Technische Universität Braunschweig

Institut für Flugführung

Versionsverwaltung

Einführung in git und GitHub

Prof. Dr.-Ing. Peter Hecker, Dipl.-Ing. Paul Frost, 25. April 2017

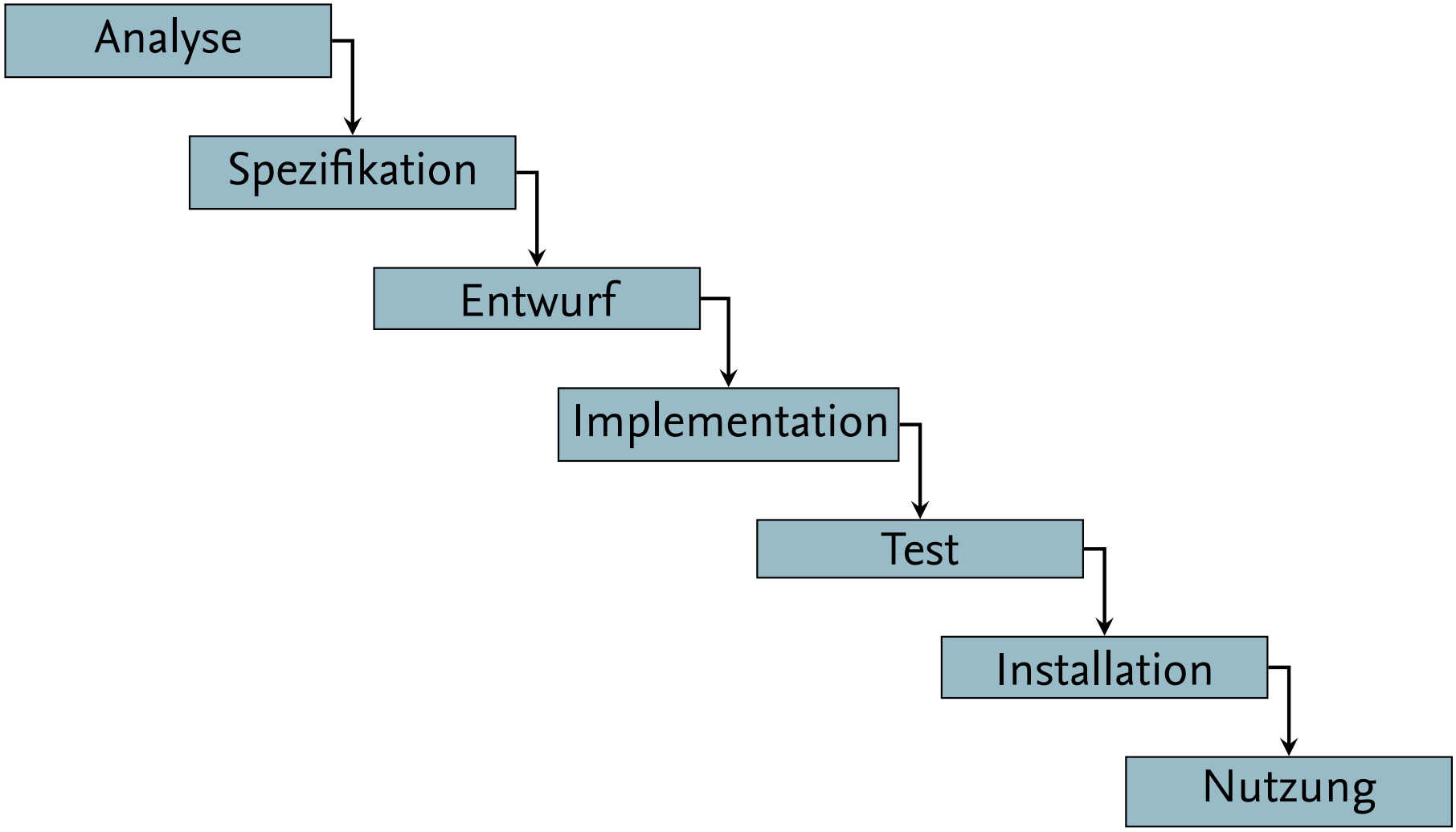
Agenda

04. April	Kick-Off
11. April	Projektmanagement
18. April	Prozessmodelle
25. April	**Versionsverwaltung**
02. Mai	Einführung Arduino/Funduino
09. Mai	Entwicklungsumgebungen und Debugging
16. Mai	Dokumentation und Testing
23. Mai	Dateieingabe und -ausgabe
30. Mai	GUI-Erstellung mit Qt
06. Juni	Exkursionswoche
13. Juni	Bibliotheken
20. Juni	Netzwerke
27. Juni	Projektarbeit
04. Juli	Projektarbeit
11. Juli	Vorbereitung der Abgabe

Teil I

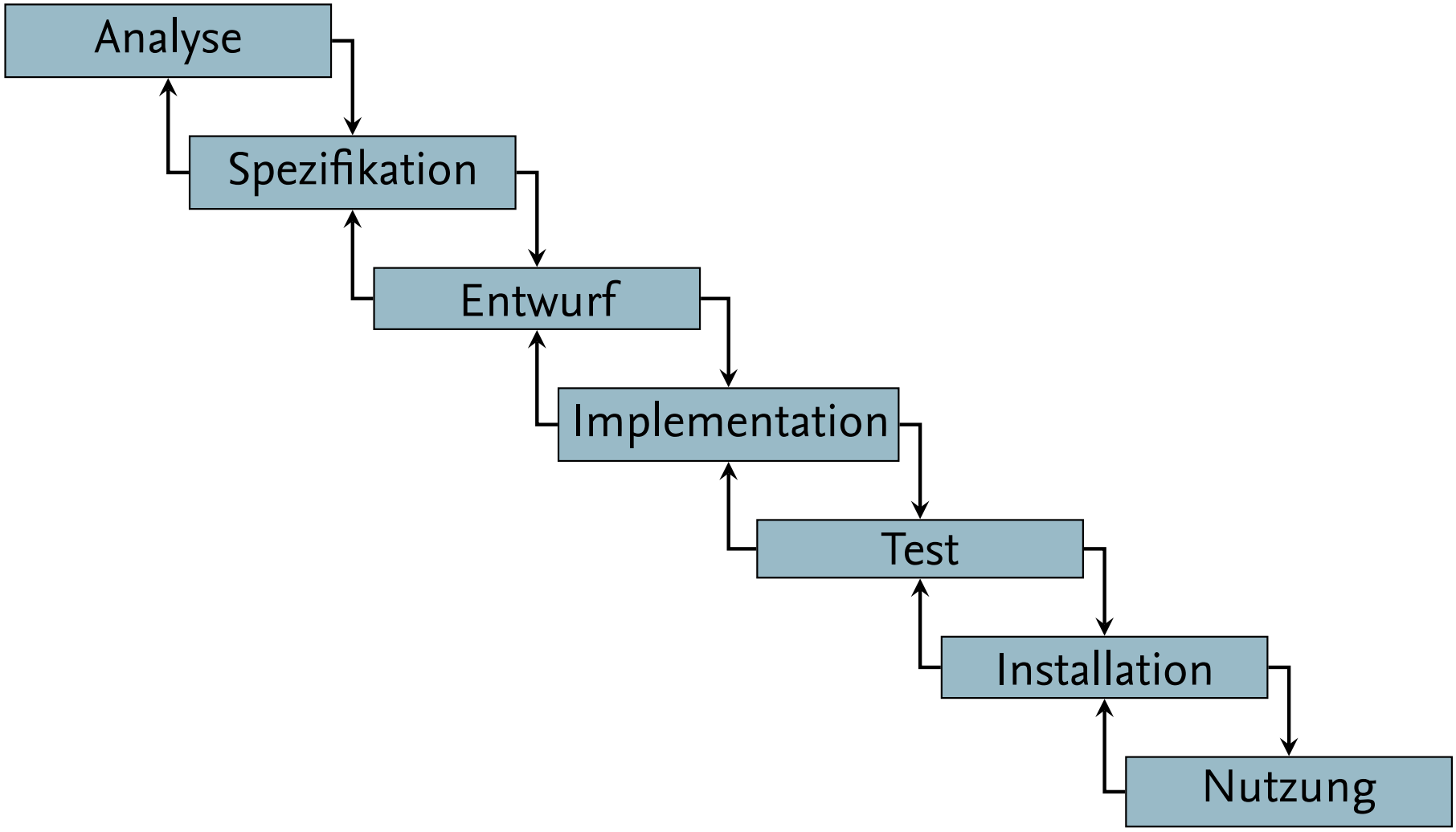
Wiederholung

Das Wasserfallmodell[1]



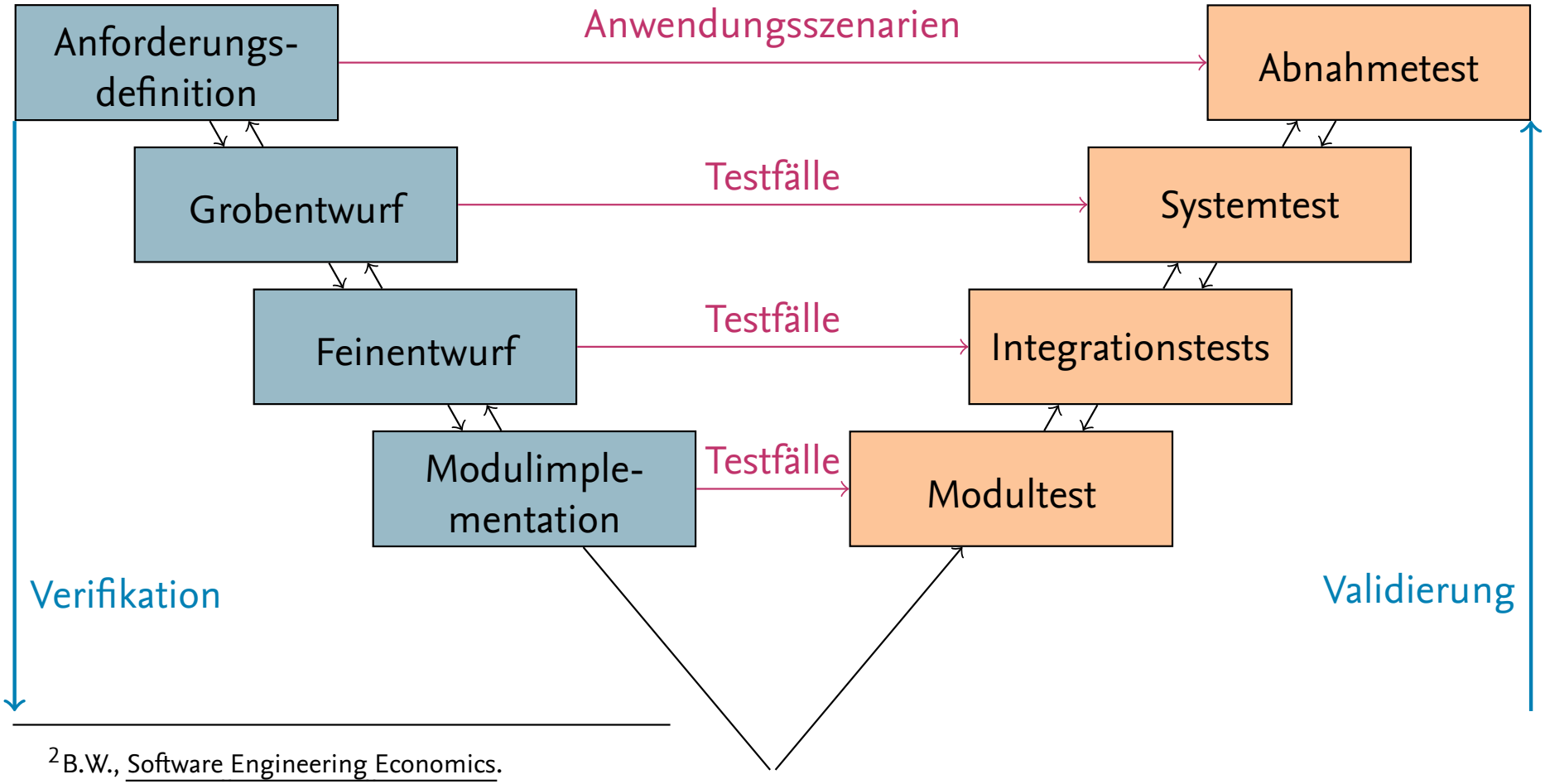
[1] tware Engineering Economics. 1981.

Das Wasserfallmodell[1]



ngineering Economics.

Vorgehensmodell (V-Modell)[2,3,4]



[2] B.W., Software Engineering Economics.

[3] Boehm B.W. Verifying and Validating Software Requirements and Design Specifications, in IEEE Software. 1984.

[4] Helmut Balzert. Lehrbuch der Software-Technik. 1998.

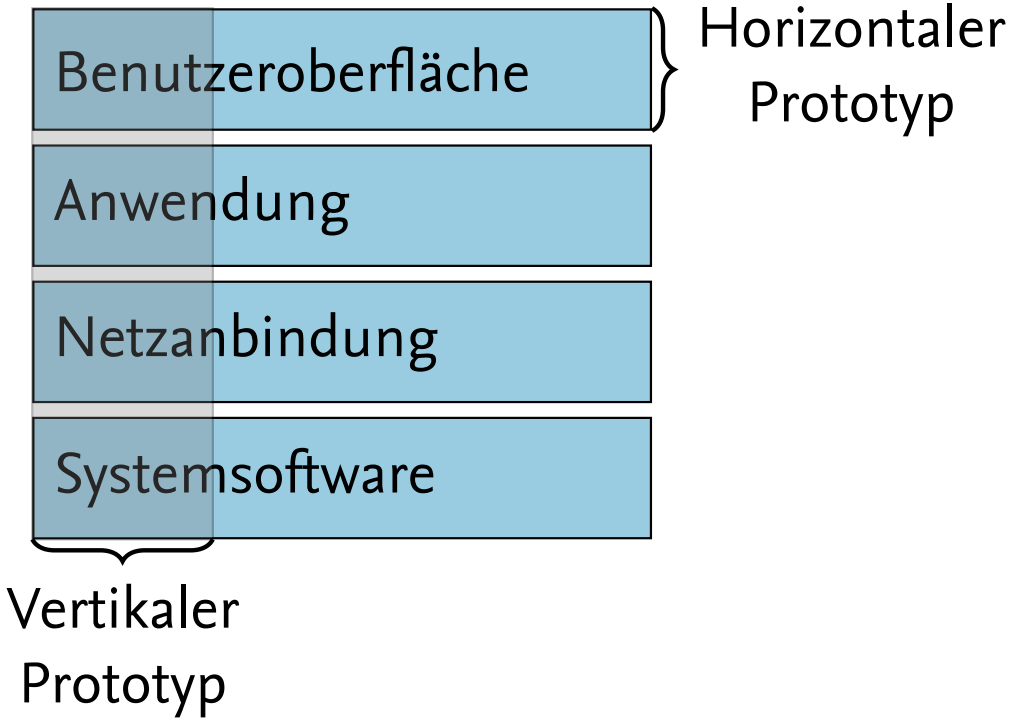
Prototypen-Modell[5]

Horizontaler Prototyp

- Realisiert spezifische Ebenen
- Ebene wird vollständig realisiert

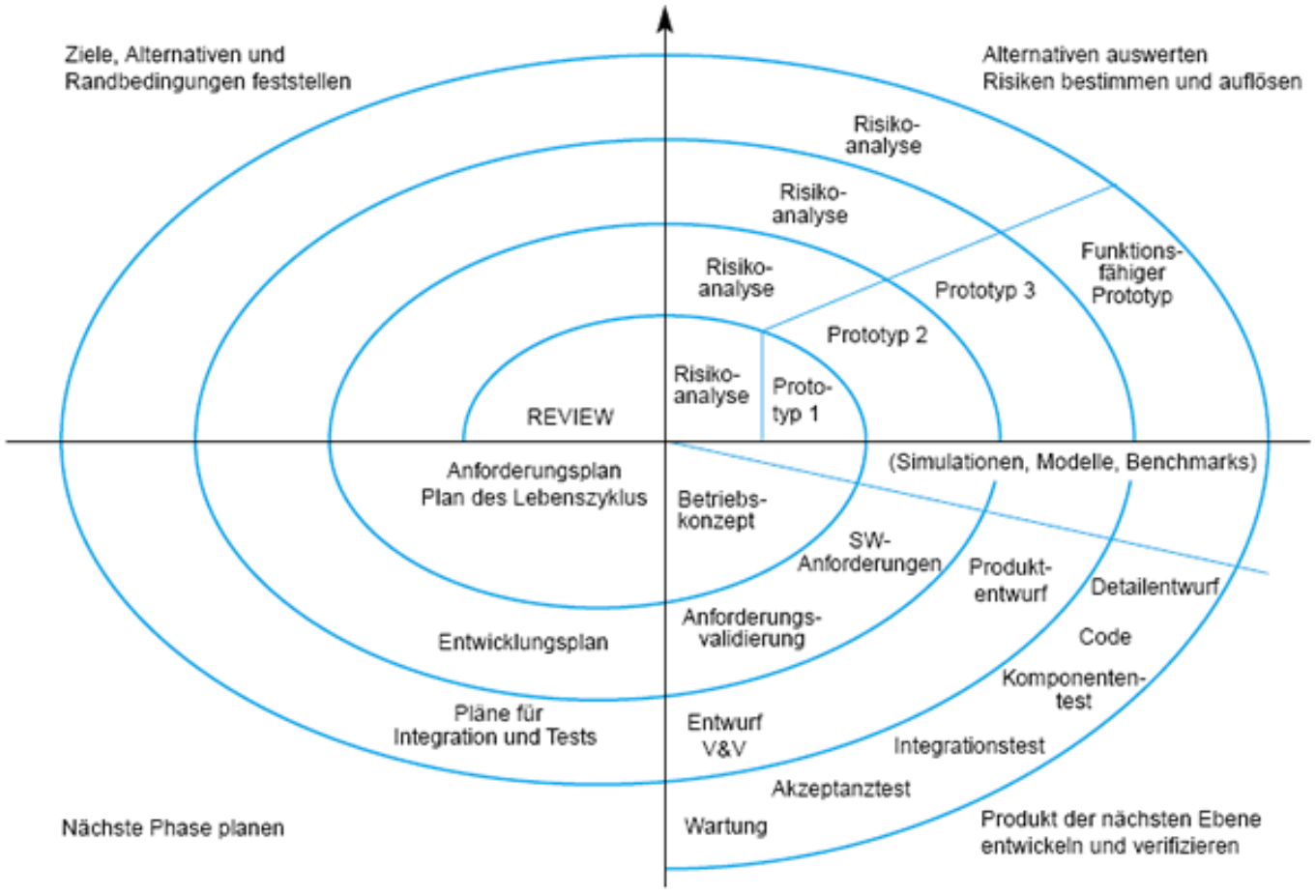
Vertikaler Prototyp

- Umfasst alle Ebenen des Systems
- Ebenen werden partiell realisiert



[5] Balzert, Lehrbuch der Software-Technik.

Spiralmodell



Scrum

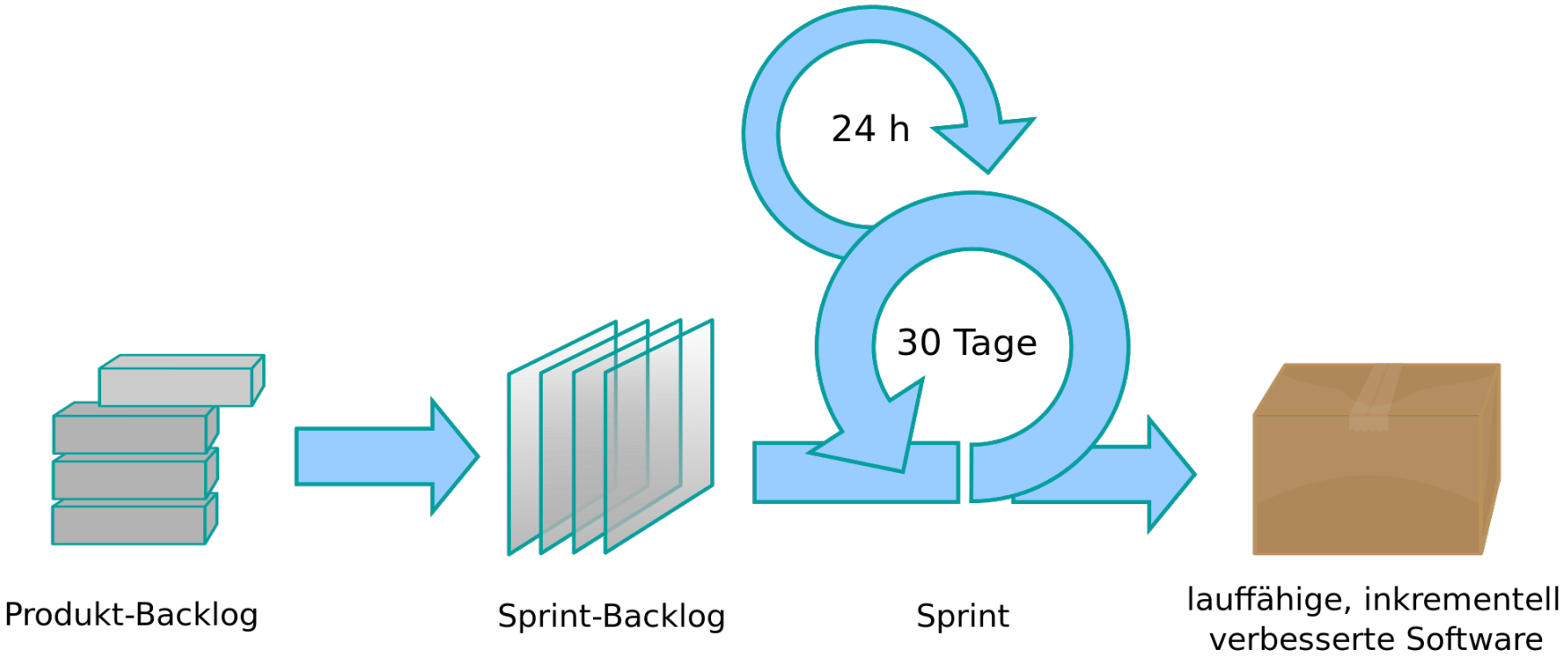


Abbildung 1: Scrum-Prozessverlauf[6]

[6] Von Scrum_process.svg: Lakeworksderivative work: Sebastian Wallroth (talk) - Scrum_process.svg, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=10772971

Scrum

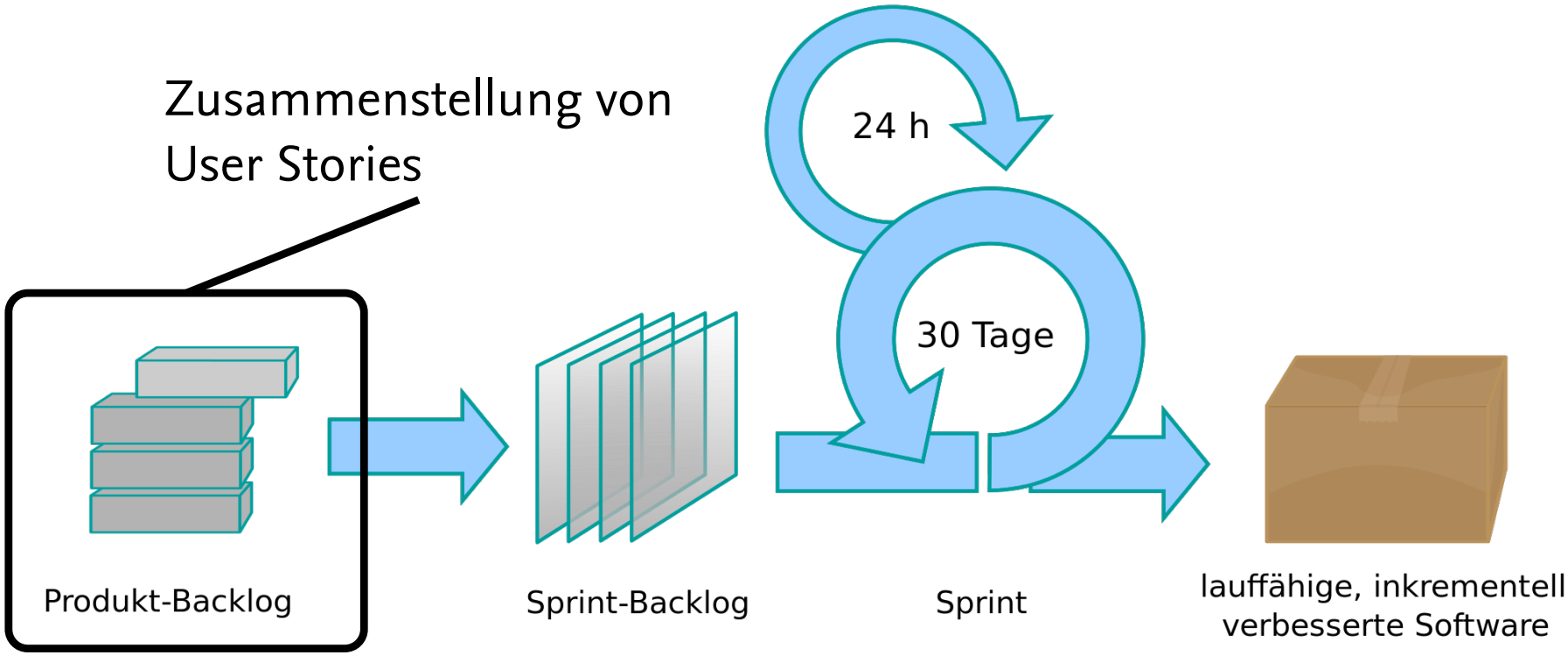


Abbildung 1: Scrum-Prozessverlauf[6]

[6] Von Scrum_process.svg: Lakeworksderivative work: Sebastian Wallroth (talk) - Scrum_process.svg, CC BY-SA 3.0, [illegible]ikimedia.org/w/index.php?curid=10772971

Scrum

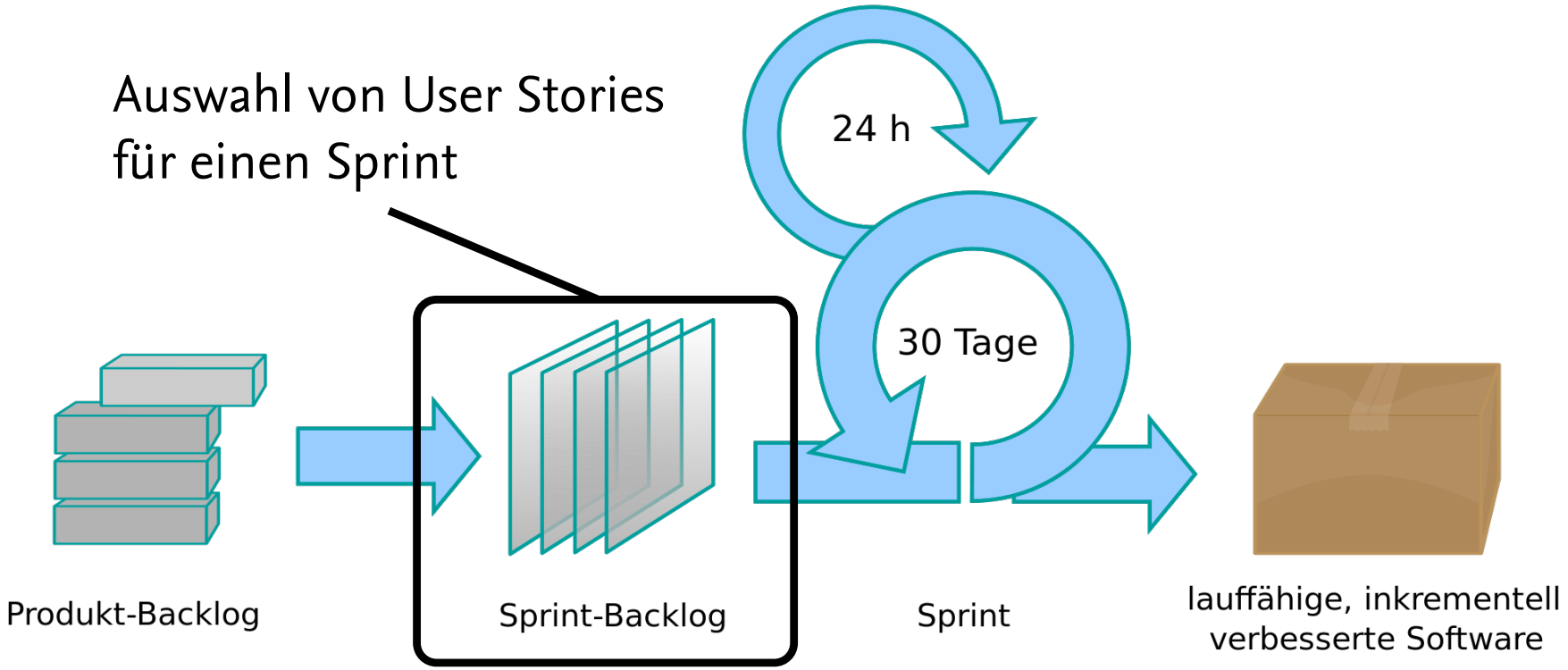


Abbildung 1: Scrum-Prozessverlauf[6]

[6] Von Scrum_process.svg: Lakeworksderivative work: Sebastian Wallroth (talk) - Scrum_process.svg, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=10772971

Überblick Prozessmodelle[7]

Prozessmodell	Primäres Ziel	Antreibendes Moment
Wasserfall-Modell	minimaler Managementaufwand	Dokumente
V-Modell	maximale Qualität	Dokumente
Prototypen-Modell	Risikominimierung	Code
Evolutionäres Modell	minimale Entwicklungszeit	Code
Inkrementelles Modell	minimale Entwicklungszeit	Code
Objektorientiertes Modell	Zeit- und Kostenminimierung	Wiederverwendung
Spiralmodell	Risikominimierung	Risiko

[7] Balzert, Lehrbuch der Software-Technik.

API-Prozessmodell

Ziel

- Risiko minimieren
- Minimale Entwicklungszeit
- Wenig Bürokratie
- Geeignet für Neueinsteiger

Auswahl

- Ein für API angepasstes Spiralmodell
- Software-Entwicklung soll evolutionär oder inkrementell erfolgen

API-Spiralmodell

Teil A:

- Ziele für den kommenden Zyklus formulieren

Teil B:

- Risiken formulieren (falls vorhanden)
- Maßnahmen ausarbeiten und ggf. Risiken einbeziehen
- Kleinere Prototypen entwickeln

Teil C:

- Ziele ausarbeiten
  - Feinentwurf und/oder
  - Programmieren und/oder
  - Prototypen entwickeln

Teil D:

- Review des aktuellen Zyklus (Ziele erreicht? Qualität?)
- Planung des folgenden Zyklus

API-Spiralmodell

Beispiel

Ziele	Matrikel-Nr., Vorname und Name über Studentenausweis einlesen
Risiken	Daten sind nicht auslesbar
Maßnahmen	a) Prototyp: Testaufbau erstellen b) Auslesen des Studentenausweises testen c) ggf. andere Karte auslesen
Ergebnisse	a) Testaufbau fertiggestellt b) Studentenausweis kann nicht ausgelesen werden c) Auslesen der anderen Karten erfolgt problemlos
Review	*Inhalte hierfür folgen am 16.05.17*
Nächster Zyklus	Verknüpfung zwischen Student und Funduino-Kit

Teil II

Versionsverwaltung

Versionskontrolle?

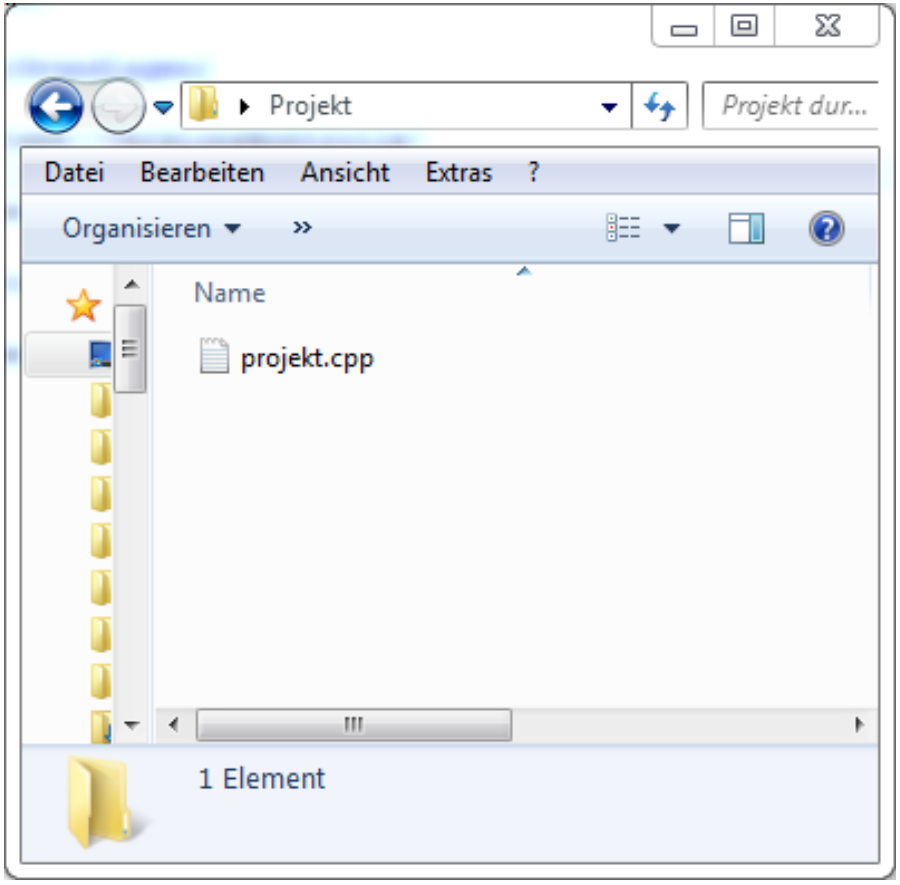


Abbildung 2: Versionsverwaltung durch Copy und Paste

Versionskontrolle?

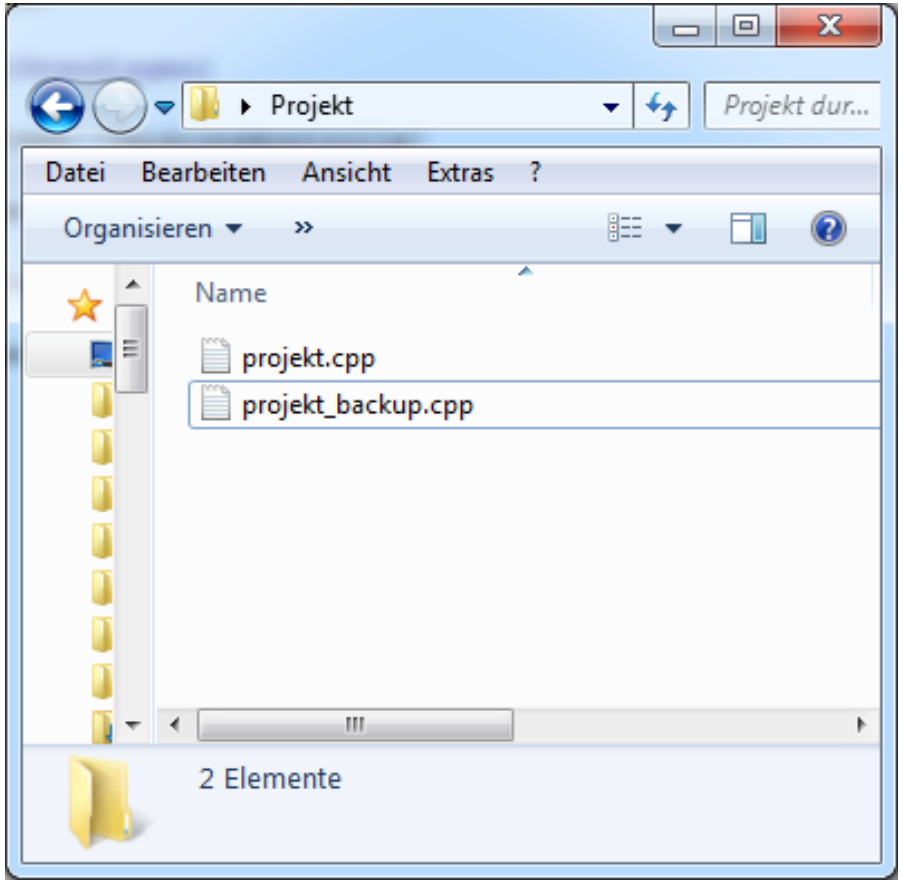


Abbildung 2: Versionsverwaltung durch Copy und Paste

Versionskontrolle?

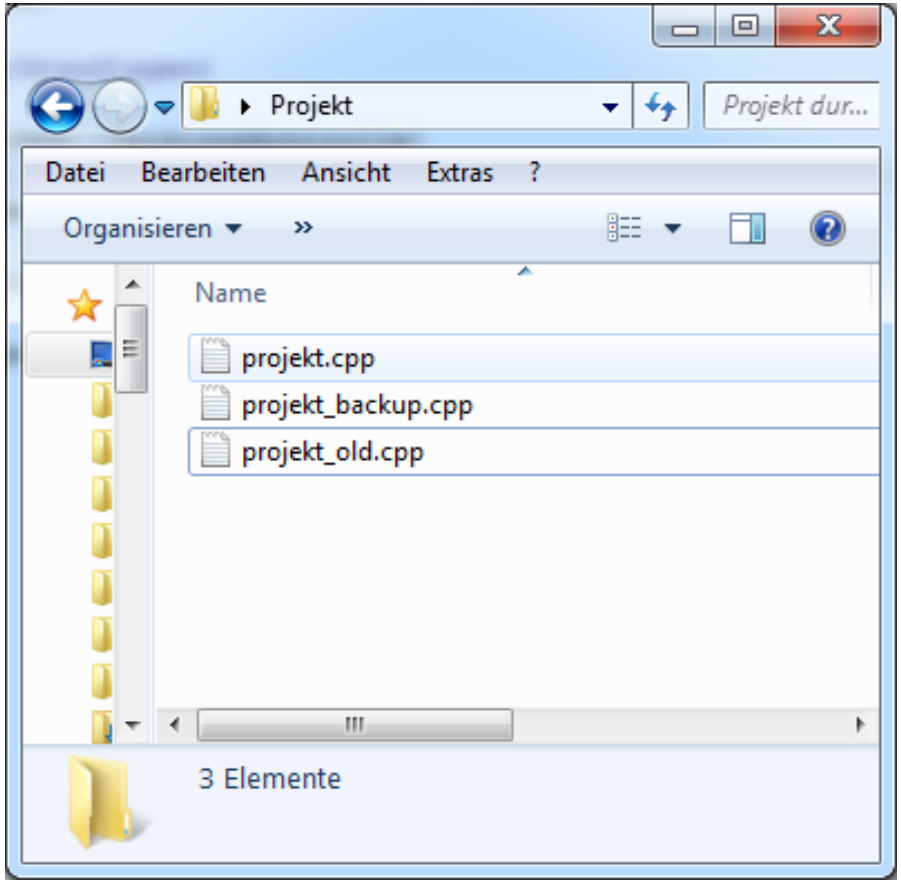


Abbildung 2: Versionsverwaltung durch Copy und Paste

Versionskontrolle?

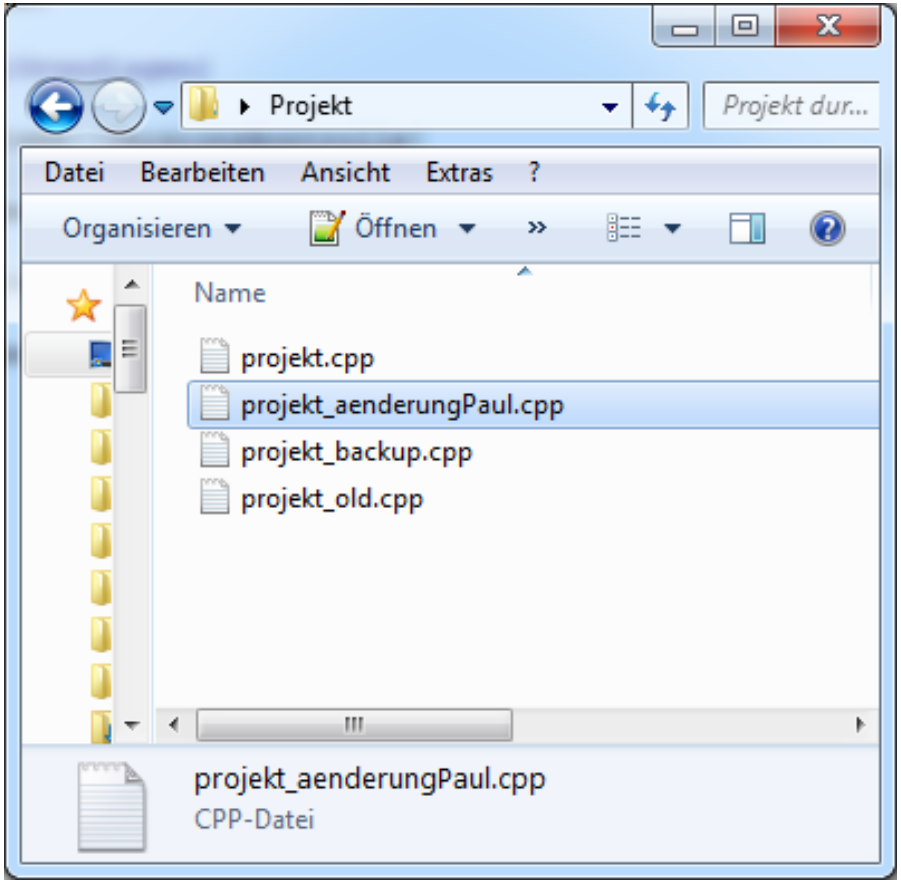


Abbildung 2: Versionsverwaltung durch Copy und Paste

Versionskontrolle?

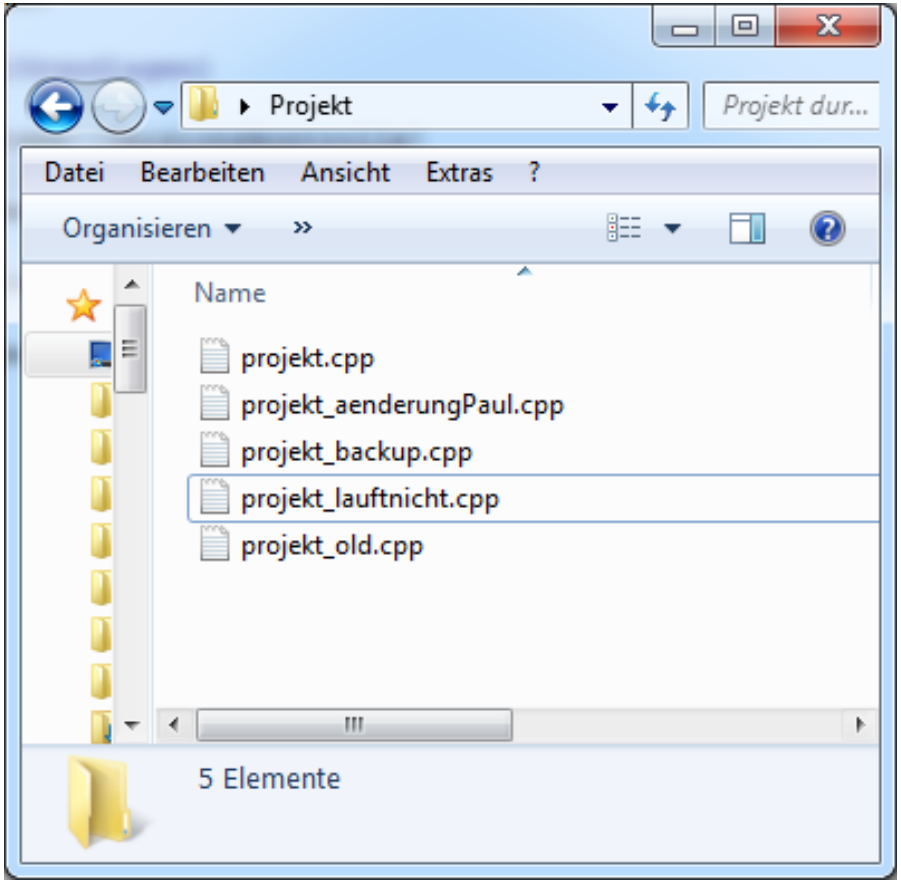


Abbildung 2: Versionsverwaltung durch Copy und Paste

Lösungsansatz: Dateiumbenennung

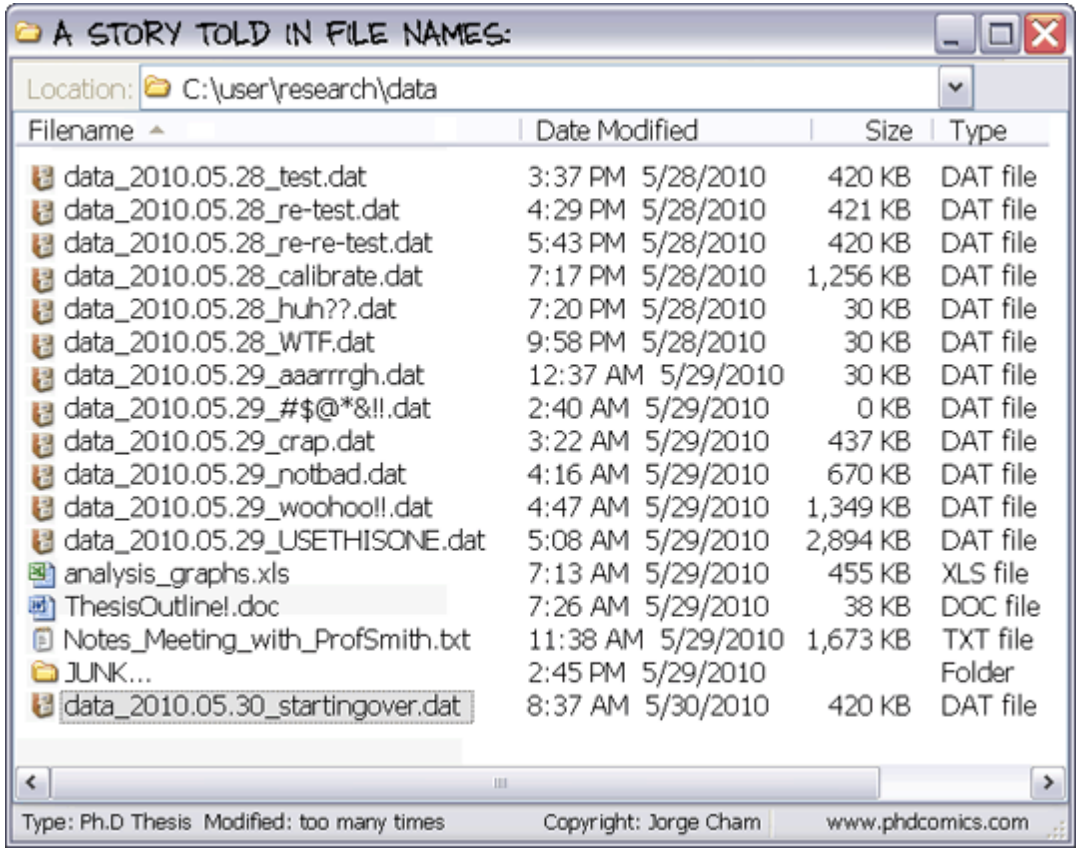


Abbildung 3: Versionierung mit Dateinamen
“Piled Higher and Deeper” by Jorge Cham
`http://www.phdcomics.com/comics.php?f=1323`

Quellcodeaustausch?

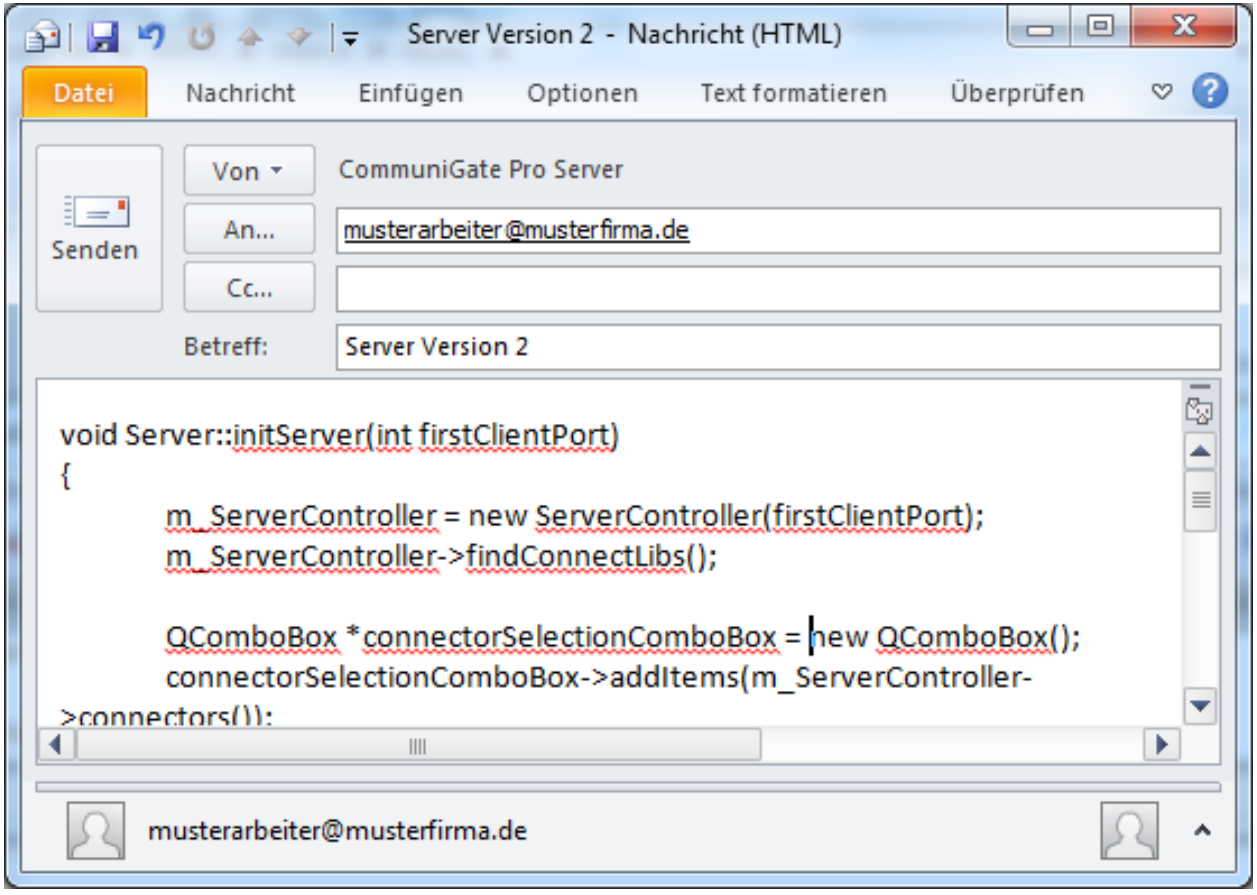
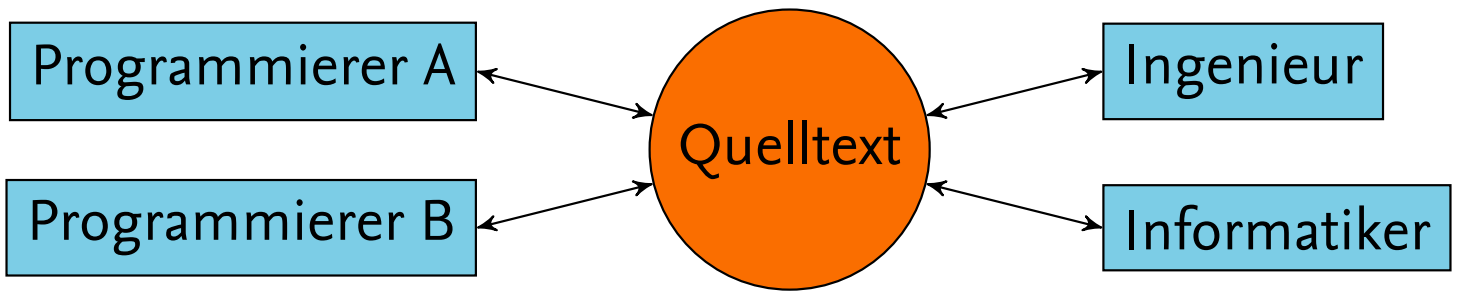


Abbildung 4: Quellcodeaustausch per E-Mail

Softwareentwicklung im Team

Ausgangsszenario

- Implementierung einer Steuerungssoftware
- Softwareentwicklung im Team
  - Programmierer
  - Ingenieure
  - Informatiker
- Arbeit an gemeinsamer Quelltextbasis



Überblick über den Quelltext

- Unstrukturierte Softwareentwicklung
- Übersicht über Quelltext fehlt
- Typische Fragen
  - „Hat noch jemand von der Datei eine alte Version? Ich habe was geändert, und das klappt nicht!“
  - „Hattest du da gerade auch was geändert? Das habe ich jetzt überschrieben!“
  - „Wer hat denn den Code geschrieben?!“

⇒ Ordnender Prozess für die Bearbeitung von Quelltext nötig

Ziele geordneter Entwicklung

- Parallele Bearbeitung von Quelltext
  - Verschiedene Entwickler
  - Unterschiedliche Computer/Arbeitsplätze
  - Verschiedene Betriebssysteme
- Nachvollziehbarkeit von Änderungen
  - Dokumentation des Bearbeiters
  - Ausschluss von Änderungen
  - Zurücksetzen auf frühere Dateiversion
- Releaseverwaltung (Veröffentlichung)
  - Markierung von Versionsständen
  - Archivierung von Veröffentlichung

$\Rightarrow$ Versionsverwaltung notwendig

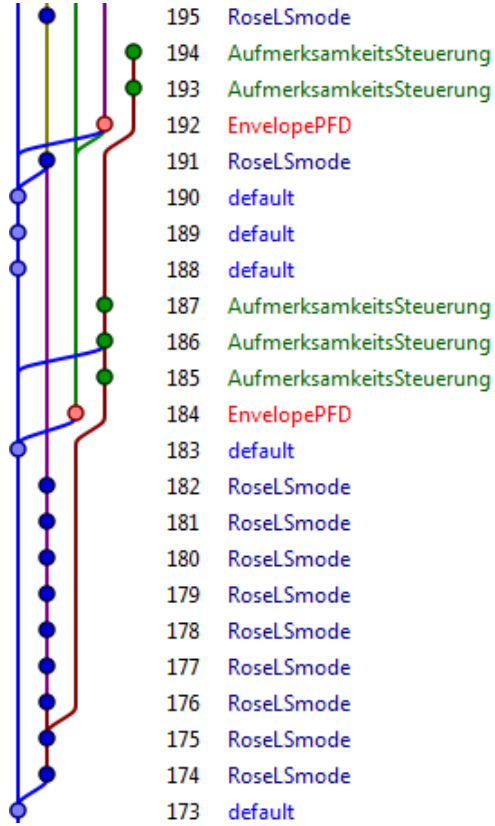


Abbildung 5: Versionsbaum

Lösungsansatz: Externe Werkzeuge

Periodische Sicherung auf zentralem Server

- Zu bestimmter Zeit (z. B. täglich oder stündlich) werden die Dateien extern gespeichert
- Umständlicher Zugriff
- Vergleich zwischen Sicherungen aufwändig

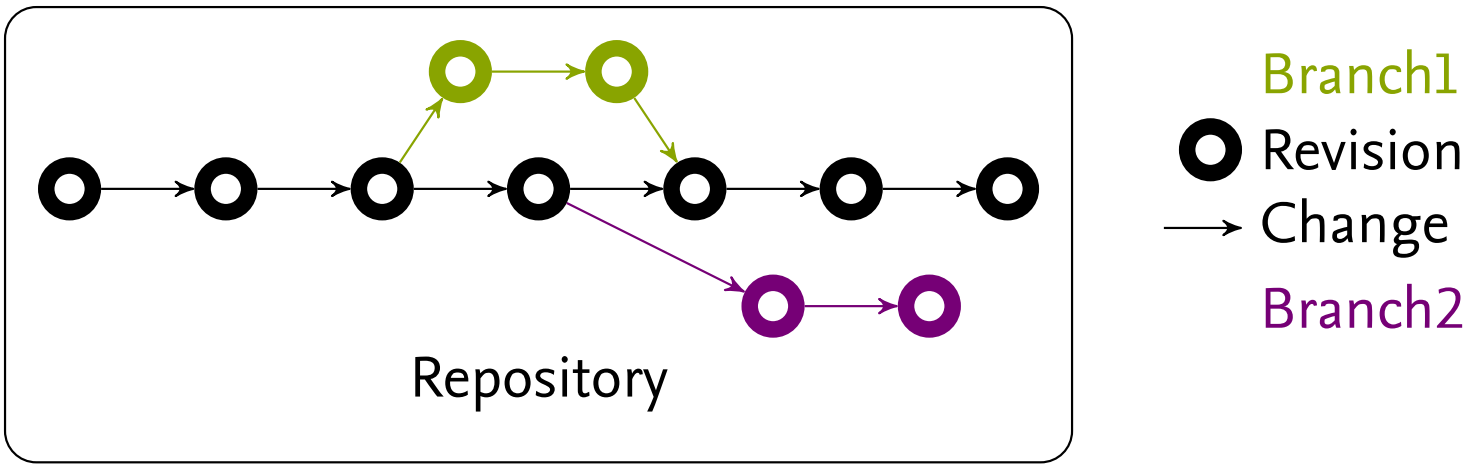
Versionskontrollsysteme (Version Control System, VCS)

- Sicherung von Dateiversionen und Änderungen zu vom Nutzer bestimmtem Zeitpunkt
- Unterstützung verschiedener Entwickler
- Dokumentation von Änderungen (inkl. Zeit und Autor)

Aufbau eines Versionskontrollsystems

Speichern von Änderungen zwischen Dateiversionen

- Eine Dateiversion ist eine Revision
- Änderungen werden als Change bezeichnet
- Die Sammlung aller Änderungen ist das Repository
- Ein Entwicklungszweig heißt Branch



Änderungen

- Entwickler nimmt Änderung am Quelltext vor
- Gespeichert wird nur der Unterschied zur alten Version
  - Aufgabe des VCS: Herausfinden der Änderung (Patch)
  - Lösung: Diff-Algorithmus sucht zeilenweise Änderungen an Textdateien
  - Problem: Keine Änderungen bei Binärdateien (z. B. Bilder, PDFs)

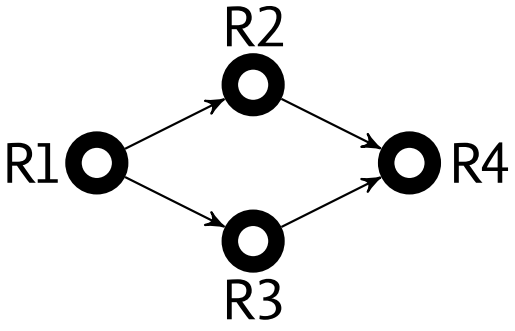
```
--- Version_SS2011.txt 2014-04-19 09:51:20.507788191 +0200
+++ Version_SS2015.txt 2015-04-19 09:51:51.725847652 +0200
@@ -1,5 +1,5 @@
Anwendungsorientierte Programmierung für Ingenieure
-SS 2011
+SS 2015
-Montags, 13:15 - 14:45 Uhr, Pk 15.1
+Dienstags, 16:45 - 18:15 Uhr, Pk 15.1
```

Änderungen im Versionskontrollsystem

- Übermittlung an das Repository: Commit/Check-In
  - Erzeugt eine neue Revision mit den Änderungen
  - Speichert Änderung, Autor und Zeit
- Angabe einer Änderungsmitteilung
  - Dokumentation und Nachweisführung
  - Manche Projekte schreiben spezielles Format vor oder ermöglichen Hervorhebungen oder Beziehungen durch Markup zu erstellen
- Andere Entwickler können auf die neue Revision zugreifen: Update
- Parallele Änderungen an Dateien können zu Konflikten führen

Konflikte im Versionskontrollsystem

- Konflikte entstehen, wenn mehrere Entwickler die selbe Datei ändern
- Kombination verschiedener Änderungen: Merge
  - Änderung unterschiedlicher Zeilen
    $\Rightarrow$ Automatisches Zusammenführen
  - Überschneidende Änderung
    $\Rightarrow$ Konflikt
- Konfliktauflösung durch Entwickler
  - Welche Änderungen sollen wie kombiniert werden?
  - Unterstützung durch 3-Way-Merge-Programme
  - Ergebnis wird als Merge hinzugefügt



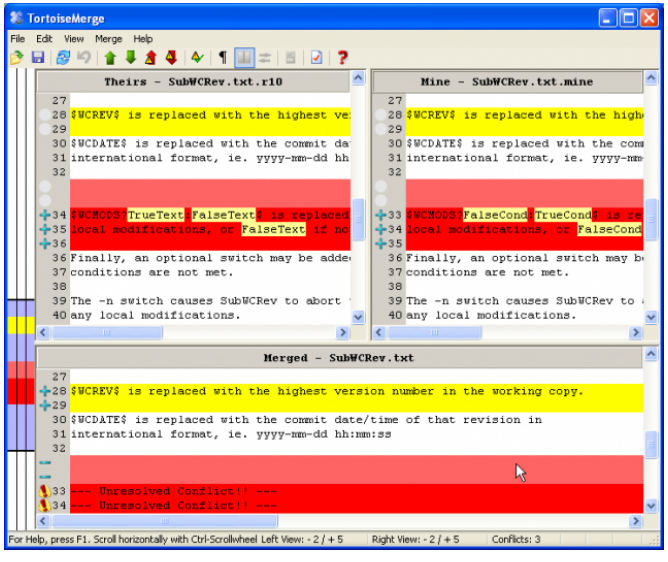


Abbildung 6: Konflikt

Vorteile von Versionskontrollsystemen

Nachvollziehbarkeit

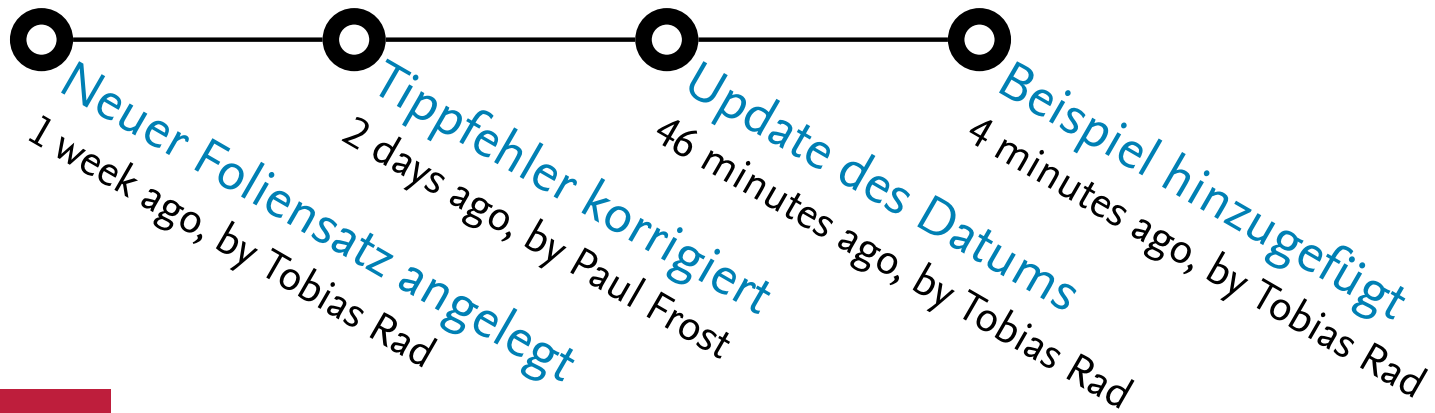
Wer hat was wann geändert?

Synchronisation

Änderungen verschiedener Entwickler werden zusammengeführt

Sicherheit

Wiederherstellbarkeit alter Versionen bei Fehlern



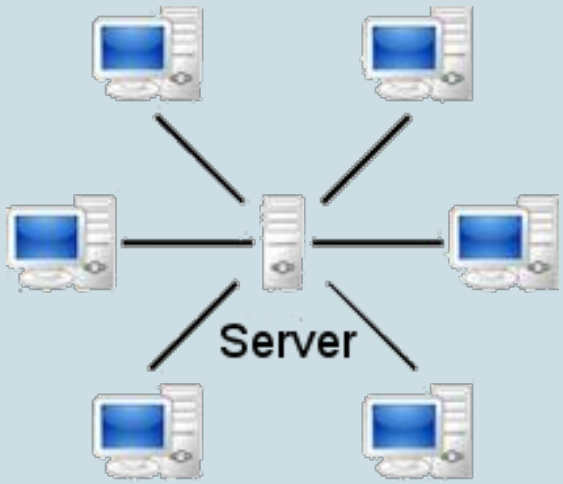
Varianten von Versionskontrollsystemen

Lokal

Revisionen werden im lokalen Dateisystem gespeichert

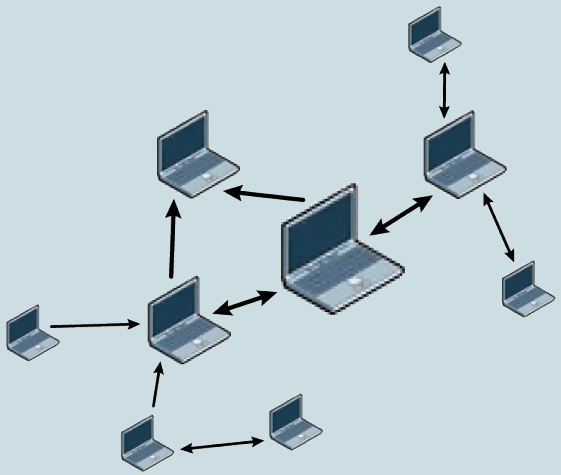
Zentral

Revisionen werden zentral auf einem Server verwaltet



Verteilt

Jede Instanz ist eine Kopie des gesamten Repositorys

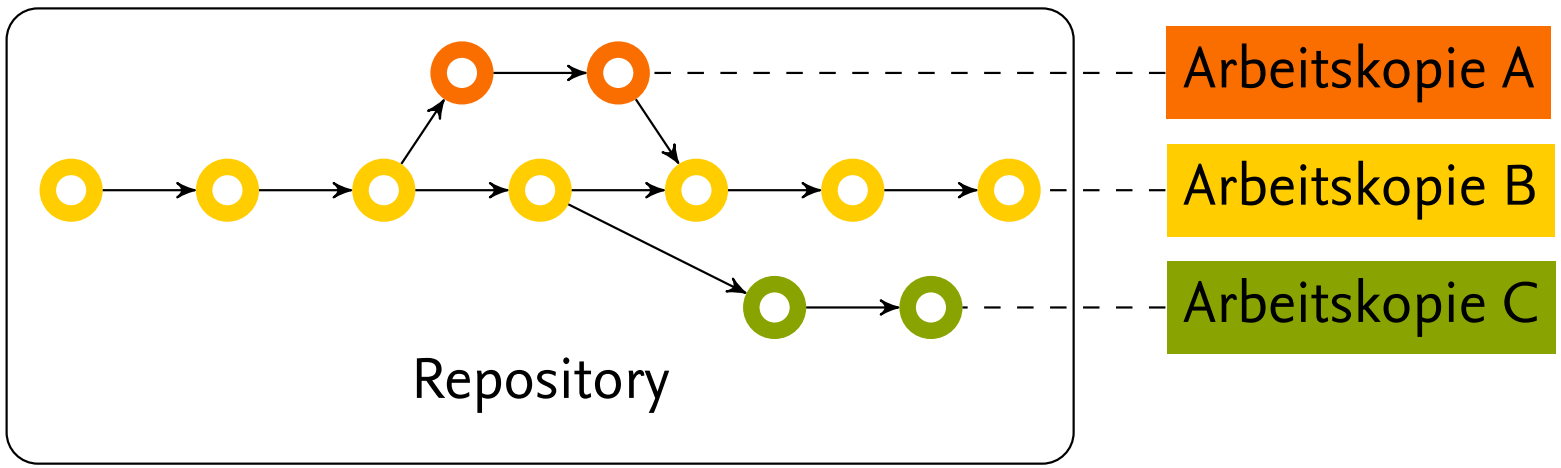


Lokale Versionskontrollsysteme

- Einfachste Möglichkeit des Versionskontrollsystems
- Änderungen und Verwaltungsinformationen werden lokal gespeichert
  - Kein geregelter Austausch zwischen Entwicklern
  - In der Regel nur Verwaltung einzelner Dateien
  - Überblick über gemachte Änderungen dennoch möglich
  - Verwendung heutzutage z. B. für Konfigurationsdateien

Zentrale Versionskontrollsysteme: Funktion

- Zentraler Server auf dem alle Revisionen gespeichert werden
- Authentifizierung möglich (z. B. über Benutzername und Passwort)
- Ein Benutzer hat auf seinem Rechner eine Arbeitskopie
  - Synchronisation mit Server ⇒ Update der Arbeitskopie
  - Entwickler verändern ihre Arbeitskopie
  - Einchecken der Änderungen beim zentralen Server



Verteilte Versionskontrollsysteme: Funktion

- Kein zentrales Repository erforderlich
  - Jeder Entwickler hat eine vollständige Kopie des Repository
  - Änderungen und Commits werden ins lokale Repository gepflegt
  - Einfaches Erzeugen und Mergen von Entwicklungszweigen
  - Synchronisation von Änderungen zwischen verschiedenen Entwicklern
- Durch Absprache kann ein Repository als Hauptrepository fungieren
  - Bitte um Übernahme von Commit aus eigenem Repository: Pull-Request
  - Testen von Änderungen vor dem Zusammenführen
  - Gezielte Übernahme einzelner Änderungen möglich (Cherry-Picking)
  - Am Ende erfolgt ein Merge mit dem Hauptentwicklungszweig

$\Rightarrow$ Verteilte Versionskontrollsysteme versuchen die Vorteile von lokaler und zentraler Versionskontrolle zu verbinden

Zusammenfassung

- Software-Entwicklung im Team benötigt Methode zur Verwaltung von Änderungen
- Kommunikation und Absprache zwischen den Entwicklern bleibt notwendig
- Spezialisiertes Werkzeug zum Änderungsmanagement: Versionskontrollsysteme
  - Ermöglichen paralleles Programmieren mehrerer Entwickler
  - Erhöhen Fehlertoleranz und Nachvollziehbarkeit
  - Einfache Bedienung durch Integration in graphische Oberflächen
- Verschiedene Versionskontrollsysteme vorhanden

Arbeit mit Konsolenapplikationen

Wichtige Konsolenbefehle

`cd` Ordner wechseln

`ls` Ordnerinhalt anzeigen

`..` Übergeordneter Ordner

Syntax für Konsolenbefehle

- `$ befehl argumente`
- `$ programm argumente`

git-Installation überprüfen

Eingabe des Befehls:

```
$ git --version
```

Git einrichten

Mit `$ git config` können Einstellungen vorgenommen werden.

Ausprobieren

- `$ git config --global user.name "Max MusterstudentInnenX"`
- `$ git config --global user.email "mmusterstudentinnenx@uni.de"`

Help — Hilfe anzeigen

Mit `$ git -h` wird die Hilfefunktion aufgerufen.

Die Eingabe von

```
git init -h
```

führt zu

```
git init [-q | --quiet] [--bare]
    [--template=<template_directory>]
    [--separate-git-dir <git dir>]
    [--shared[=<permissions>]] [directory]
```

Ausprobieren

- `$ git init -h` eingeben

Init

Repository anlegen

Mit `$ git init` wird in einem bestehenden Ordner ein Repository angelegt.

Ausprobieren

- Ordner „Testrepo“anlegen
- Ordner in Kommandozeile öffnen
- `$ git init` eingeben

Init

Repository-Status

Repository: Testrepo

Commit — Übermittlung von Änderungen

Mit `$ git commit` wird der aktuelle Stand von ausgewählten Dateien im Repository gespeichert.

```
git commit <Dateien> -m "Eine Nachricht"
```

Ausprobieren

- Readme.md anlegen
- „# Überschrift der Startseite“ in Readme.md eintragen
- `$ git add Readme.md` eingeben
- `$ git commit Readme.md -m "Readme hinzugefuegt"` eingeben

Commit

Repository-Status

Readme hinzugefügt

Repository: Testrepo

Commit

Repository-Status

Clone — Repository klonen

Mit `$ git clone` wird das gesamte Repository geklont.
Achtung: Wird ein Repository lokal geklont, muss evtl. der Server neu gesetzt werden (siehe `$ git remote`).

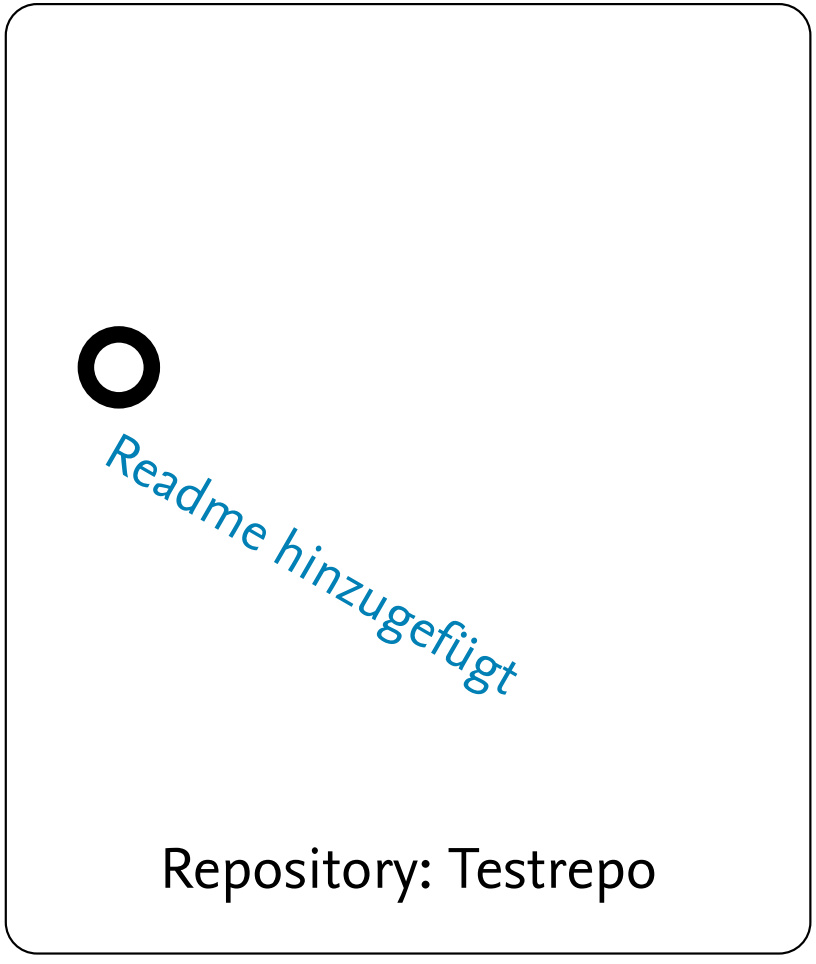
```
git clone Repository [Neuer_Repository_Ordner]
```

Ausprobieren

- In das Übergeordnete Verzeichnis wechseln
- `$ git clone Testrepo Testrepoclone` eingeben

Clone

Repository-Status



Remote — Verknüpftes entferntes Repository

Mit `$ git remote` wird ein entferntes Repository verknüpft, welches anschließend für `$ git push` oder `$ git pull` verwendet werden kann. Dieses Repository sollte ein Server sein (angelegt mit `$ git init --bare`).

Standardname: „origin“.

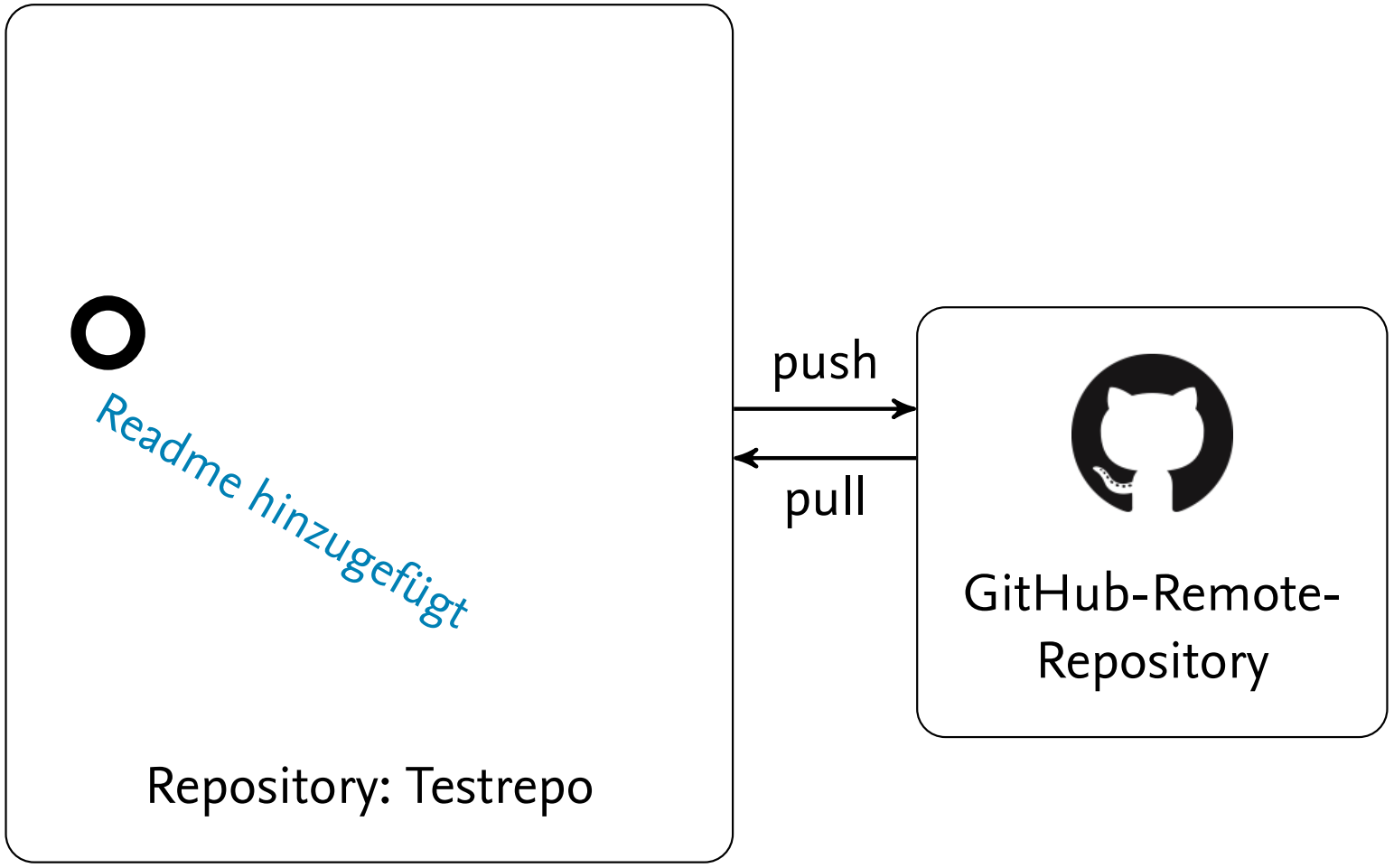
```
git remote add Name_Verknuepfung Repository_URL_Verzeichnis
```

Ausprobieren

- Legen Sie in GitHub ein eigenes öffentliches Repository an
- Verknüpfen Sie dieses GitHub-Repository mit dem Testrepo

Remote

Repository-Status



Push

Stand hochladen

Mit `$ git push` wird ein bestimmter Zweig oder alle Zweige eines Repositorys zum entfernten Repository hochgeladen.

Wichtig: Hierfür muss das lokale Repository auf dem aktuellen Stand sein (siehe `$ git pull`).

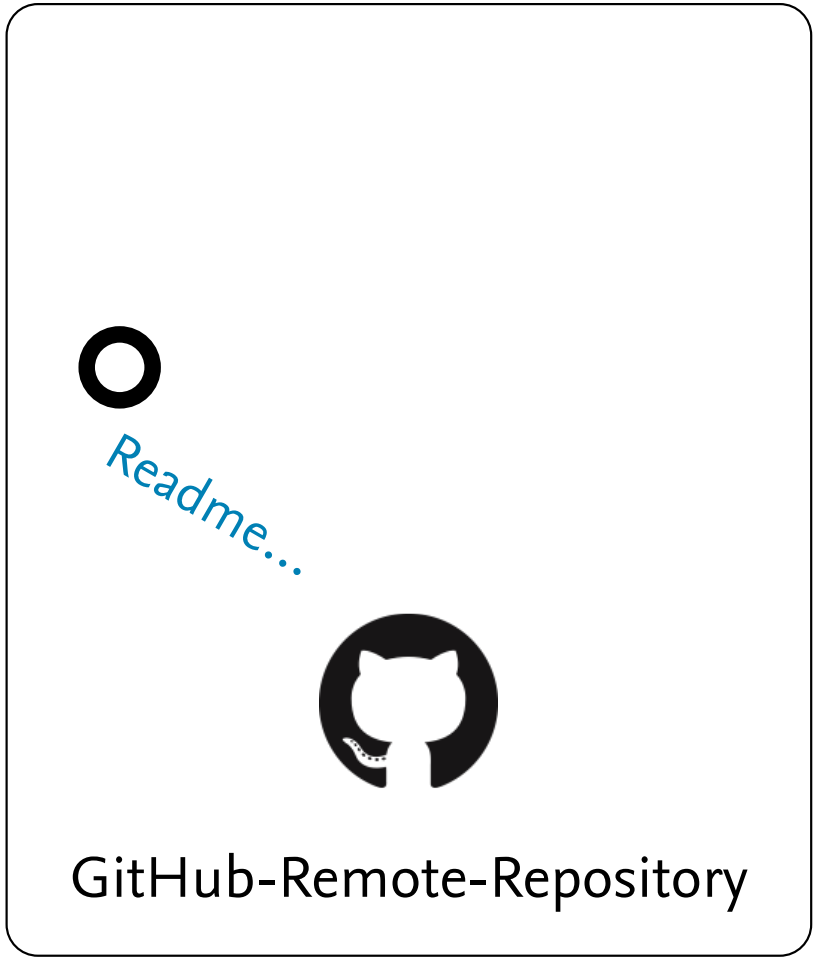
```
git push Name_Verknuepfung [Name_Zweig]
```

Ausprobieren

- Laden Sie den aktuellen Stand auf das GitHub-Repository

Push

Repository-Status



Pull

Stand herunterladen

Der Befehl `$ git pull` kombiniert die Befehle `$ git fetch` und `$ git merge`. Somit wird der Stand vom entfernten Repository heruntergeladen und mit der lokalen Version zusammengeführt.

Wichtig: Es können Konflikte entstehen.

```
git pull Name_Verknuepfung [Name_Zweig]
```

Ausprobieren

- Verändern Sie über GitHub etwas am Text in der Datei Readme.md
- Laden Sie den Stand von GitHub herunter

Praxisdomonstration: Teamarbeit über git

Checkout

Zweig wechseln/Zweig erstellen

Mit dem Befehl `$ git checkout` kann der Stand eines anderen Zweigs geladen werden.

Wichtig: Für einen Wechsel müssen alle Änderungen über den Befehl `$ git commit` Eingecheckt sein.

```
git checkout Name_Zweig
```

Existiert der Zweig noch nicht muss dieser über die Option `-b` erstellt werden.

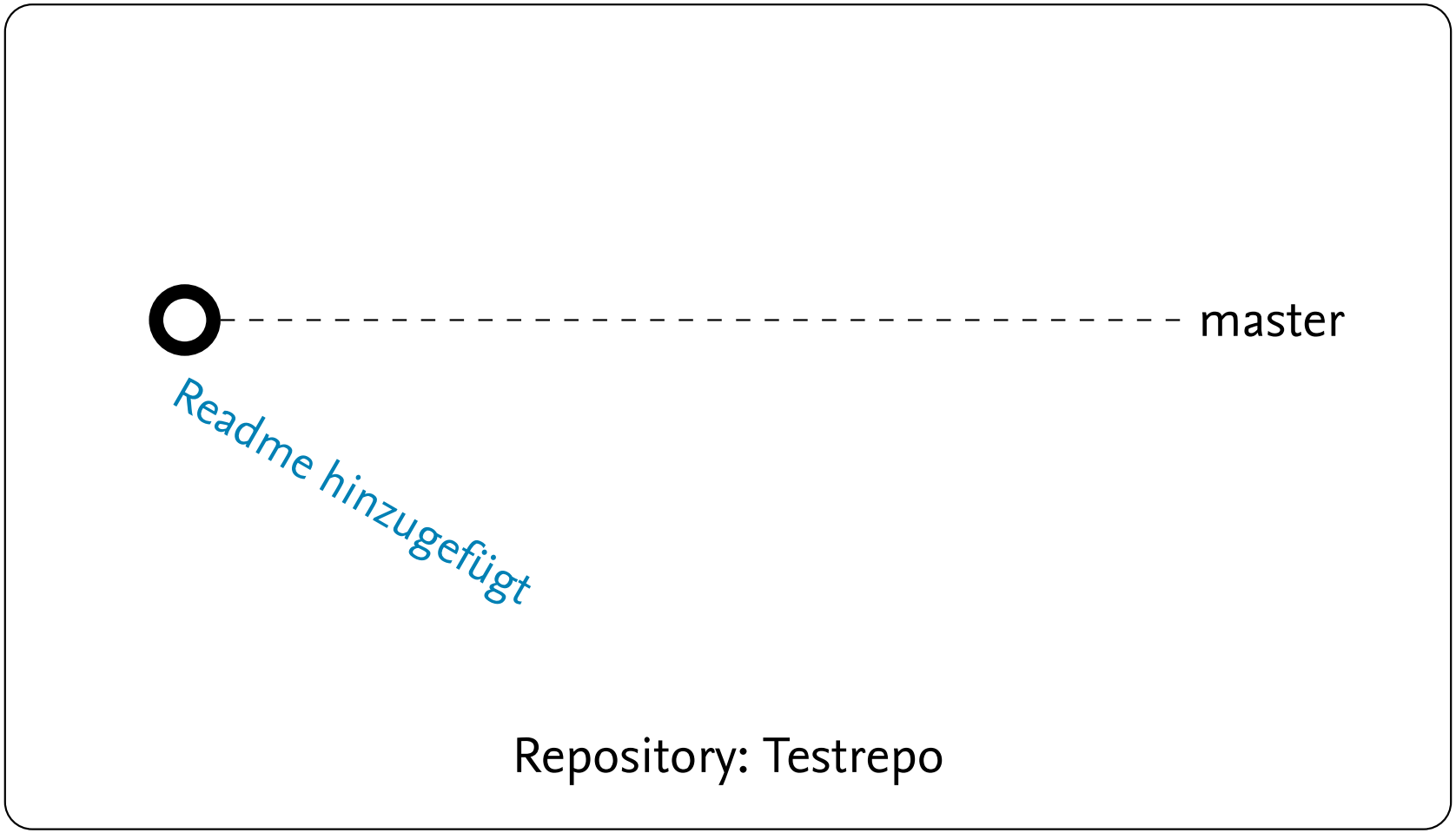
```
git checkout -b Name_neuer_Zweig
```

Ausprobieren

- Erstellen Sie über den Befehl `$ git checkout` einen neuen Zweig

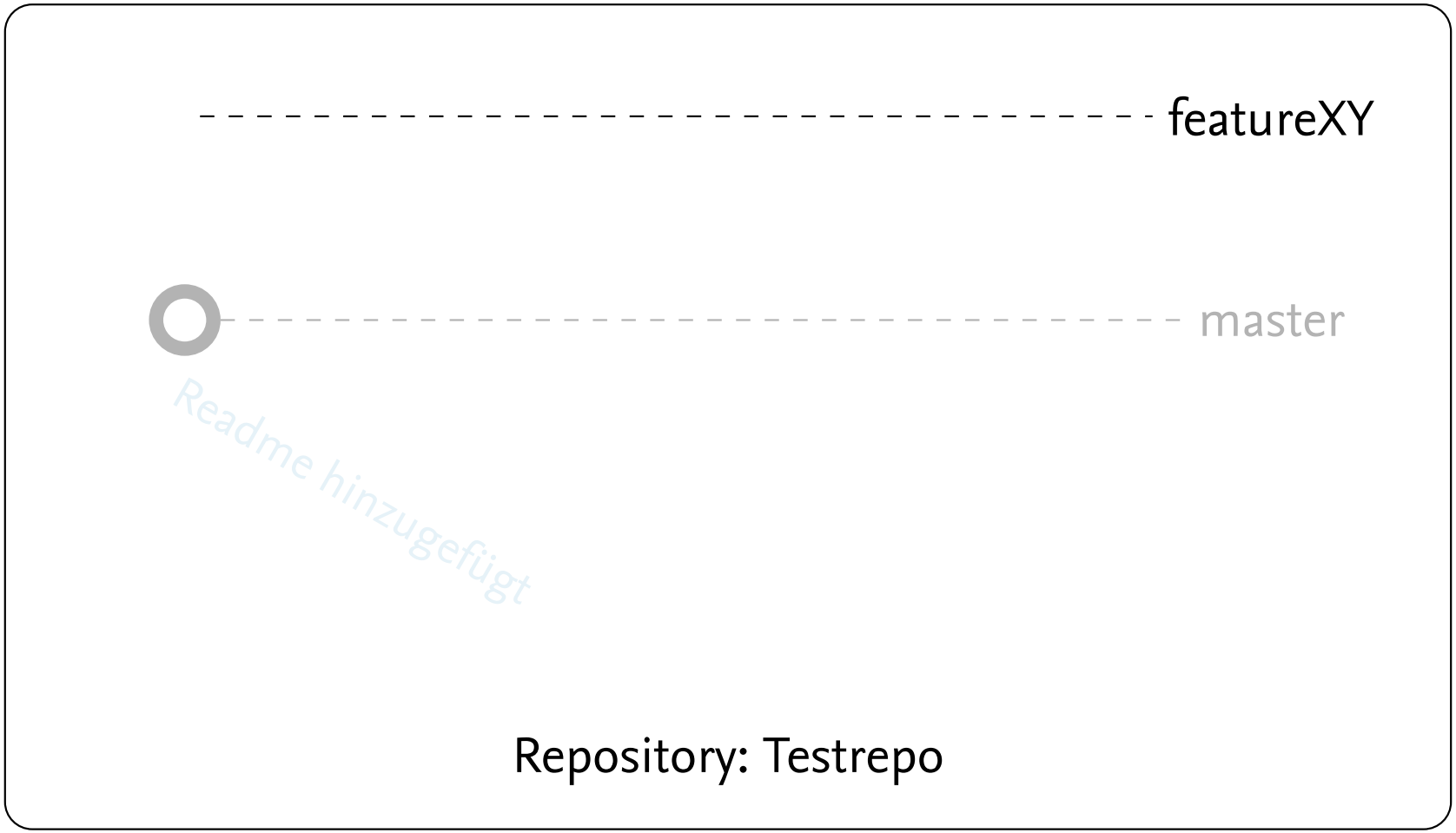
Checkout

Repository-Status



Checkout

Repository-Status



Checkout Änderungen widerrufen

Der Befehl `$ git checkout` kann gleichzeitig dafür verwendet werden, Änderungen bis zu dem letzten Commit wiederherzustellen. Hierzu wird die Option `--` als Option vor einer Liste mit Dateien gesetzt.

```
git checkout -- <Dateien>
```

Ausprobieren

- Modifizieren Sie die Readme.md
- Widerrufen Sie die Änderungen

Merge

Zweige zusammenführen

Der Befehl `$ git merge` Führt die eingecheckten Änderungen zweier Zweige zusammen. Anschließend werden die Änderungen über ein Commit bestätigt.

Wichtig: Die Änderungen eines anderen Zweigs werden in den aktiven Zweig überführt!

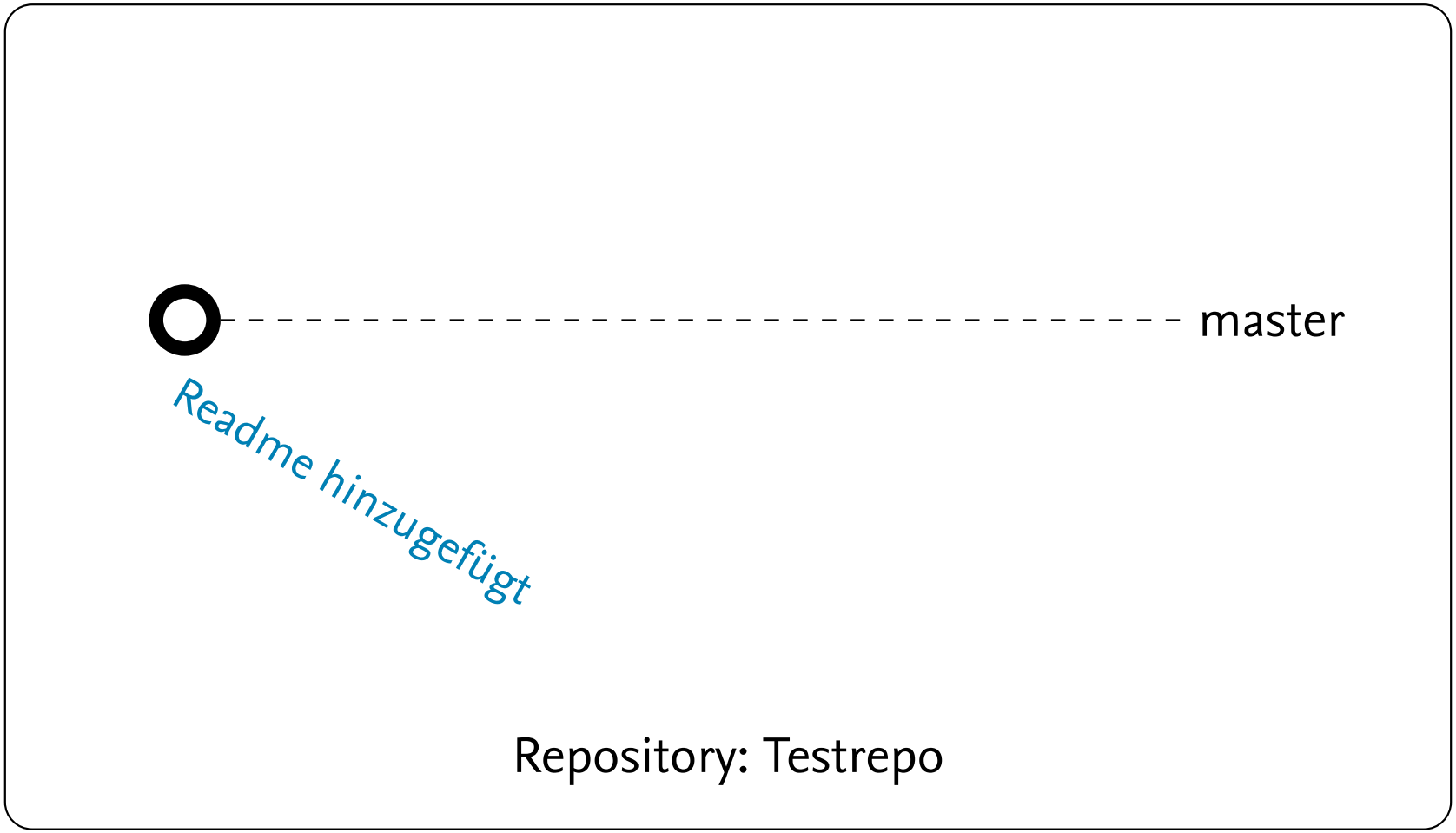
```
git merge Name_Zweig
```

Ausprobieren

- Checken Sie 2 neue Dateien in den neu erstellten Zweig ein
- Wechseln in den master-Zweig
- Überführen Sie die Änderungen des anderen Zweigs in den master-Zweig

Merge

Repository-Status

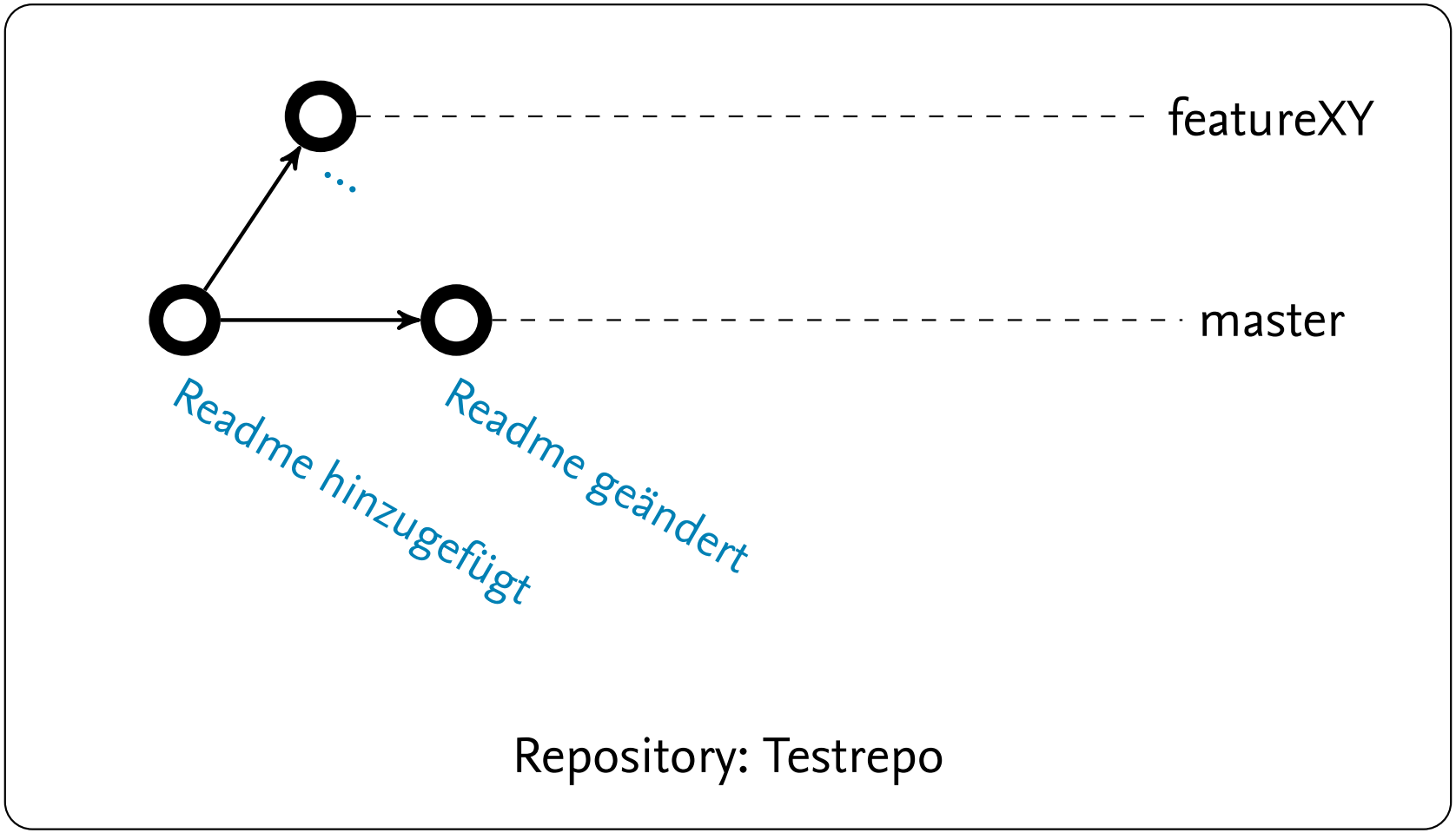


Merge

Repository-Status

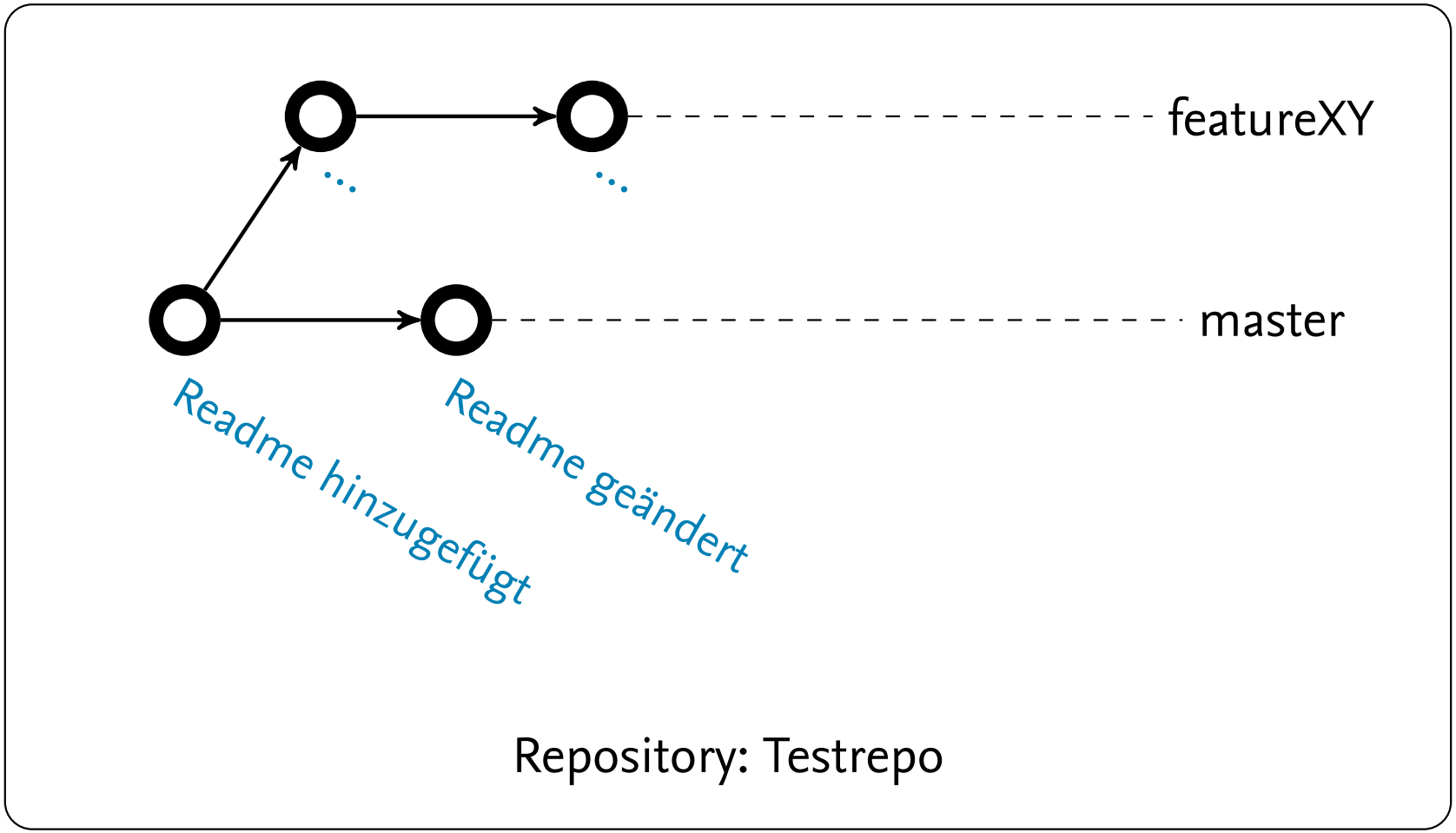
Merge

Repository-Status



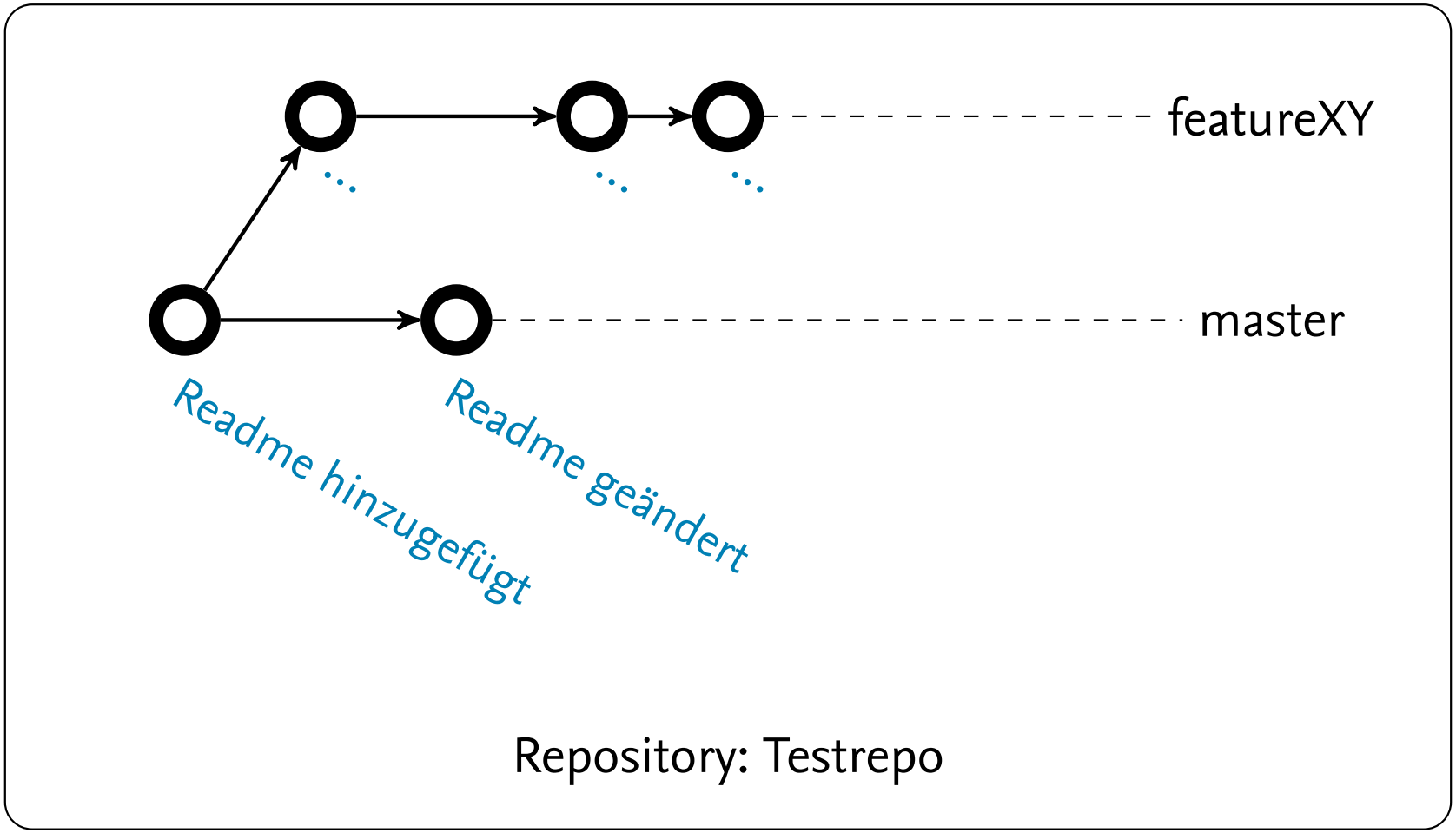
Merge

Repository-Status



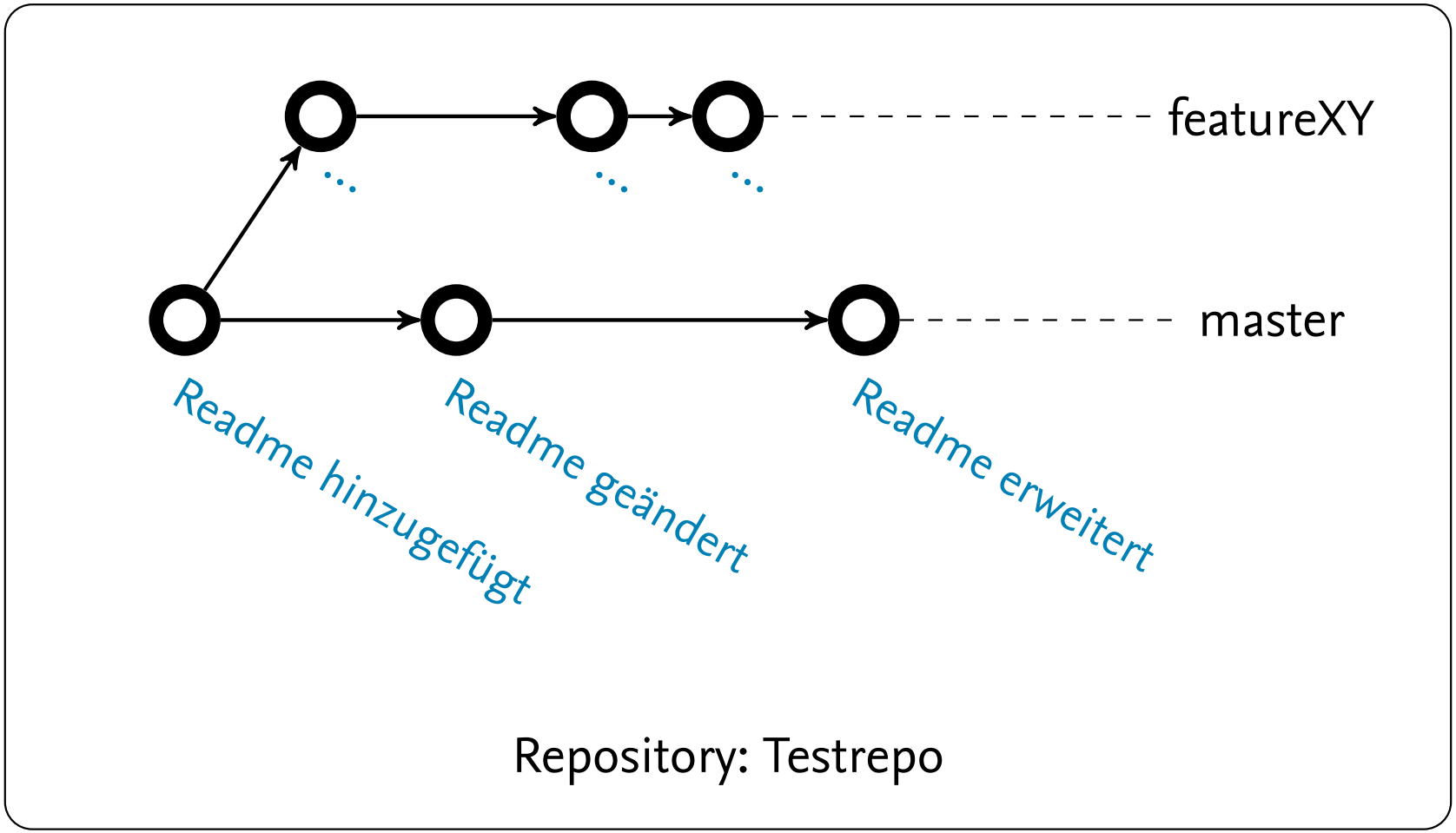
Merge

Repository-Status



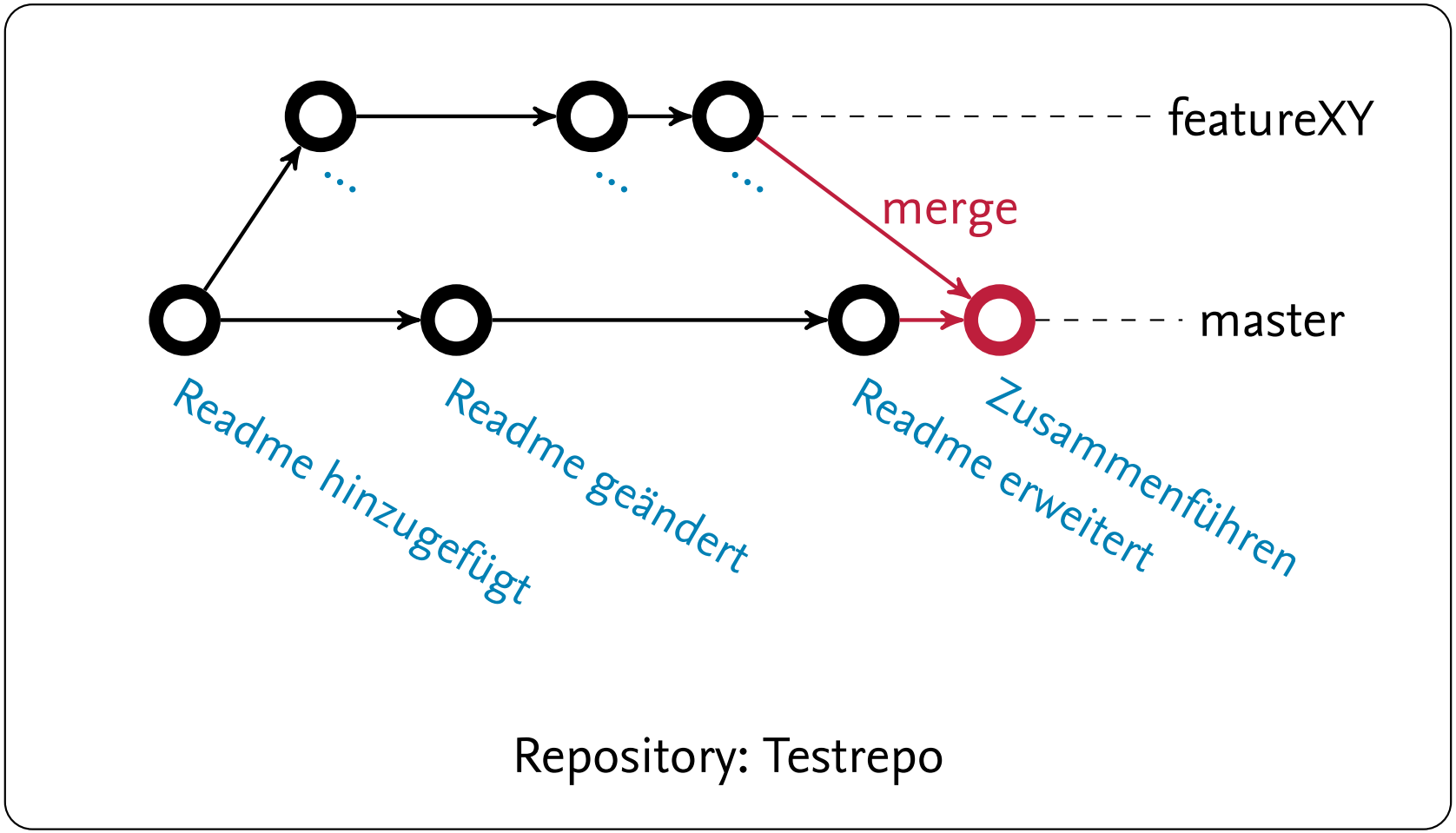
Merge

Repository-Status



Merge

Repository-Status



Konflikte

- Konflikte sind Änderungen in gleichen Abschnitten einer Datei in unterschiedlichen Zweigen
- Tauchen bei dem Zusammenführen von Zweigen auf
- Müssen händisch aufgelöst werden

Kommunikation erforderlich!

Konflikte lösen

Dateiinhalt einer konfliktbehafteten Datei:

```
Bis hierhin ist alles konfliktfrei
<<<<<<< eigeneId:sample.txt
eigene Version des Konflikts
=======
andere Version des Konflikts
>>>>>>> andereId:sample.txt
Ab hier ist es wieder konfliktfrei
```

Datei kann über den Editor angepasst werden. Markierungen entfernen!

Ausprobieren

- Verändern Sie über GitHub etwas am Text in der Datei Readme.md
- Nehmen Sie eine andere Änderung lokal an der Datei Readme.md vor
- Laden Sie den Stand von GitHub herunter

Ignore-Datei

Dateien ignorieren

In der Datei `.gitignore` können Dateien von der Versionierung ausgeschlossen werden. Diese Datei liegt in dem Repositoryordner und kann auch versioniert werden.

```
odner/
datei.tex
*.exe
Dateien_mit*
```

Unter Umständen sinnvoll für:

- Binärdateien
- Temporäre Dateien (*.log)
- Gebaute Dateien (*.dll, *.exe)

Pull-Request GitHub Feature

- Andere Projektentwickler von den Änderungen in Kenntnis setzen
- Änderungen können eingesehen und kommentiert werden
- Review-Prozess kann beginnen
- Unterschiede zu dem Hauptzweig können begutachtet werden
- Weitere Commits können aufgenommen werden
- Abschließend erfolgt ein Merge
- Der Zweig kann im Anschluss gelöscht werden

⇒ Möglichkeit zur Kommunikation

Praxisdomonstration: Pull-Request

Zusammenfassung

- Bedeutung von Versionsverwaltung
- Unterschied zwischen
  - lokaler
  - zentraler
  - verteilter

  Versionsverwaltung
- Arbeit mit git

Teil III

Projektarbeit

Einarbeitung Git

Aufgabe 1

1. Falls noch nicht erfolgt, bearbeiten Sie die Ausprobieren-Aufgaben des Abschnitts Git-Übung

Aufgabe 2

1. Laden Sie die Diagrammbilder in das Wiki-Repository
2. Fügen Sie die hochgeladenen Bilder in die Dokumentation ein.

Dokumentation Projektverlauf

Aufgabe 3

1. Erstellen Sie im Wiki Ihres Projekts die Seite Projektverlauf
2. Dokumentieren Sie den bisherigen Verlauf des Projekts

Fragen?

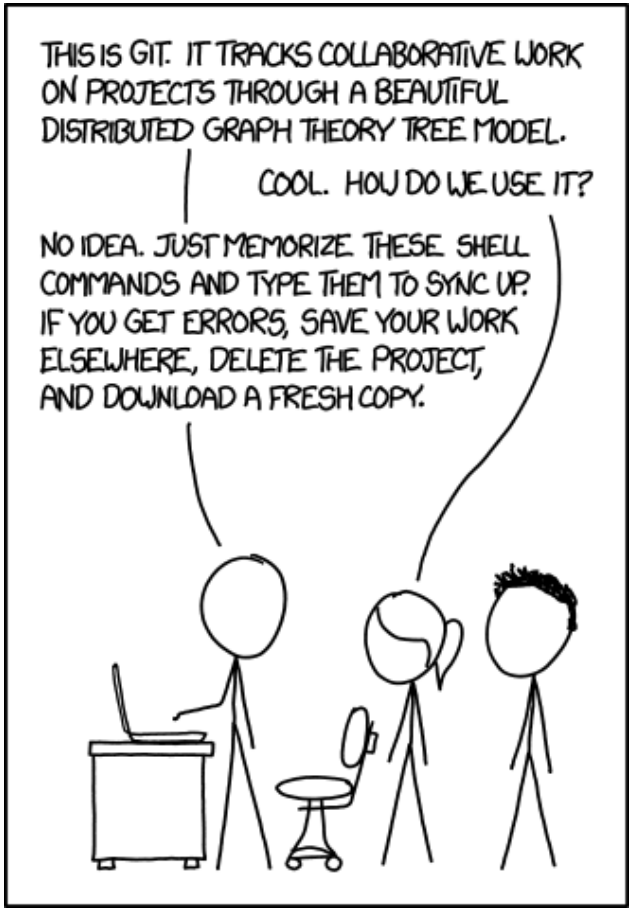


Abbildung 7: Git Comic von xkcd: `https://xkcd.com/1597/`