

Pflichtmodul 12: Entwurf von Softwaresystemen

Werkzeuge und Techniken des Softwareentwurfs

Clemens Sauerwein
Wintersemester 2018/19

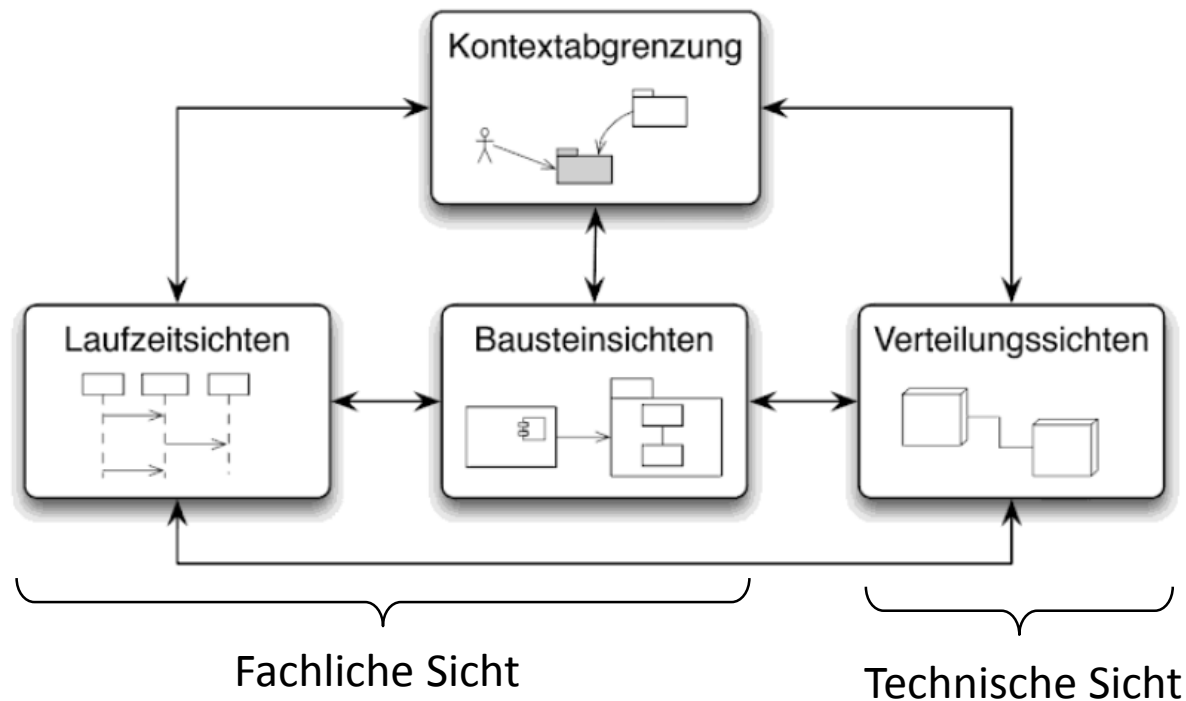
Inhalte

1. UML Sequenzdiagramme
2. Dokumentation von Source Code
3. Versionskontrollsysteme
4. Zusammenfassung

1. UML-Sequenzdiagramme

Beschreibung von Softwarearchitekturen

- Softwarearchitekturen werden meist auf Basis des **Komponenten**-begriffs beschrieben
- Das System wird in verschiedenen **Sichten** beschrieben

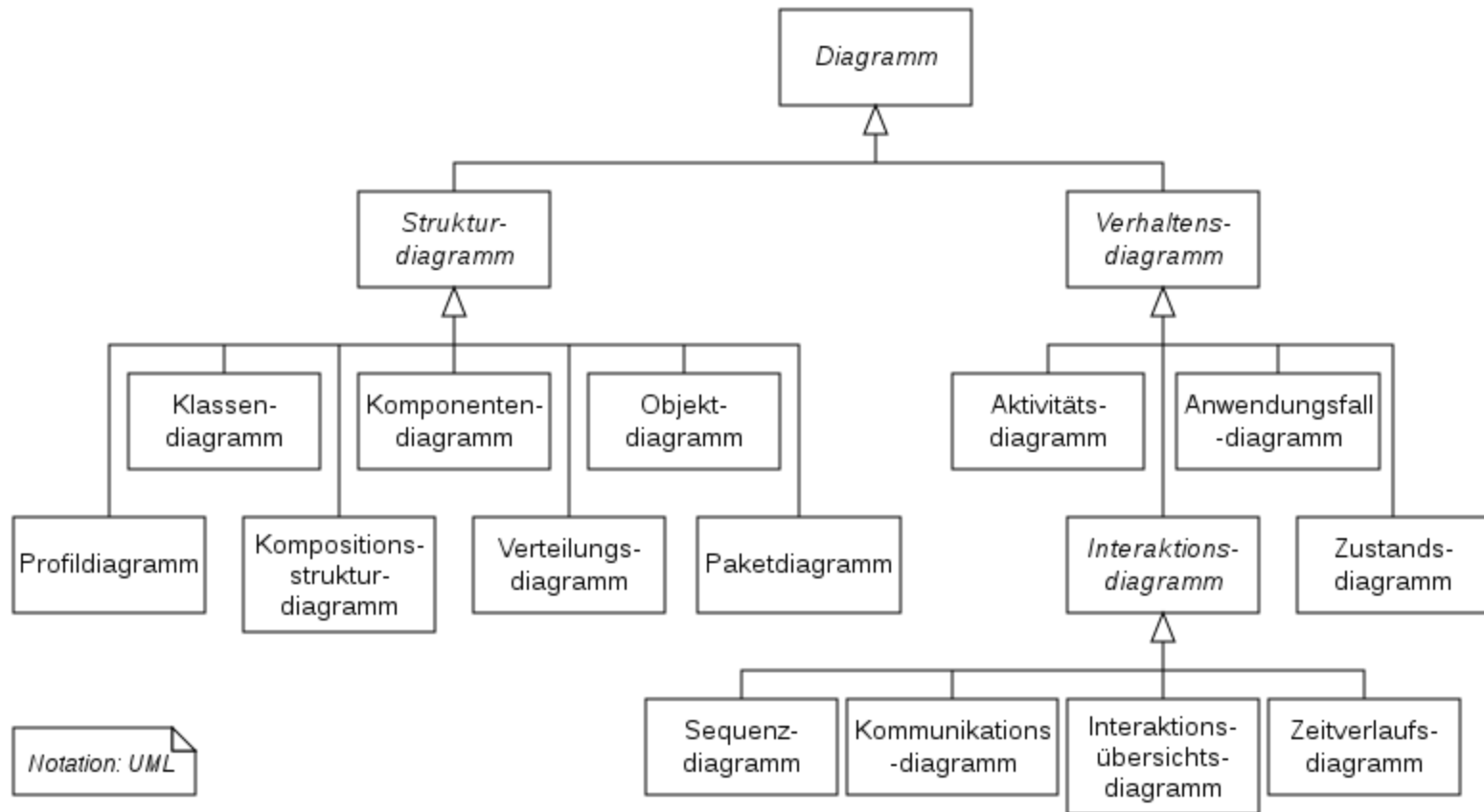


UML Klassendiagramme

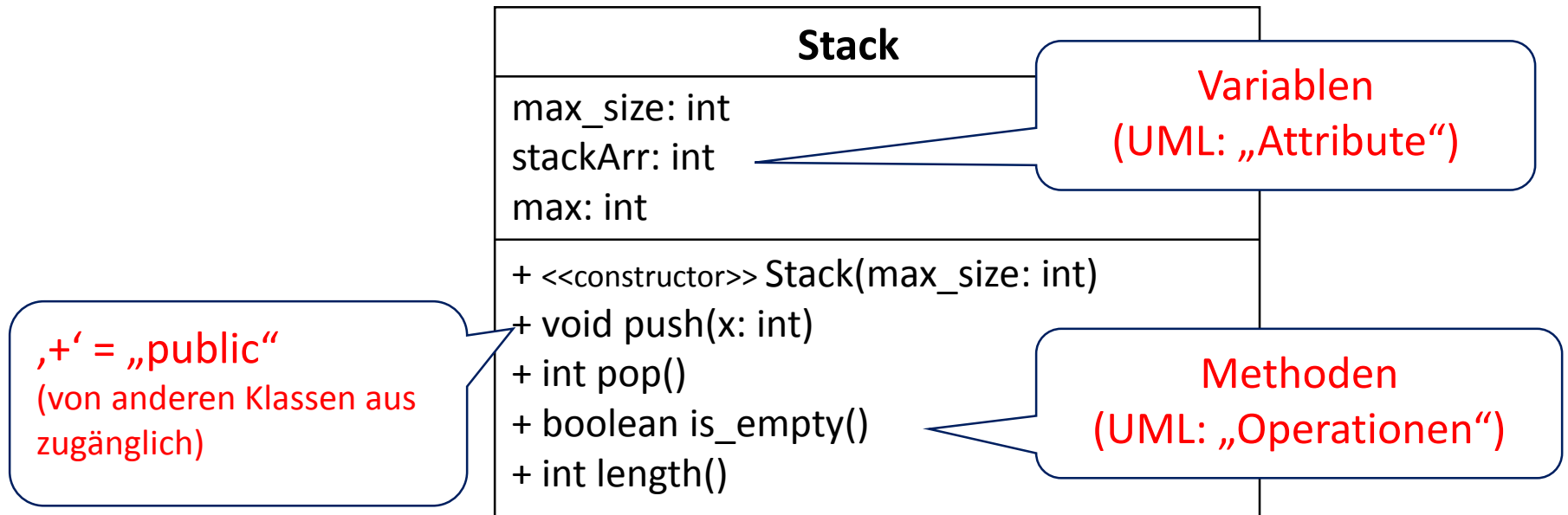
- Unified Modeling Language (UML) – eine standardisierte Sammlung von Notationen zur graphischen Beschreibung von Softwaresystemen
- Diagramme zur Beschreibung von Klassen und ihren Beziehungen
- **Literaturempfehlung**
 - UML@Classroom – Eine Einführung in die objektorientierte Modellierung
 - *Martina Seidl, Marion Scholz, Christian Huemer, Gerti Kappel.*
 - *Dpunkt, 2012.*
 - <http://www.uml.ac.at/de/>



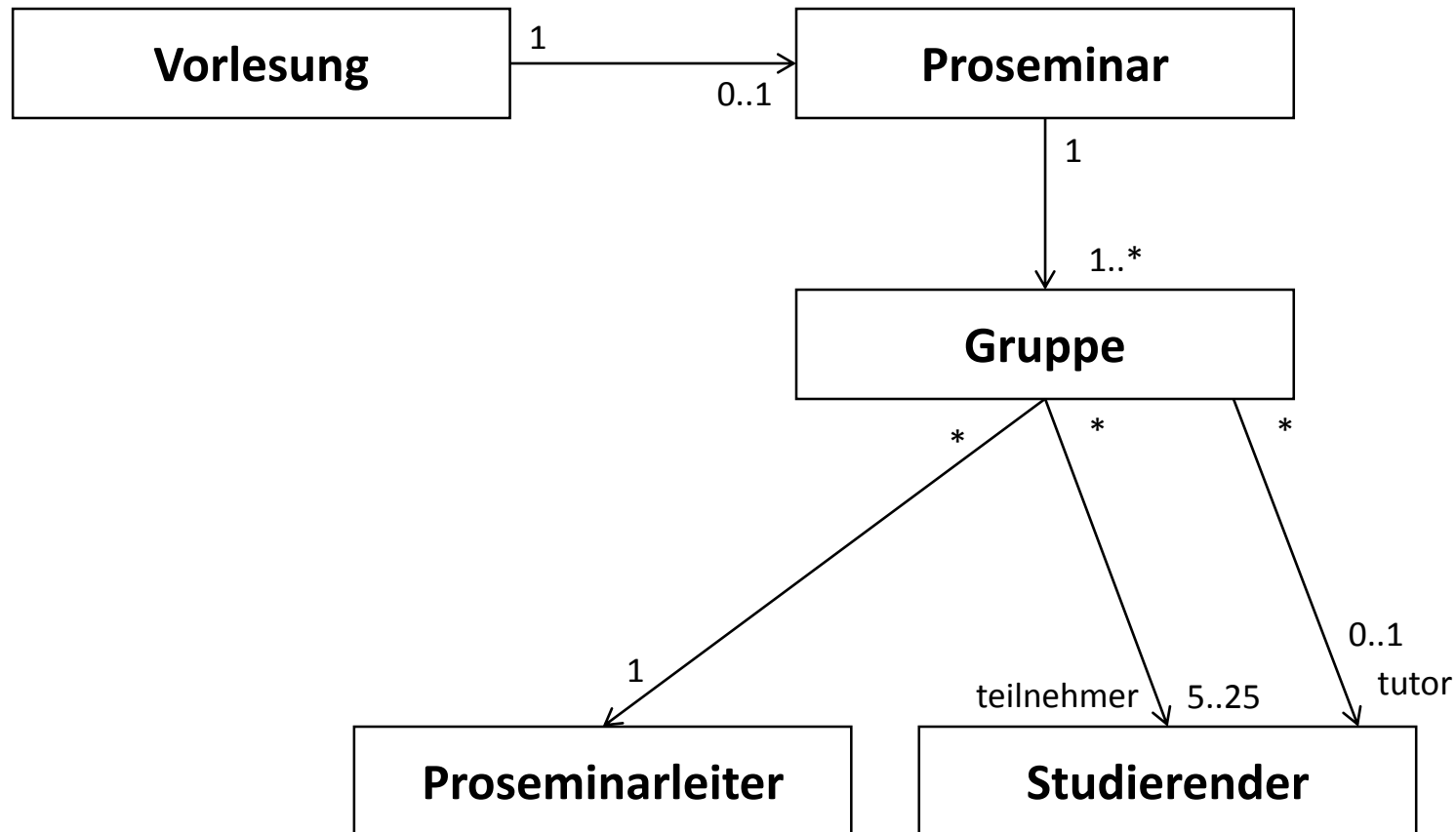
UML Diagrammarten



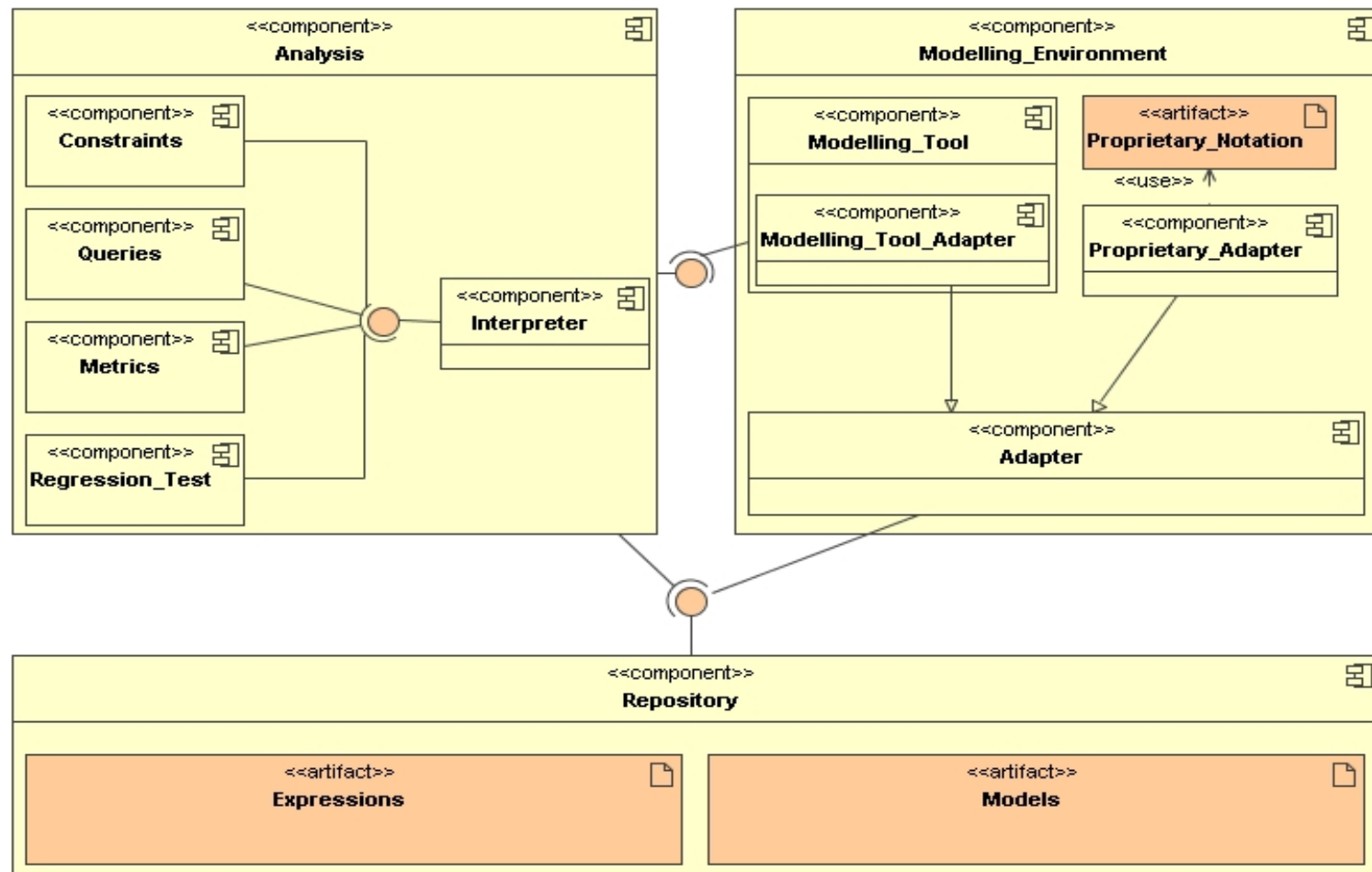
UML Klassendiagramm



Beispiel: (Fachliches) UML Klassendiagramm



Beispiel: UML Komponentendiagramm

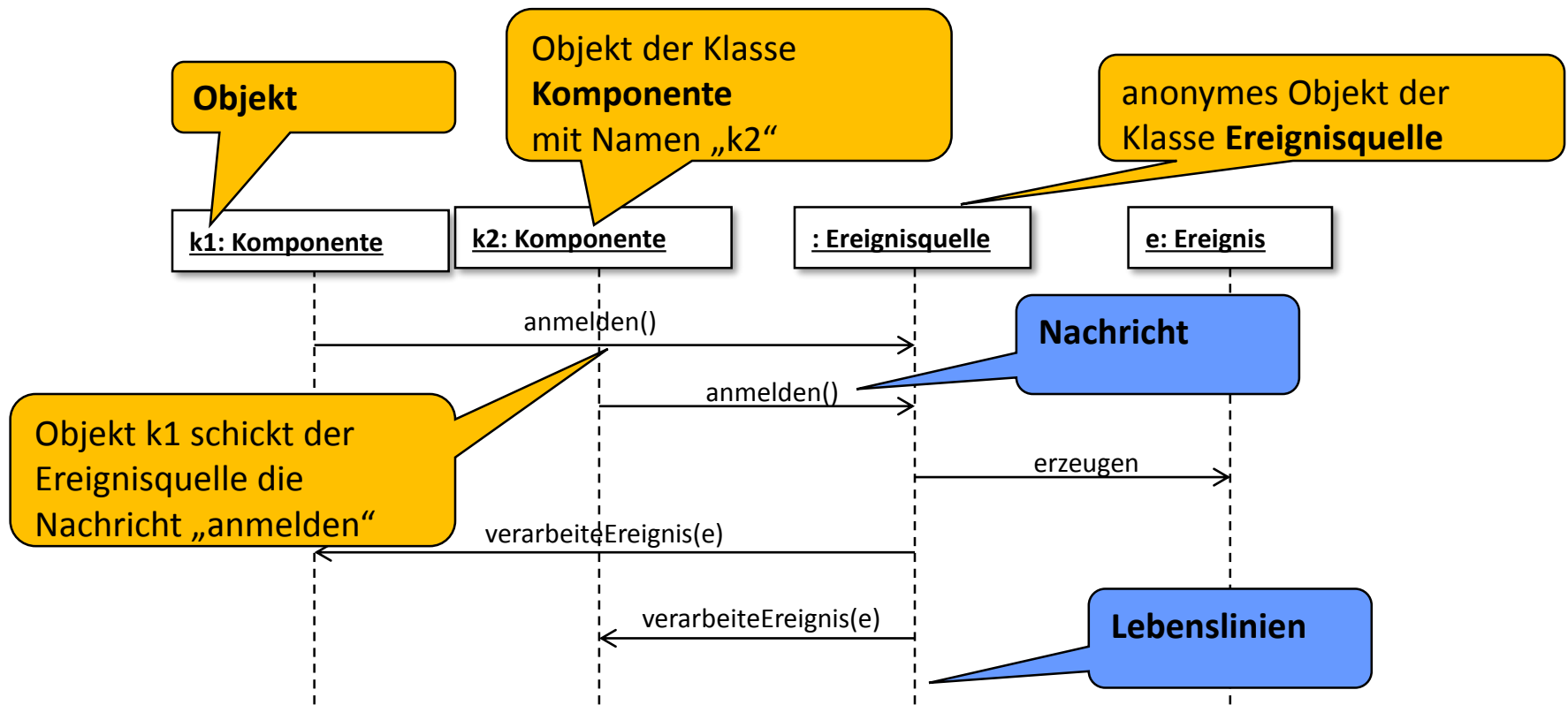


UML Sequenzdiagramm

- Beschreibt komplexe Interaktionen zwischen Objekten in bestimmten Rollen
- Beschreibt die zeitliche Abfolge dieser Interaktionen
- Wird zur Beschreibung der fachlichen Sicht (Laufzeitsicht) verwendet

UML Sequenzdiagramm

Graphische Notation zur Darstellung des Verhalten von Objekten

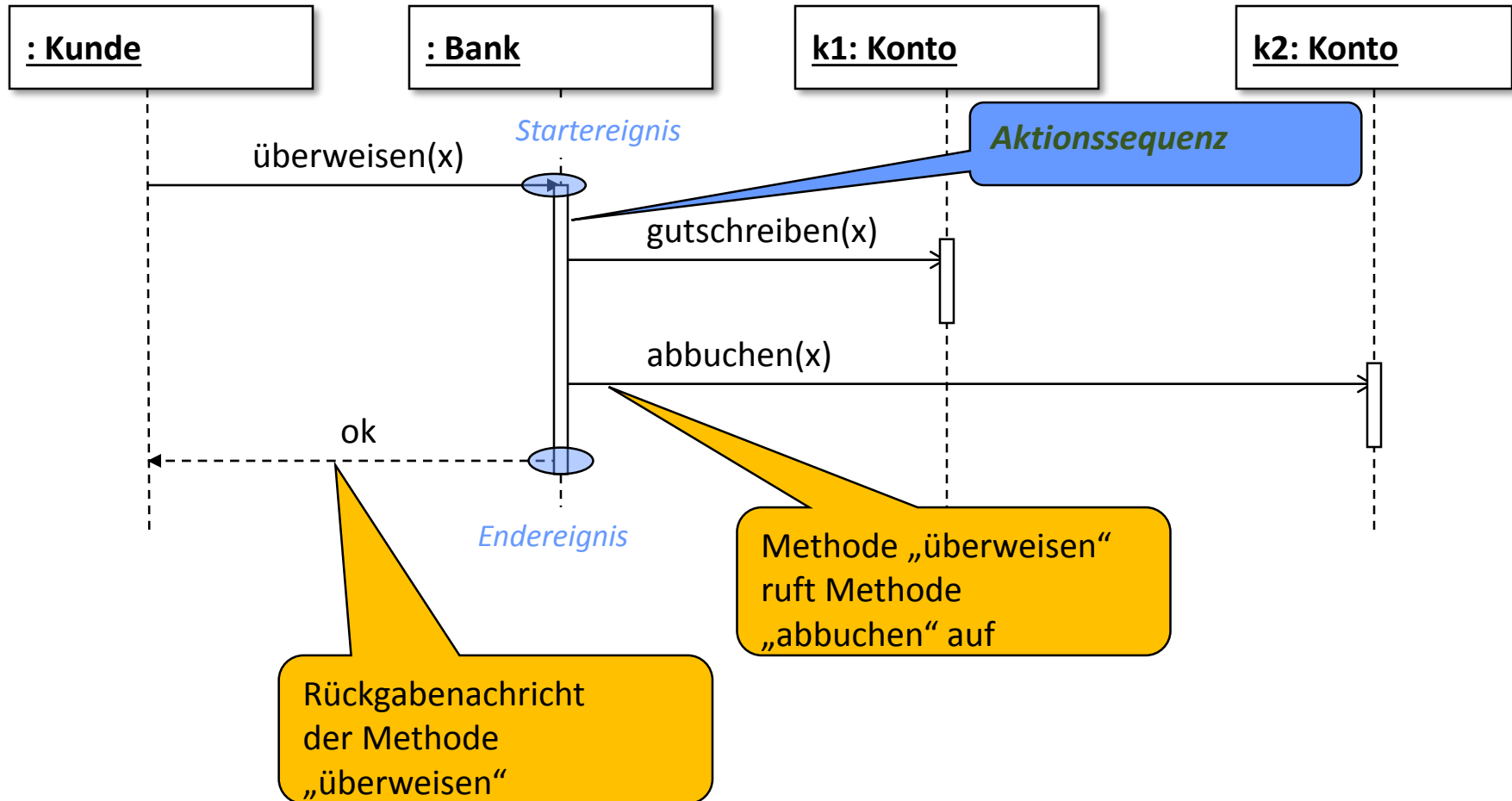


Die Anordnung der Nachrichtenpfeile von oben nach unten stellt den Nachrichtenfluss im System dar

Nachrichten

- Allgemein ist eine **Nachricht eine Informationseinheit**, die von einem Senderobjekt zu einem Empfängerobjekt gesendet wird.
- In unserem Kontext sind Nachrichten
 - **Methodenaufrufe**
 - Z.B. anmelden() oder gib_kontostand()
 - der Pfeil zeigt vom Aufrufer zum aufgerufenen Objekt
 - von Methoden **zurückgegebene Objekte oder Werte**
 - z.B. "1000 €"
 - Übertragung von **Signalen** (Zeitereignisse)

Nachrichten und Methoden



Verwendung von Sequenzdiagrammen

- Wir verwenden Sequenzdiagramme im folgenden, um *beispielhafte* Nachrichtenflüsse darzustellen
 - Eine prinzipielle Art und Weise, wie Objekte miteinander kommunizieren, wird dargestellt
 - Das Diagramm ist keine exakte Repräsentation von Programmabläufen
 - Nicht alle beteiligten Objekte und Nachrichten sind dargestellt
 - Parameter der Nachrichten dürfen fehlen
- Allgemeiner werden Sequenzdiagramme im Systementwurf dazu eingesetzt, Nachrichten und Objektkommunikation beispielhaft und abstrakt darzustellen

2. Dokumentation von Source Code

Code-Konventionen

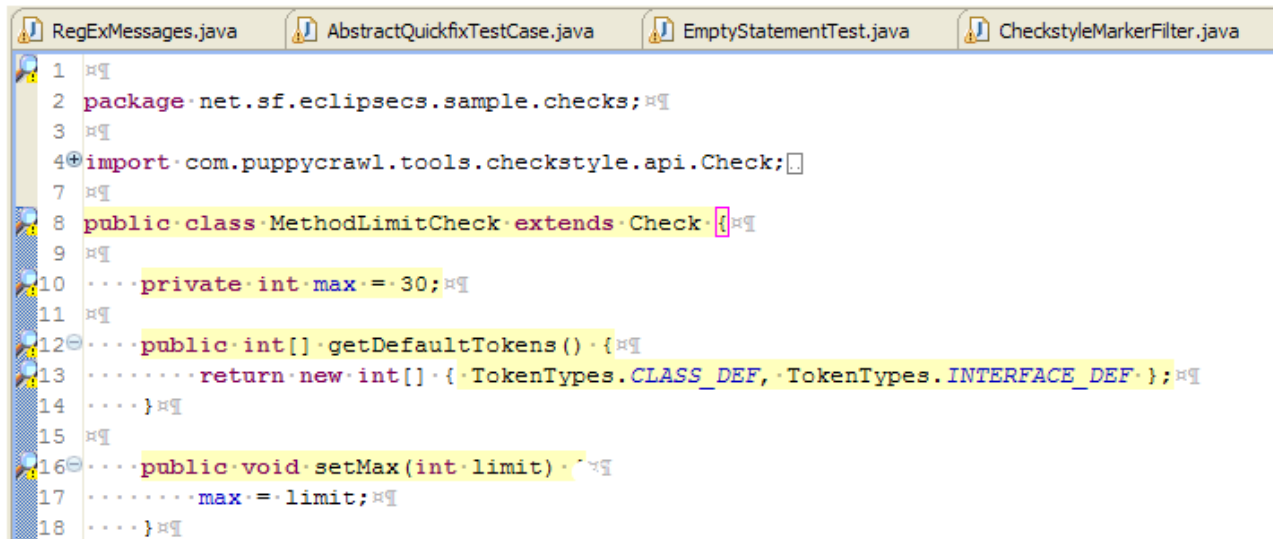
- Satz von Regeln, nach welchen der Quelltext eines Programms erstellt wird
- Diese Regeln beziehen sich auf
 - Einrückung von untergeordneten Programmelementen
 - Positionierung umschließender Syntaxelemente ({, }, begin, end)
 - Einsatz von Kommentaren
 - Namenskonventionen für Symbole
 - Reihenfolge der Deklaration von Symbolen
 - Länge und Umfang von Symbolen
 - Schachtelungstiefe untergeordneter Programmelemente
 - Maximale Zeilenlänge
 - ...
- Regeln darüber hinaus werden für die statische Qualitätssicherung von Code verwendet (in Softwareentwicklung & Projektmanagement VO)

Code-Konventionen cont.

- Hintergrund:
 - 80% der Lebenszeit eines Softwareprodukts entfallen auf die Wartung
 - Code wird von vielen Autoren bearbeitet
 - Einhalten von Standards bei Auslieferung des Source-Codes
- Verbesserung von
 - Lesbarkeit
 - Verständlichkeit
 - Wartbarkeit

Code Konventionen cont.

- Beispiele:
 - „Code Conventions for the Java Programming Language“
 - Google Java Style Guide
 - Hungarian Notation
 - Linux Kernel Coding Style
- Unterstützende Werkzeuge (Beispiel)
 - Checkstyle <http://checkstyle.sourceforge.net/>



```
1 package net.sf.eclipsecs.sample.checks;
2
3
4 import com.puppycrawl.tools.checkstyle.api.Check;
5
6
7
8 public class MethodLimitCheck extends Check {
9
10     private int max = 30;
11
12     public int[] getDefaultTokens() {
13         return new int[] { TokenTypes.CLASS_DEF, TokenTypes.INTERFACE_DEF };
14     }
15
16     public void setMax(int limit) {
17         max = limit;
18     }
19 }
```

Dokumentation mit Javadoc

- Erstellung von (HTML-)Dokumentationsdateien aus Java-Quelltextdateien
 - Steuerung durch spezielle Tags mit Metadaten
- Ziel: Unterstützung des Vertragsprinzips
 - Spezifikation von Schnittstellen
- Verwendung z.B. in der Java-API-Dokumentation
 - <http://docs.oracle.com/javase/8/docs/api/>

Javadoc-Tags

Tag & Parameter	Ausgabe	Verwendung in
<code>@author name</code>	Beschreibt den Autor.	Klasse, Interface
<code>@version version</code>	Erzeugt einen Versionseintrag. Maximal einmal pro Klasse oder Interface.	Klasse, Interface
<code>@see reference,</code> <code>{@link reference}</code>	Erzeugt einen Link auf ein anderes Element der Dokumentation.	Klasse, Interface, Instanzvariable, Methode
<code>@param name description</code>	Parameterbeschreibung einer Methode.	Methode
<code>@return description</code>	Beschreibung des Returnwerts einer Methode.	Methode
<code>@throws, @exception</code> <code>classname</code> <code>description</code>	Beschreibung einer Exception, die von dieser Methode geworfen werden kann.	Methode
<code>@deprecated description</code>	Beschreibung einer veralteten Methode, die nicht mehr verwendet werden sollte.	Methode
<code>{@code ...}</code>	Buchstabengetreue Formatierung mit dem Quelltextzeichensatz.	Klasse, Interface, Instanzvariable, Methode

Javadoc Verwendung

```
/**
 * A bank account is used to store the current balance and limit of a client.
 * Its main purpose is to manage transactions and it guarantees a properly synchronized
 * state of balance.
 * @author Ruth Breu
 * @version 1.0
 */
public class BankAccount {

    private int balance;
    private int limit;

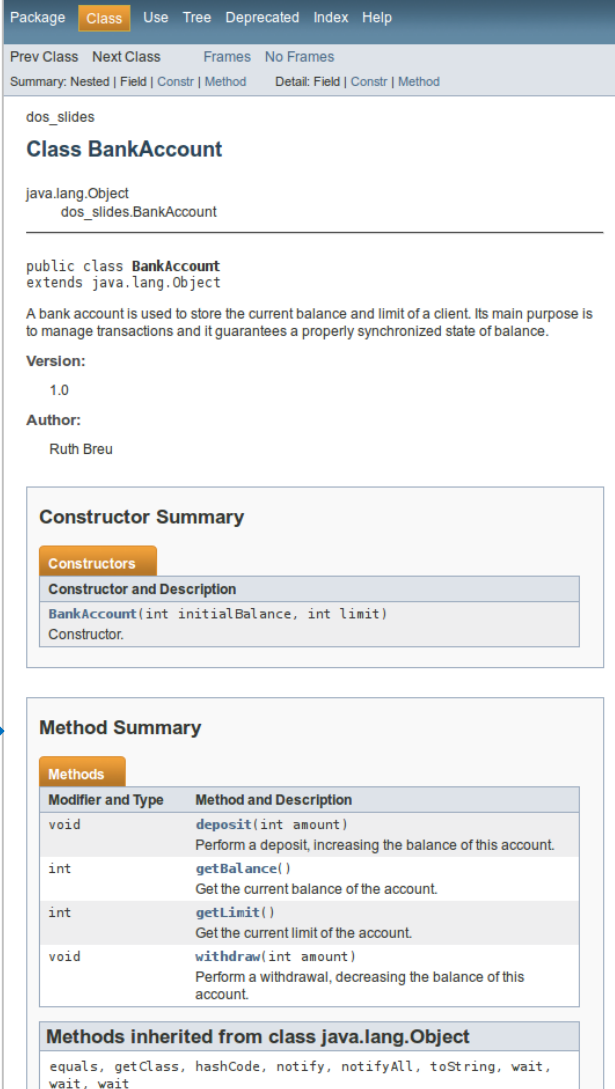
    /**
     * Constructor.
     * @param initialBalance
     *     The balance of a client when this account is created/opened. Must not be negative,
     *     otherwise an {@code IllegalArgumentException} is thrown.
     * @param limit
     *     The limit to be used for this account. Must not be greater than zero, otherwise an
     *     {@code IllegalArgumentException} is thrown.
     */
    public BankAccount(int initialBalance, int limit) {
        if(initialBalance < 0) throw new IllegalArgumentException("Positiver Betrag erforderlich");
        if(limit > 0) throw new IllegalArgumentException("Negativer Betrag erforderlich");

        this.balance = initialBalance;
        this.limit = limit;
    }

    /**
     * Get the current balance of the account.
     * @return The balance.
     */
    public int getBalance() {
        return this.balance;
    }

    // ...
}
```

javadoc command



Package **Class** Use Tree Deprecated Index Help

Prev Class Next Class Frames No Frames

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

dos_slides

Class BankAccount

java.lang.Object
dos_slides.BankAccount

public class **BankAccount**
extends java.lang.Object

A bank account is used to store the current balance and limit of a client. Its main purpose is to manage transactions and it guarantees a properly synchronized state of balance.

Version:
1.0

Author:
Ruth Breu

Constructor Summary

Constructors

Constructor and Description
BankAccount (int initialBalance, int limit) Constructor.

Method Summary

Methods

Modifier and Type	Method and Description
void	deposit (int amount) Perform a deposit, increasing the balance of this account.
int	getBalance () Get the current balance of the account.
int	getLimit () Get the current limit of the account.
void	withdraw (int amount) Perform a withdrawal, decreasing the balance of this account.

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

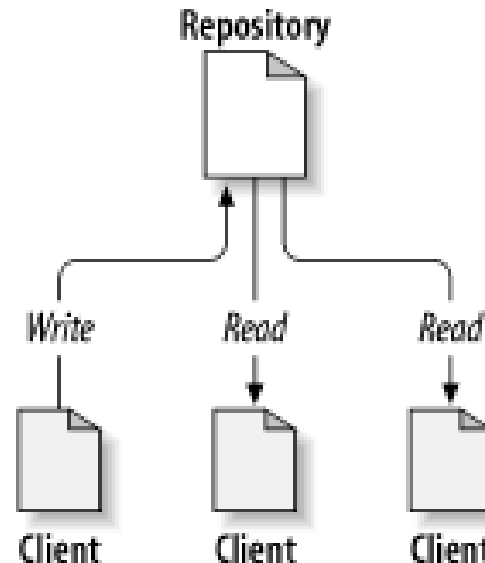
3. Versionskontrollsysteme

Ziele der Versionskontrolle

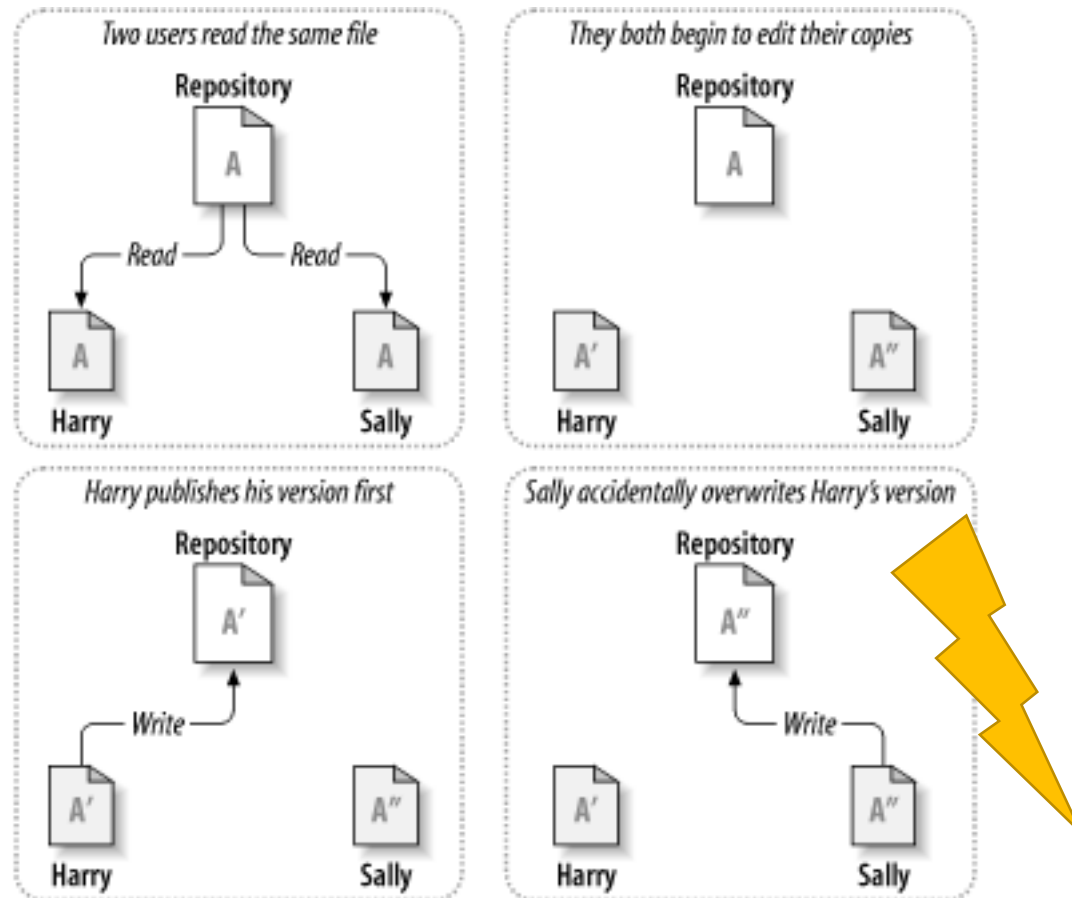
- **Nachvollziehbarkeit**
 - Wie und wann hat sich ein Fehler ins System eingeschlichen?
- **EntwicklerInnenkoordinierung**
 - Wie können mehrere EntwicklerInnen gleichzeitig an der Codebasis arbeiten?
- **Releasemanagement & Archivierung**
 - Wie markiere einen bestimmten Zustand als Version?
- **Wiederherstellbarkeit**
 - Wie bekomme ich mein Filesystem in einen alten bugfreien Zustand?

Das Code-Repository

- Zentraler Speicher der Systemdaten
 - Dateisystem in Baumstruktur (Ordner und Dateien)
- Beliebig viele Clients können lesend und schreibend auf das Repository zugreifen



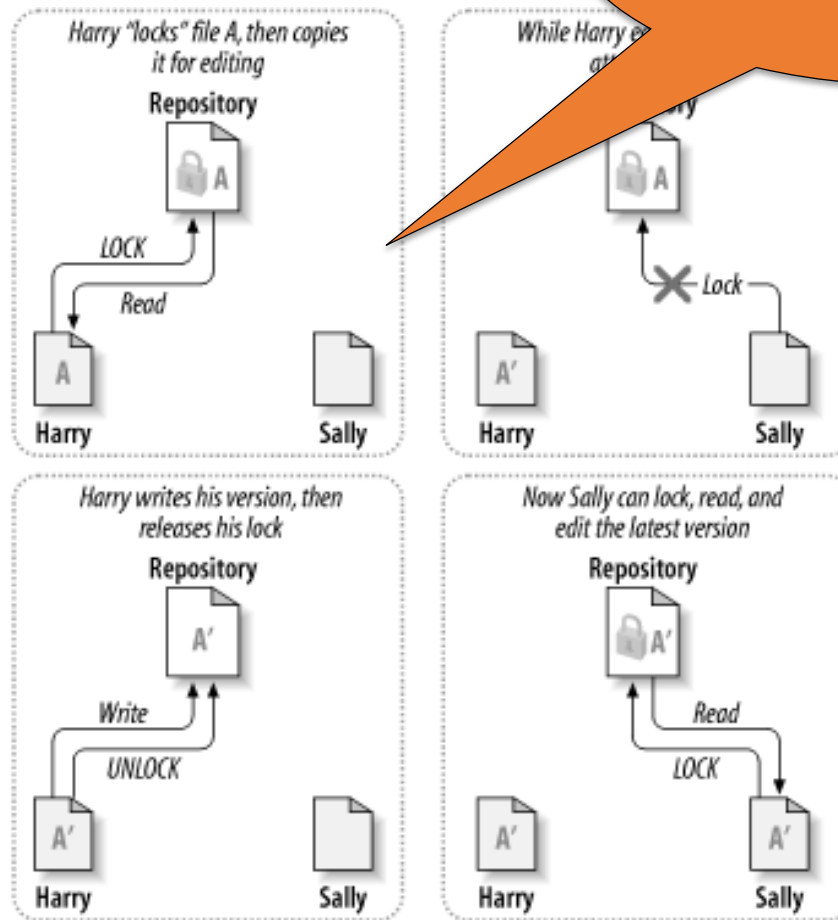
Das fundamentale Problem eines Filesystems



Lösungsansatz 1

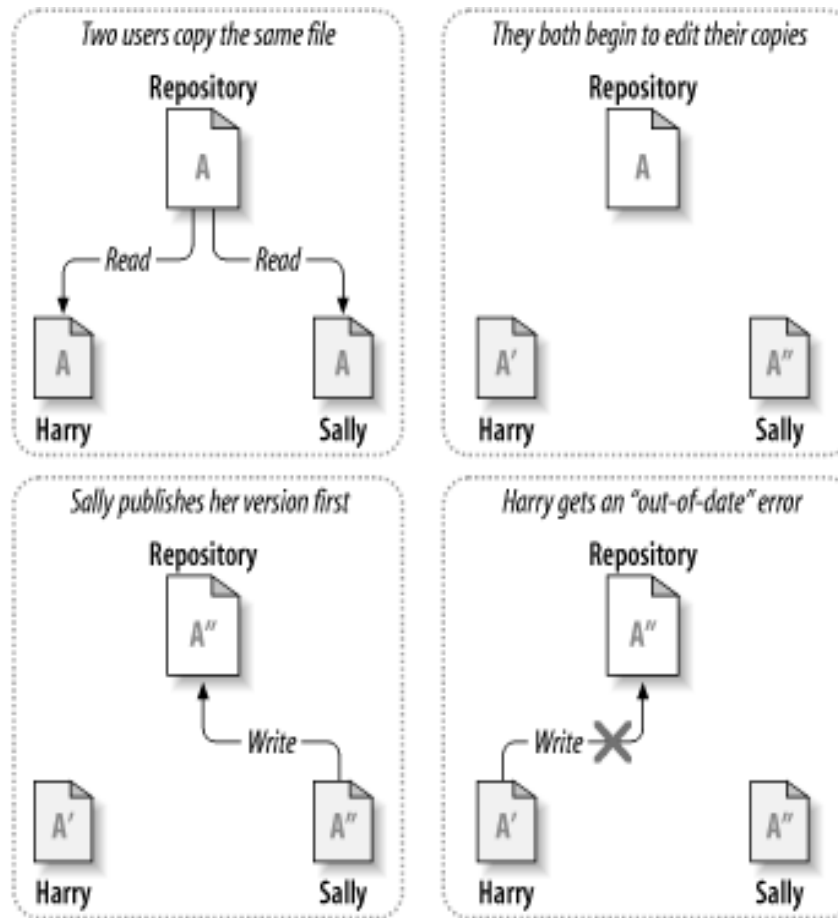
- **Pessimistisches Locking**

- Umständlich
- Kein paralleles Arbeiten möglich!



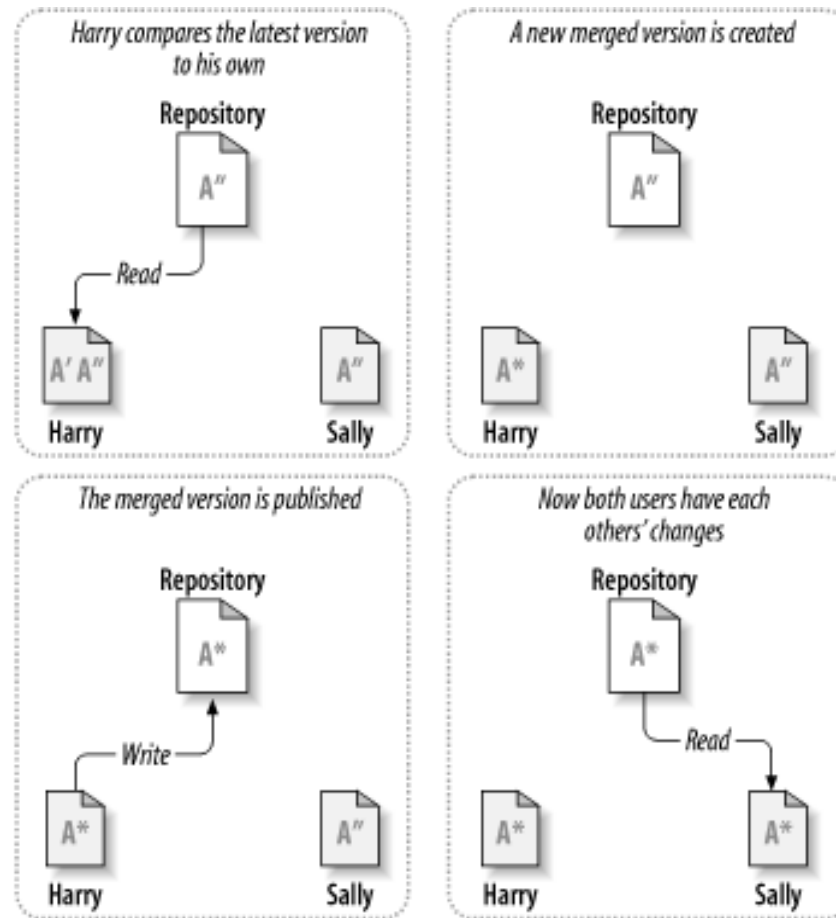
Lösungsansatz 2

- **Optimistisches Locking**



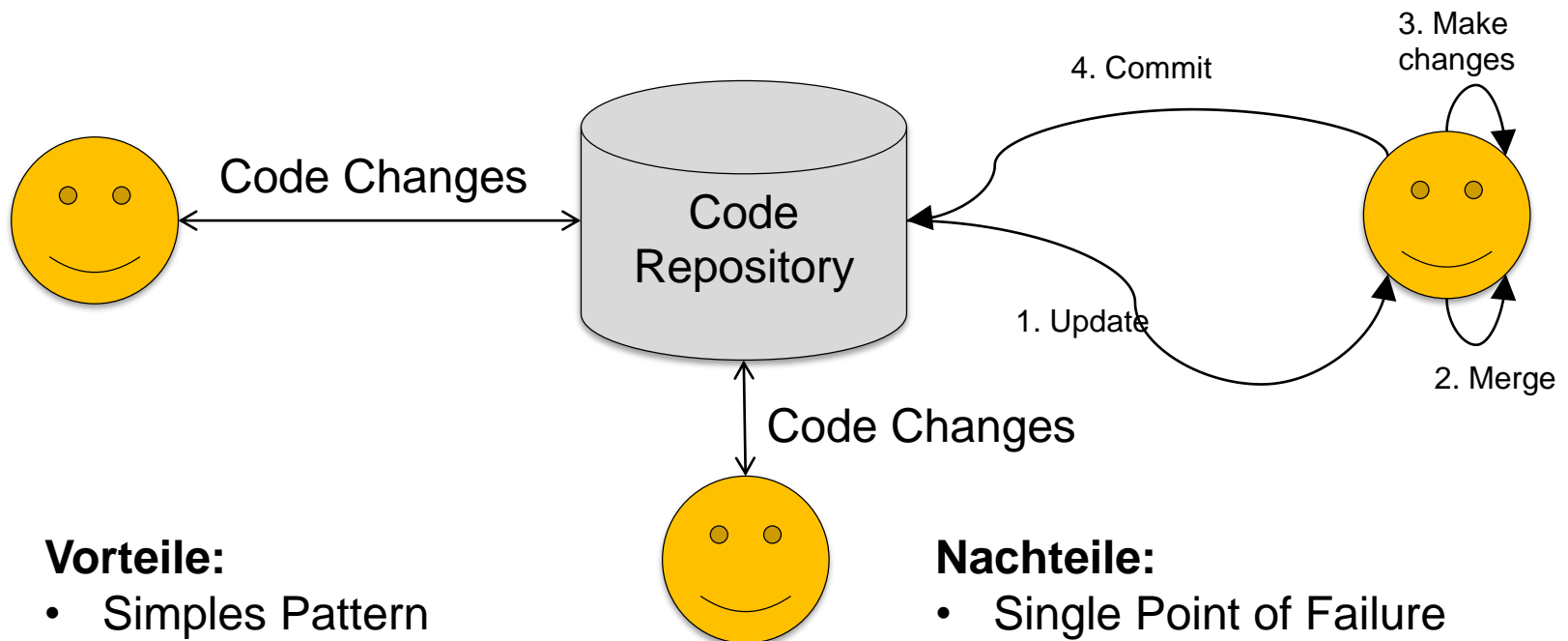
Lösungsansatz 2 – cont.

- **Optimistisches Locking – cont.**



Arten von Versionskontrollsysteme

- **Zentrales Versionskontrollsystem** (z.B. CVS, SVN,...)



Vorteile:

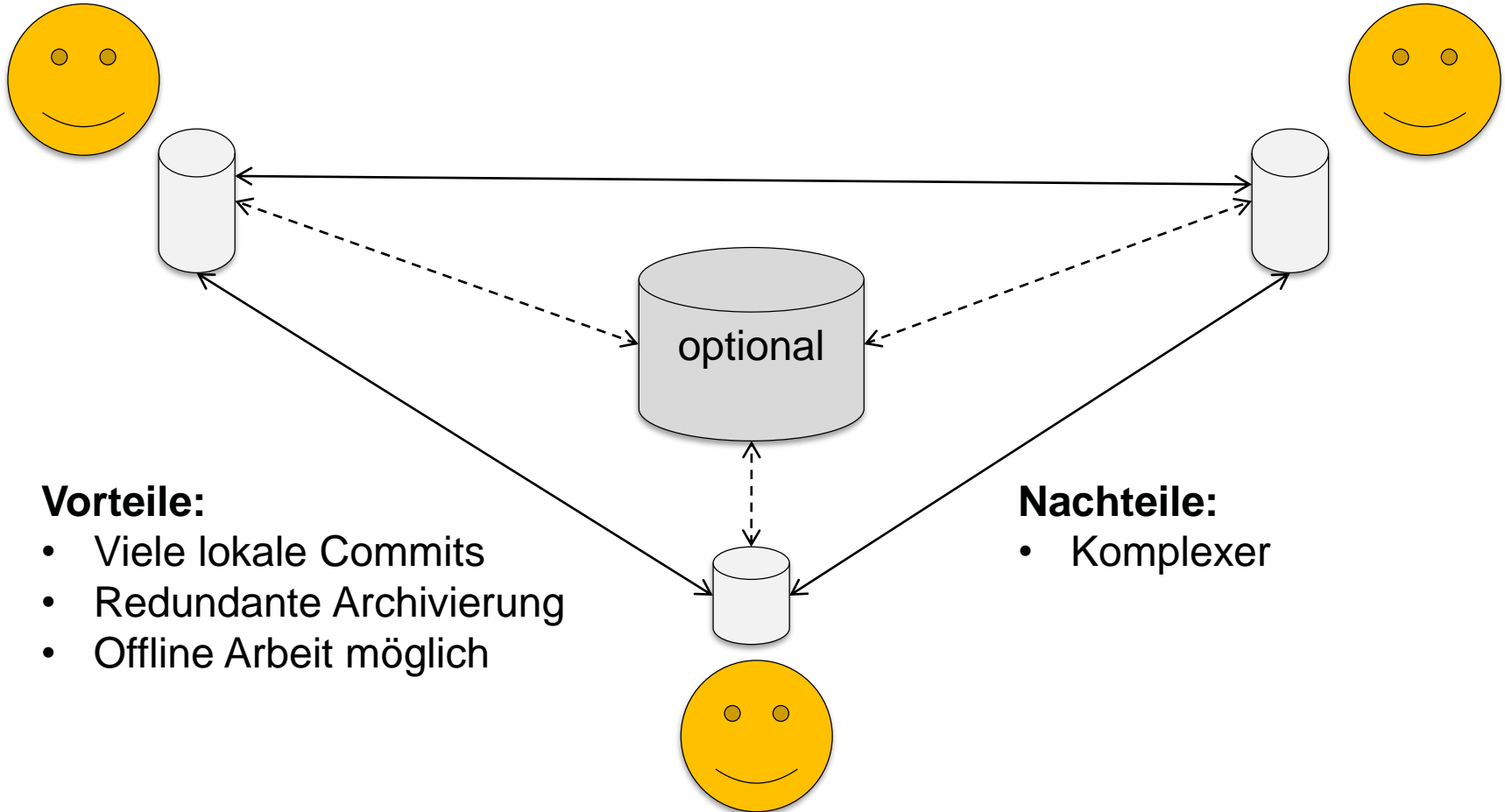
- Simple Pattern

Nachteile:

- Single Point of Failure
- Repository muss online sein
- Seltene Commits

Arten von Versionskontrollsysteme cont.

- **Dezentrales Versionskontrollsystem (z.B. GIT)**



Vorteile:

- Viele lokale Commits
- Redundante Archivierung
- Offline Arbeit möglich

Nachteile:

- Komplexer

Versionskontrollsystem Terminologie

- **Commit Operation**
 - Fügt Änderungen zum Repository hinzu
- **Branch**
 - Entwicklungszweig
- **Merge Operation**
 - Vereint verschiedene Versionen
- **Tag**
 - Markiert einen bestimmten Status (z.B. Release Management)

Versionskontrollsystem Tools

- **Features**

- Interaktion mit Repositories (Commit, Branch, Merge,...)
- Vergleich von Historie
- Rollback
- Sprung zwischen Branches
- Authentifizierung

- **Tool Arten**

- Command-line Tools
- Integration mit IDEs
- Desktop Clients
- Web-Apps mit GitHub, GitLab, ...
 - Bieten Projektmanagement Funktionen

Command Line GIT

GIT(1)

Git Manual

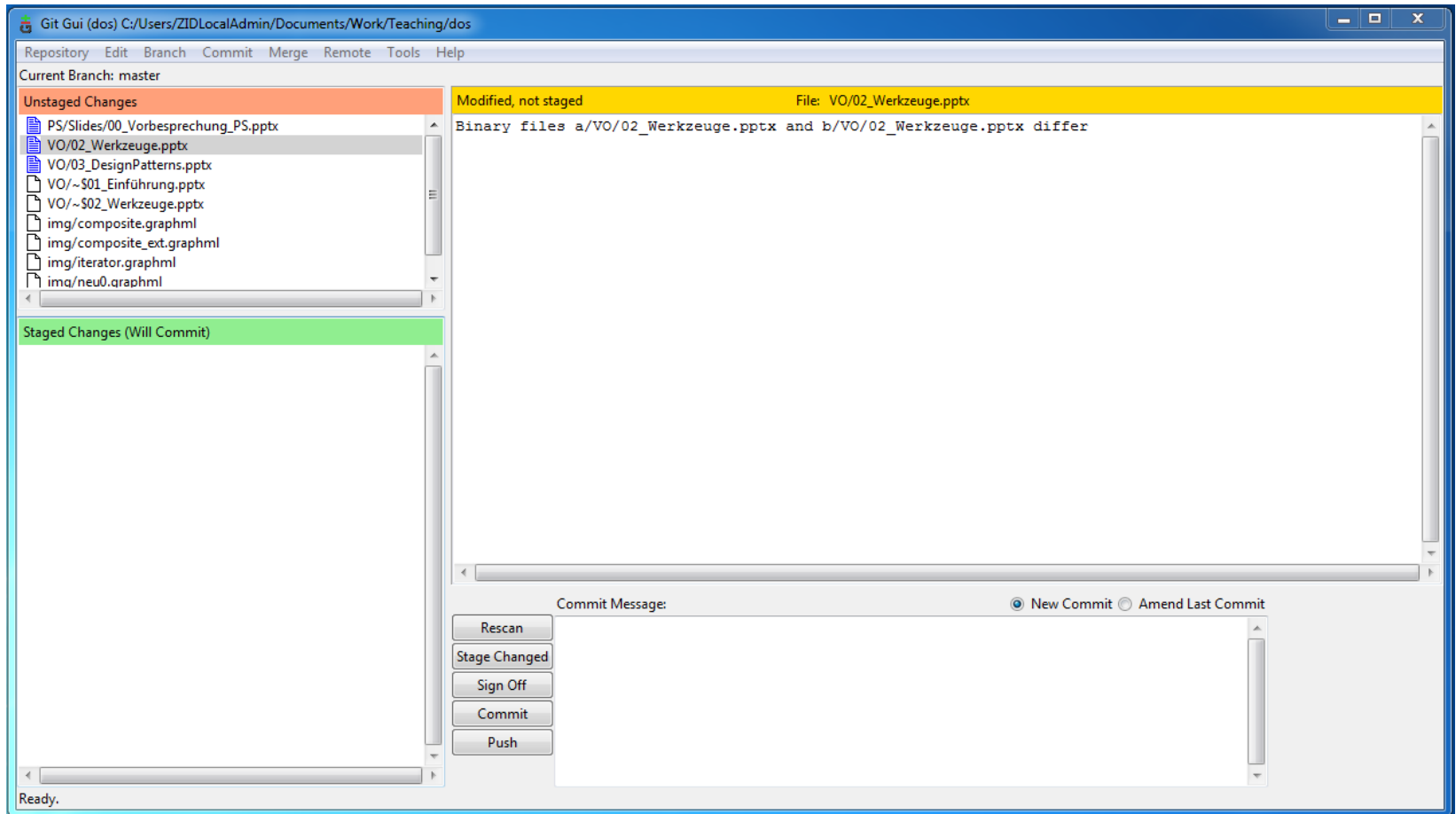
NAME

git - the stupid content tracker

SYNOPSIS

```
git [--version] [--help] [-C <path>] [-c <name>=<value>]  
      [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]  
      [-p|--paginate|--no-pager] [--no-replace-objects] [--bare]  
      [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]  
      [--super-prefix=<path>]  
      <command> [<args>]
```

GIT-GUI



GitLab Universität Innsbruck

GITLAB @ UIBK



Sign in with your University of Innsbruck account.

New to Gitlab at UIBK ? *Neu bei Gitlab bei UIBK ?*

If you are a member of the University and already have an account but are new to Gitlab, you must first [opt in](#). Click on that link to enable your account for this service.

- ▶ (Klicken Sie hier, die deutsche Version zu sehen)

No UIBK account ? *Kein UIBK-Konto ?*

If you do not have an account, ask a faculty member or project coordinator to request one for you.

- ▶ (Klicken Sie hier, die deutsche Version zu sehen)

LDAP Server der Universität Innsbruck

LDAP Server der Universität Innsbruck Username

Password

☐ Remember me

Sign in

<https://git.uibk.ac.at/>

Historie Populärer Versionskontrollsysteme

Lokale Versionskontrollsysteme, z.B. SCCP (1972)

CVS - Concurrent Versions System (1989 – 2008)

- Wurde in der Betriebssystementwicklung eingesetzt
- Entwicklung eingestellt

SVN – Subversion (2000 - heute)

- Nachfolger von CVS und sollte Fehler beheben

GIT (2005 - heute)

- Eigenentwicklung von Linux Erfinder Linus Torvald
- Wird zur Linux Kernel Entwicklung eingesetzt

CVS - Concurrent Versions System



- Zentrales Repository
- Delta-Kodierung
 - Differenzen zwischen den Dateiversionen werden gespeichert
- Nur Änderungen der Dateien werden gespeichert
 - D.h. Änderungen können nicht auf einen bestimmten Commit zurückgeführt werden
- Umbenennungen und Dateibewegungen werden nicht versioniert
- Keine „Gesamtversionsnummer“
- Umgang mit Verzeichnissen ist sehr kompliziert
- Link: <https://savannah.nongnu.org/projects/cvs>

SVN - Subversion



- Als Rewrite des Concurrent Versions System (CVS) gedacht
 - Lässt aber Features aus, z.B. Tagging
 - Braucht aber mehr Speicher
- Gesamtversionsnummer „Revision“
- Keine spezielle Semantic für Tagging/Branching
 - Nur durch implizite Filestruktursemantik
 - REPORURL/projectX/trunk
 - REPORURL/projectX/ranches
 - REPORURL/projectX/tags
- Link: <https://subversion.apache.org/>

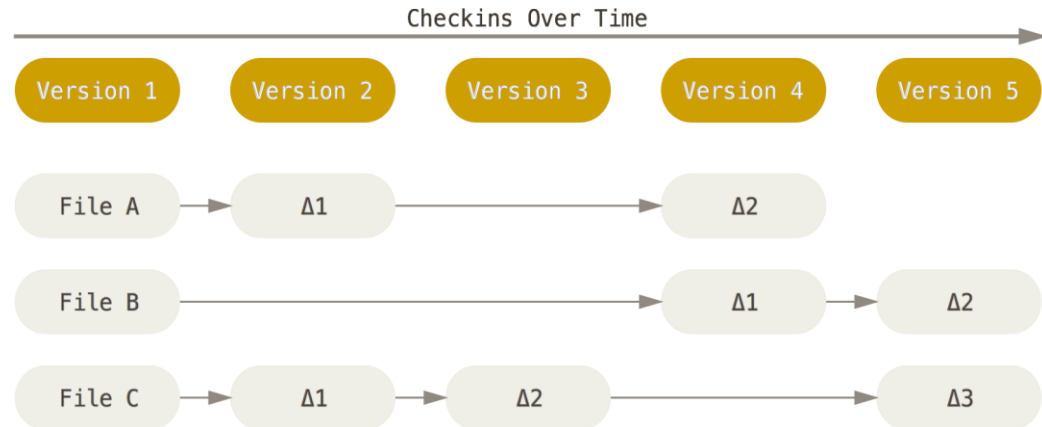
GIT



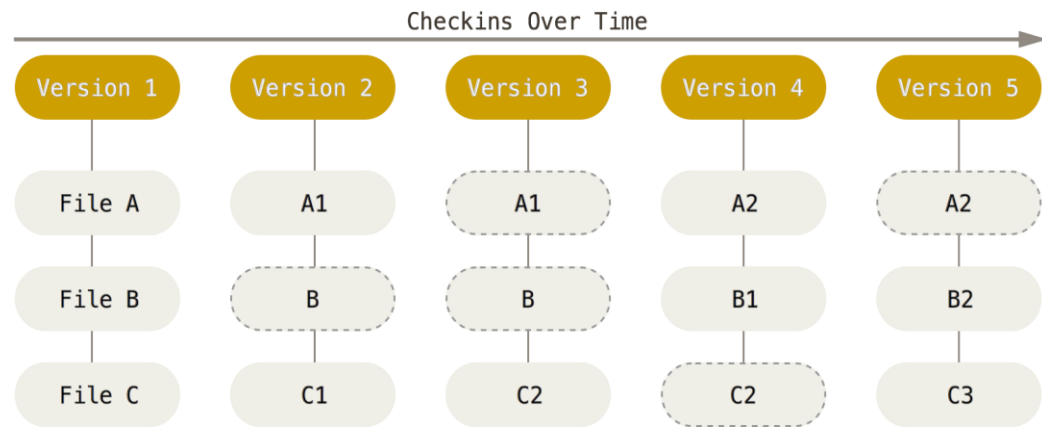
- Verteiltes VCS
- Sehr flexible Workflows
 - Kann SVN/CVS Workflow emulieren
- Lokale Commits
 - Performance Vorteile
 - Offline Arbeiten möglich
- Commits identifiziert via Hash
 - Nötig da kein zentraler Versionszähler möglich
- Extrem populär durch GitHub: „Social Coding“
- Link: <https://git-scm.com/>

Subversion vs. GIT

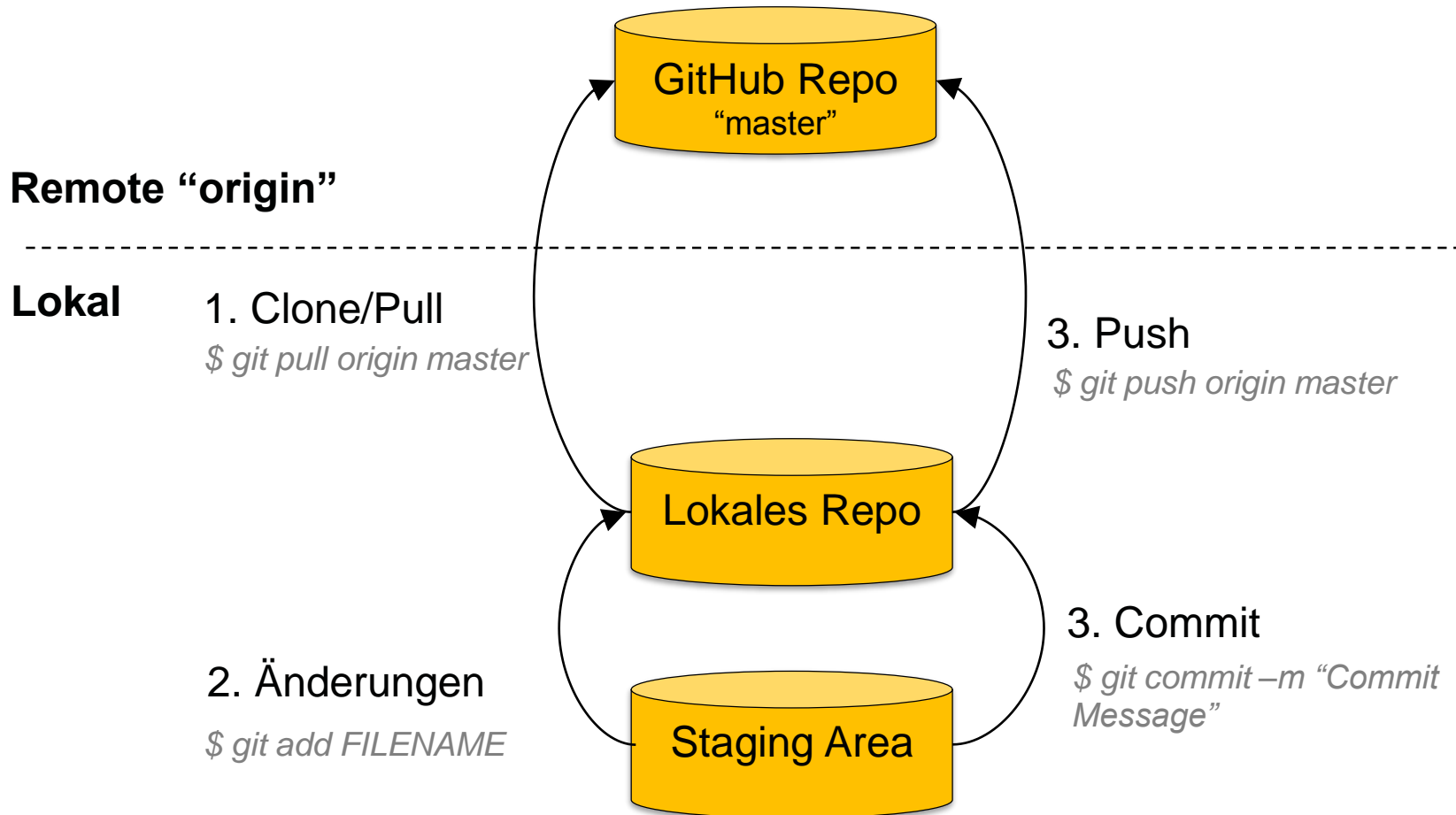
CVS/SVN:
Änderungen auf
File-basis gespeichert



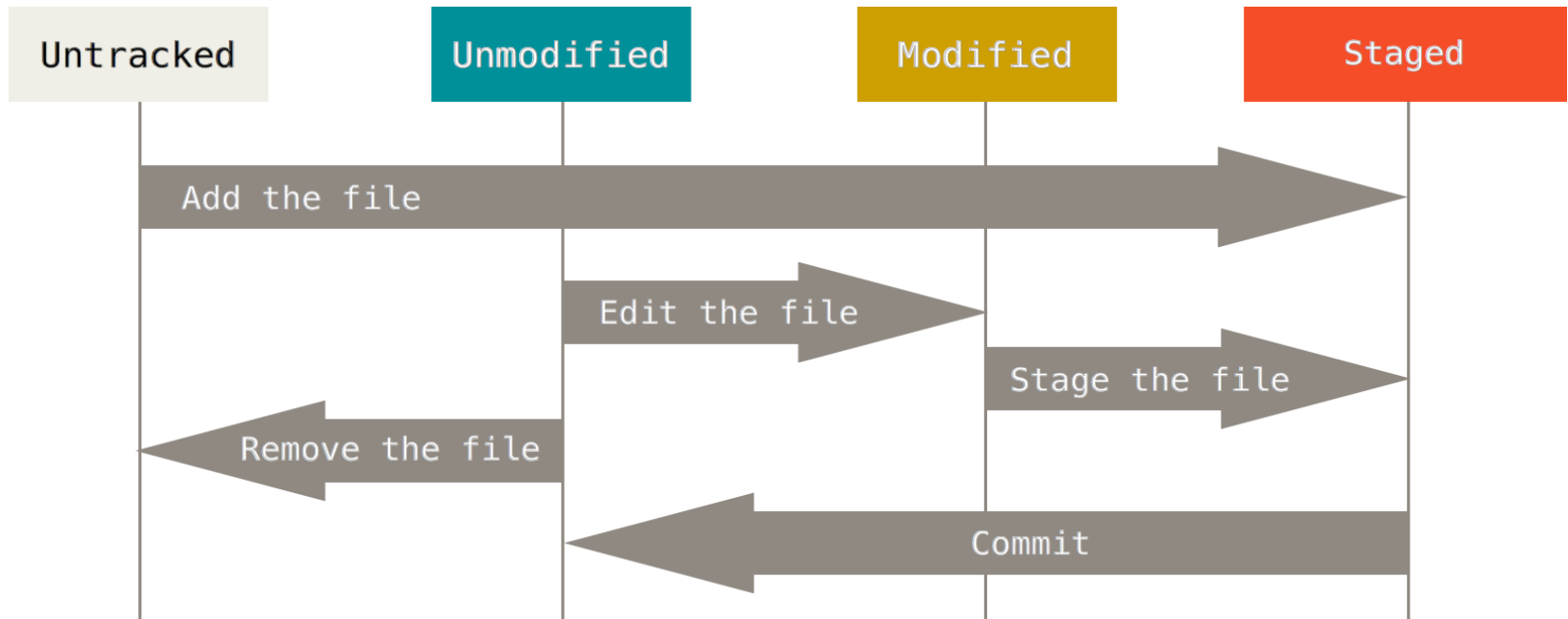
Git:
Snapshots vom gesamten
Filesystem pro Version



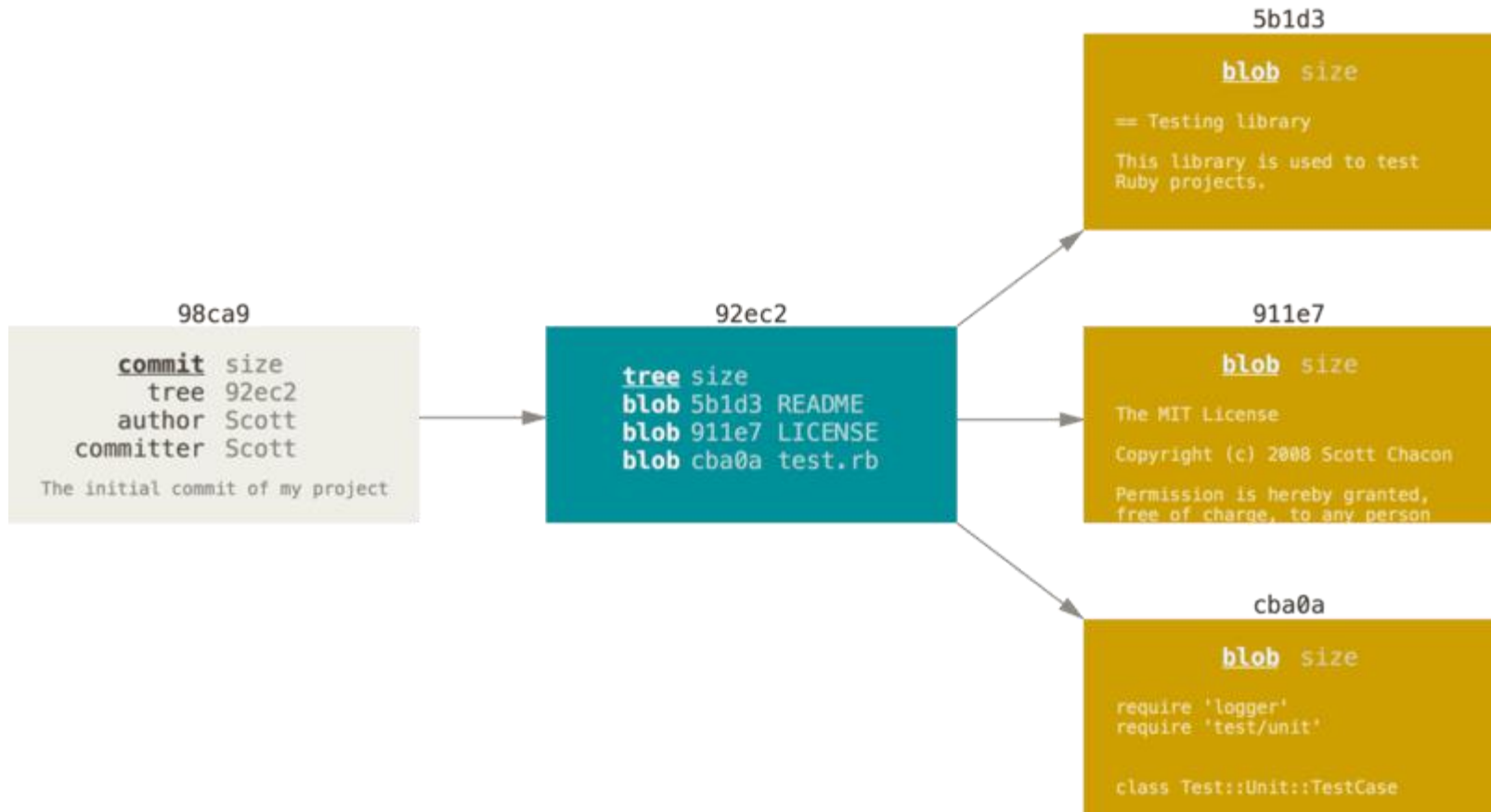
GIT Zyklus



GIT Änderungszyklus



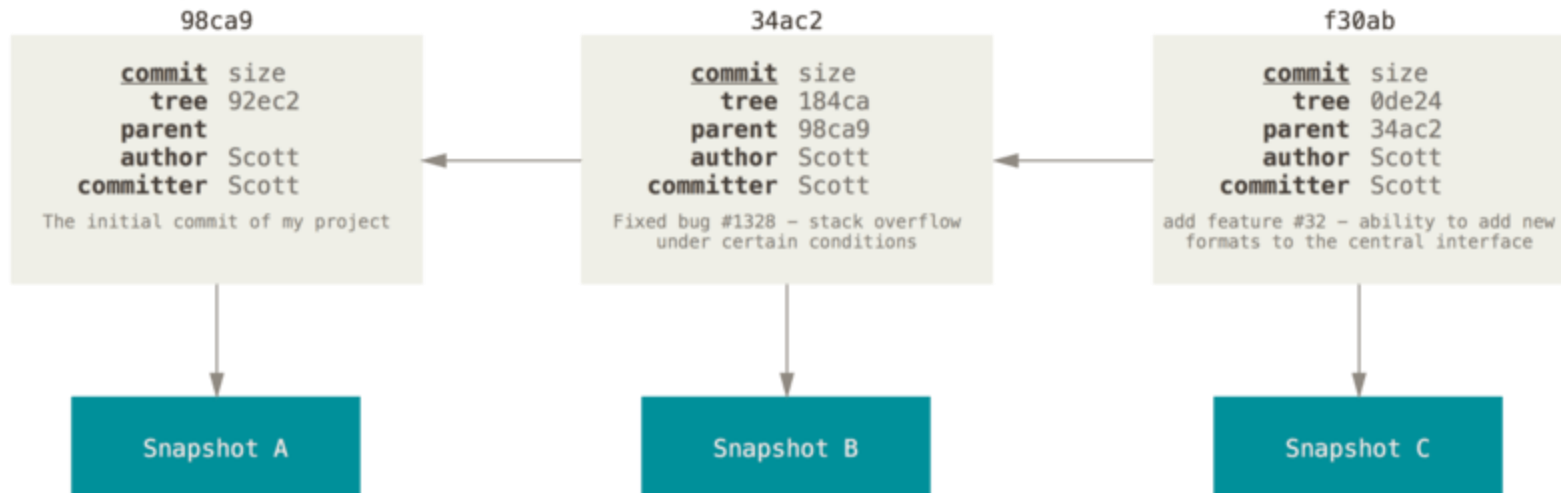
GIT Internals: GIT Tree



```
$ git add README test.rb LICENSE
$ git commit -m 'The initial commit of my project'
```

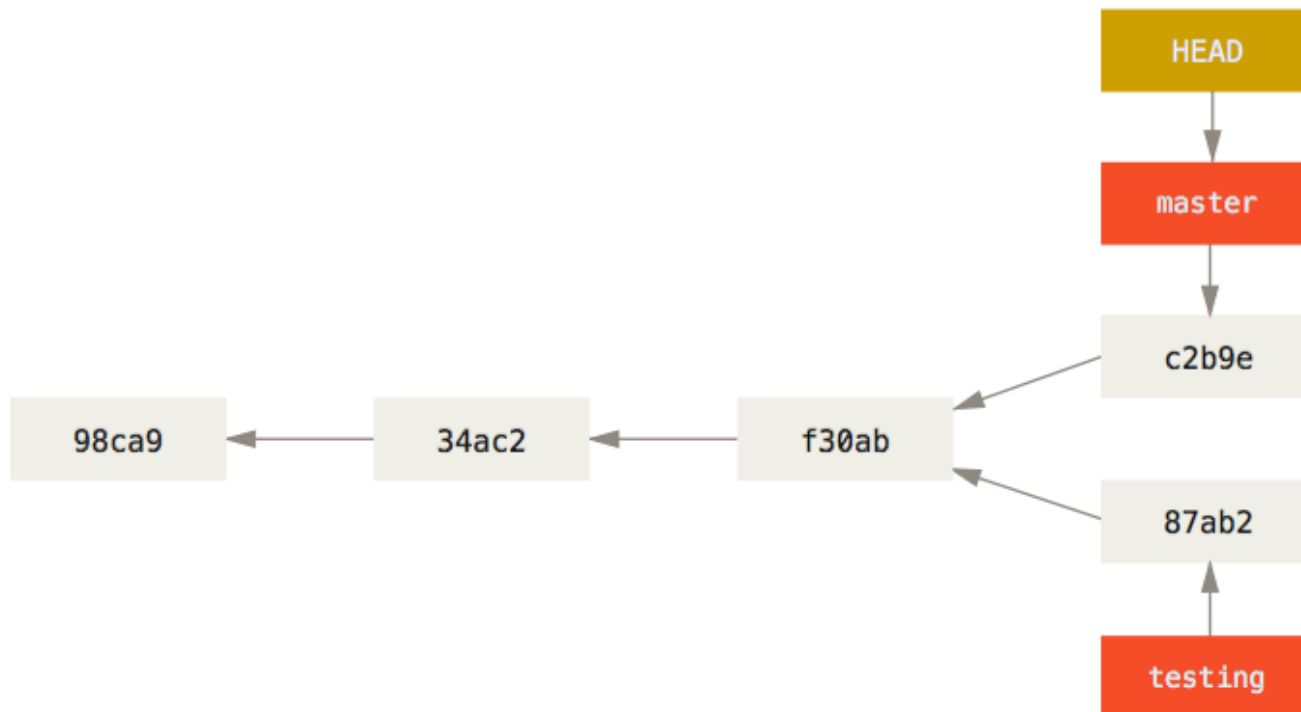
GIT Internals: Commits

Commit Sha-1 Hash

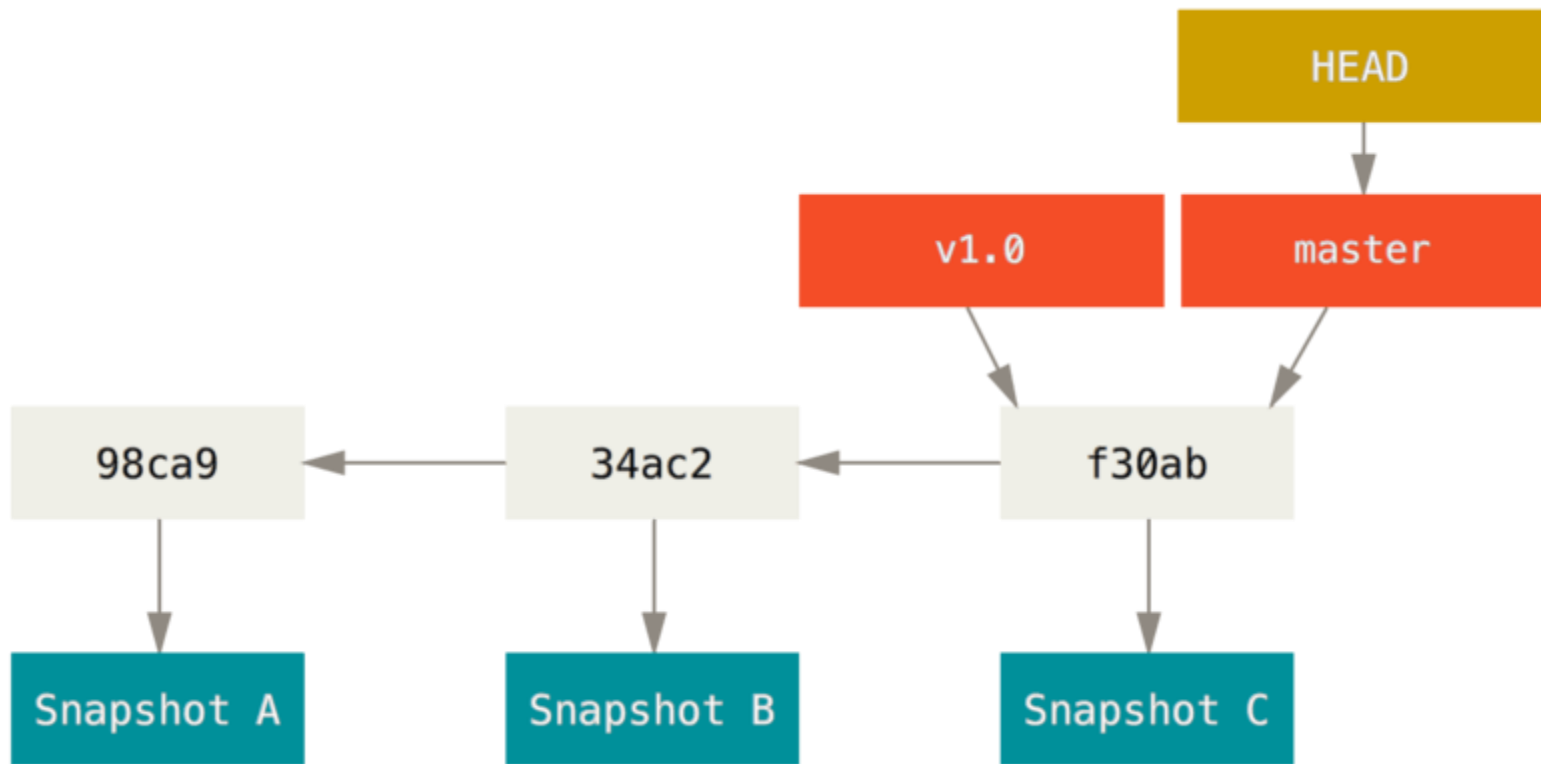


GIT Internals: GIT Branches

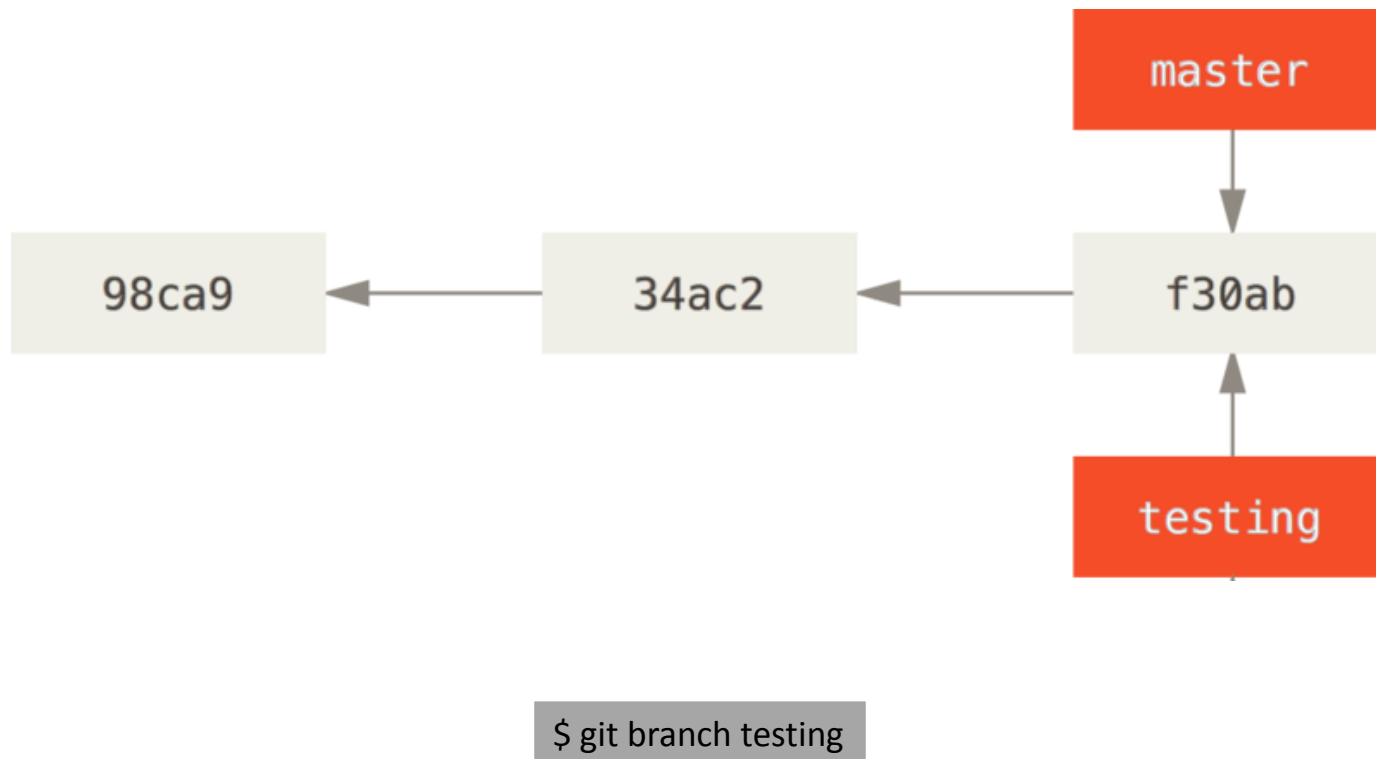
- Ein Branch ist ein leichtgewichtiger Pointer auf einen Commit
- HEAD bezeichnet den Commit auf dem gearbeitet wird



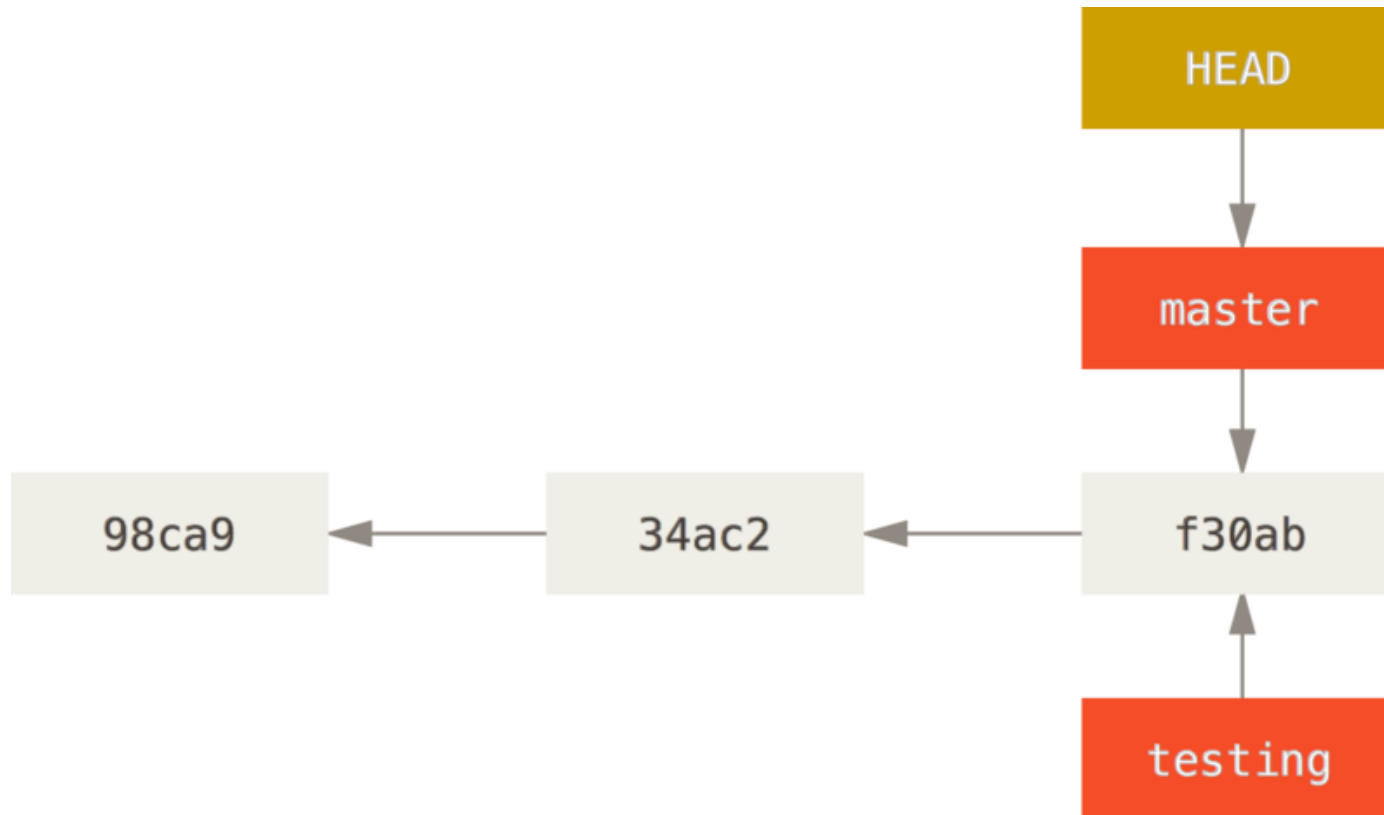
GIT Internals: Ein „Branch“ und seine „Commit“-Historie



GIT Internals: Branching



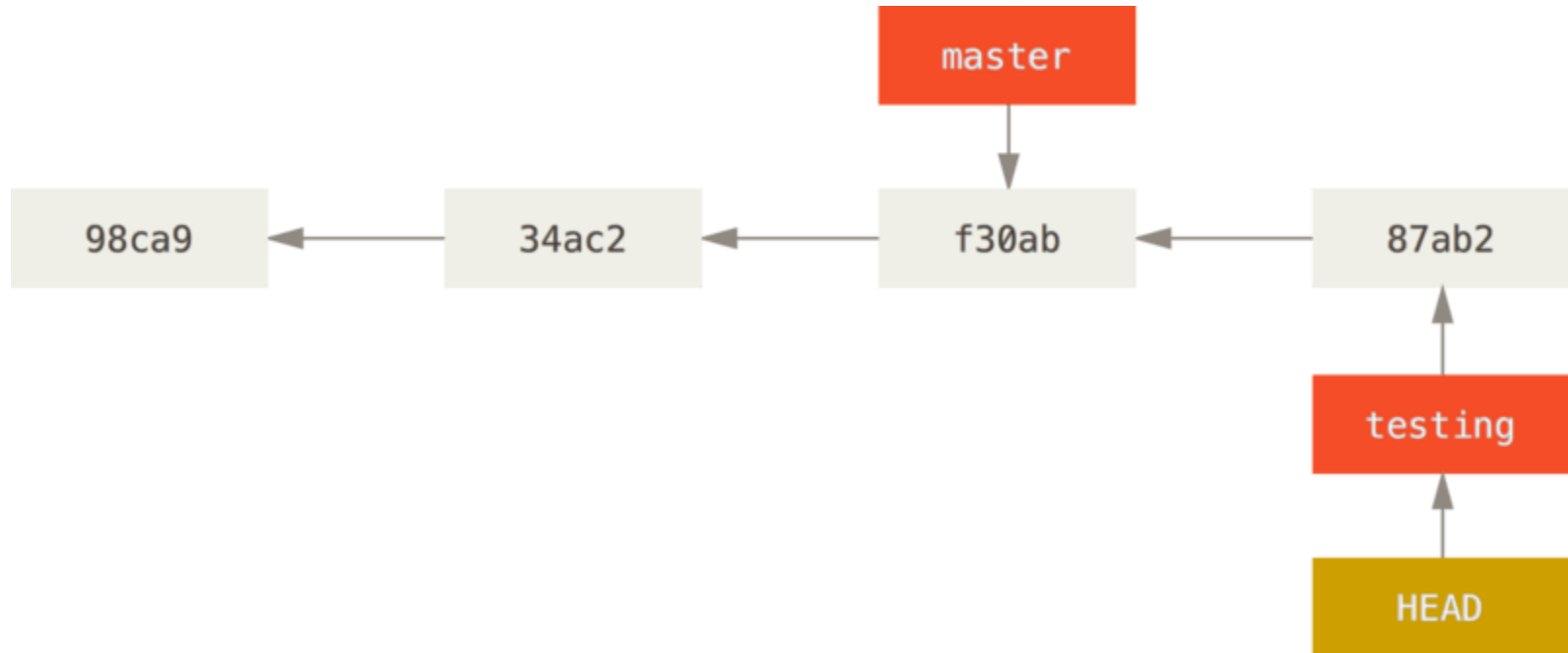
GIT Internals: Head pointing to a branch



GIT Internals: Switching Branches

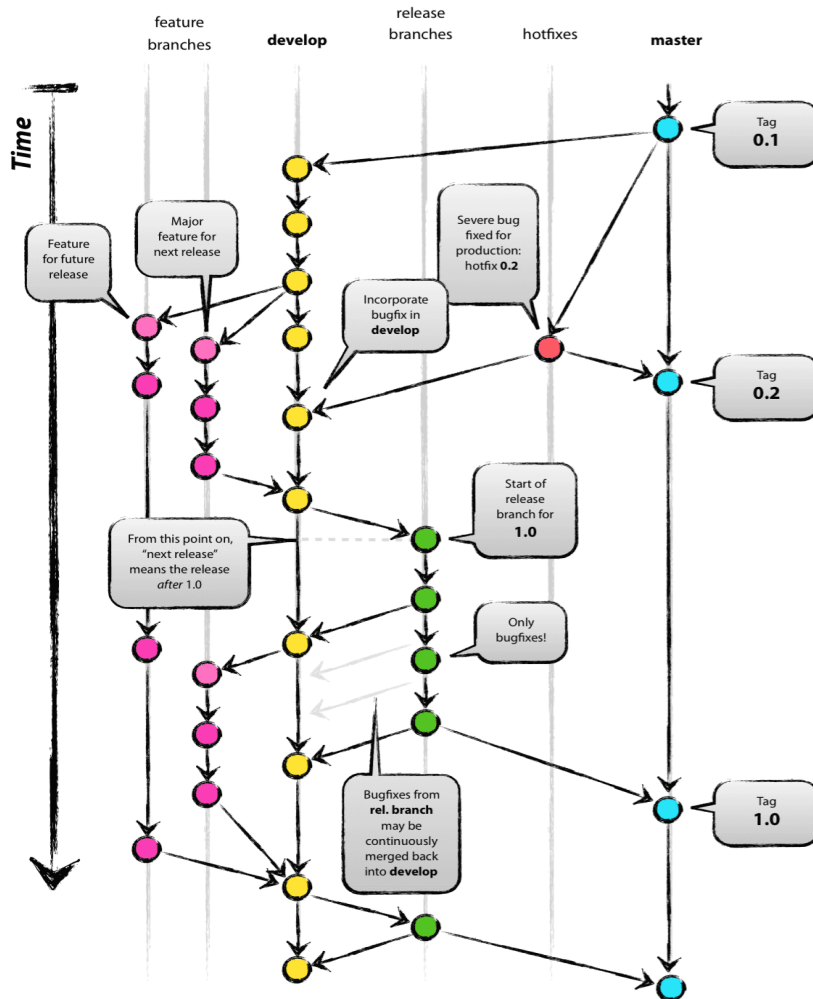


GIT Internals: Branches and Commits



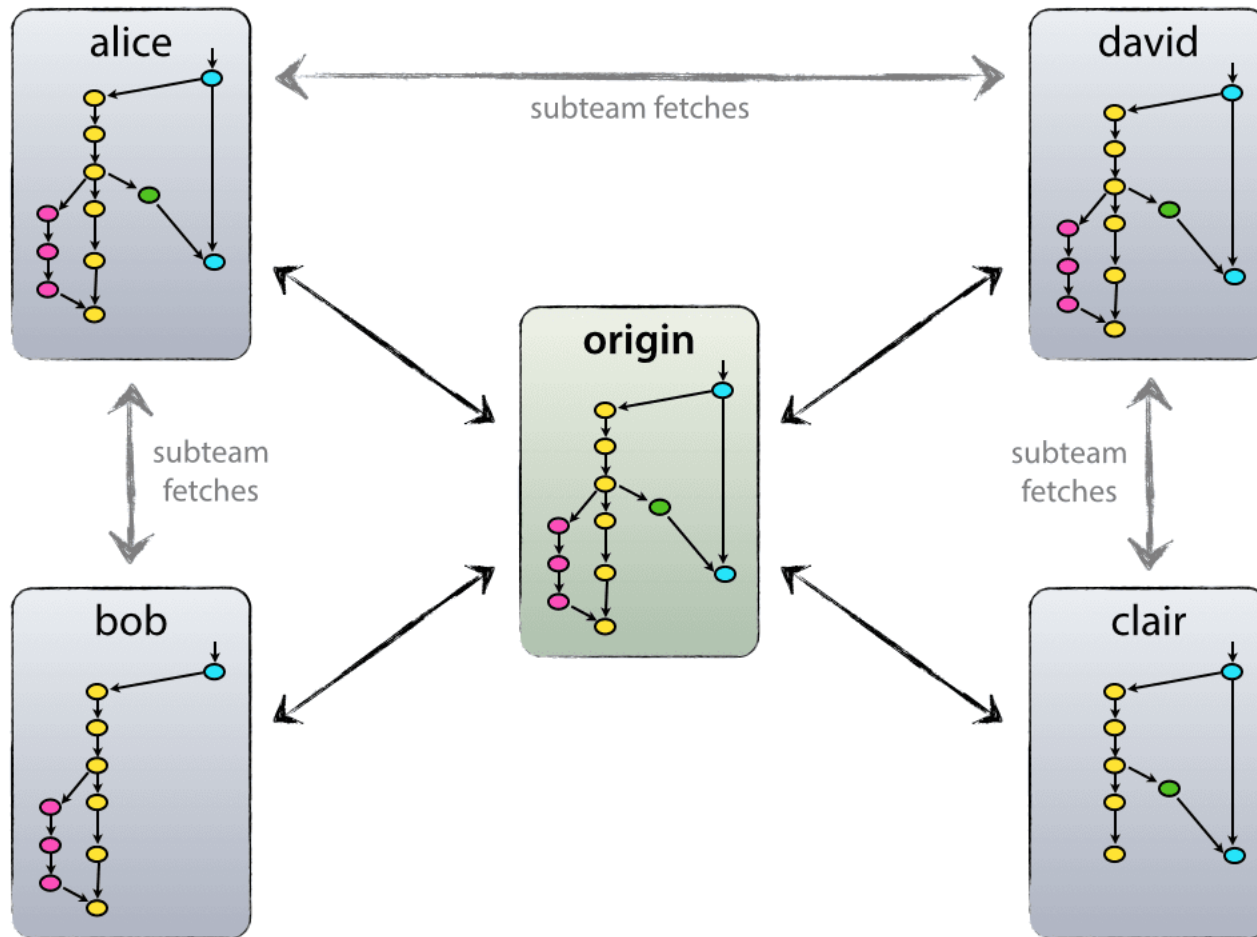
```
$ vim test.rb  
$ git commit -a -m 'made a change'
```

GIT Branching in der Praxis

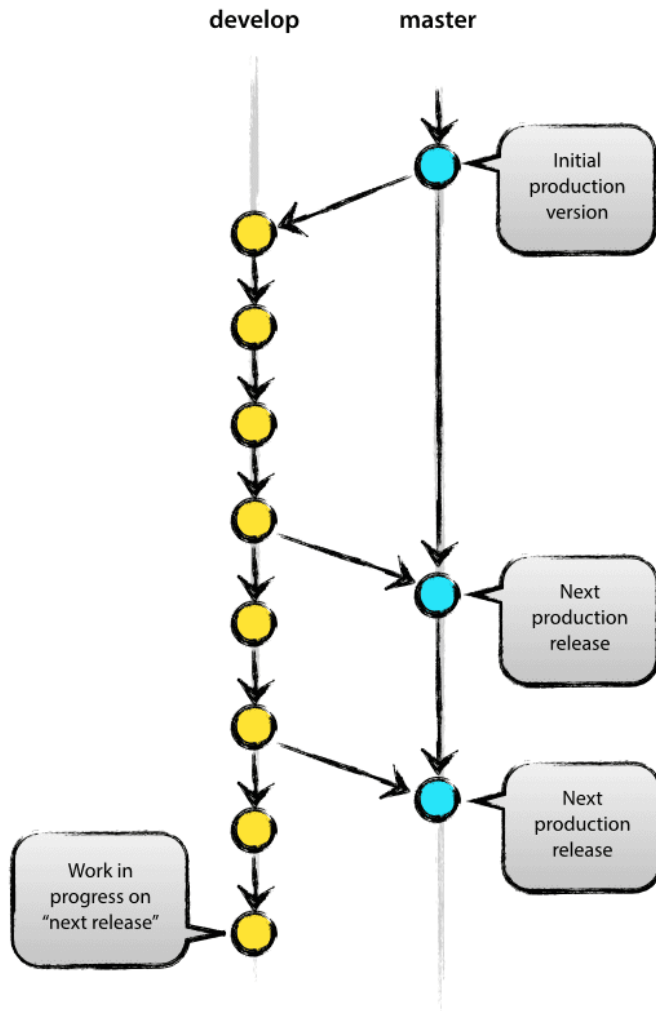


- Ein viel verwendetes Model in der Industrie.
Im Folgenden vereinfacht:
 - Keine Release-Banches
 - Keine Hotfixes

GIT Branching in der Praxis



GIT Branching Modell – Stabile Branches



Es gibt 2 Branches die immer existieren:

1. **Master:** Die fertige „production-ready“ Version der Software
2. **Develop:** der aktuelle Stand der Entwicklung für das nächste Release

Jeder **Merge** nach Master ist ein **Release!**
Jedes Release bekommt einen **Tag**.

GIT Branching Modell – Feature branches

feature
branches develop

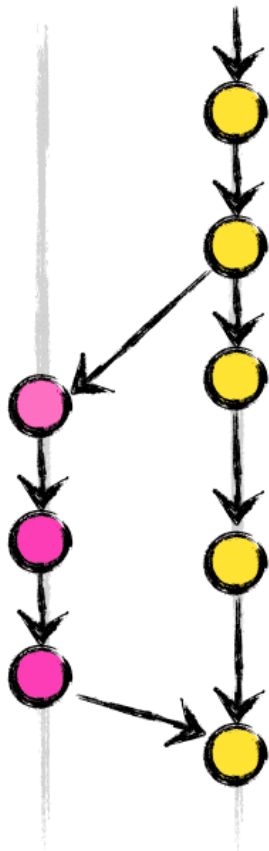
Also wird in „**Develop**“ entwickelt?

Nein!

Es ist nicht klar wann ein Feature released wird, darum
Trennung in Sub-Banches.
Erlaubt es parallel zu arbeiten

Wichtig

- mergen mit **git merge --no-ff Branchname**
- nach dem Merge Feature-Branch löschen



Wichtige GIT Kommandos

- ***\$ git status***
 - Zeigt den momentanen Stand im Zyklus an
- ***\$ git add <file name>***
 - Fügt File(s) zum Staging Area (Index) hinzu
- ***\$ git checkout <commit>***
 - selektiert commit/branch an dem gearbeitet werden soll
 - *D.h. es lädt den Stand des Commits in die Working Copy Ordner*
- ***git fetch <quelle>***
 - Lädt Änderungen aus Quelle in lokales Repository
 - *Integriert sie NICHT mit lokalem Stand*

Wichtige GIT Kommandos cont.

- ***\$ git merge <commit>***
 - Versucht den lokalen Stand mit dem ausgewählten commit zu mergen
- ***\$ git pull <remote name> <branch>***
 - Kombiniert *fetch* und *merge*
 - Beispiel: *\$ git pull origin master*
- ***git push <ziel>***
 - Transferiert lokalen Stand zu Remote Repository
- ***\$ git branch <branchname>***
 - Erstellt neuen Branch

Git Tips

- Häufiges Committen von kleinen abgeschlossenen Arbeitspaketen (wenige Codezeilen!)
 - Erhöht die Nachvollziehbarkeit
- Erst wenn man im Team den klassischen zentralen Workflow versteht sollte man auf komplexere Mechanismen umsteigen



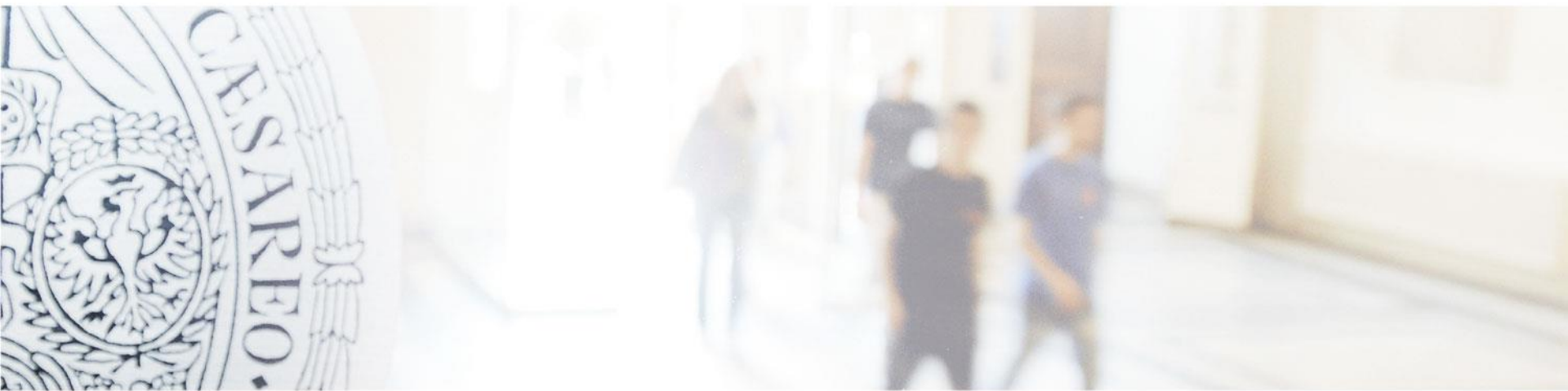
	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT
MESSAGES GET LESS AND LESS INFORMATIVE.

4. Zusammenfassung

Zusammenfassung

- UML Sequenzdiagrammen
 - Beschreiben komplexe Interaktionen zwischen Objekten in bestimmten Rollen
 - Beschreibt die zeitliche Abfolge dieser Interaktionen
 - Wird zur Beschreibung der fachlichen Sicht (Laufzeitsicht) verwendet
- Dokumentation von Source Code
 - Code Conventions: Satz von Regeln, nach welchen der Quelltext eines Programms erstellt wird
 - Java Docs: Erstellung von (HTML-)Dokumentationsdateien aus Java-Quelltextdateien
- Versionskontrollsysteme
 - Zentrales vs. Dezentrales Versionskontrollsystem
 - CVS, SVN, GIT,...
 - GIT Branching in der Praxis



Kontakt

Dipl.-Ing. Clemens Sauerwein, M.Sc., PhD

Raum 3S03, Technikerstraße 21a, 6020-Innsbruck, Austria

Sprechstunde: nach Vereinbarung!

Telefon: 0043 512 507 53345

Email: Clemens.Sauerwein@uibk.ac.at