Programmierkonzepte in der Physikalischen Chemie

4. Infrastruktur

UNI FREIBURG

Albert-Ludwigs-Universität Freiburg

Dr. Till Biskup

Institut für Physikalische Chemie
Albert-Ludwigs-Universität Freiburg
Wintersemester 2013/14

Übersicht

UNI FREIBURG

Infrastruktur

Versionsverwaltung

- Versionsverwaltungssysteme
- Grundbegriffe der Versionsverwaltung
- Grundlegende Arbeit mit einer Versionsverwaltung
- Praktisches Arbeiten mit Git

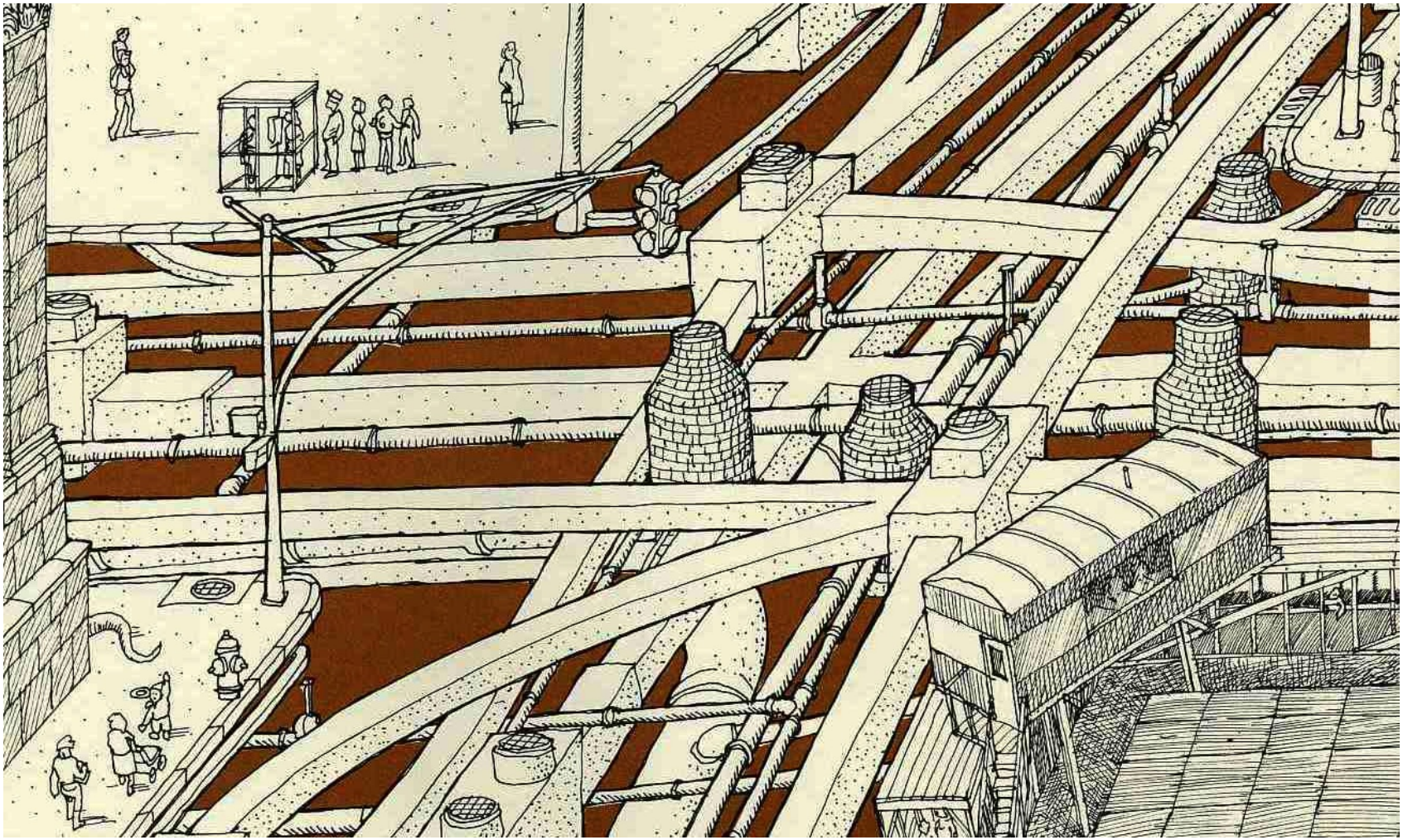
Planung und Dokumentation (Wiki)

- Mögliche Gliederung
- Wiki-Software

Bug-Verwaltung

Infrastruktur

Oft unsichtbar, aber überlebenswichtig



David Macaulay: Unter einer Stadt. dtv, München, 1977

Infrastruktur (allg.)

personelle, sachliche und finanzielle Ausstattung,
um ein angestrebtes Ziel zu erreichen

- personell: Wir sind meist auf uns alleine gestellt.
- finanziell: Programmierung ist meist ein Hobby.
- sachlich: Es gibt Hilfsmittel, die das Leben erleichtern.
  - Versionsverwaltung (VCS)
  - Planung und Dokumentation (Wiki)
  - Bug-Verwaltung

Schubert, Klein: Das Politiklexikon. 5., aktual. Aufl. Bonn: Dietz 2011

Infrastruktur

Ziel: Das Leben erleichtern

UNI FREIBURG

Ist Infrastruktur wirklich wichtig/notwendig?

- Abhängig von der Größe eines Projektes
- Für Auswertesoftware gilt in der Regel: Ja!

Ziel: Das Leben erleichtern

- Balance zwischen dem „Overhead“,
  den Infrastruktur erzeugt, und ihrem Nutzen
- Nutzen muss für die Anwender klar ersichtlich sein

☛ Eigene Erfahrung:
Infrastruktur wirkt befreiend und steigert die Produktivität

Wichtige Aspekte in der Umsetzung

- Verwendung von Standard-Komponenten
  - keine Insel- oder Speziallösungen!
  - idealerweise freie Software (kostenneutral)
  - ausreichend getestet
  - einfach zu integrieren
  - hohe Chance auf Interoperabilität
- Einfache Bedienbarkeit
  - Nur was möglichst einfach und intuitiv bedienbar ist, wird in der Praxis auch eingesetzt werden.
- Hohe Verfügbarkeit
  - Programmierung oft auch nachts und am Wochenende
  - Erreichbarkeit von außerhalb der Arbeit sicherstellen

Eigener Server (Linux, Beispielkonfiguration)

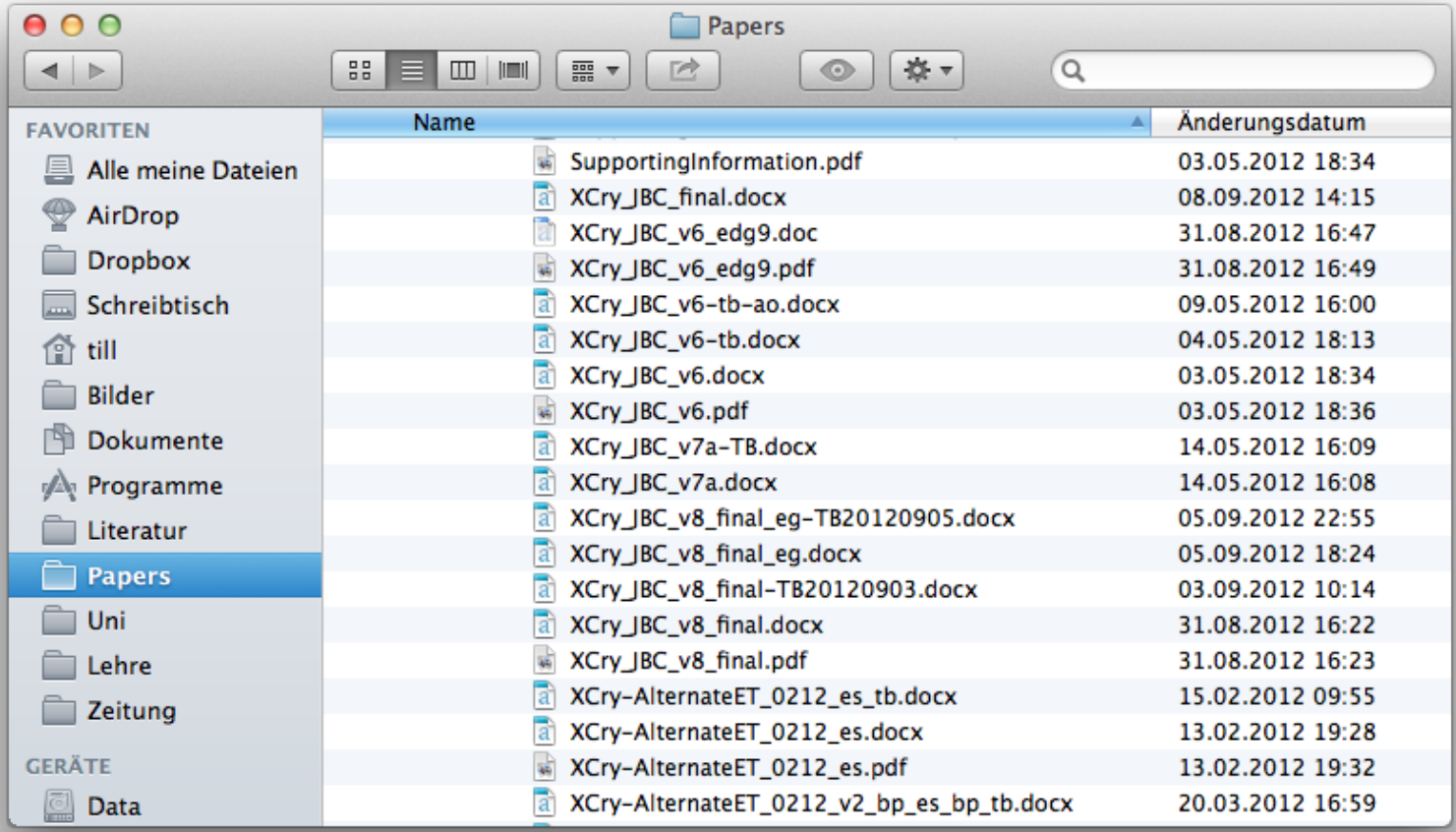
VCS	Git (mit gitolite und gitweb)
Wiki	DokuWiki
Bugs	BugZilla

Webbasierte Lösungen

- SourceForge
  - Freie Plattform
  - zugrundeliegende Software ebenfalls Open Source
- GitHub
  - Kommerzielle Plattform
  - Für öffentliche Projekte kostenfrei

Versionsverwaltung

Voraussetzung für zentrale Programmierkonzepte



Versionsverwaltung

engl. *version control system* (VCS), System zur Erfassung von Änderungen an Dokumenten oder Dateien.

- Alle Versionen werden in einem Archiv gesichert.
- Jeweils mit Zeitstempel und Benutzerkennung
- Jede Version kann später wiederhergestellt werden.

- ☛ Typischer Einsatz: Softwareentwicklung
- ☛ VCS hat erst einmal nichts mit Versionsnummern zu tun.

Versionsverwaltung

Aufgaben einer Versionsverwaltung

UNI FREIBURG

- Protokollierung der Änderungen
  - Wer hat wann was (warum) geändert?
- Wiederherstellung
  - Beliebige alte Zustände einer Datei wiederherstellbar
  - (Versehentliche) Änderungen rücknehmbar
- Archivierung
  - Zustände eines Projektes archiviert
  - Bestimmter Zustand jederzeit wieder herstellbar
- Zugriffskontrolle und -koordination
  - Wichtig bei gemeinsamer Arbeit an einem Projekt
  - Zusammenführung unterschiedlicher Änderungen
- Verwaltung mehrerer Zweige
  - Gerade in der Softwareentwicklung bedeutsam

Warum Versionsverwaltung?

- Auswirkungen auf die Art des Programmierens
  - Befreiend: Funktionierende Vorversion immer erreichbar
  - Strukturierend: Klarer Verweis auf eine Version möglich
- Unumgänglich für verteilte Programmierung
  - Zusammenführung unterschiedlicher Änderungen
  - Verantwortliche für Änderungen nachvollziehbar

Warum Versionsverwaltung als einzelner Nutzer?

- Historie einer Entwicklung verfügbar
- Auswirkungen auf die Art des Programmierens (s.o.)

☞ Voraussetzung für zentrale Programmierkonzepte

Lokale Versionsverwaltung

- Oft Versionierung nur einer Datei
- Heute noch in Büroanwendungen (z.B. Word & Co.)

Zentrale Versionsverwaltung

- Server-Client-Konzept
- Rechteverwaltung beschränkt Zugriff
- Versionsgeschichte nur an einem Speicherort

Verteilte Versionsverwaltung

- Kein zentraler Speicher, dezentrale Versionsgeschichte
- Jeder pflegt eigenen Speicherort mit ei(ge)ner Historie

Concurrent Versions System (CVS)

- Erste populäre zentrale Versionsverwaltung (ab 1989)
- Konzepte in viele kommerzielle Produkte übernommen

Subversion (SVN)

- Quasi Reimplementierung von CVS (ab 2000, 2004: 1.0)
- Viele Verbesserungen

Git

- Dezentrale Versionsverwaltung („everything is local“)
- Vorteil: Für Einzelkämpfer genauso gut wie im Team

Grundbegriffe der Versionsverwaltung

Repository	(zentraler) Speicherort der versionierten Dateien
Arbeitskopie	Lokale Kopie eines Repositorys
Revision	einzelner der Versionsverwaltung bekannter Stand
Head	Neueste Revision eines bestimmten Branches
Branch	Zweig, Abspaltung von einer anderen Version Branches können parallel weiterentwickelt werden Hauptzweig: *trunc* (SVN) bzw. *master* (Git)
Tag	frei wählbarer Bezeichner für eine Revision z.B. nach außen kommunizierte Versionsnummer

Grundbegriffe der Versionsverwaltung (Fortsetzung)

checkout	Holen einer Version aus dem Repository
commit	Übertragen einer Version in das Repository
merge	Zusammenführen unterschiedlicher Versionen
diff	Vergleich zweier Versionen

Verteilte Versionsverwaltungssysteme

push	Übertragen einer Version in ein anderes (ggf. entferntes) Repository
pull	Holen einer Version aus einem anderen (ggf. entfernten) Repository

Bevor es losgehen kann

- Installation
  - Die relevanten Programme existieren für jede Plattform
  - Versionsverwaltung findet auf der Kommandozeile statt
  - Es gibt Integrationen in viele gängige IDEs
- Grundkonfiguration
  - Personalisierung mit Name und Email-Adresse
- Arbeitskopie anlegen
  - aus einem bestehenden Repository
  - aus einem neuen Repository

☛ Personalisierung ist von herausragender Bedeutung:
Wer ist für welche Änderung verantwortlich?

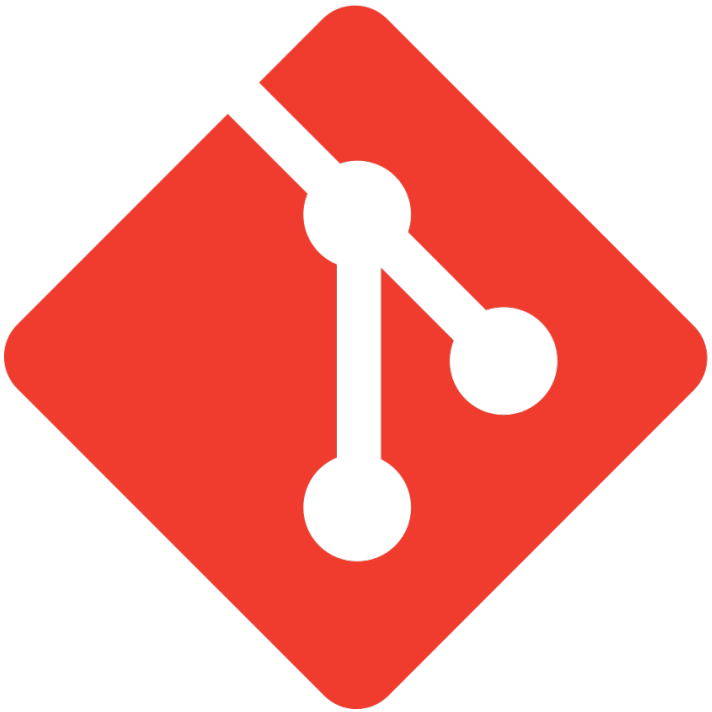
Tägliche Arbeit

- Regelmäßig auf Änderungen überprüfen
  - Nur wichtig, wenn mehrere Leute parallel entwickeln
- Änderungen regelmäßig einchecken
  - Bei verteilten Versionsverwaltungen zusätzlich pushen
  - Je kleiner die Änderungen, desto größer die Freiheit
  - Nur funktionierende Versionen einchecken (aber: s.u.)
- Größere Änderungen in eigenem Zweig (Branch)
  - Ermöglicht die Entwicklung ohne Notwendigkeit, dass jede Version „funktionieren“ muss.
  - Zweige können später zusammengeführt werden.

Tägliche Arbeit (Fortsetzung)

- Wichtige Revisionen markieren (Tag)
  - z.B. veröffentlichte Versionen
  - Revisionen, die man direkt ansprechen können möchte
  - Tags „kosten“ nichts
- Änderungen weitergeben (Patches, Pull Requests)
  - Mehrere Leute arbeiten an einem Projekt
  - Man ist selbst nicht der Hauptentwickler
  - Man hat selbst nicht Zugriff auf das Hauptrepository
  - Weitergabe von Änderungen an den Hauptentwickler

☛ Im Folgenden geht es um Git.



git

Besonderheiten von Git

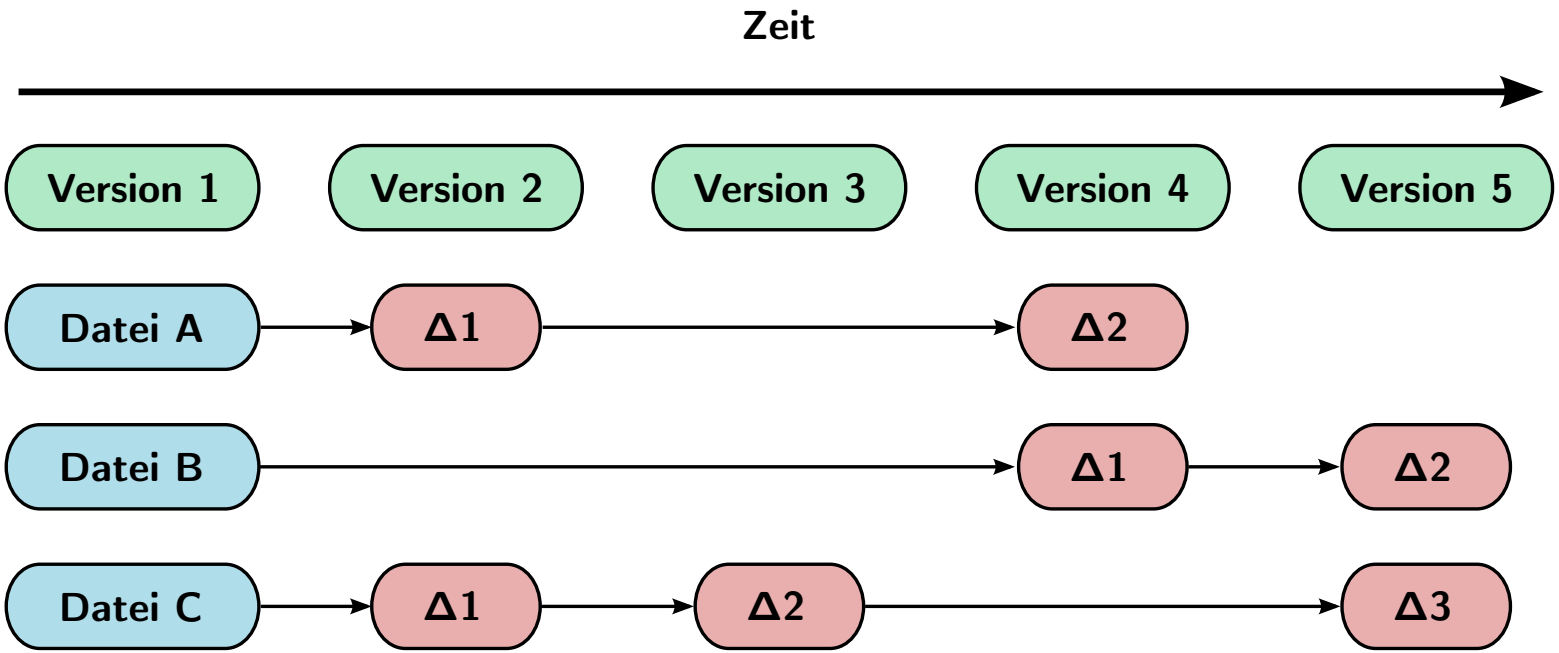
- Verteiltes Versionsverwaltungssystem
  - Lokal, unabhängig von Serverinfrastruktur
  - Dadurch sehr schnell
- Optimiert für Branching und Merging
  - Eröffnet ganz neue Entwicklungsmuster

Vorteile von Git

- Geeignet für kleine und große Projekte
  - Sowohl eine Skriptsammlung als auch der Linux-Kernel
- Perfekt für den lokalen Einsatz durch Einzelpersonen
  - Später problemlos auf mehrere Entwickler erweiterbar

Herkömmliche Versionsverwaltungssysteme

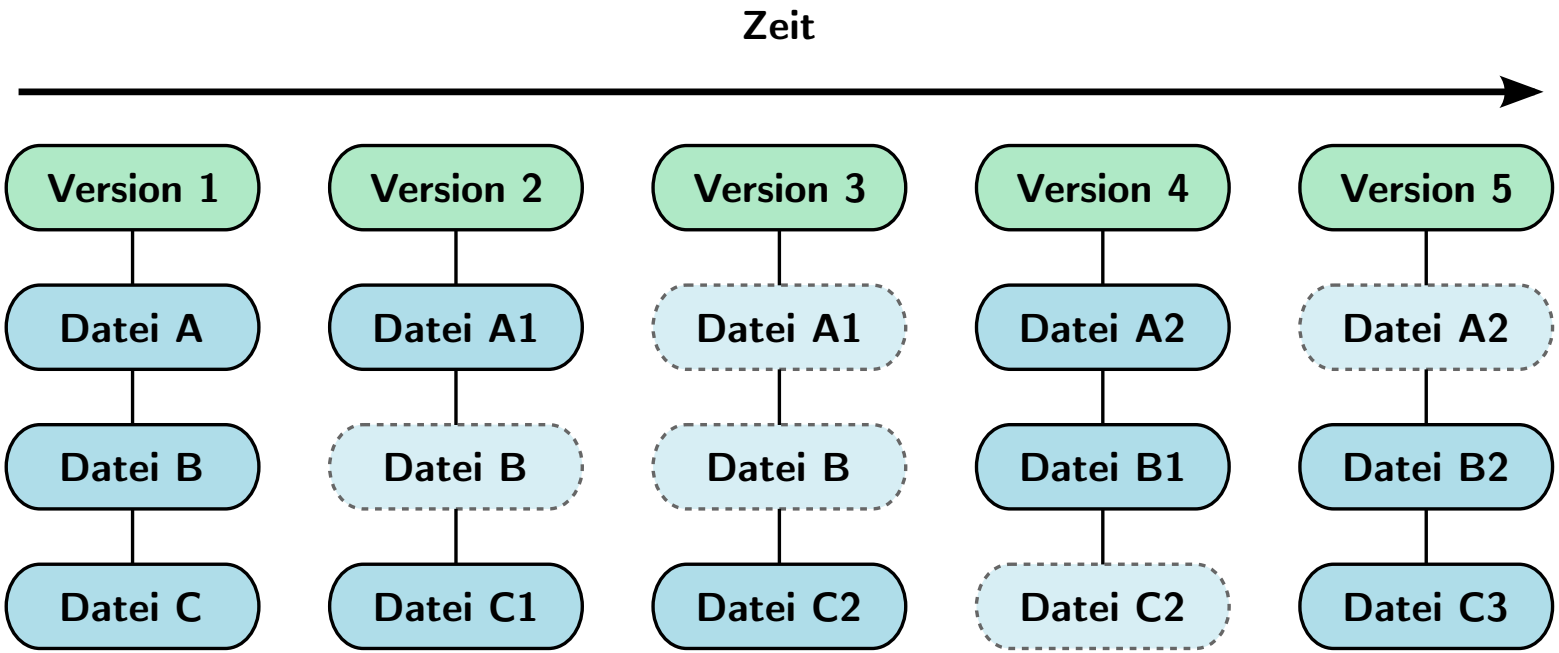
- Daten werden als Änderungen an einzelnen Dateien einer Datenbasis gespeichert



`http://git-scm.com/book/`, verändert

Git

- Daten werden als eine Historie von Momentaufnahmen (Snapshots) des Projektes gespeichert



`http://git-scm.com/book/`, verändert

- Git bedient sich am Einfachsten auf der Kommandozeile.
- Für die Darstellung komplexer Historie gibt es GUIs.

Struktur der Befehle

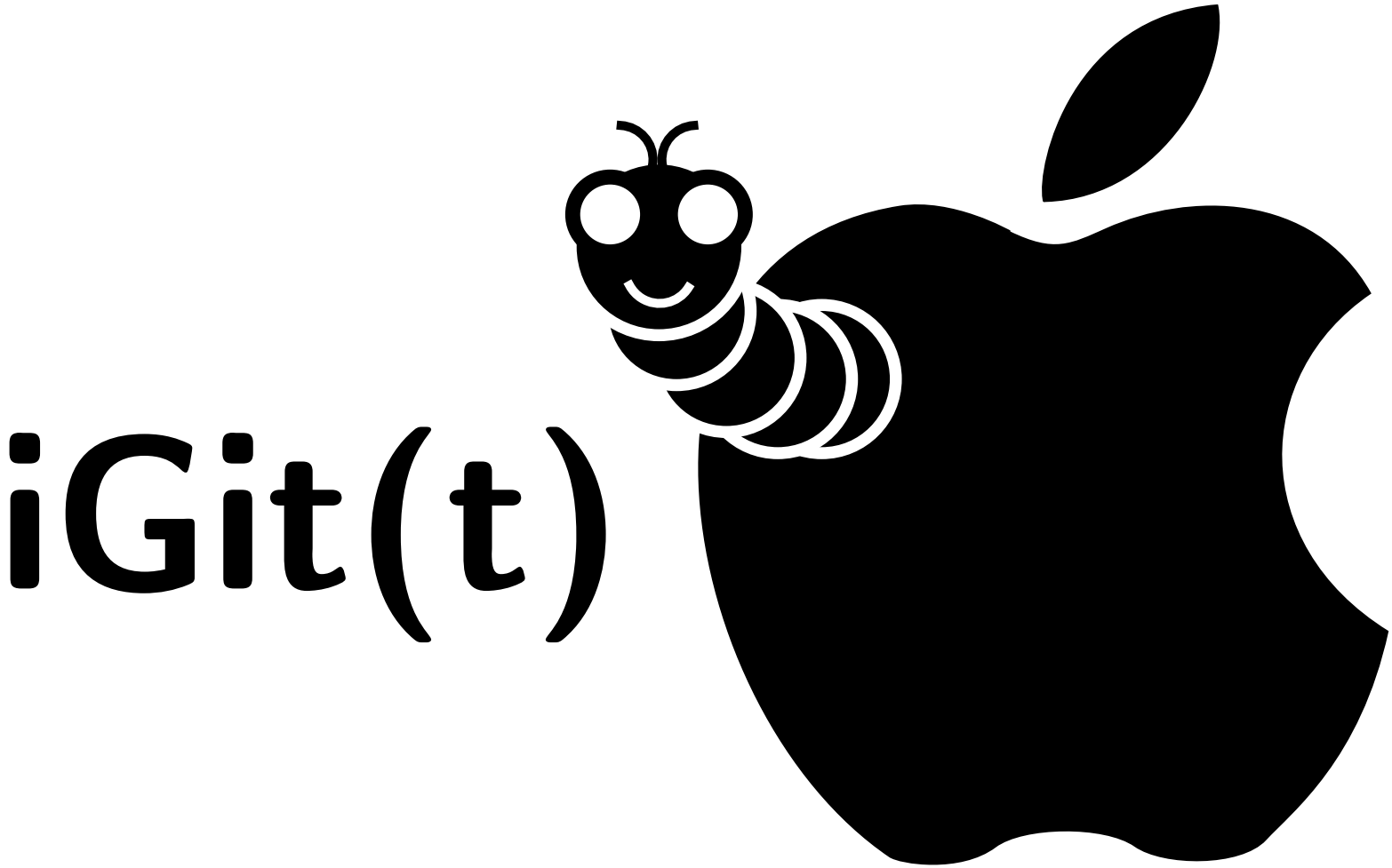
Listing 1: Grundstruktur von Git-Befehlen

```
git <befehl> <optionen>
```

Hilfe zu Git-Befehlen

Listing 2: Hilfe zu Git-Befehlen

```
git help <befehl>
```


iGit(t)

Erster Schritt: Repository erzeugen

1 Klonen eines bestehenden Repositorys

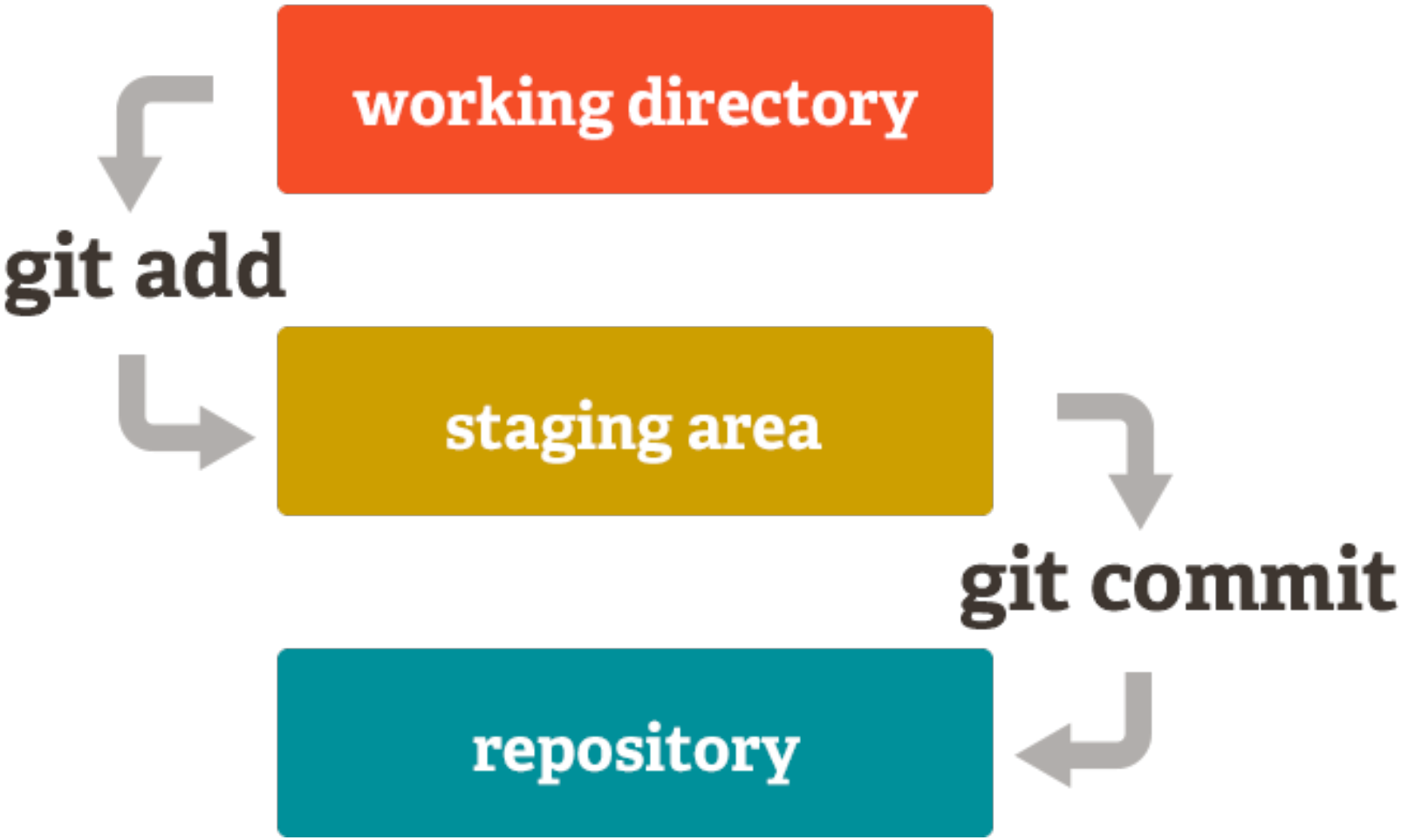
Listing 3: Bestehendes Git-Repository klonen

```
$ git clone <url> <path>
Cloning into '<path>'...
```

2 Neues (leeres) Repository erzeugen

Listing 4: Neues Git-Repository erzeugen

```
$ git init <path>
Initialized empty Git repository in <path>/.git/
```



http://git-scm.com/

Grundlegender Arbeitsprozess

1. Dateien im Arbeitsverzeichnis werden bearbeitet.
   - working directory
2. Dateien werden für den nächsten Commit markiert.
   - Snapshots werden zur „Staging Area“ hinzugefügt.
   - Dateien hinzufügen: `git add`
   - Dateien entfernen: `git remove`
3. Ein Commit wird angelegt.
   - Die in der Staging Area vorgemerkten Snapshots werden dauerhaft in der lokalen Datenbank gespeichert.
   - Zu jedem Commit wird ein kurzer Kommentar angegeben.
   - `git commit`

git status – wissen, was los ist

Listing 5: "git status" auf leerem Repository

```
1 $ git status
2 # On branch master
3 #
4 # Initial commit
5 #
6 nothing to commit (create/copy files and use "git add" to track)
```

- Keine Änderungen seit dem letzten Commit.
- Zeigt den aktuellen Zweig („branch master“) an.
- Sieht direkt nach einem Commit quasi gleich aus.

git status – wissen, was los ist

Listing 6: "git status" nach ersten Änderungen

```
1  $ git status
2  # On branch master
3  #
4  # Initial commit
5  #
6  # Untracked files:
7  #   (use "git add <file>..." to include in what will be committed)
8  #
9  # README
10 nothing added to commit but untracked files present (use "git add" to track)
```

- Änderungen seit dem letzten Commit („Untracked files“).
- Gibt Hilfestellung, was zu tun wäre („git add“).

git add – Dateien zum Commit vormerken

Listing 7: Alle Dateien zum Commit vormerken

```
git add
```

- Merkt alle Änderungen zum Commit vor.

Listing 8: Einzelne Dateien zum Commit vormerken

```
git add <filepattern>
```

- Zum Vormerken einzelner Dateien.
- Nützlich zum separaten Commit von Änderungen

git rm – Dateien zum Löschen vormerken

Listing 9: Einzelne Dateien löschen

```
git rm <filepattern>
```

- Löscht Dateien aus Index und Arbeitsverzeichnis
- Nur möglich ohne Änderungen an diesen Dateien.
  - Überschreiben durch zusätzlichen Parameter „–f“
- Akzeptiert die üblichen Platzhalter für Muster (*, ...)
- Funktioniert nicht für Dateien, die bereits gelöscht wurden.
  - Mögliches Workaround: `git commit -a`

git commit – Vorgemerkte Dateien ins Repository übertragen

Listing 10: Vorgemerkte Dateien ins Repository übertragen

```
git commit -m '<commit comment>'
```

- Schreibt die Änderungen („git add“, „git rm“) ins Repository
- Commit-Kommentar ist obligatorisch
- Wird die Option „-m“ (mit Kommentar) nicht angegeben, wird der voreingestellte Editor geöffnet.
- Zusätzliche Option „-a“ erspart vorheriges „git add“
  - Sparsam einsetzen: Man will *nicht immer* alles committen

Versionsverwaltung

Branching und Merging – die besondere Stärke von Git

git branch – Arbeiten mit Verzweigungen

Listing 11: Liste aller existierenden Zweige anzeigen

```
git branch
```

- Zeigt Liste aller existierenden Zweige
- Aktueller Zweig wird hervorgehoben

Listing 12: Neuen Zweig anlegen

```
git branch <branchname>
```

- Legt neuen Zweig mit dem Namen `<branchname>` an
- Optionen: „–m“ benennt um, „–d“ löscht

git checkout – Zwischen Versionen wechseln

Listing 13: Auf anderen Zweig wechseln

```
git checkout <branch>
```

- Wechselt auf den angegebenen Zweig `<branch>`

Listing 14: Änderungen im Arbeitsverzeichnis verwerfen

```
git checkout -- <file>
```

- Überschreibt die aktuelle Fassung der Datei `<file>` mit der letzten Fassung aus dem Repository

git merge – Mehrere Versionen oder Zweige zusammenfassen

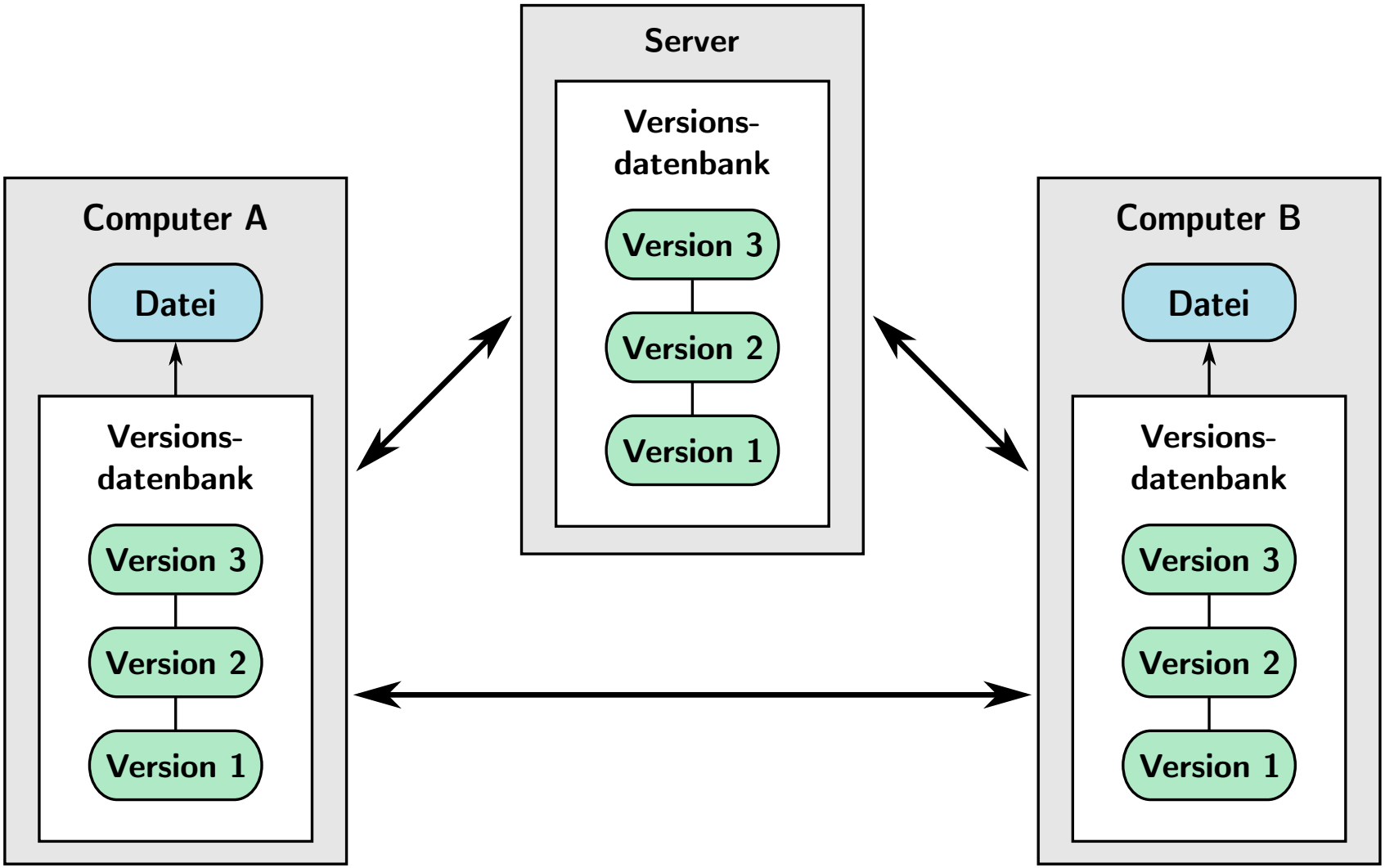
Listing 15: Anderen Zweig mit aktuellem Zweig verbinden

```
git merge <branch>
```

- Verbindet den angegebenen Zweig mit dem aktuellen.
- Konflikte werden auf der Kommandozeile ausgegeben
  - Dateien beinhalten übliche Marker: <<<, ===, >>>
  - Konflikte können (müssen) manuell behoben werden.
- „git pull“ führt automatisch ein „git merge“ durch.
  - Normalfall: „fast forward“ – aktuelles Arbeitsverzeichnis ist ein Vorgänger des entfernten Repositorys

Versionsverwaltung

Pull und Push – Interaktion zwischen Repositories



git pull – Änderungen von entferntem Repository holen

Listing 16: Änderungen von entferntem Repository holen

```
git pull [<repository>] [<branch>]
```

- Es gibt bei Git keinen Server und keinen Klienten.
- Ein Repository kann mit mehreren entfernten Repositories abgeglichen werden.
- In der Konfiguration können Kurznamen für die entfernten Repositories festgelegt werden
- Der Zweig (*branch*) muss nicht immer angegeben werden.

git push – Änderungen in entferntes Repository übertragen

Listing 17: Änderungen in entferntes Repository übertragen

```
git push [<repository>] [<branch>]
```

- Symmetrisch zu `git pull`
- Setzt Schreibrechte auf dem entfernten Repository voraus
- Alternative: „pull request“
  - Leserechte sind weitaus häufiger als Schreibrechte

☛ Rechteverwaltung ist kein integraler Bestandteil von Git.
Es gibt aber viele externe Programme dafür.

git log – letzte Änderungen anzeigen

Listing 18: Letzte Änderungen anzeigen

```
1  $ git log
2  commit 99f745ce29564753ac3c7e720728b2e3b04f689a
3  Author: Till Biskup <till@till-biskup.de>
4  Date:   Tue Nov 19 21:22:23 2013 +0100
5
6      Updated list of new features
7
8  commit e0d059cbb984dcf9ef5ed3e09872d20a67f5578f
9  Author: Till Biskup <till@till-biskup.de>
10 Date:   Tue Nov 19 21:20:09 2013 +0100
11
12     Added new importer to list of known file formats
```

- Commits werden durch Hashes identifiziert
- Zu jedem Commit Autor, Datum und Commit-Nachricht

.git/config – Konfiguration auf Repositoryebene

Listing 19: Beispiel für eine config-Datei im .git-Verzeichnis

```
1 [core]
2     repositoryformatversion = 0
3     filemode = true
4     bare = false
5     logallrefupdates = true
6     ignorecase = true
7 [remote "fr"]
8     url = git@sw2.chemie.uni-freiburg.de:trEPRtoolbox
9     fetch = +refs/heads/*:refs/remotes/fr/*
```

- Liegt im .git-Verzeichnis des Repositorys
- Enthält Einstellungen auf Repositoryebene
- U.a. Kürzel für entfernte Repositories („remote“)

.gitignore – Dateien und Dateimuster ignorieren

Listing 20: Beispiel für eine .gitignore-Datei

```
1  # Can ignore specific files
2  .DS_Store
3  *.aux
4  *.log
5  *.nav
6  *.out
7  *.pdf
8  *.snm
9  *.toc
10 *.vrb
11
12 # Use wildcards as well
13 *~
```

- Liegt im Wurzelverzeichnis des Repositorys
- Enthält Einstellungen auf Repositoryebene

~/.gitconfig – Konfiguration auf Nutzerebene

Listing 21: Beispiel für eine .gitconfig-Datei

```
1  [user]
2      name = Till Biskup
3      email = till@till-biskup.de
4  [core]
5      editor = vim
6  [color]
7      status = auto
8      branch = auto
9  [alias]
10     ci = commit
11     st = status
12     co = checkout
```

- Liegt im Heimverzeichnis des Nutzers
- Enthält zentrale Einstellungen auf Nutzerebene
- Wichtig für die obligatorische Personalisierung

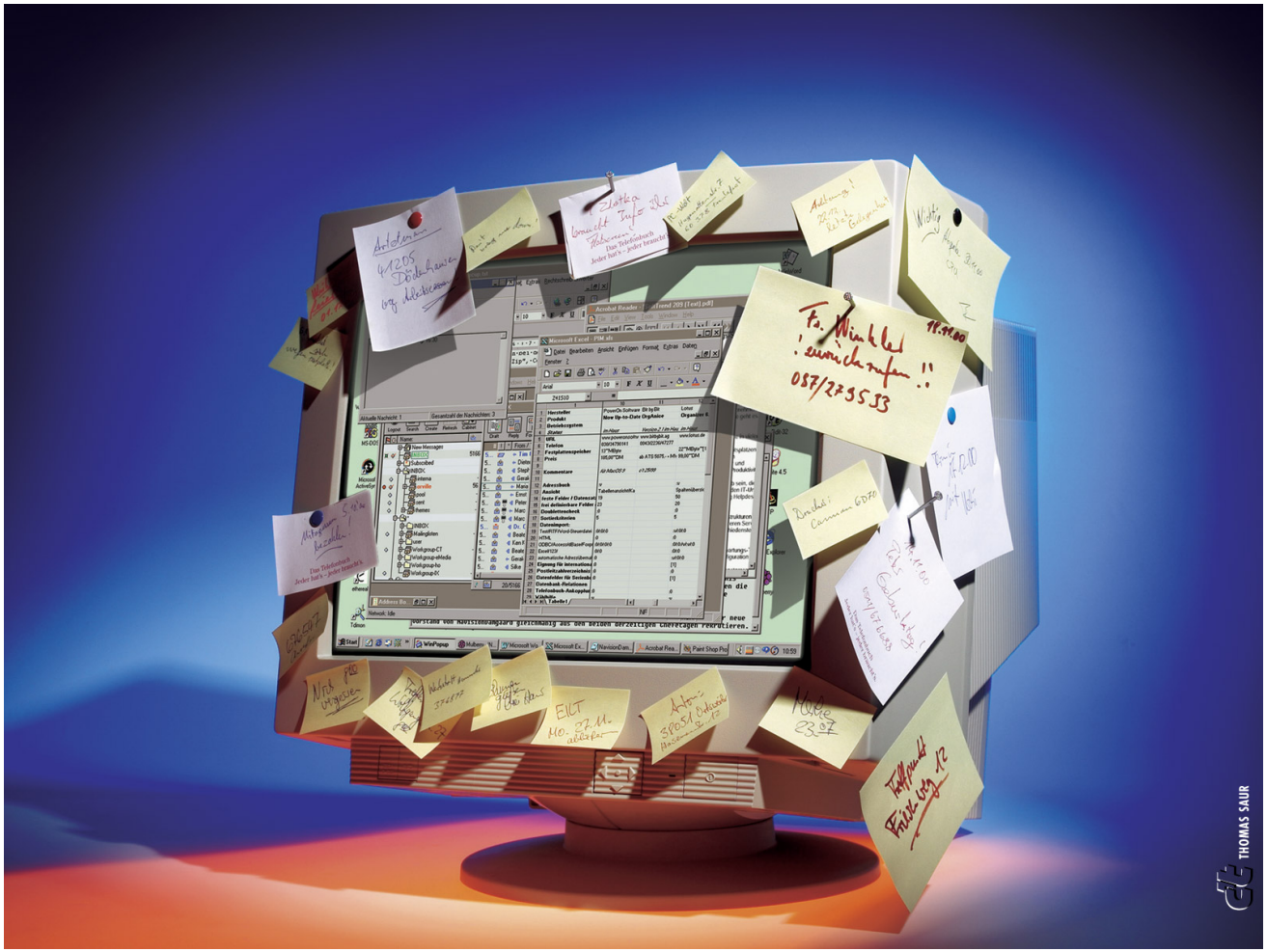
Nochmal: Warum Versionsverwaltung?

- Auswirkungen auf die Art des Programmierens
  - Befreiend: Funktionierende Vorversion immer erreichbar
  - Strukturierend: Klarer Verweis auf eine Version möglich
- Unumgänglich für verteilte Programmierung
  - Zusammenführung unterschiedlicher Änderungen
  - Verantwortliche für Änderungen nachvollziehbar
- Zugewinn auch als Einzelnutzer
  - Historie einer Entwicklung verfügbar
  - Auswirkungen auf die Art der Programmierung (s.o.)

☛ Voraussetzung für zentrale Programmierkonzepte

Planung und Dokumentation (Wiki)

Konzeptionelle Dokumentation jenseits des Quellcodes



c't, Thomas Saur, Heise Zeitschriftenverlag

Viele verschiedene Arten

- Im Quellcode
  - Schnittstellen-Dokumentation
  - Quellcode-Dokumentation (einzelne Zeilen)
- Im gleichen Verzeichnis wie die Funktionen/Toolbox
  - `README`
  - `INSTALL`
- Nutzerhandbuch
  - Beschreibung der Verwendung jeder Funktion
- Konzepte
- Beispiele

Unterschiedliche Einteilung

- nach Zielgruppe
  - Programmierer
  - Anwender
- nach Inhalt
  - Quellcode-Dokumentation
  - Schnittstellen-Dokumentation
  - Dokumentation der Konzepte
  - Installation, Bedienung, ... (Anwenderdokumentation)
- nach Medium
  - im Quellcode
  - in Dateien neben dem Quellcode
  - getrennt vom Quellcode (Webseite, ...)

Konzeptionelle Dokumentation

- Eine Schnittstellen-Dokumentation reicht meist nicht aus.
- Grundlegende Konzepte und Ideen sollten zusammenhängend beschrieben werden.
- Ein statisches Dokument ist oft nicht flexibel genug.

- Ein Wiki ist eine mögliche Lösung.
  - Flexibel
  - Erlaubt einfache Aktualisierungen
  - Geeignet als primäre Informationsquelle für Anwender

- ☛ Hat sich in der Praxis bewährt
- ☛ Qualität ist (auch hier) eine Frage persönlicher Disziplin

Planung und Dokumentation (Wiki)

Mögliche Gliederung

UNI FREIBURG

- Gliederung nach Zielgruppe
  - Endanwender
  - Entwickler
- Wiki gleichzeitig als Webpräsenz des Projektes
  - Ggf. Zugangsbeschränkung für Entwicklerbereich
  - Entscheidung frühzeitig treffen:
    Einfluss auf Struktur, Qualität und Formulierungen
- konkreter Vorschlag für eine Gliederung
  - Basiert auf eigener (langjähriger) Erfahrung
  - Nur ein Vorschlag
  - Setzt eine hierarchische Struktur (z.B: DokuWiki) voraus

Planung und Dokumentation (Wiki)

Mögliche Gliederung

UNI FREIBURG

- Dokumentation (Anwender)
  - Aufgaben
  - Funktionen
  - Beispiele
  - HOWTOs
  - Installation
  - Erste Schritte
- Entwicklung (Programmierer)
  - Konzepte
  - Ideen
  - Planung
  - Dokumentation
  - Changelog
- FAQ (Anwender)

Planung und Dokumentation (Wiki)

Mögliche Gliederung – Dokumentation (Anwender)

UNI FREIBURG

- Aufgaben
  - Was kann ich mit dem Programm/der Toolbox machen?
- Funktionen
  - Funktionsreferenz zum Nachschlagen
- Beispiele
  - Konkrete Beispiele für typische Aufgaben
- HOWTOs
  - kurze, anwendungsorientierte Tutorien

- Installation
  - knappe, vollständige Installationsanleitung
- Erste Schritte
  - Den Neuling mit dem Programm vertraut machen

Planung und Dokumentation (Wiki)

Mögliche Gliederung – Entwicklung (Programmierer)

UNI FREIBURG

- Konzepte
  - grundlegende Konzepte schriftlich niederlegen
- Ideen
  - Was gibt es an Ideen für die weitere Entwicklung?
- Planung
  - Konkrete Planung für die Umsetzung der nächsten Schritte
- Dokumentation
  - interne Abläufe, „call graphs“, etc.

- Changelog
  - Seite mit der Liste aller Änderungen für alle Versionen

Planung und Dokumentation (Wiki)

Welche Wiki-Software?

UNI FREIBURG

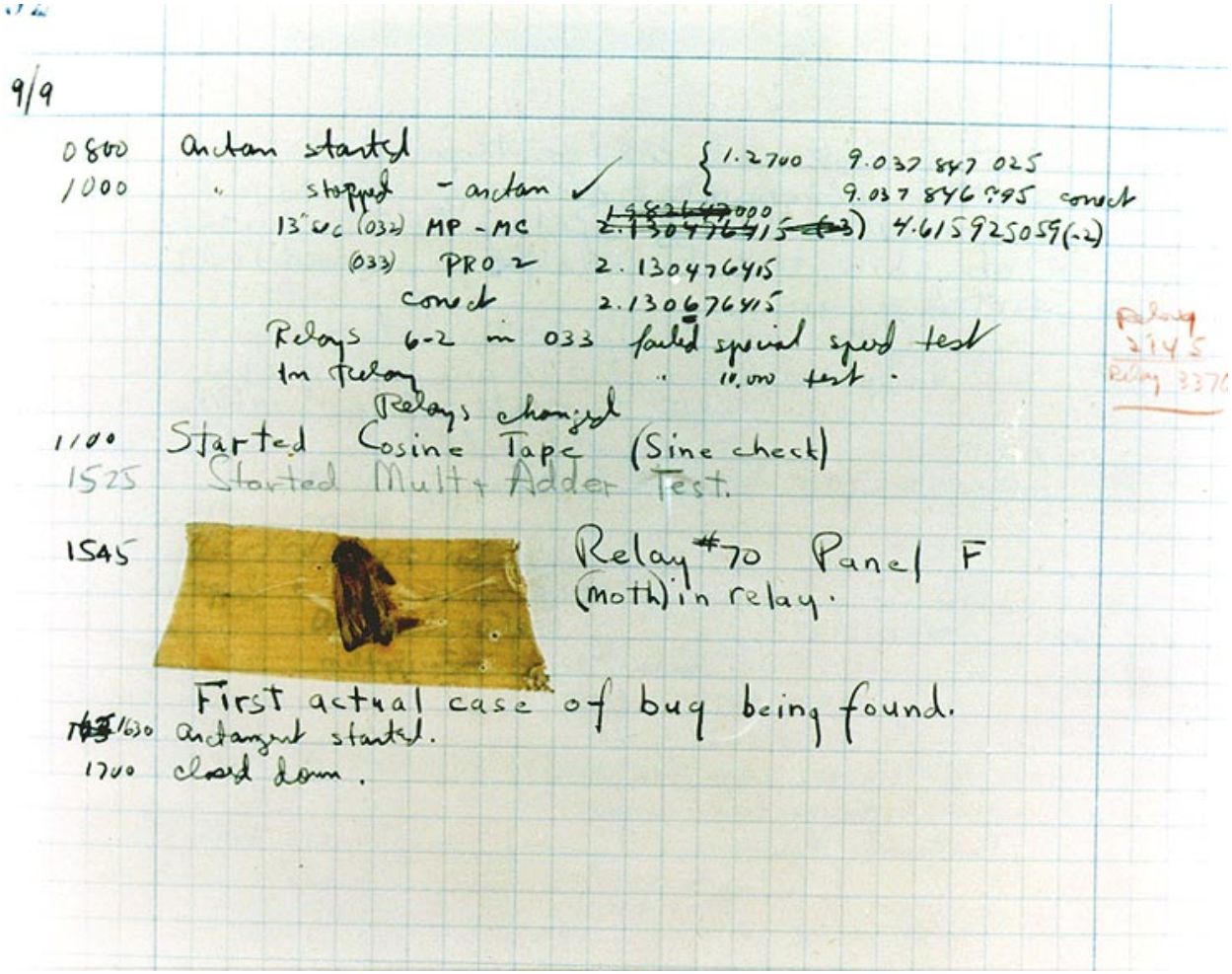
- Welche Wiki-Software?
  - Grundsätzlich ist jede Wiki-Software nutzbar
  - Sollte gut unterstützt sein und aktiv entwickelt werden
- DokuWiki hat konzeptionell einige Vorteile
  - keine Datenbank, deshalb einfach zu kopieren
  - hierarchisch
  - einfache Syntax
  - Grundsystem sehr schlank
  - Einfach lokal installierbar (und synchronisierbar)
  - Läuft auf (fast) jeder Plattform

☛ Letztlich eine Frage persönlicher Vorlieben

☛ Was gefällt und praktisch ist, wird genutzt werden.

Bug-Verwaltung

Bugs sind eine unvermeidliche Begleiterscheinung



U.S. Naval Historical Center Online Library Photograph NH 96566-KN

Bug-Verwaltung

Bugs sind eine unvermeidliche Begleiterscheinung

UNI FREIBURG

Bug

Programmfehler oder Softwarefehler,
allgemein ein Fehlverhalten von Computerprogrammen

- Völlige Fehlerfreiheit für Software, die eine gewisse Komplexitätsgrenze überschreitet, ist weder erreichbar noch beweisbar.

☛ Stabilität und Robustheit sind wichtiger als Fehlerfreiheit.

☛ Wird in der nächsten Lektion behandelt.

Bug-Verwaltung

Bugs sind eine unvermeidliche Begleiterscheinung

Das Problem des Ariane-5-Jungfernflugs

- Fehler bei der Typkonversion einer Variablen
  - zulässiger Wertebereich wurde überschritten
  - Routine wurde komplett von der Ariane 4 übernommen
  - für die Trajektorie der Ariane 4 war bewiesen worden, dass der Wertebereich nie überschritten wird
- Ausnahme (*exception*) wurde nicht korrekt abgefangen
  - vom Flugcomputer als Steuerbefehle interpretiert
  - Destabilisierung der Flugbahn und Selbstzerstörung
- Resultat: Verlust von Satelliten im Wert von 500 Mio. USD
- Ironie der Geschichte
  - Das auslösende System war für den Flug nicht wichtig.
  - Aktivität während des Fluges Relikt aus Ariane-4-Zeiten.

Bug-Verwaltung

Bugs sind eine unvermeidliche Begleiterscheinung

UNI FREIBURG

Softwarefehler sind ein ernsthafter wirtschaftlicher Faktor

- Mehr als ein Drittel des IT-Budgets wird auf Fehlerbehebungen verwendet.
- Hauptproblem: Software, die bereits beim Anwender ist

Softwarefehler sind ein Problem in der Wissenschaft

- Auswertung und Interpretation hängt von Software ab.

Wichtig: Sinnvoller Umgang mit Softwarefehlern

- Systeme zur Erkennung und Behandlung von Fehlern
- Fehler immer ernst nehmen

Warum eine Bug-Verwaltung?

- Um den Überblick zu behalten.
- Um einen Ort zu haben, wo alle Bugs auflaufen.
- Um dem jeweiligen Berichter ein Maximum an Transparenz zu gewährleisten.
- Um Bugs zu archivieren – damit lassen sich doppelte Berichte handhaben.

☛ Um den Programmierer zu entlasten.

☛ Lässt sich bis zu einem gewissen Grad für „Feature Requests“ verwenden.

Beispiele für freie Bugtracker

- Reine Bugtracker
  - Bugzilla
  - Mantis
- Projektmanagement-Systeme
  - Trac
  - Roundup
  - Redmine

☛ Projektmanagement-Systeme bringen meist eine Integration für VCS und ein eigenes Wiki mit.

☛ Reine Bugtracker sind modularer verwendbar.

Was gehört in einen brauchbaren Bug-Report?

- Möglichst präzise Fehlerbeschreibung
  - Kurze Beschreibung des Nutzers
  - Fehlerausgabe des Programms
- Betriebssystem
  - Name
  - Version, Service Pack, etc.
- Version des verwendeten Programms
  - Beispiel Matlab: Matlab-Version (2012a, 2013b, ...)
  - Beispiel PHP: installierte PHP-Version
  - Voraussetzung: Nutzer muss an diese Information kommen

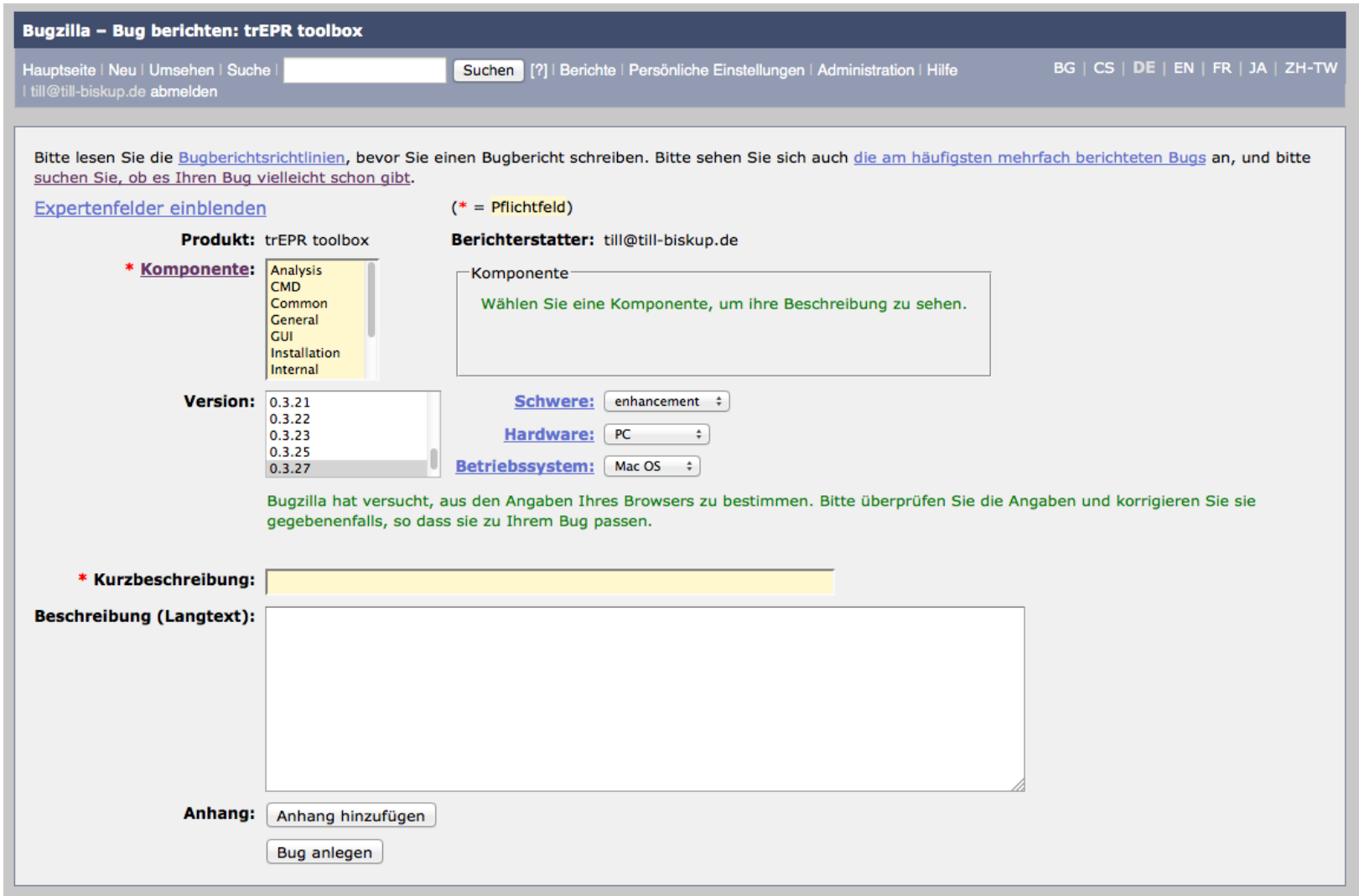
Was gehört in einen brauchbaren Bug-Report? (Fortsetzung)

- Version der verwendeten Toolbox
  - Setzt eine sinnvolle Funktion voraus, diese Information anzuzeigen
- Kontext des Fehlers
  - Komponente, in der der Fehler auftrat (IO, CLI, GUI, ...)
  - durchgeführte (oder beabsichtigte) Aufgabe
- Schwere des Fehlers
  - blocker, critical, major, minor, enhancement
  - Hilft bei der Einschätzung der Wichtigkeit und Dringlichkeit

☛ Lässt sich alles bei BugZilla eintragen.

Bug-Verwaltung

BugZilla - Beispiel für die Maske zum Anlegen eines Bugreports

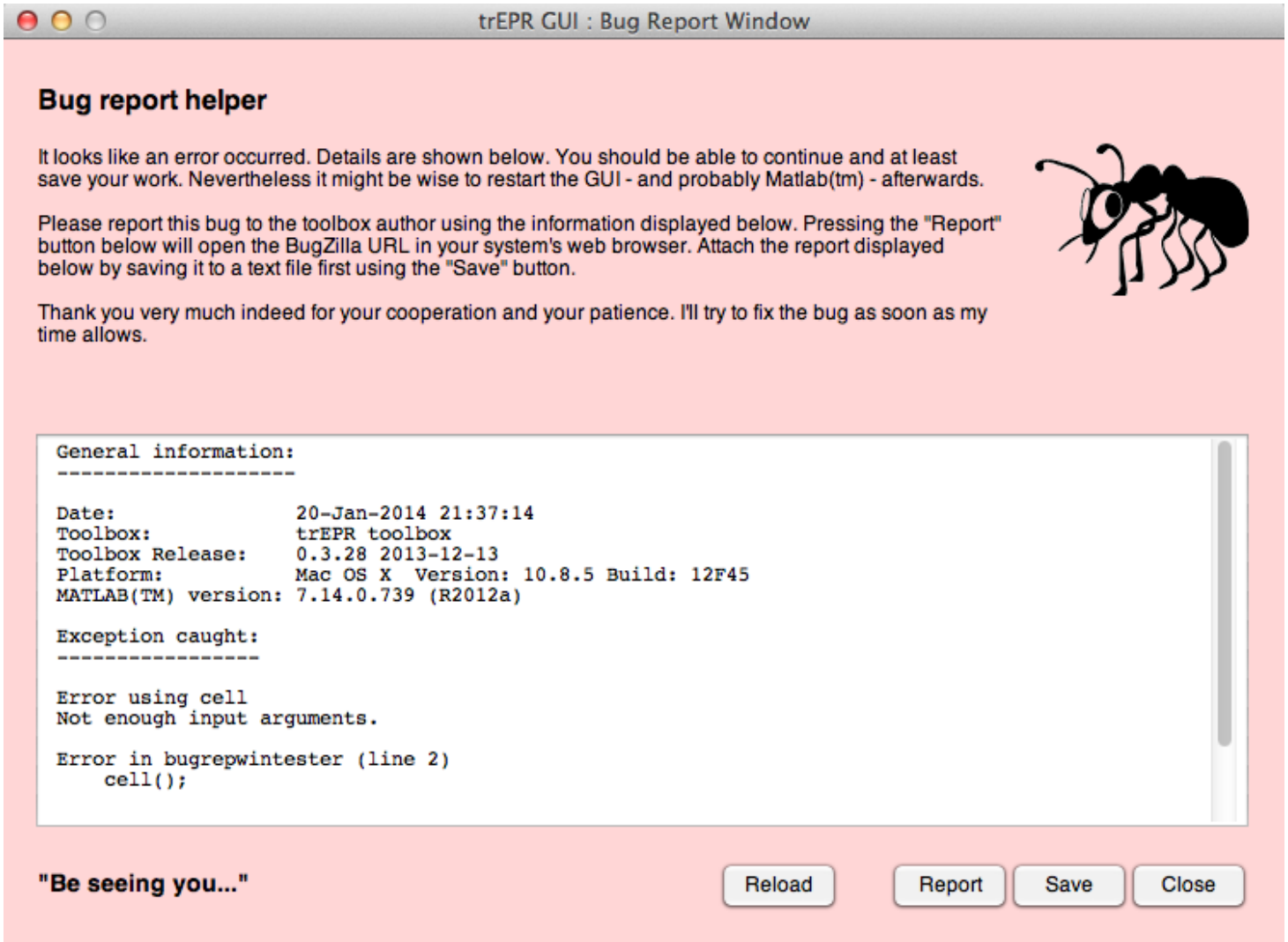


Wie bekommt man als Entwickler brauchbare Fehlerberichte?

- So einfach wie möglich für den Nutzer
  - Offensichtliche Hinweise auf die Bugverwaltung
  - GUIs: Bugverwaltung im Browser per Klick öffnen
- So vollständig wie möglich
  - Funktion, die alle relevanten Informationen sammelt
  - Komplette Fehlerausgabe mit speichern („call stack“)
- Transparente und zügige Bearbeitung der Fehlerberichte
  - Bugverwaltungssoftware verwenden
  - Oft großer Effekt mit wenig Aufwand erzielbar
  - Jeden Fehlerbericht ernst nehmen
  - Kommentarfunktionen verwenden für Rückfragen etc.

Bug-Verwaltung

Ein Beispiel für eine Hilfsfunktion



Bug-Verwaltung

Eine vernünftige Fehlerkultur verbessert Programme

UNI FREIBURG

Bugs sind eine unvermeidliche Begleiterscheinung

- Bugs sind normal und (meist) keine Katastrophe.
- Je früher Fehler erkannt werden, desto besser.

Fehlerberichte so einfach wie möglich machen

- Anwender sind (wie alle Menschen) faul.
- Nur Einfaches und Intuitives wird genutzt.

Motivation und Ermutigung der Anwender

- Verantwortung des Entwicklers
- Zügige Rückmeldung und Behebung

Infrastruktur (allg.)

personelle, sachliche und finanzielle Ausstattung,
um ein angestrebtes Ziel zu erreichen

Ziel: Das Leben erleichtern

- Nutzen muss für die Anwender klar ersichtlich sein

☞ Eigene Erfahrung:
Infrastruktur wirkt befreiend und steigert die Produktivität

Schubert, Klein: Das Politiklexikon. 5., aktual. Aufl. Bonn: Dietz 2011

Ausblick

Das war's. Wie geht's weiter?

UNI FREIBURG

So long, and thanks for all the fish.

Vorschau: Robuster und schneller Code

- Robustheit
- Schnelligkeit

Douglas Adams