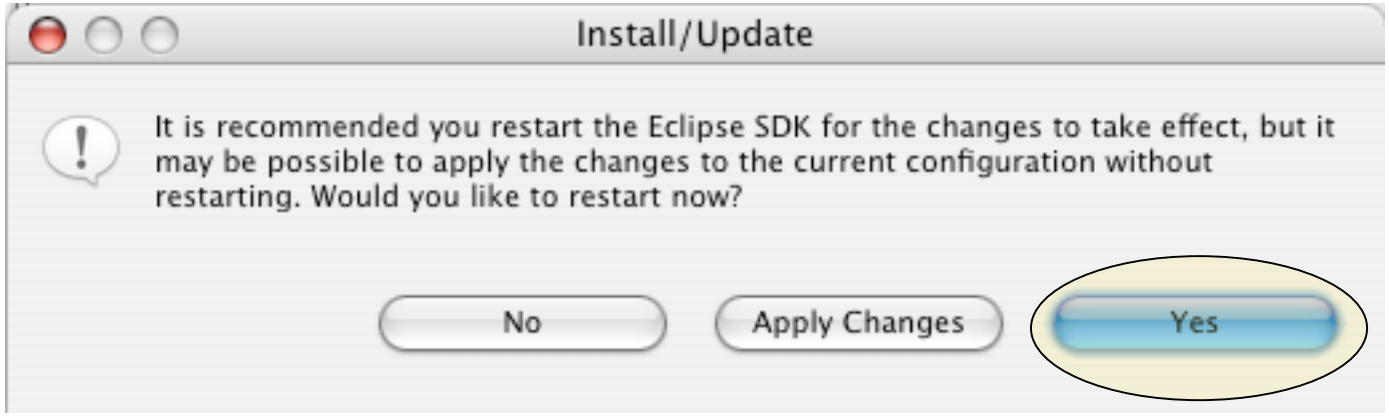
Subversive installieren



Subversive installieren

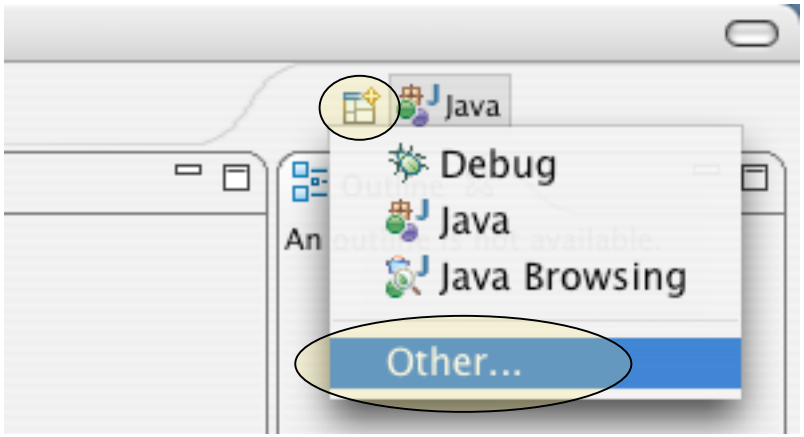
Subversive installieren

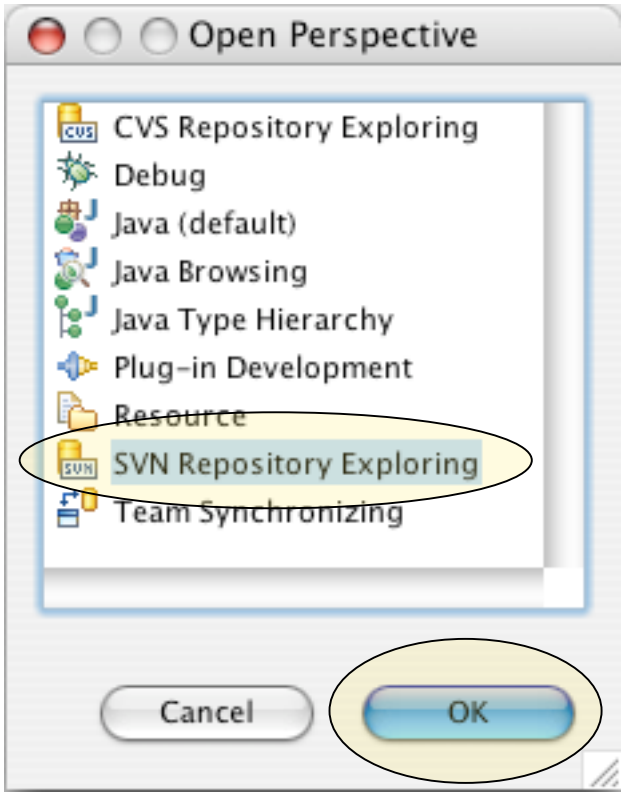
- Eclipse neu starten



Subversive installieren

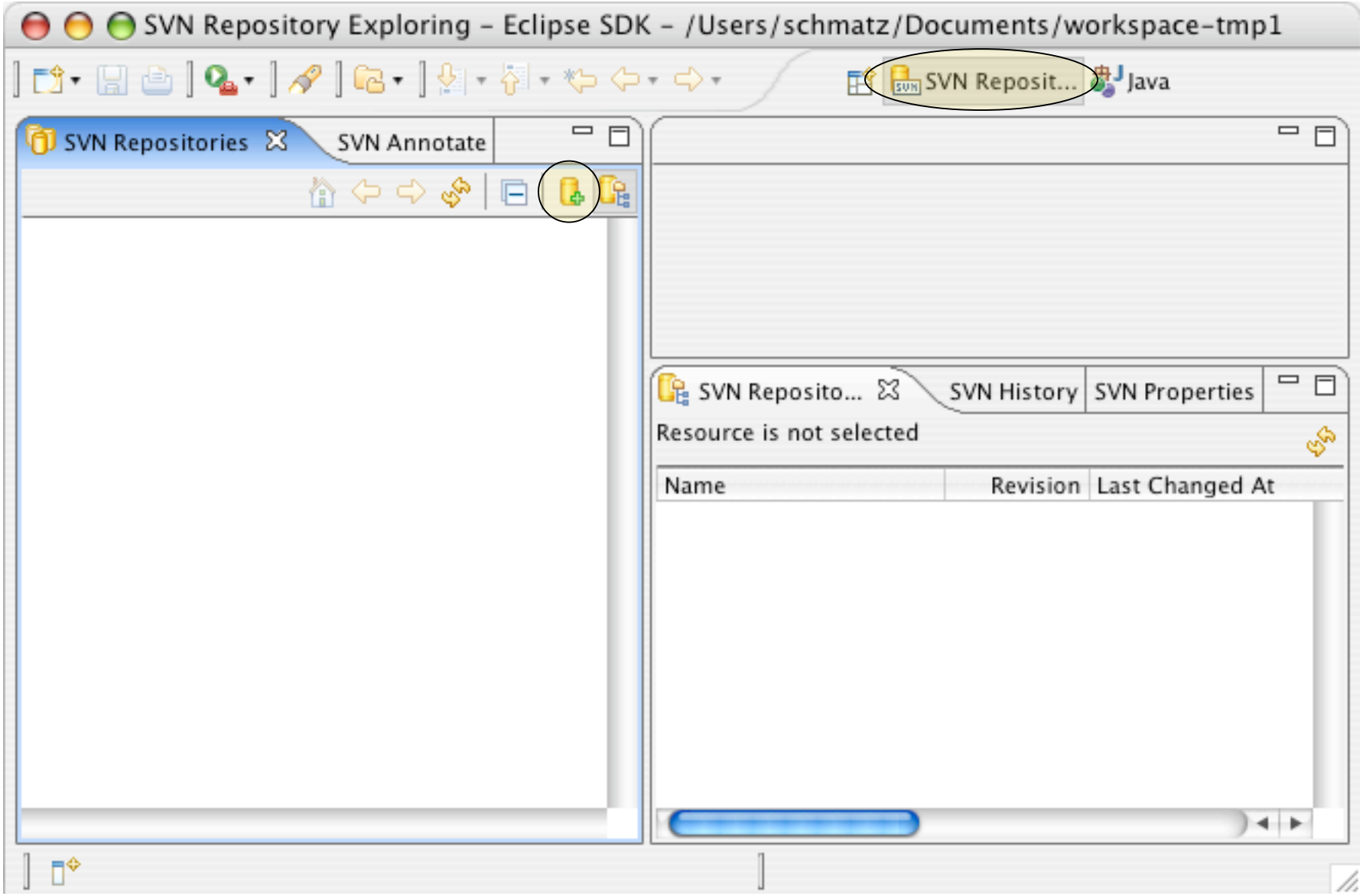
- SVN Perspective hinzufügen



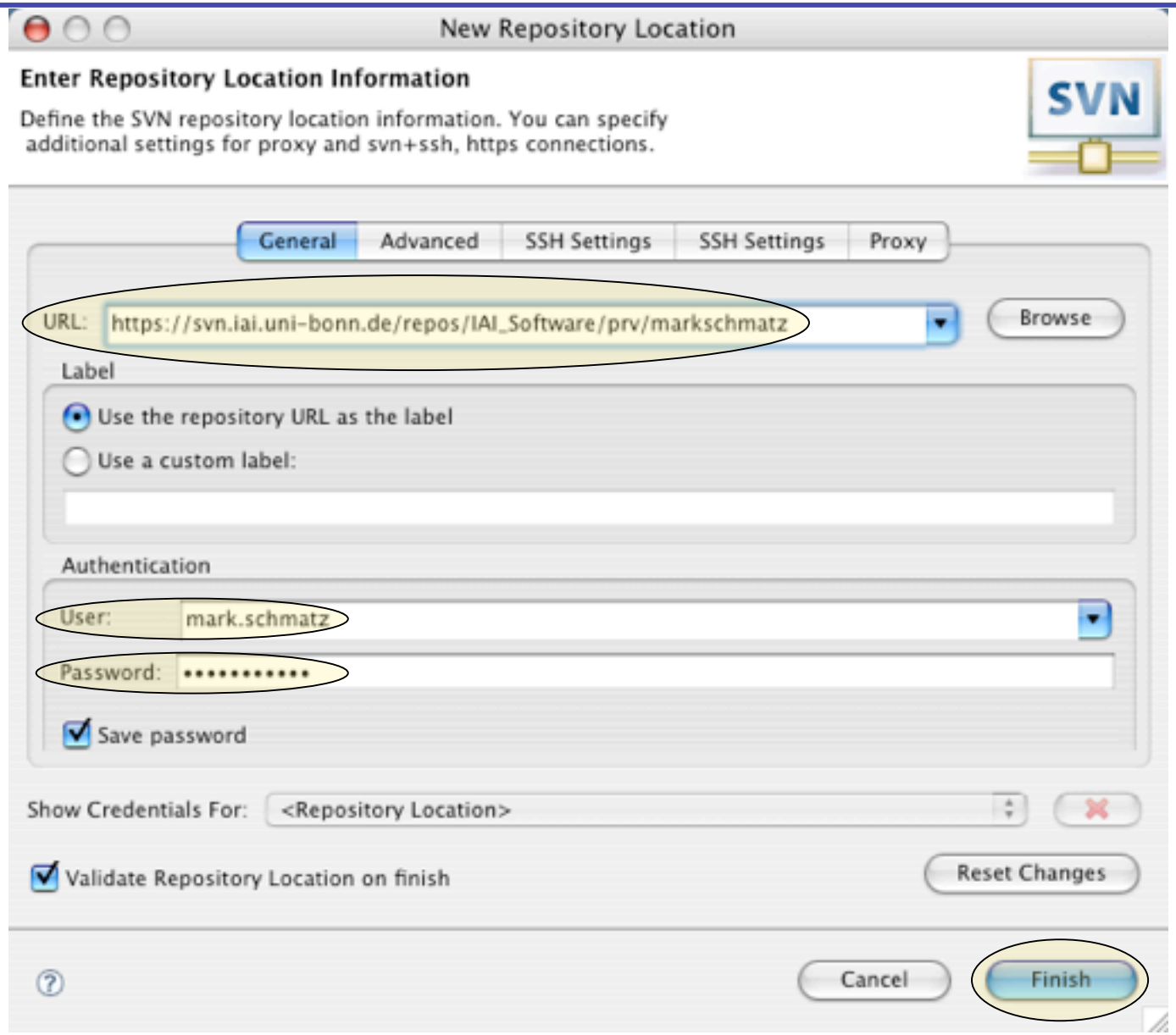


Repository einrichten

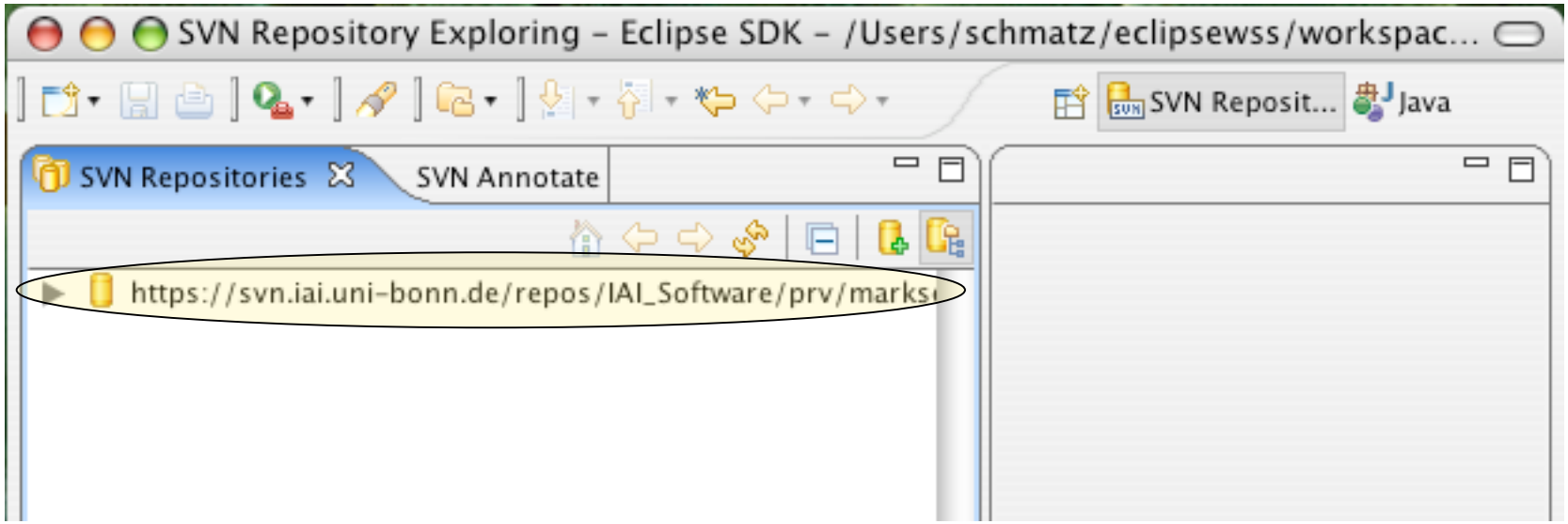
Repository einrichten



Repository einrichten



Repository einrichten



Subversive Plugin

Demo

Links

- Subversion: subversion.tigris.org
- Subversive: www.polarion.org
  - Plugin Download Site: http://www.polarion.org/projects/subversive/download/1.1/update-site/
- SVN Buch:
  - http://svnbook.red-bean.com/nightly/en/svn-book.pdf
- SWT Wiki (SoSe 2007):
  - http://butterbur03.iai.uni-bonn.de/rootstiki/Lecture_SWT_SS2007
- SVN Wiki:
  - http://butterbur03.iai.uni-bonn.de/rootstiki/KB_SVN
- SVN FAQ:
  - http://butterbur03.iai.uni-bonn.de/rootstiki/tiki-index.php?page=KB_SVN_FAQ