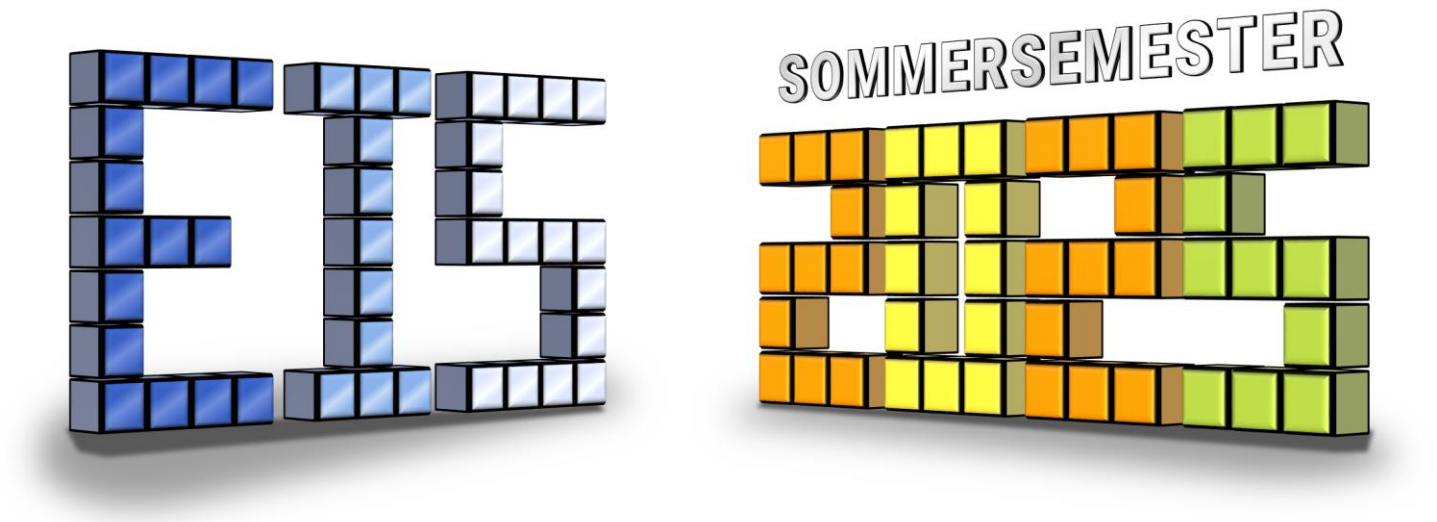
EINFÜHRUNG IN DIE SOFTWAREENTWICKLUNG

Sommersemester 2025



Foliensatz #4

Versionsverwaltung

Michael Wand
Institut für Informatik
Michael.Wand@uni-mainz.de

Übersicht

Inhalt heute

- Organisatorisches
- Programmiersprachen
  - Python + Tools
  - C/C++
  - Java/Scala
- Der Softwareentwicklungsprozess
  - Probleme & grobe Ansätze
- Versionsverwaltung

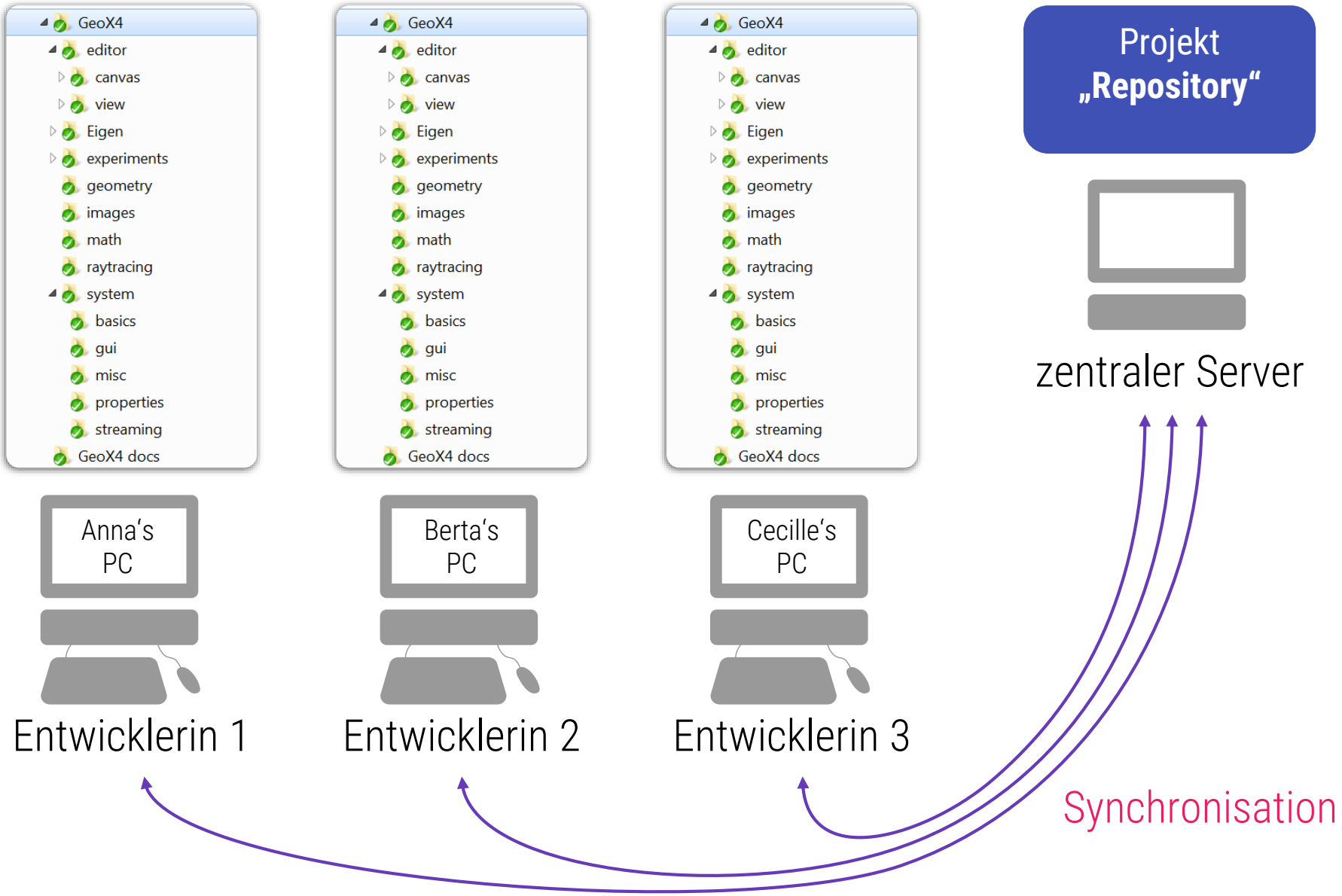
Versionsverwaltung

Versionsverwaltung

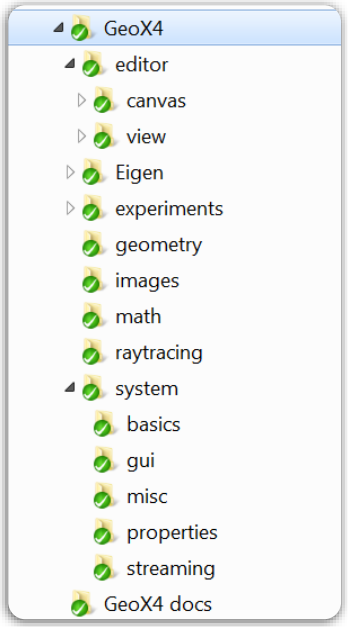
Ziele

- Alle Änderungen nachverfolgen
  - „Gestern tat es noch… :-(“
  - Alte Versionen können restauriert / verglichen werden
- Kollaboration
  - Mehrere Entwickler/innen
  - Synchronisation von verschiedenen Änderungen
  - „Konflikte“ (widersprüchliche Änderungen) werden erkannt
- Kommunikation
  - Zentraler Aufbewahrungsort („Cloud“)
  - Alternativ: dezentrale Synchronisation

Prinzip (Beispiel CVS, SVN)



Prinzip (Beispiel CVS, SVN)



Projekt
„Repository“

update

Update holt Änderungen
Konflikte werden markiert!
Merging: Konflikte beheben

commit

Neue Version hochladen
Neuster Stand nötig (vorher update)
Änderungen fest (Transkation)
Änderungshistorie (immutable history)

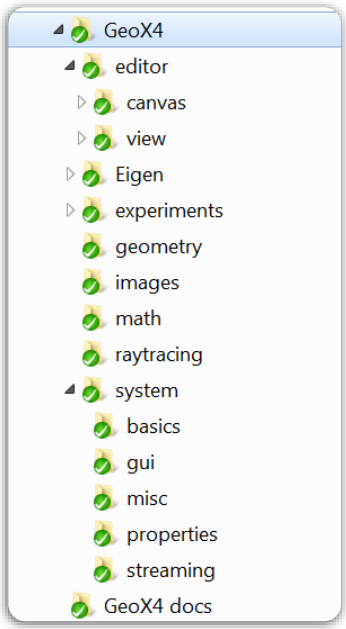
zentraler Server

Anna‘s PC

Entwicklerin

Synchronisation

Subversion (SVN)



init, import

Repo. anlegen, initial füllen

Projekt **„Repository“**

update

commit

checkout

Code holen (bestimmte Version)

zentraler Server

add: Dateien für Versionierung markieren

revert: Änderungen verwerfen

rename, move, delete: Dateien ändern

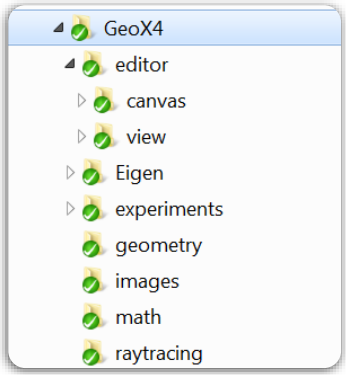
Anna‘s PC

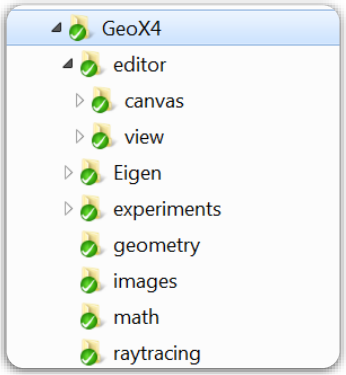
Entwicklerin

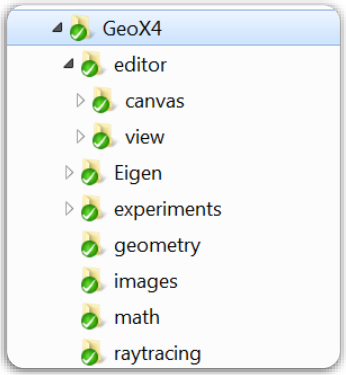
Synchronisation

Verteilte (dezentrale) Versionsverwaltung

Prinzip (Beispiel HG, GIT)



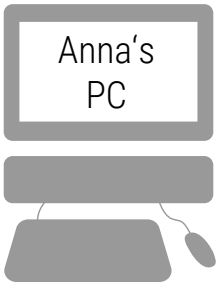


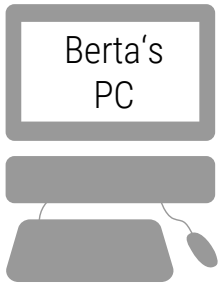


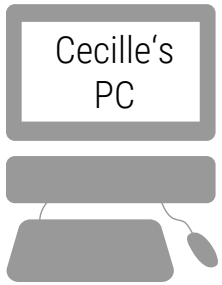
Anna's **Repository**

Berta's **Repository**

Cecille's **Repository**



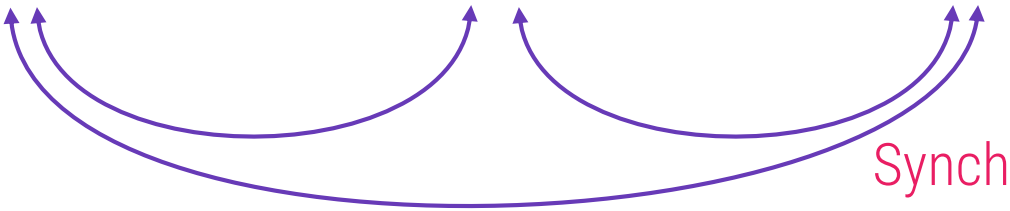




Entwicklerin 1

Entwicklerin 2

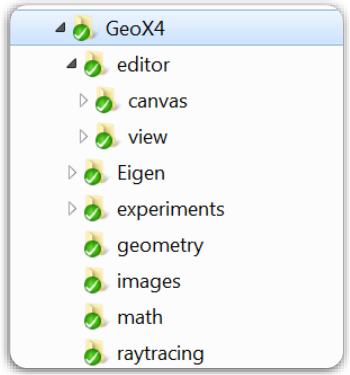
Entwicklerin 3



Synchronisation

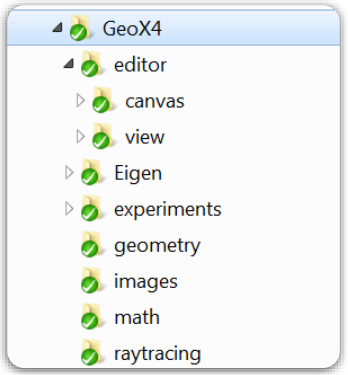
kein
zentraler Server
nötig
(optional)

Mercurial (HG), GIT ähnlich



push
(Anna)

pull
(Anna)



update, commit, add,move, delete
genau wie SVN

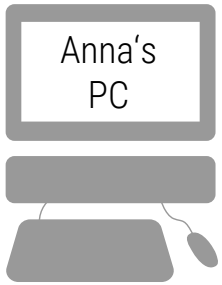
wie SVN

Anna's **Repository**

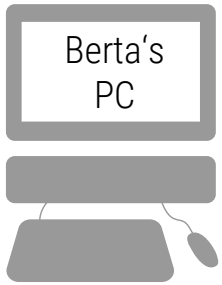
Berta's **Repository**

Branching:
Verschiedene Repoentwicklungen werden unterschieden

merge
nach pull Branches zusammen-führen



Entwicklerin 1



Entwicklerin 2

Zusätzlich:
Synchronisation der Repo's.

Was ist git?

In den Übungen

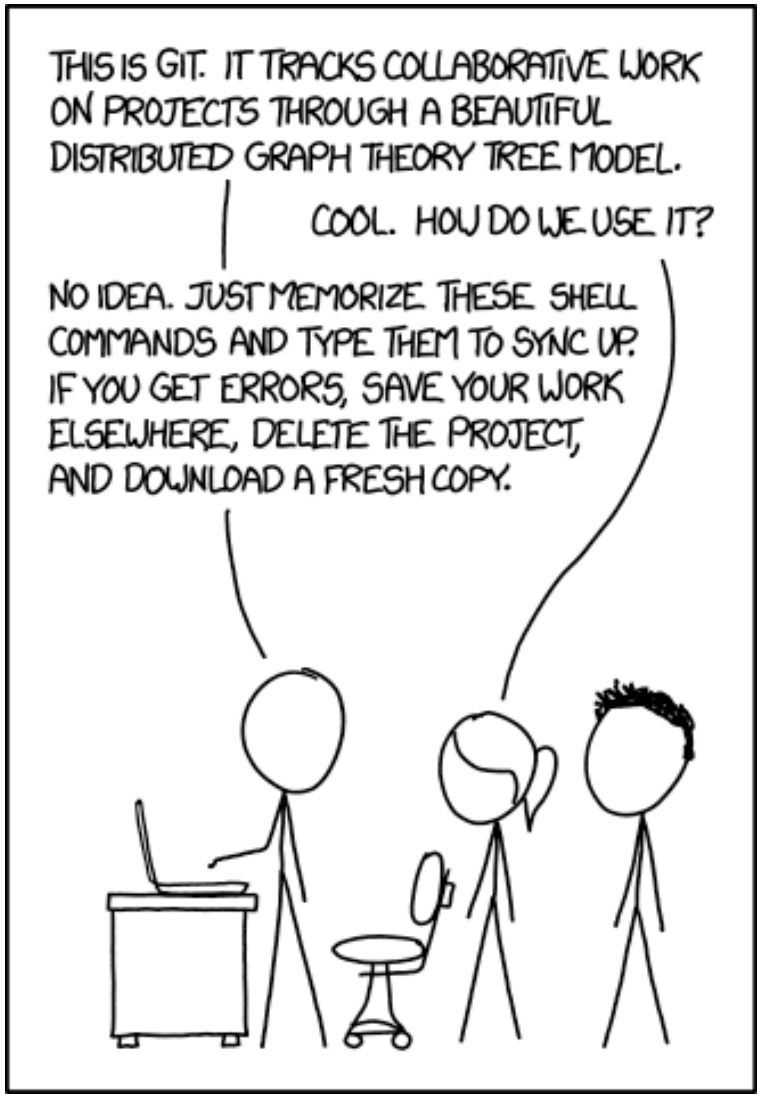
- Benutzung von „git“
- Gleiches Prinzip wie „hg“
  - Verteilte Versionsverwaltung
  - Mehr Features
  - Etwas schwerer zu bedienen
  - „Industriestandard“
  - Hauptvorteil: „History rewriting“ möglich
    - Außerdem: Vorübergehendes „Branching“ bevorzugt
    - Kann im Kleinen destruktiv sein
    - Bei großen Projekten wichtig, um Ballast zu entfernen

git commands

`git init`	Neues Repo erzeugen (im akt.Verzeichnis)
`git status`	Status ausgeben – geänderte Dateien etc.
`git clone` </path/to/my/repo>	Repo clonen (aus Dateisystem)
`git clone` <username@host:/path/to/repo>	Repo clonen (remote)
`git add` <filename>	(Geänderte) Datei für commit einreihen
`git commit -m` "<commit message>"	commit (mit „Nachricht“, nur „ge-add-ete“)
`git commit -a`	commit (auch alle geänderten)
`git push origin master`*)	Hauptbranch an Herkunftsrepo schicken
`git push --all origin`	Alle Branches an Herkunftsrepo schicken
`git pull`	Änderungen holen und mit akt. Stand mergen

*) Hauptzweig wird inzwischen auch oft „main“ genannt

Mehr zu git in den Übungen...



https://xkcd.com/1597

Entwicklungsprozesse mit (D)VC-Systemen

Disclaimer:

Meine praktische Erfahrung beschränkt sich auf

- kleine Teams
- zentrale Systeme/Ansätze
- Wenig Branching
- Akademischen Umfeld
- „unkritische“ Systeme

Zentrale Vorgehensmodelle

Zentrales Repository

(für die one-woman show)

Wie nutze ich Versionsverwaltung?

Zentrale Systeme

- Annahme:
  - zentrales VCS (z.B. SVN) oder zentral genutztes DVCS (z.B. GIT mit zentral genutztem GitLap-Repo)
- Einzelentwickler
  - Alles ins SVN (GIT, HG, CVS, Perforce™, …)
  - Ich weiß noch nächsten Sommer was ich gestern getan habe
  - Auch für Seminararbeiten, B.Sc./M.Sc.-Thesis, etc.
  - Vorteile
    - Versionshistorie
    - Persistenz
    - Einfaches Backup (hoffentlich! – nicht vergessen)

Zentrales Repository

(für ein kleines Team)

Wie nutze ich Versionsverwaltung?

Zentrale Systeme

- Betrachte nun Entwicklung im kleinen Team

Ein zentrales Repo für‘s Team

- Moderne Begriffe (früher „alternativlos“)
  - „Trunk-based Development“
  - „Mono-Repository“
- Einfaches Modell
  - Relativ leicht zu verwalten und zu beherrschen
  - Komplexe Vorgehensmodelle nicht abgebildet
  - Komplexere Varianten von TBD mit MR aber auch in modernen/großen Projekten üblich

Wie nutze ich Versionsverwaltung?

Ein zentrales Repo für‘s Team

- Mögliches Entwicklungsmodell
  - Morgens updaten
  - Abends einchecken
  - Intensive persönliche Absprache (kleine Teams!)
- Tags/Branches für Meilensteine
  - „Release 2.0“ o.ä. gesondert markieren
  - Typischerweise wenig Branching während der Entwicklung
- Soziales Protokoll für commits (einchecken)
  - Rule #1: Do not break the system!!!
  - Schlechten / ungetesteten Code „abschalten“ (→ Architektur!)
  - Besondere Vorsicht bei wichtigem / sensitivem Code

Wie nutze ich Versionsverwaltung?

Zentrale Systeme

- Do not break things: Sensitiver Code
  - Annahme: Modulares System
    - „abschaltbare“ Module / Versionsauswahl
  - Mehr und weniger wichtige Module
    - Z.B. wenige zentrale Komponenten, viele optionale Erweiterungen
  - Soziale Protokolle für kritische Änderungen
    - z.B. mehr-Augen-Prinzip für essentielle/zentrale Teile
    - z.B. besonders ausführliche Tests
    - z.B. Aufgaben nur für erfahrene Teammitglieder
    - z.B. „Code-Ownership“ – Verantwortung für Komponenten
      - Besonders hohe Verantwortung für kritische Komp.

Wie nutze ich Versionsverwaltung?

Zentrale Systeme

- Vorteile des einfachen zentralen Ansatzes
  - Alle auf dem gleichen Stand
  - Wenig Divergenz
- Nachteile
  - Größere Änderungen umständlich
  - Man kann sich nicht auf „seinen Teil“ konzentrieren
  - Schwierigkeiten, mehrere Versionen zu pflegen
    - z.B. Release und Development
- „Trunk-based Development“ auch komplexer möglich
  - Durchaus auch heute noch populär
  - Ich diskutiere hier nur die Basic-1990er-Variante…

Verteilte Repositories

Wie nutze ich Versionsverwaltung?

Verteilte Versionsverwaltung

- Branching
  - Entwickler/in (oder Sub-Team) nutzt eigenen „Branch“
  - Änderungen sind isoliert
  - „Merging“ wenn alles fertig ist
- Vorteil
  - Komplexere Änderungen am Stück ohne Ablenkungen
- Nachteil
  - Zusammenführen kann schief gehen
  - „Code Divergenz“
- Absprache & strukturiertes Vorgehen unabdingbar!

Verteilte Repositories
(Extrembeispiel)

Wie nutze ich Versionsverwaltung?

Verteilte Versionsverwaltung

- Extremes Branching: „Pull requests“
  - Größere Änderung wird vorbereitet
  - An „Code Owner“ (dem das Repo „gehört“) geschickt
  - Vom „Code Onwer“ gemerged nach Prüfung
- Im Open-Source Bereich oft zu finden
  - Sehr lose gekoppelte Entwicklung möglich
  - Prüfung von u.U. anonymen Beiträgen
- Management (!) & Design (!)
  - Modularisierung & Strukturierung
  - Architektur auf Erweiterbarkeit ausgelegt

Verteilte Repositories
(typische professionelle Projekte)

Wie nutze ich Versionsverwaltung?

Verteilte Versionsverwaltung

- Mittelweg: Pull-Requests in „normalen“ Teams
  - Struktur wie im Open-Source-Bereich
    - Branches, Pull-Requests
  - „Code-Reviews“ vor Merging
    - Vom „Code-Owner“, von Peers
    - Alternative: Pair-Programming
  - Test-Driven-Development
    - „Unit-Tests“ für jede Komponente („Unit“)
    - Automatisierter Check
    - Merge nur wenn frei von Testfehlern „passing the tests“
    - Automatisierbar (run tests on push/commit)
- Ein Repo pro Developer aber zentrales Repo sammelt alles

Wie nutze ich Versionsverwaltung?

Verteilte Versionsverwaltung

- Management
  - Strukturiertes Vorgehen
  - Absprache
- Kein Cargo Cult
  - Struktur muss zum Menschen passen
  - Anreize (Tests, Code-Reviews)
  - Kreative Freiräume
  - Robuste und klare Gesamtstruktur
- Schweres Problem – Rezepte sind nur ein Teil!
  - Erfahrung & Empathie unersetzlich