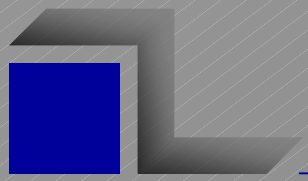


# Software Configuration Management (SCM)

- Motivation
- Definition
- Konzepte und Aufgaben
  - Einzelarbeit
  - Namensgebung und Nummerierung
  - Teamarbeit
- Anleitung
- Aufgabe

# Motivation: Problemsituationen beim Arbeiten im Team

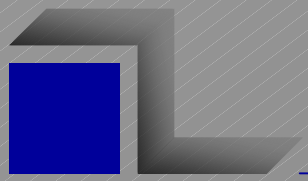


- Mehrere Entwickler ändern die gleichen Klassen
- Die Weiterentwicklung einer funktionierenden Version geht schief
  - "Das hat aber schon mal funktioniert..."
  - "Gestern ging es noch, seitdem haben wir nur ... geändert!"

  → *typische Prüfungssituation!*

Andere Problemsituation

- Ein Kunde hat noch Änderungswünsche / Fehler in einer alten, längst weiterentwickelten Version, will aber nicht zu einer neueren wechseln

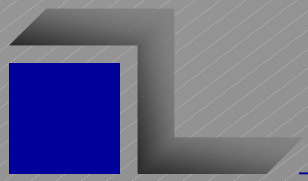


# Zwei Ziele des Software Configuration Management (SCM)



**Versionen kontrollieren**

**Parallelarbeit koordinieren**

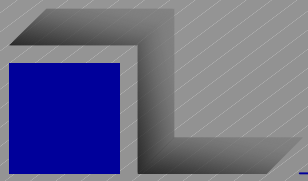


# Definition



*Software Configuration Management* heißt die Disziplin, die Konfiguration eines Systems zu bestimmten Zeitpunkten zu erfassen, um Änderungen daran systematisch und kontrolliert durchzuführen und die Integrität und die Nachvollziehbarkeit von Konfigurationen während der gesamten Lebensdauer sicherzustellen.

A Project of the Software Engineering Coordinating Committee:
*Guide to the Software Engineering Body of Knowledge - SWEBOK*.
Version 1.00 (Trial Version) – May 2001
(SWEBOK is an official service mark of the IEEE); www.swebok.org



# Konzepte und Aufgaben des SCM 1/2



- Ein Verzeichnis (*Repository*) enthält alle Versionen aller Dateien, die zu einem Programm gehören
- Verwaltet wird nicht nur Programmcode, sondern auch Anforderungsdokumente, Designdokumente, Benutzerdokumentation etc.
- Bei Bedarf auch Datenbank-Zustände!

# Konzepte und Aufgaben des SCM 2/2



- Restriktives Modell: SCCS, RCS, ...
  - Entwickler holen sich bestimmte Versionen von Dateien (*check out*)
    - → zum Lesen / Benutzen (beliebig viele Entwickler gleichzeitig)
    - → zum Schreiben (Verbessern, Ändern, Weiterentwickeln etc.): maximal ein Entwickler zu einem Zeitpunkt
  - Geändert werden können nur Dateien, die zum Schreiben aus dem Repository geholt wurden (*check in*)
- Optimistisches Modell: CVS, Subversion (SVN), ...
  - Entwickler holen sich bestimmte Versionen von Dateien (*check out* / *update*)
  - Geändert werden können alle Dateien (*commit*); Annahme: Es werden selten Konflikte zwischen Änderungen auftreten
- Geänderte Dateien werden nach korrektem Abschluss der Änderungen kommentiert mit neuer Versionsnummer wieder ins Repository gestellt (*check in* / *commit*)

# Konzept für das restriktive Modell: Dateien und Verzeichnisse



**Heinz**

Lokales Verzeichnis
- Binaries aller zuletzt freigegebenen Dokument-Versionen
- Quellen der aktuell von Heinz bearbeiteten Dokumente

Repository:
Alle Versionen
aller Dateien

**Gerd**

Lokales Verzeichnis
- Kopie aller freigegebenen Versionen (Binaries)
- Quellen der aktuell von Gerd bearbeiteten Dokumente

**Hugo**

Lokales Verzeichnis
- Binaries aller zuletzt freigegebenen Dokument-Versionen
- Quellen der aktuell von Hugo bearbeiteten Dokumente

Binaries:
- Im Fall von Java: .class-/.jar-Dateien
- Im Fall von z.B. C/C++: .o-Dateien

# Konzept für das optimistische Modell: Dateien und Verzeichnisse



**Heinz**

Verzeichnis
- Quellen aller zuletzt freigegebenen Dokument-Versionen; Heinz darf alle bearbeiten

Repository: Alle Versionen aller Dateien

**Gerd**

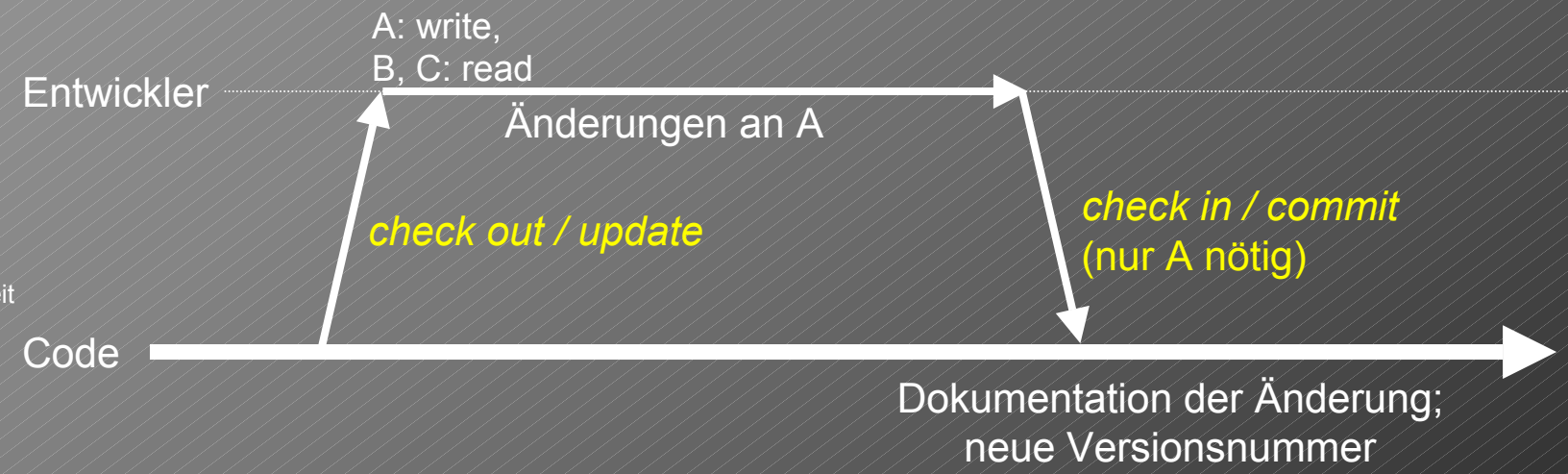
Verzeichnis
- Quellen aller zuletzt freigegebenen Dokument-Versionen; Gerd darf potenziell alle bearbeiten

**Hugo**

Verzeichnis
- Quellen aller zuletzt freigegebenen Dokument-Versionen; Hugo darf potenziell alle bearbeiten

# Konzept: Einzelarbeit

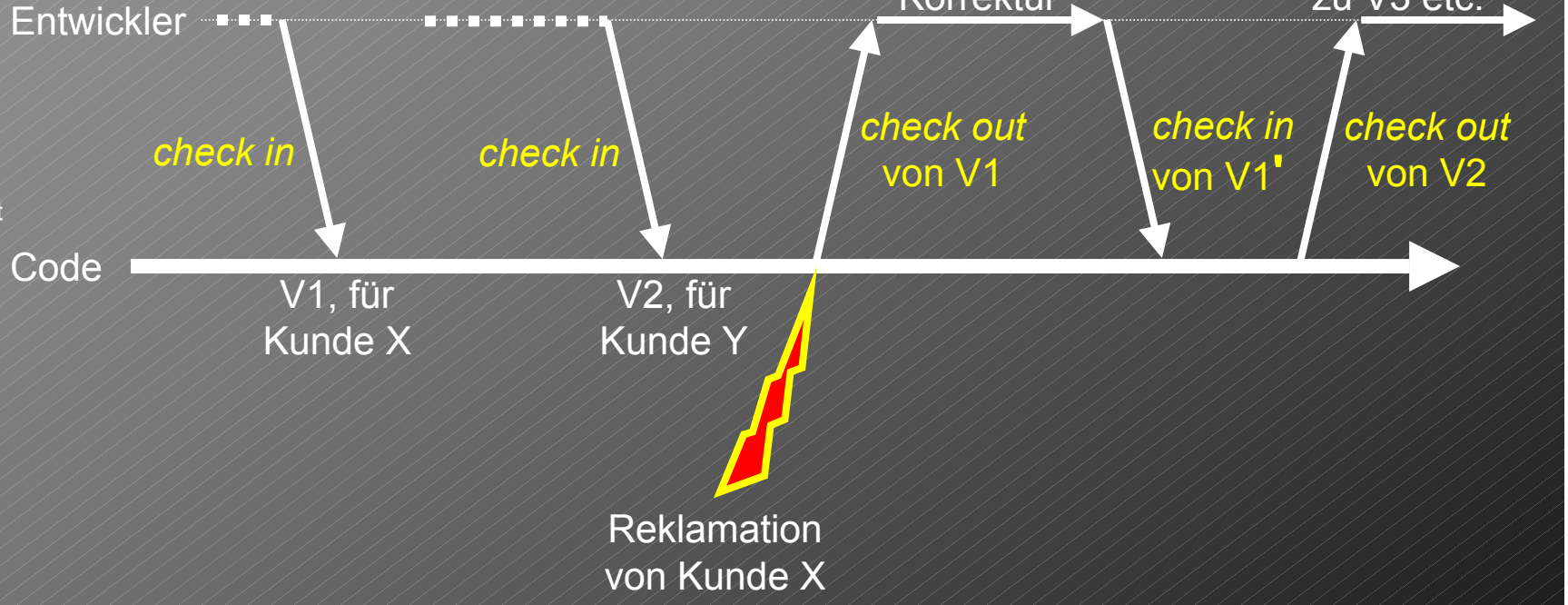





zeitlicher Verlauf

# Konzept: Einzelarbeit, Rückgriff auf alte Version

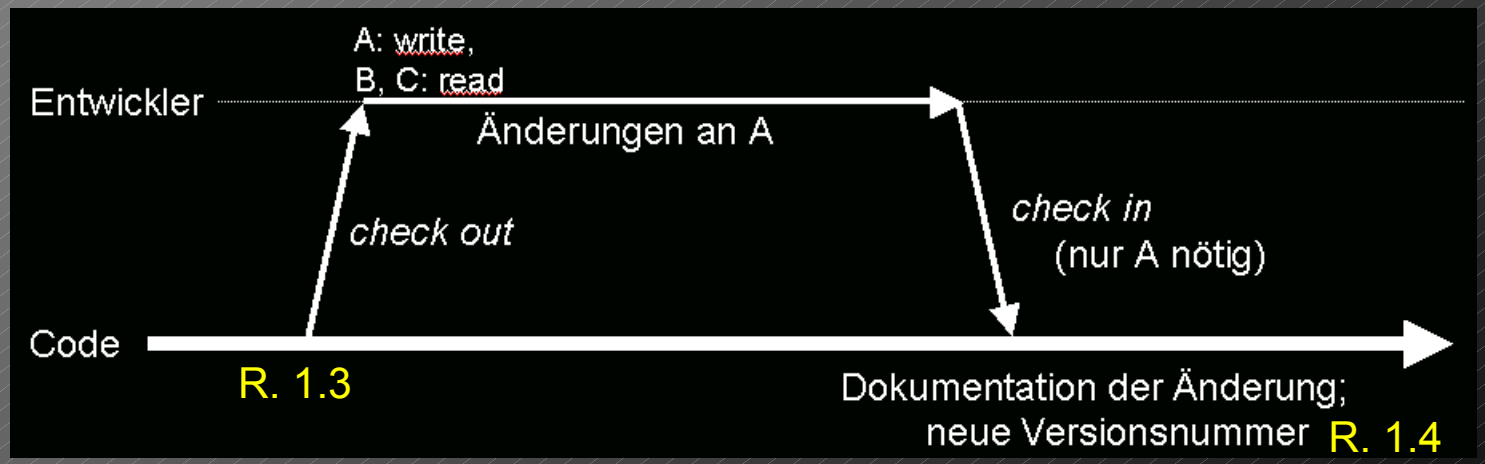




# Namensgebung, Nummerierung 1/2: *traditionelle Variante*



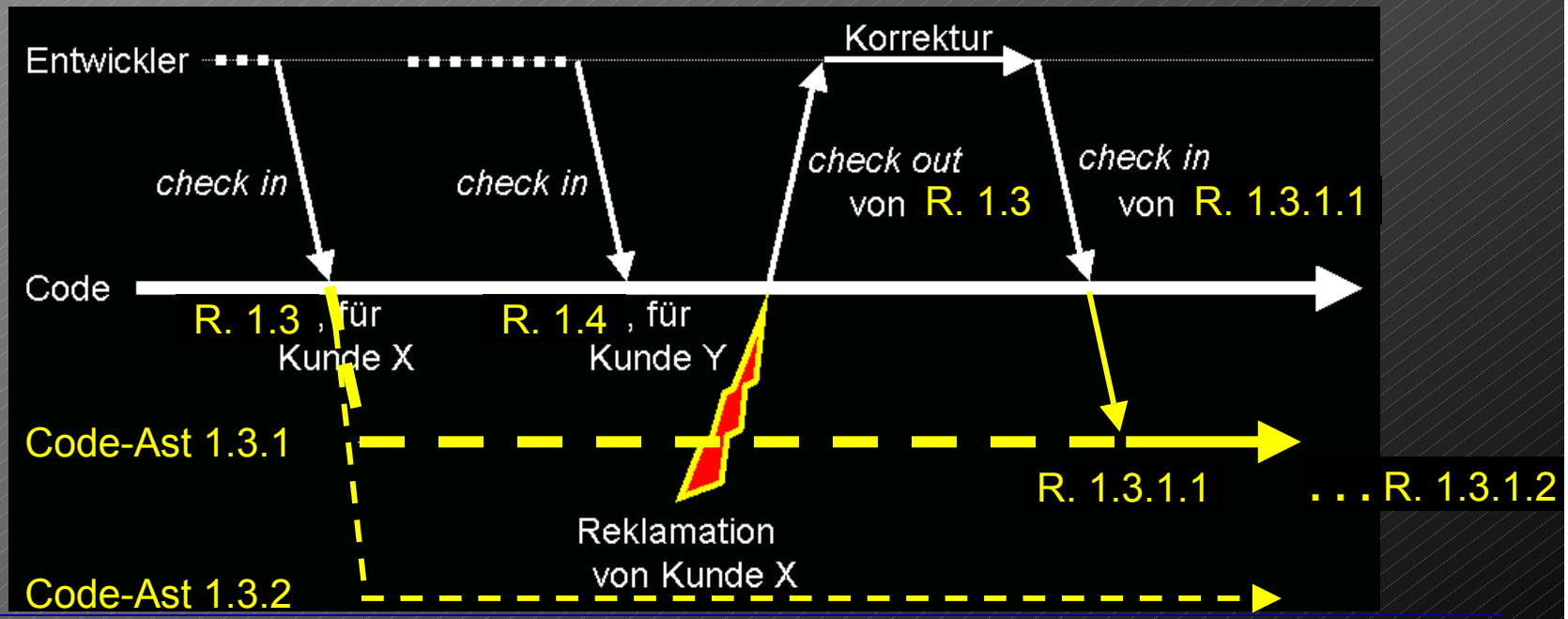
- Oft gebrauchter Begriff: "Version", korrekte Bezeichnung: *Revision*
- Nummerierungsschema für Revisionen: 1.1, 1.2, 1.3, ..., 2.1, 2.2, ...; Unterscheidung
  - *Release*-Nummer: die Stelle vor dem Punkt
  - *Level*-Nummer: die Stelle nach dem Punkt
- Eine neue Release-Nummer bezeichnet eine neue Revision mit wesentlichen Änderungen/Erweiterungen, die z.B. an Kunden ausgeliefert wird



# Namensgebung, Nummerierung 2/2: *traditionelle Variante*



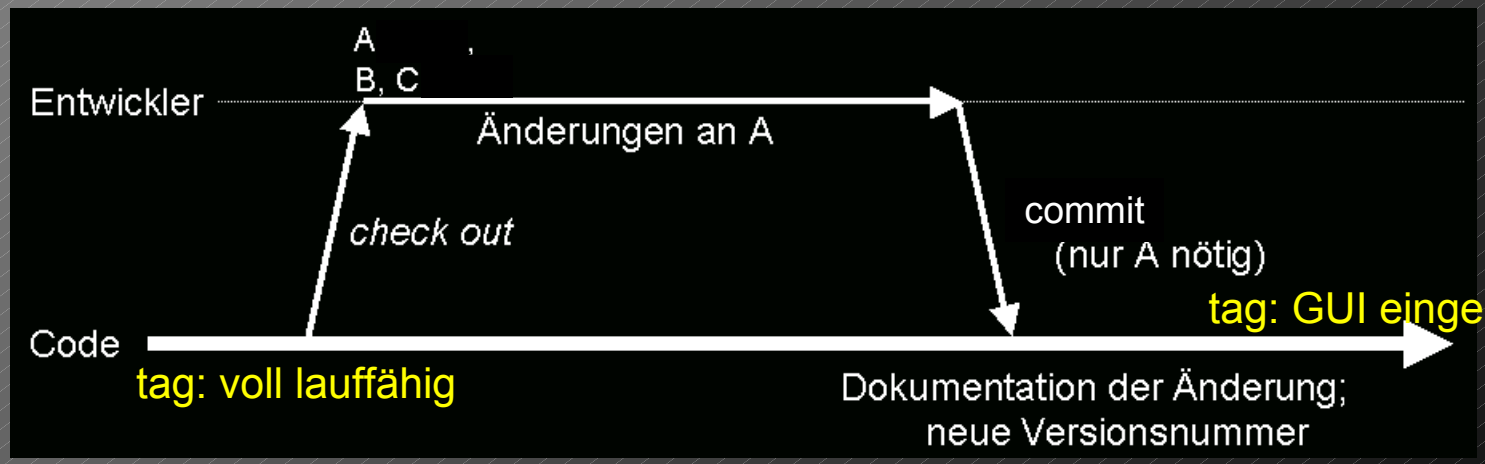
- Eine neue Release-Nummer bezeichnet eine neue Revision mit wesentlichen Änderungen/Erweiterungen, die z.B. an Kunden ausgeliefert wird
- Änderungen an früheren Revisionen/Releases können nicht nachträglich in den Code-Stamm übernommen werden
- Für sie wird (jeweils) ein neuer Ast parallel zum Stamm angelegt



# Namensgebung 1/2: *mit Subversion*



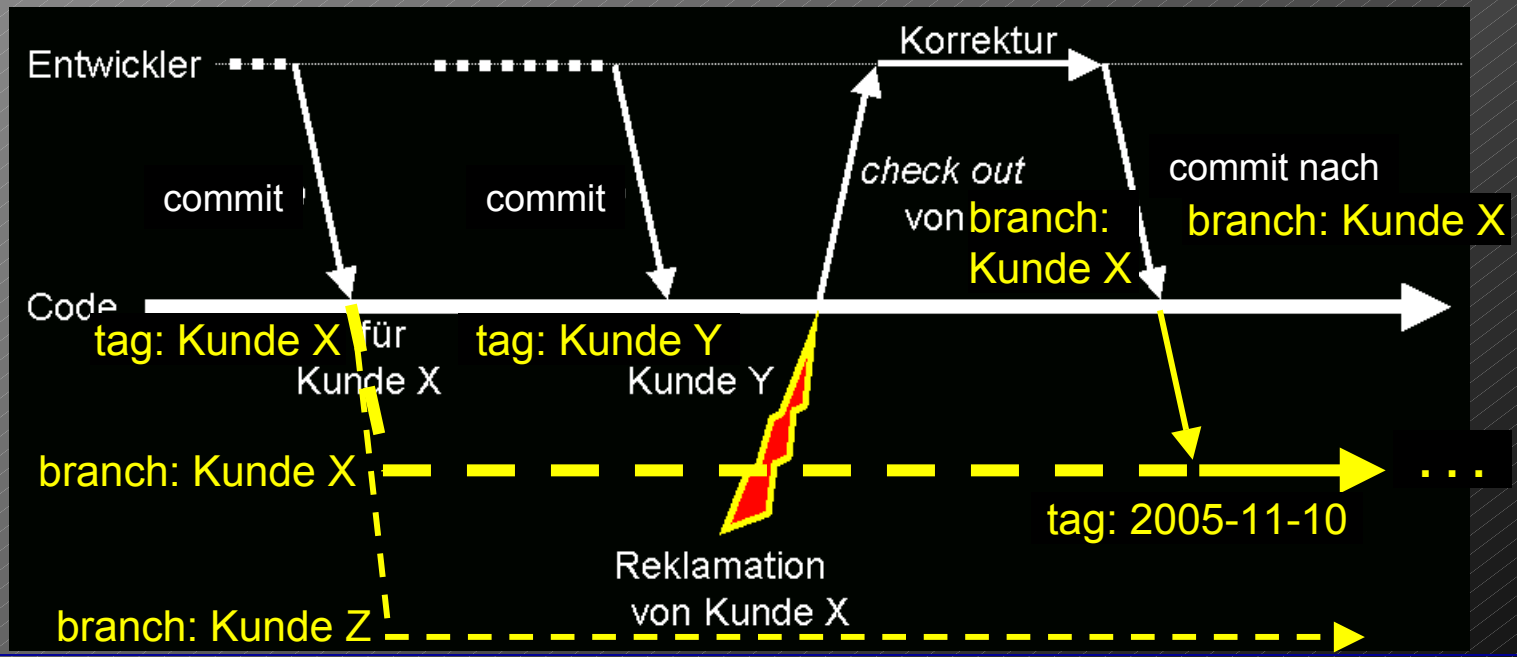
- Wir sprechen von *Revisionen* (nicht "Versionen")
- Anstelle von Nummern werden Namen (*tags*) verwendet, diese dürfen aber Nummern beinhalten...
- Subversion vergibt intern "globale" Release-Nummern: für alle Änderungen im Repository zu einem Zeitpunkt eine Nummer

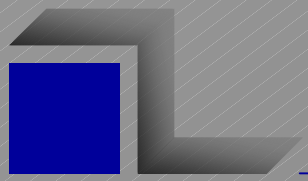


# Namensgebung 2/2: *mit Subversion*



- Ein *tag* (Kennzeichen) bezeichnet eine wichtige Revision, die z.B. an Kunden ausgeliefert wird
- Ein *branch* wird benutzt, um eine Revision weiterzuentwickeln
- Der *trunk* (Stamm) ist ein spezieller *branch*, für die hauptsächliche Entwicklung
- Ein *Merge* von *branches* und/oder *trunk* ist möglich



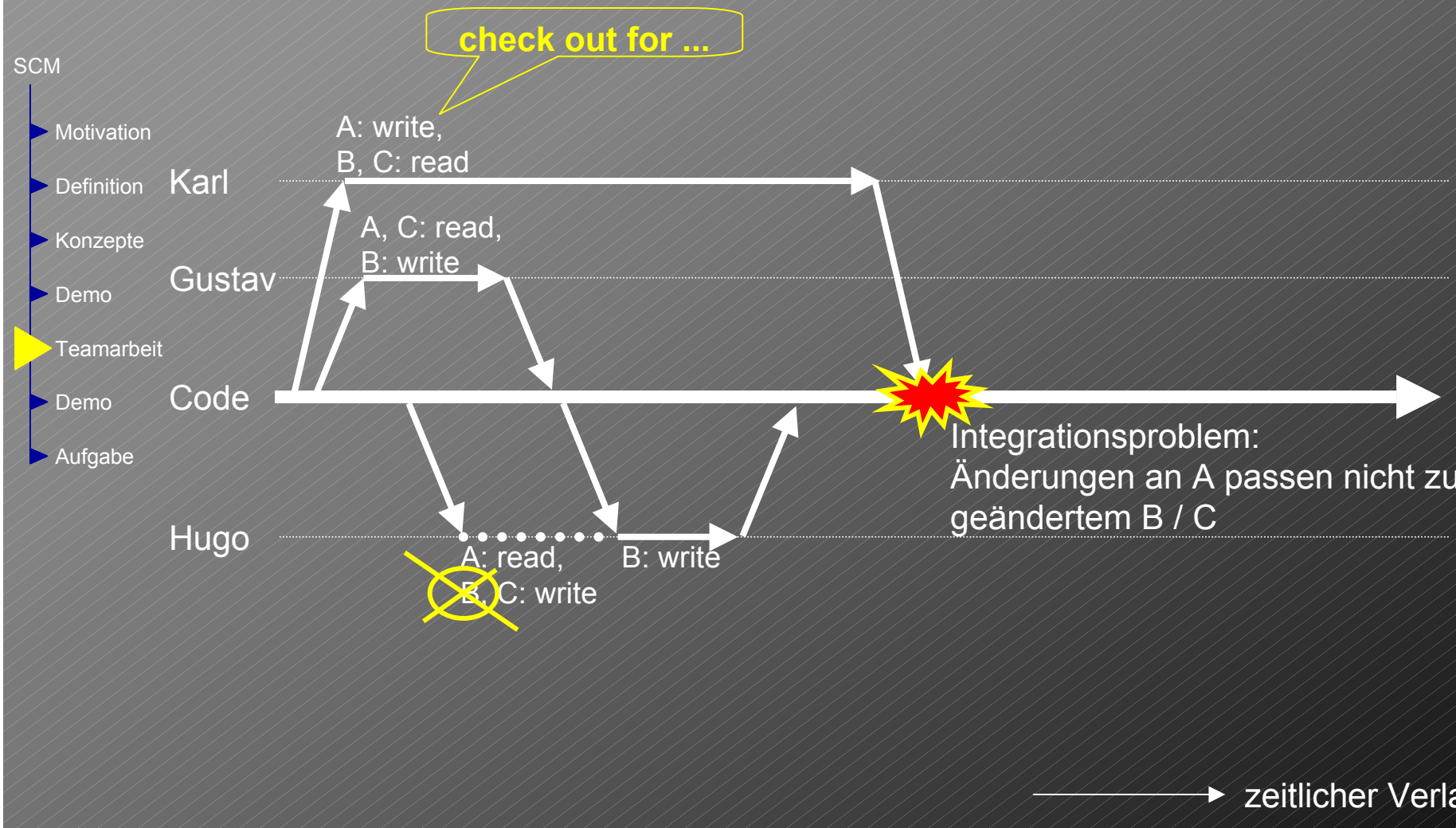




# Fragen?

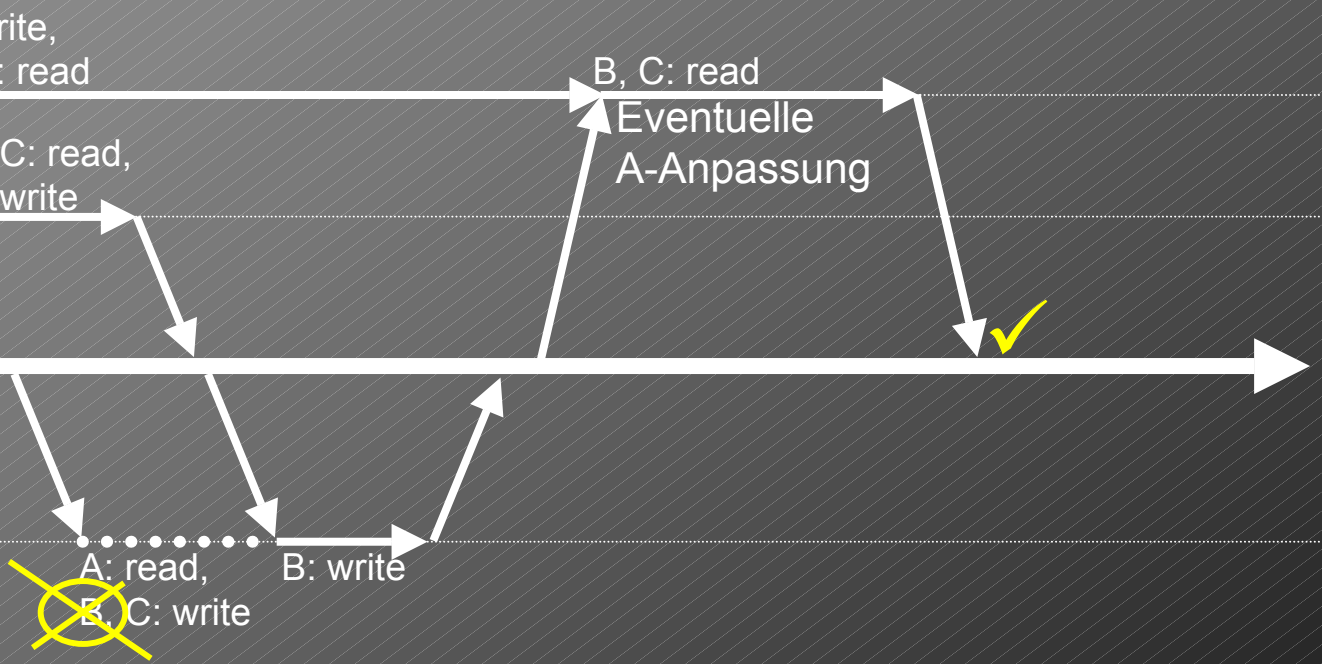
# Konzept: Teamarbeit, restriktives Modell





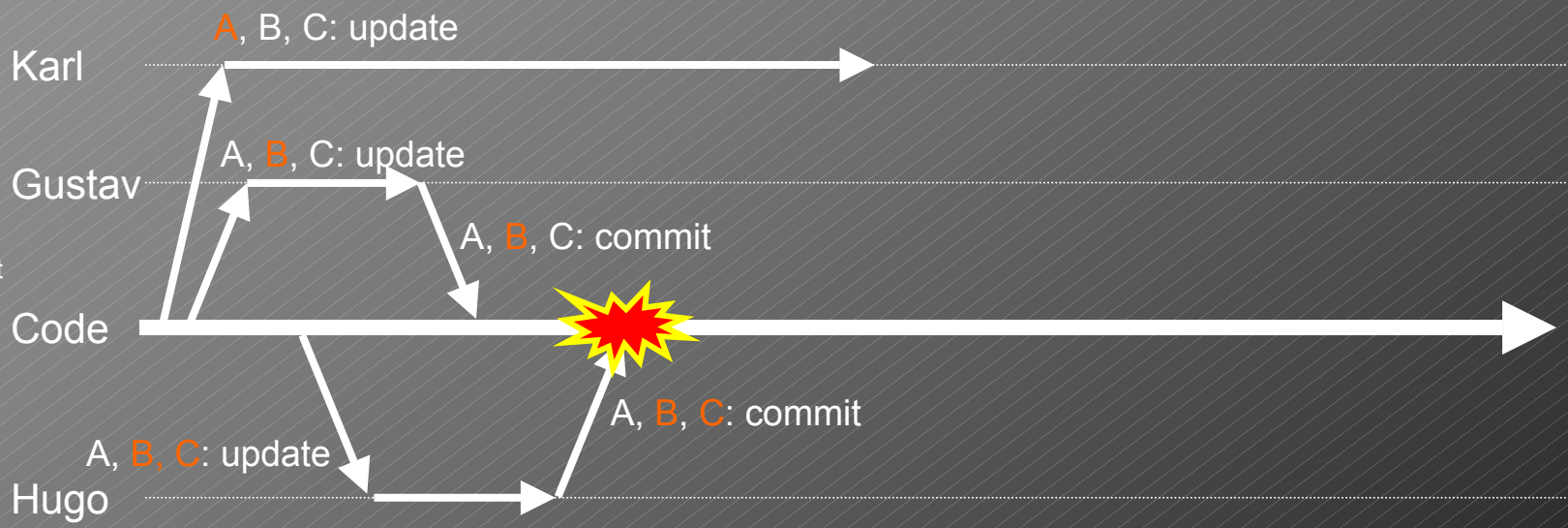
# Konzept: Teamarbeit, restriktives Modell





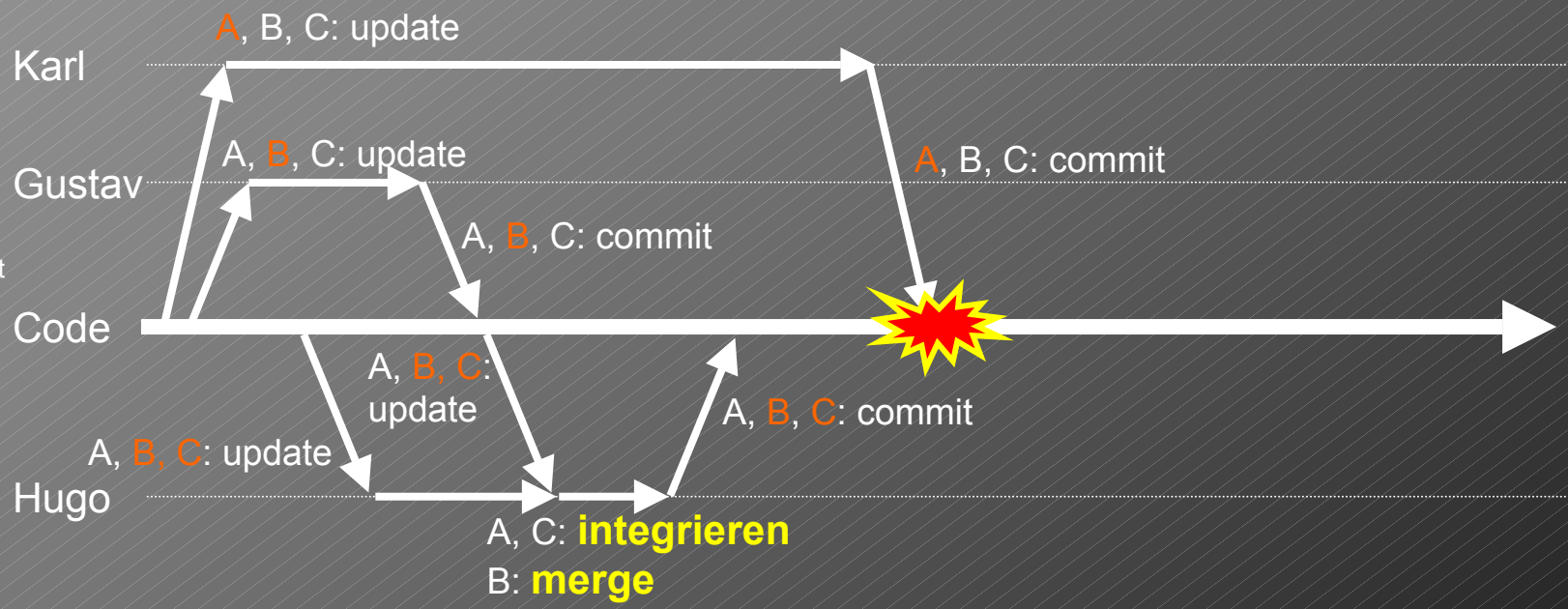
# Konzept: Teamarbeit, *optimistisches Modell*





# Konzept: Teamarbeit, *optimistisches Modell*



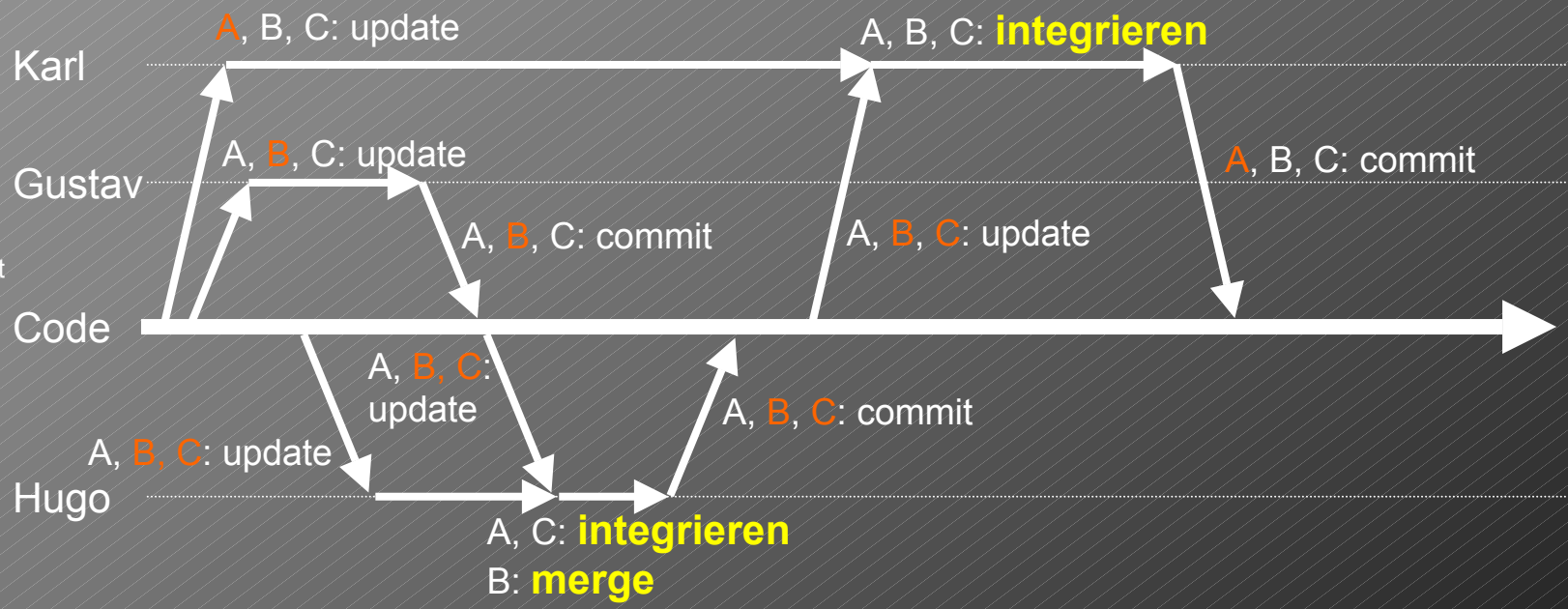



**Auch der merge kann schief gehen!**
**(muss aber nicht: "optimistisch")**

zeitlicher Verlauf

# Konzept: Teamarbeit, *optimistisches Modell*





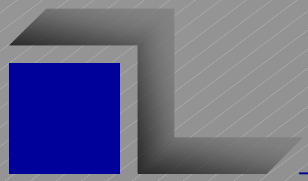

**Auch der merge kann schief gehen!**
**(muss aber nicht: "optimistisch")**

zeitlicher Verlauf

# Anleitung:
# idealer Workflow, optimistische Variante
## [Eclipse]



1. Machen Sie ein *Update* auf Ihren Workspace, bevor Sie mit irgendwelchen Arbeiten beginnen, um sich den aktuellen Zustand (des trunks/eines branches) zu holen
2. Arbeiten Sie (wie gewohnt) in Ihrem lokalen Workspace
3. Wenn Ihre Arbeit abgeschlossen ist
   a) Aktualisieren Sie Ihren Workspace mit *Update*, um sicherzustellen, dass zwischenzeitliche Änderungen anderer Projektmitarbeiter keinen Konflikt mit Ihren Arbeitsergebnissen verursachen
   Gibt es Konflikte, übernehmen Sie den fremden Code, wo möglich und (zusammen mit Ihren Änderungen) sinnvoll und lösen Sie die Konflikte
   Stellen Sie sicher, dass Ihr Code weiterhin korrekt funktioniert
   b) Stellen Sie Ihre Änderungen mit *Commit* ins Repository
   Wiederholen Sie vorher Schritt a), falls dieser sehr lange gedauert hat



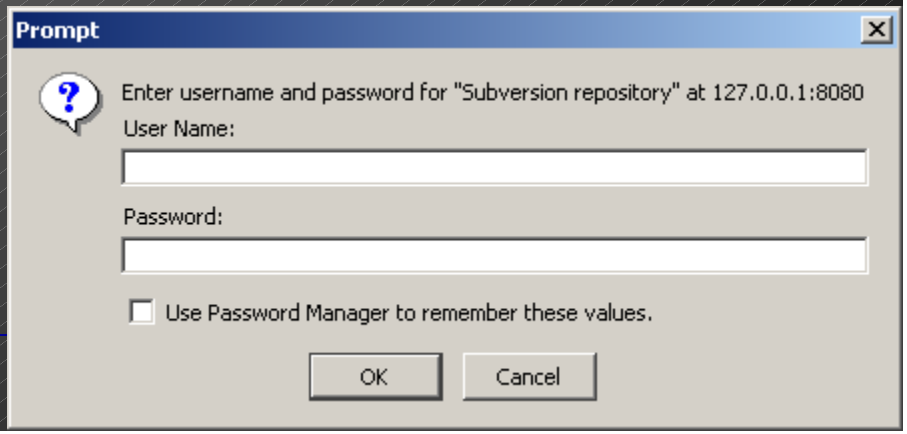


# Fragen?

# Anleitung: Vorbereitung 1/3



- Das jeweilige Gruppenverzeichnis ist festgelegt:
  /rest/svn/see*xy*
  Folgende Rechte sollten gesetzt sein: Besitzer apache, Gruppe see*xy*
- Repository anlegen:
  - Anmelden (ssh-Client) auf jonathan.sv.fh-mannheim.de
  - Der Befehl
    svnadmin create /rest/svn/see*xy*/<*Repository-Name*>
    erzeugt das Verzeichnis /rest/svn/see*xy*/<*Repository-Name*> mit Inhalt
  - Als Besitzer des Verzeichnisses muss apache gesetzt sein:
    **→ Es gibt bereits ein Verzeichnis "repo", das Sie bitte verwenden als <*Repository-Name*> einsetzen!**
- Test per Browser, URL:
  http://jonathan.sv.fh-mannheim.de/svn-see*xy*/<*Repository-Name*>
  sollte zu folgender Passwort-Abfrage führen:

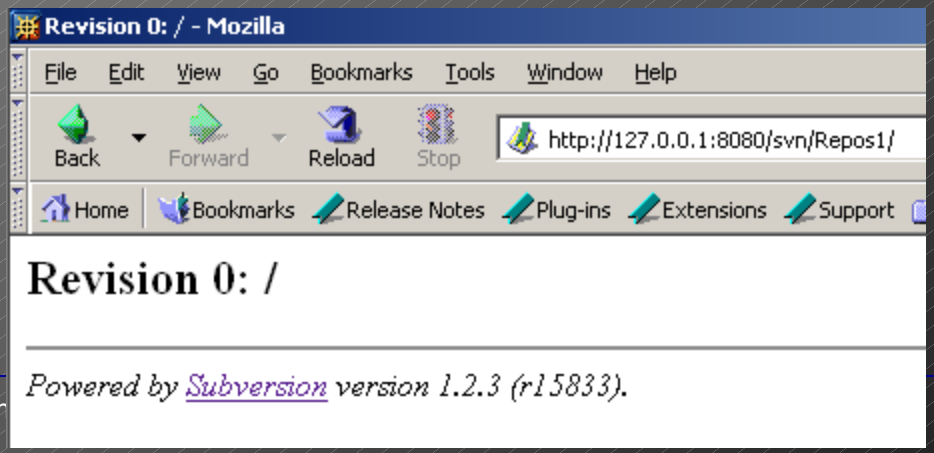


# Anleitung: Vorbereitung 2/3



- Zugriff per Passwort erlauben:
  - Passwort-Datei ist festgelegt:
    /rest/svn/see*xy*/.htpasswd
    Folgende Rechte müssen gesetzt sein: Besitzer apache, Gruppe see*xy*

  Immer noch auf jonathan:
  - Eintragen von Benutzern inkl. Passwort ("b" für Batch):
    htpasswd -bm /rest/svn/see*xy*/.htpasswd *<user1>* *<passwd>*
  - Eintragen von Benutzern mit expliziter Passwort-Abfrage:
    htpasswd -m /rest/svn/see*xy*/.htpasswd *<user2>*
  - Der Benutzername ist beliebig,
    das Passwort sollte *nicht* gleich dem Unix-Passwort sein...
- Test per Browser, gleiche URL, Passwort eingeben:



# Anleitung: Vorbereitung 3/3

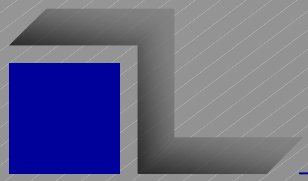


- Repository in Eclipse bekannt machen
  - Window → Show View → Other... → SVN → SVN Repository
  - Neues Repository bekannt machen:
    Kontext-Menü → New → Repository Location... → URL:
    http://jonathan.sv.fh-mannheim.de/svn-see*xy*/<*Repository-Name*>
- Verzeichnisse trunk, tags, branches anlegen:
  Nicht auf der Betriebssystem-Ebene, sondern in Eclipse im Repository-View:
  Kontext-Menü → New → New remote folder
- Das Passwort muss bei jedem Zugriff eingegeben werden
  → In Eclipse speichern lassen!
  → Deshalb *nicht* das Unix-Passwort wählen...

# Beispiel-Skript für eine Demo



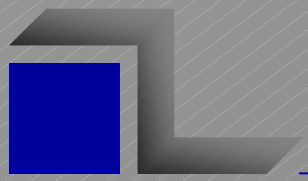
- Projekt mit Package und zwei Dateien anlegen, Projekt ins Repository
  - Klasse Calc: Kreisflächenberechnung mit 3.14
  - Klasse Output: Flächenausgabe
- Tag "Kunde1" für die aktuelle Revision setzen
- Eine Datei überarbeiten, update, commit
  - Beispielsweise Math.PI verwenden (statt 3.14)
- Beide Dateien weiterentwickeln, update, commit
  - Beispielsweise in einer neuen Datei Umfang berechnen und ausgeben
- Tag "Kunde1" auschecken, in neuen Ast "Kunde1" wiedereinchecken, "verbessern", update
  - PI verwenden
  - Neue Datei für den Umfang in den Ast kopieren
- Neuste Revision auschecken, weiterbearbeiten, update, commit



# Anmerkung



Verwenden Sie *keine* Dateien mit Namen, die sich nur durch *Groß-/Kleinschreibung* unterscheiden!



# Aufgabe



- Vollziehen Sie jeden Schritt der Demo und der Folien zusammen mit ihren Gruppenpartnern nach
- Erklären Sie sich abwechselnd, was das nächste Kommando bewirken wird, welche Version danach im aktuellen Verzeichnis / im Repository vorliegt, ob diese Version editierbar ist, welche Ausgabe Eclipse beim Start (sofern möglich) liefern wird etc.
- Führen Sie *dann erst* das nächste Kommando aus!
- Überprüfen Sie die von Ihnen erwartete Wirkung nach jedem Schritt
- Bearbeiten Sie Dateien gleichzeitig und prüfen Sie das Systemverhalten beim update / commit

Bleiben noch Fragen? Fragen Sie besser jetzt als im Projekt...!