

# Migration von Subversion nach Mercurial und Einsatz dezentraler Versionskontrolle in Unternehmen

Christoph Mewes

Otto-von-Guericke-Universität Magdeburg

17. August 2011



FAKULTÄT FÜR  
INFORMATIK

# Motivation

Subversion wurde als Versionskontrollsystem im Unternehmen eingesetzt.

## Probleme

- zentrales System
- wenig geeignet für Open Source Entwicklung
- lokaler Cache als deutlicher Overhead
- verhindert Einsatz von gewohnten Dateiverwaltungswerkzeugen

## Lösungssuche

- dezentrales System (*Open Source*)
- gute Unterstützung für Windows / GUI
- gute Performance

⇒ Mercurial (hg) als Lösung

# Motivation

Subversion wurde als Versionskontrollsystem im Unternehmen eingesetzt.

## Probleme

- zentrales System
- wenig geeignet für Open Source Entwicklung
- lokaler Cache als deutlicher Overhead
- verhindert Einsatz von gewohnten Dateiverwaltungswerkzeugen

## Lösungssuche

- dezentrales System (*Open Source*)
- gute Unterstützung für Windows / GUI
- gute Performance

⇒ Mercurial (hg) als Lösung

# Motivation

Subversion wurde als Versionskontrollsystem im Unternehmen eingesetzt.

## Probleme

- zentrales System
- wenig geeignet für Open Source Entwicklung
- lokaler Cache als deutlicher Overhead
- verhindert Einsatz von gewohnten Dateiverwaltungswerkzeugen

## Lösungssuche

- dezentrales System (*Open Source*)
- gute Unterstützung für Windows / GUI
- gute Performance

⇒ Mercurial (hg) als Lösung

# Motivation

Subversion wurde als Versionskontrollsystem im Unternehmen eingesetzt.

## Probleme

- zentrales System
- wenig geeignet für Open Source Entwicklung
- lokaler Cache als deutlicher Overhead
- verhindert Einsatz von gewohnten Dateiverwaltungswerkzeugen

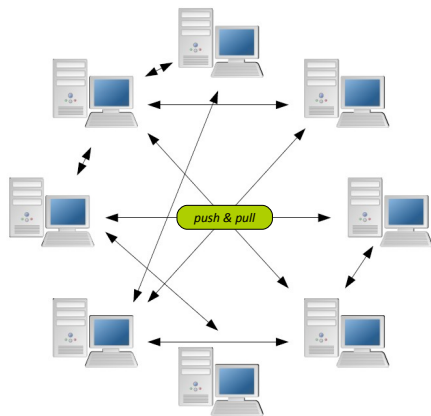
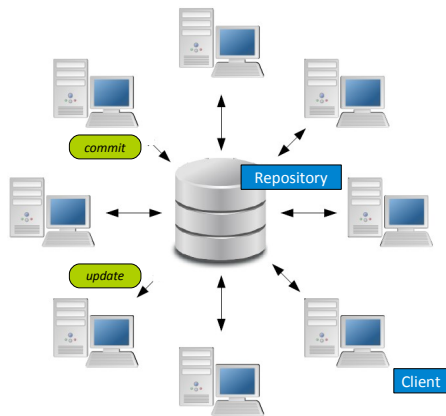
## Lösungssuche

- dezentrales System (*Open Source*)
- gute Unterstützung für Windows / GUI
- gute Performance

⇒ **Mercurial** (hg) als Lösung

# Hintergrund

## zentraler & dezentraler Workflow





## Nachteile zentraler Systeme

- Verbindung zum Server nötig
- Commits sind sofort öffentlich  
(wenn man keine Branches verwendet  $\Rightarrow$  Namenskonflikte)
- Schreibzugriff zur effizienten VCS-Nutzung nötig
- bei Mergeproblemen kein Zurücksetzen möglich

## Vorteile dezentraler Systeme

- Arbeiten auch offline möglich; lädt zum Experimentieren ein
- private Commits (bis man sie pusht)
- auch ohne Schreibzugriff VCS-Nutzung möglich
- Zurücksetzen jederzeit möglich
- „Commit before Merge“



## Nachteile zentraler Systeme

- Verbindung zum Server nötig
- Commits sind sofort öffentlich  
(wenn man keine Branches verwendet  $\Rightarrow$  Namenskonflikte)
- Schreibzugriff zur effizienten VCS-Nutzung nötig
- bei Mergeproblemen kein Zurücksetzen möglich

## Vorteile dezentraler Systeme

- Arbeiten auch offline möglich; lädt zum Experimentieren ein
- private Commits (bis man sie pusht)
- auch ohne Schreibzugriff VCS-Nutzung möglich
- Zurücksetzen jederzeit möglich
- „Commit before Merge“



## Beispiel: FeatureIDE

- Java-Projekt; Entwicklung eines Eclipse-Plugins
- Vergleich des Speicherplatzbedarf

	Subversion	Mercurial
Dateien in der Arbeitskopie	10.881	3.199
Verzeichnisse in der Arbeitskopie	16.803	835
Größe der Arbeitskopie	202 MiB	100 MiB
Größe inkl. Repository	N/A	240 MiB

- nur 38 MiB Overhead von Mercurial
- wesentlich weniger Verzeichnisse in Mercurial

## Beispiel: FeatureIDE

- Java-Projekt; Entwicklung eines Eclipse-Plugins
- Vergleich des Speicherplatzbedarf

	Subversion	Mercurial
Dateien in der Arbeitskopie	10.881	3.199
Verzeichnisse in der Arbeitskopie	16.803	835
Größe der Arbeitskopie	202 MiB	100 MiB
Größe inkl. Repository	N/A	240 MiB

- nur 38 MiB Overhead von Mercurial
- wesentlich weniger Verzeichnisse in Mercurial

# Migration

## Projektstruktur

- Websites auf Basis von SallyCMS (PHP / MySQL)
- *projektunabhängig*: Basissystem plus weitere Komponenten
- *projektspezifisch*: Umsetzung des Designs und Kundenwünschen
- parallele Entwicklung von Projekten

## Ziele

- Migration bestehender Projekte
- automatisiertes Erzeugen neuer Projekte (ein Repo. pro Projekt)
- stetige Aktualisierung von Projekten
- häufiges und schnelles Zurückmergen von Korrekturen

## Projektstruktur

- Websites auf Basis von SallyCMS (PHP / MySQL)
- *projektunabhängig*: Basissystem plus weitere Komponenten
- *projektspezifisch*: Umsetzung des Designs und Kundenwünschen
- parallele Entwicklung von Projekten

## Ziele

- Migration bestehender Projekte
- automatisiertes Erzeugen neuer Projekte (ein Repo. pro Projekt)
- stetige Aktualisierung von Projekten
- häufiges und schnelles Zurückmergen von Korrekturen

# Migration /2

## Konvertierung bestehender Projekte

### Vorgehensweise

- Unteilbarkeit von hg-Repositories
- daher: Zerlegung des SVN-Repositories in viele kleine hg-Repositories
- 1-mal hg convert pro Projekt

### Ergebnisse

- einmaliger Aufwand
- erzeugte Repositories nicht für Langzeitentwicklung geeignet
- ⇒ nach Bedarf für einzelne Projekte durchführen

# Migration /2

## Konvertierung bestehender Projekte

### Vorgehensweise

- Unteilbarkeit von hg-Repositories
- daher: Zerlegung des SVN-Repositories in viele kleine hg-Repositories
- 1-mal hg convert pro Projekt

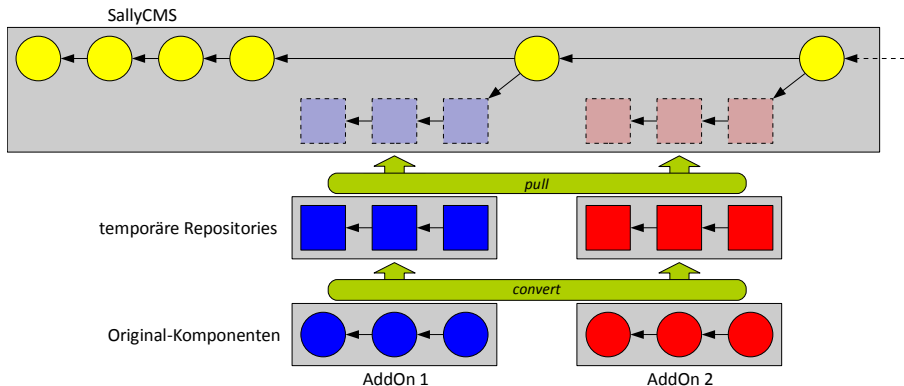
### Ergebnisse

- einmaliger Aufwand
- erzeugte Repositories nicht für Langzeitentwicklung geeignet
- ⇒ nach Bedarf für einzelne Projekte durchführen

# Migration /3

## Erzeugung neuer Projekte

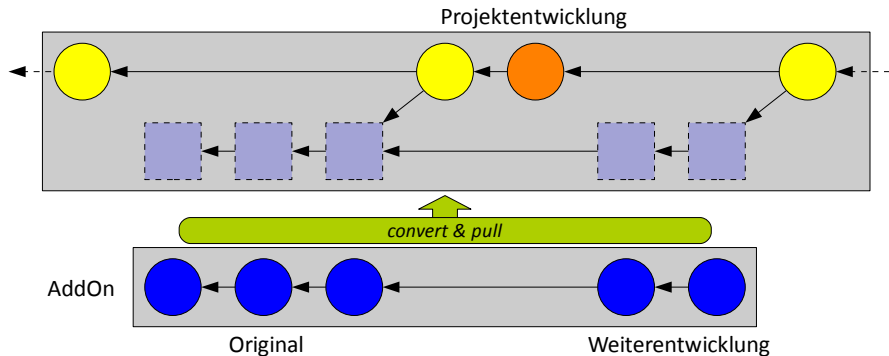
- Projekte als Klone des Basisrepositories
- Mergen der Komponenten in die Basis
- Verzeichnis-Convention vermeidet Konflikte
  - ▶ ⇒ automatisierbar
  - ▶ Vorbereitung der Komponenten nötig



# Migration /4

## Aktualisierung von Projekten

- Ziel: Korrekturen und neue Features in Projekte einbringen
- manueller Aufwand kritisch für Aktualisierungshäufigkeit  
⇒ möglichst automatisieren
- technisch: hg pull & hg merge der Komponenten  
⇒ Stärken von Mercurial ausnutzen



# Migration /5

## Zurückmergen

### Ziele

- Korrekturen aus einem Projekt in die Basis zurückspielen
- um mehrfaches Beheben eines Problems vermeiden

### Probleme

- Finden von Commits, die zurückgemergt werden müssen
- Export dieser Commits und Übertragung in die Basis

### Ergebnisse

- Toolunterstützung möglich
- automatisches Generieren von Patches
- dennoch: größtenteils **manueller** Vorgang

# Migration /5

## Zurückmergen

### Ziele

- Korrekturen aus einem Projekt in die Basis zurückspielen
- um mehrfaches Beheben eines Problems vermeiden

### Probleme

- Finden von Commits, die zurückgemergt werden müssen
- Export dieser Commits und Übertragung in die Basis

### Ergebnisse

- Toolunterstützung möglich
- automatisches Generieren von Patches
- dennoch: größtenteils **manueller** Vorgang

# Migration /5

## Zurückmergen

### Ziele

- Korrekturen aus einem Projekt in die Basis zurückspielen
- um mehrfaches Beheben eines Problems vermeiden

### Probleme

- Finden von Commits, die zurückgemergt werden müssen
- Export dieser Commits und Übertragung in die Basis

### Ergebnisse

- Toolunterstützung möglich
- automatisches Generieren von Patches
- dennoch: größtenteils **manueller** Vorgang

# Evaluierung

Commit-Verhalten in 24 zufällig ausgewählten Projekten

Kriterium	Subversion	Mercurial
Commits pro Tag	4,35	8,18
leere Commit-Nachricht	789 (92,8 %)	5 (0,3 %)
pro Commit geänderte Dateien	5,71	4,15

- Projekterzeugung & -aktualisierung vollständig automatisiert
- häufigere, besser dokumentierte Commits
- Bugs nur noch selten mehrfach korrigiert
- nahtlose Integration von Drittentwicklern (Open Source)
- überwiegend positive Erfahrungen mit Mercurial

# Keynotes

## Lektionen aus der Migration

Ein dezentrales System ist einem zentralen in vielen Punkten überlegen.

### **Aber:**

- gründliche Auseinandersetzung mit dezentralen Systemen nötig
- kein „business as usual“ nach einer Migration
- zwingende Überarbeitung der Repository-Struktur
- Schulung der Mitarbeiter nötig

### **Möglichkeiten & Perspektiven**

- Verbesserung des Commit-Verhaltens
- viele Automatisierungen möglich
- Integration der Open Source Community in eigene Entwicklungen
- Unterstützung für (global) verteilte Entwicklungsstandorte