

Ein Erfahrungsbericht

Erfolgreicher Umstieg auf Git

René Preißel (eToSquare)

Nils Hartmann (Techniker Krankenkasse)

VORSTELLUNG



René Preißel | Freiberuflicher
Softwarearchitekt, Entwickler und Trainer

Co-Autor des Buchs „Git: Dezentrale Versionsverwaltung
im Team – Grundlagen und Workflows“

Kontakt: rene.preissel@etosquare.de

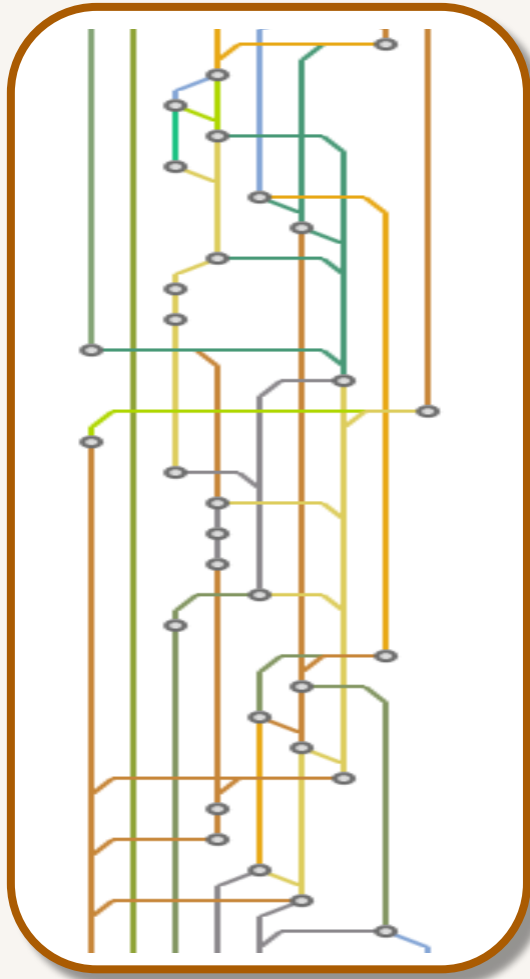


Nils Hartmann | Java-Softwareentwickler,
Techniker Krankenkasse

Schwerpunkte: OSGi, Eclipse und Build-Management

Kontakt: [nils@nilshartmann.net](mailto:nilshartmann.net)

AGENDA



- Ausgangssituation
- Git vs Synergy
- Repositories & Branches
- Entwicklungsprozess mit Git
- Tooling
- Einführung
- Fragen & Diskussionen

CONCEPT PEOPLE IT-TALK

Ausgangssituation

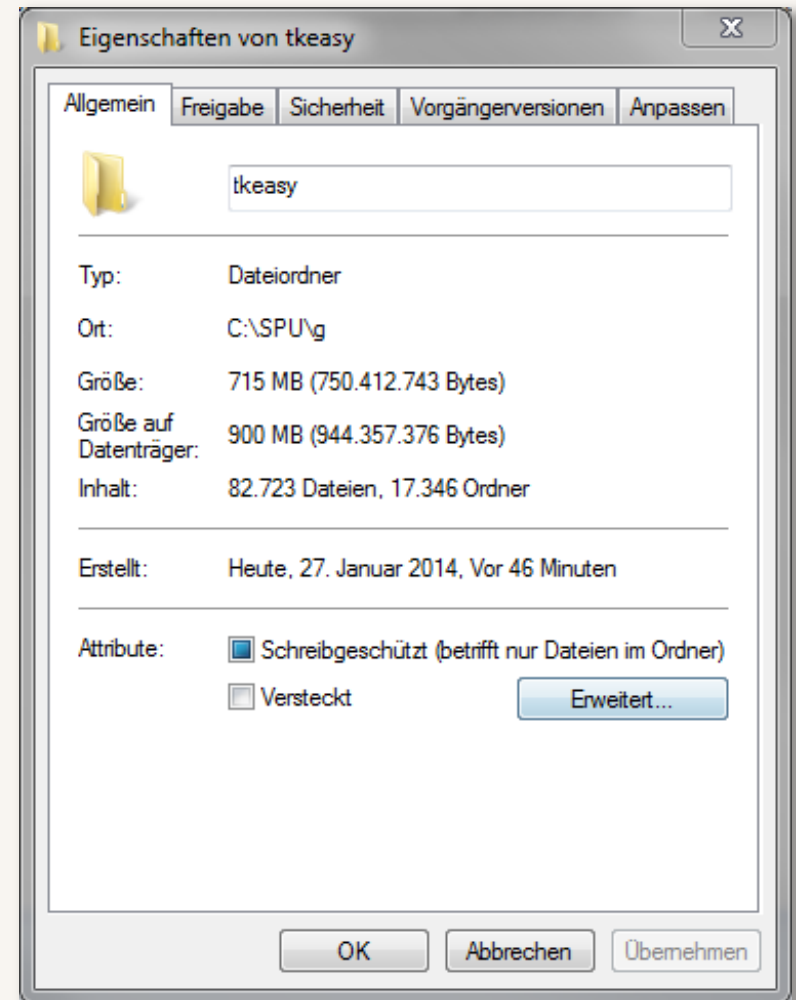
AUSGANGSSITUATION: TKEASY

TKeasy

- 1 Software • 120 Produkte
- 65.000 Java Klassen
- 700 MB Code
- 120 Entwickler

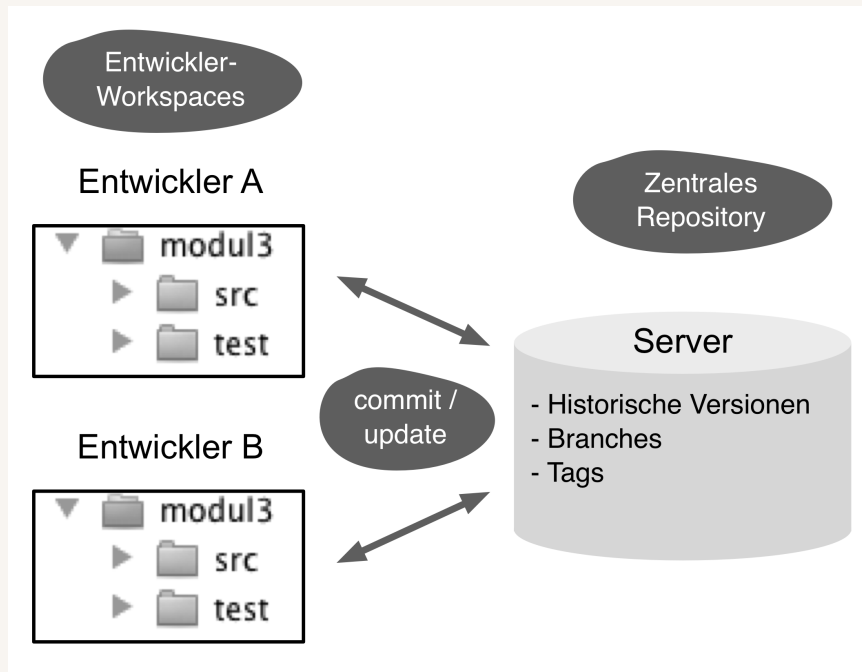
Regelmäßige Releases

- 5-6 Major Releases pro Jahr
- 1 Bugfix pro Woche



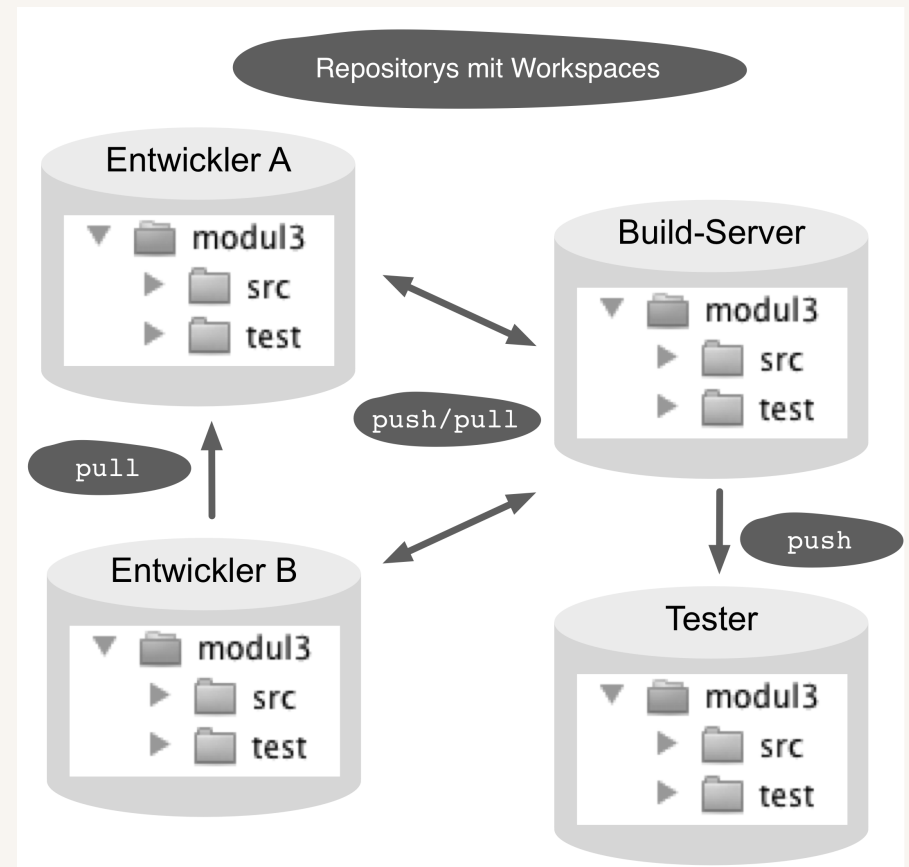
Git vs. Synergy

GIT – DEZENTRALE VERSIONSVERWALTUNG

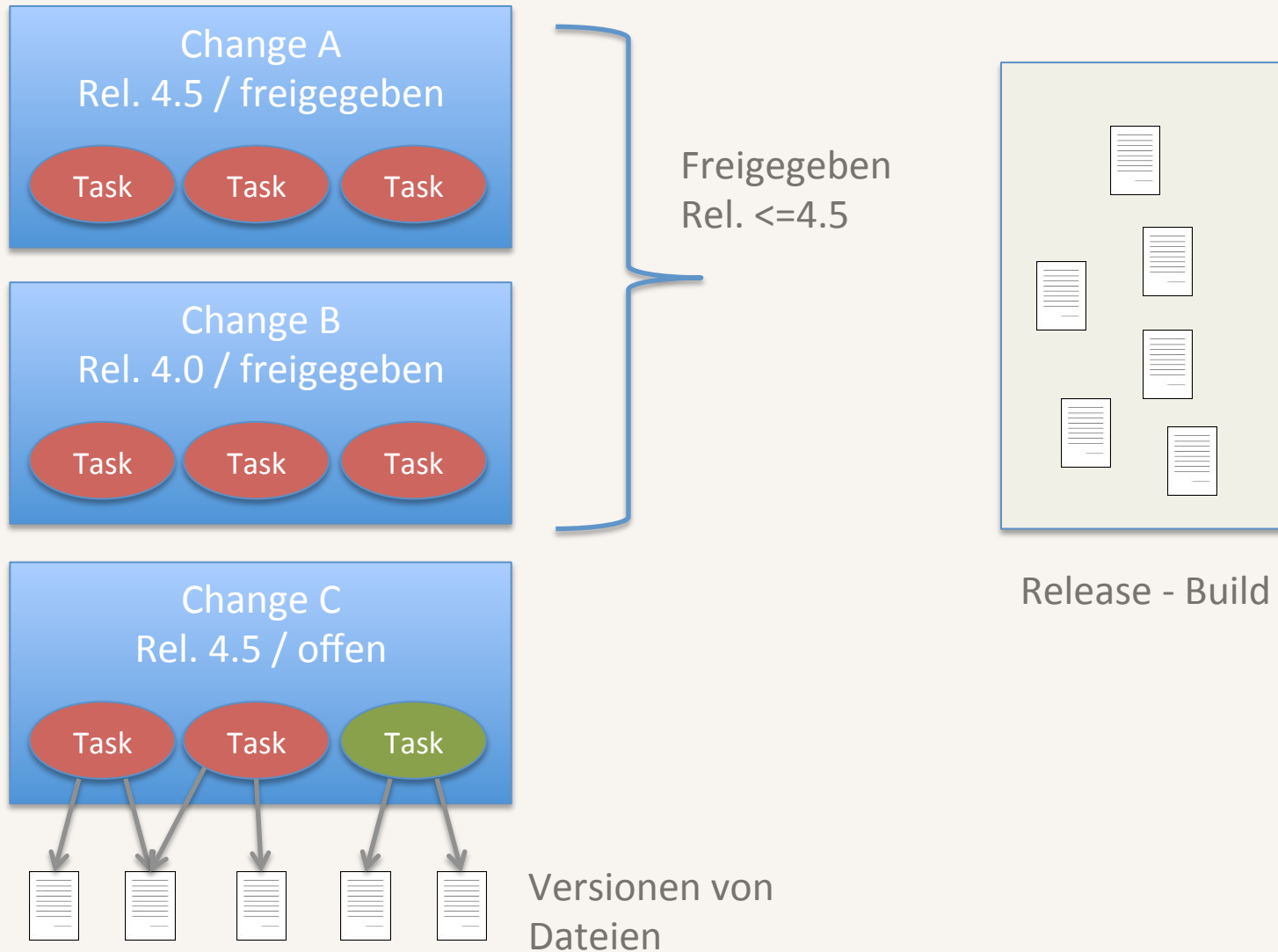


Zentraler Ansatz, z.B.
Subversion und **Synergy**

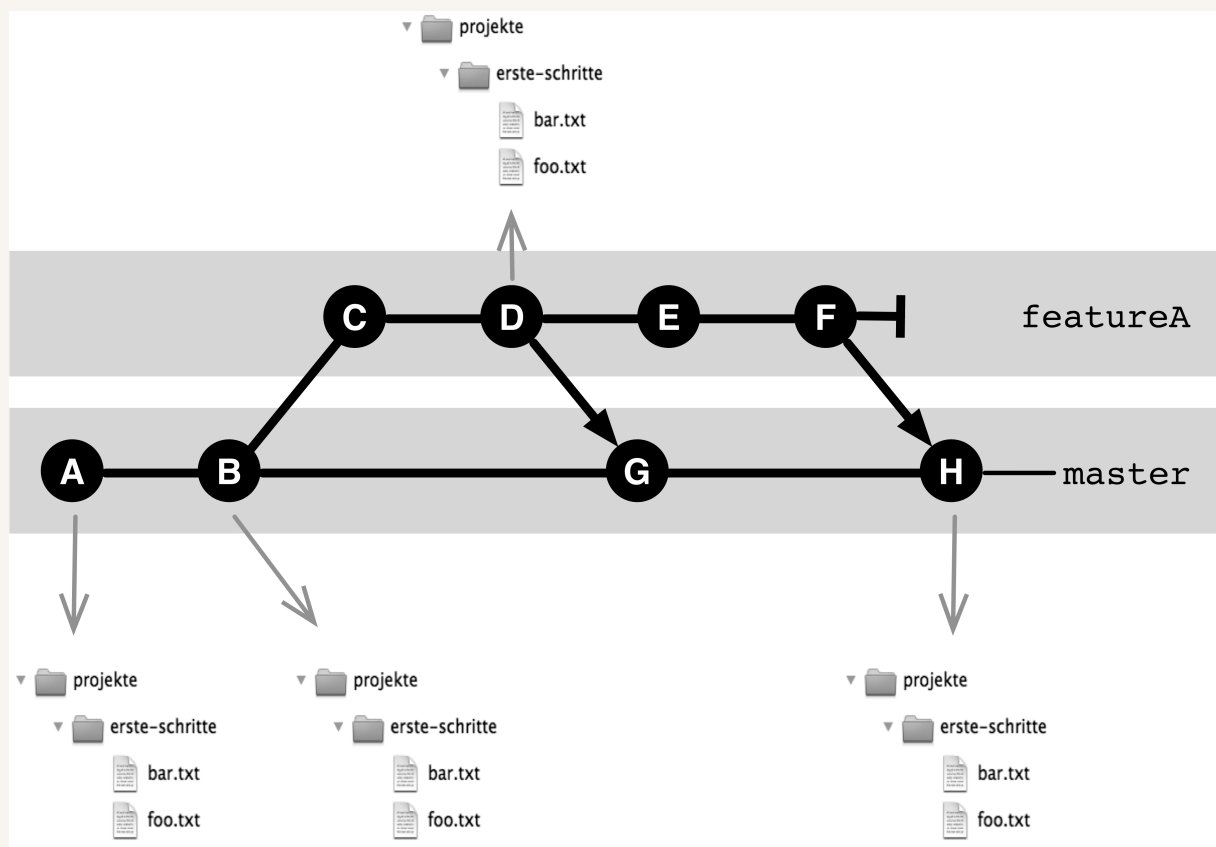
Dezentraler Ansatz



ZUSTÄNDE UND QUERIES IN SYNERGY



COMMITTS UND BRANCHES IN GIT



- Jedes Commit versioniert das gesamte Projekt
- Integrierte Änderungen müssen explizit ausgebaut werden

Repositories & Branches

REPOSITORIES – ANFORDERUNGEN

700 MB Source Code in einem Repository . . .

- + Einfache Verwaltung
- + Sehr gut für zentralen Build

- Git-Operationen dauern lange (klonen, git status etc)
- Keine feingranularen Berechtigungen möglich
- Sehr viele Commits => komplexe Historie

120 Repositories für 120 Produkte . . .

- + Git-Operationen gehen sehr schnell

- Hoher Verwaltungsaufwand
- Probleme beim Refactoring

REPOSITORIES - ANFORDERUNGEN

Verwaltung zentrale Repositories

- Weboberfläche
 - Source Code Suche
 - Verlinkung aus Stacktraces etc
- Rechteverwaltung
- Qualitätssicherung / Code Review
- REST API / Plug-ins

REPOSITORY-VERWALTUNG: ALTERNATIVEN

GitHub Enterprise

- Volltext-Suche
- (Eingeschränkte) Rechteverwaltung

Gerrit Code Review

- „Amend-basierter“ Workflow
- Konfigurierbare Code-Review-Regeln

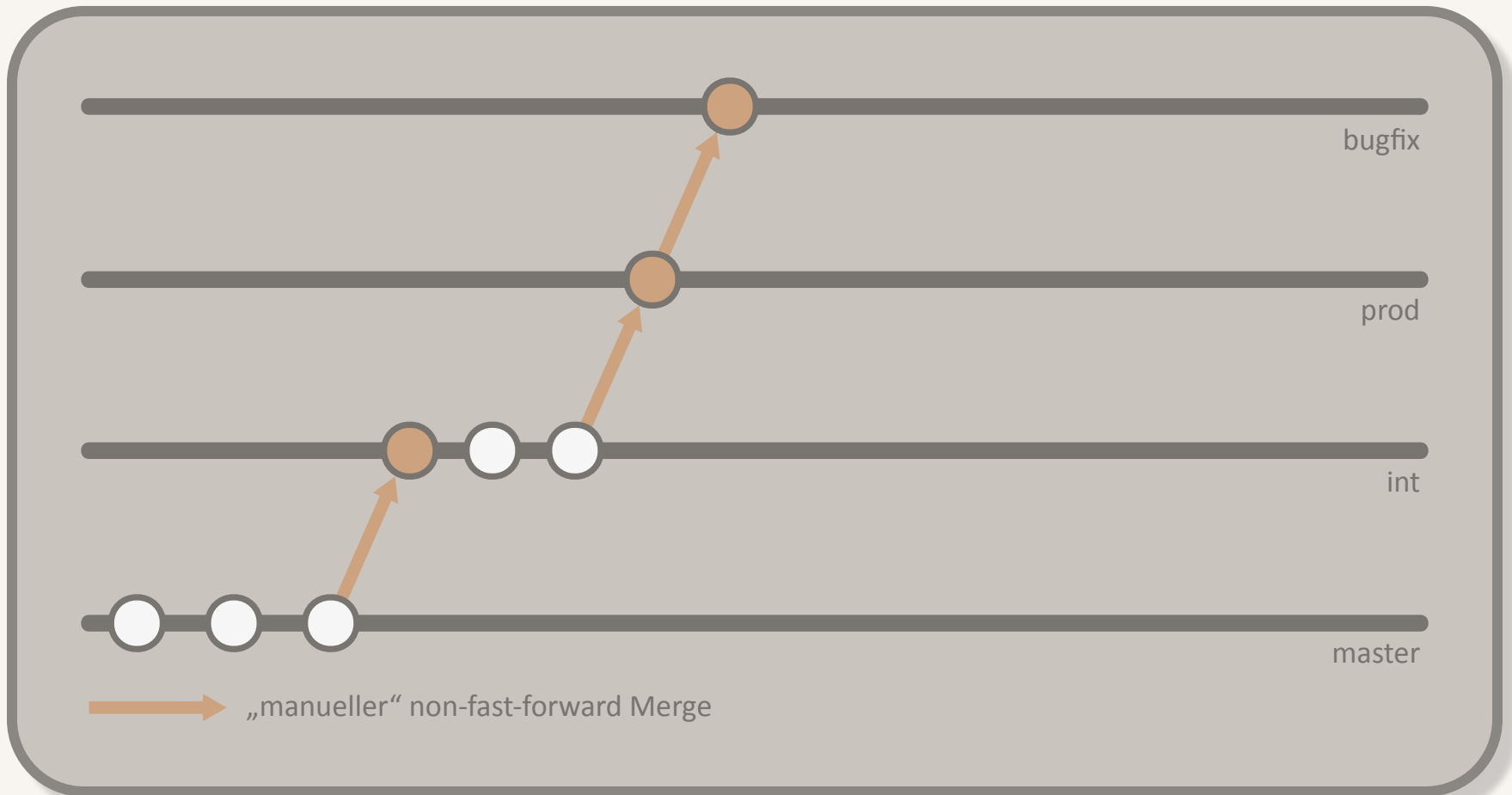
Atlassian Stash

- Feingranulares Berechtigungssystem
- Konfigurierbare Code-Review-Regeln
- Bei Projekt-Start noch nicht verfügbar



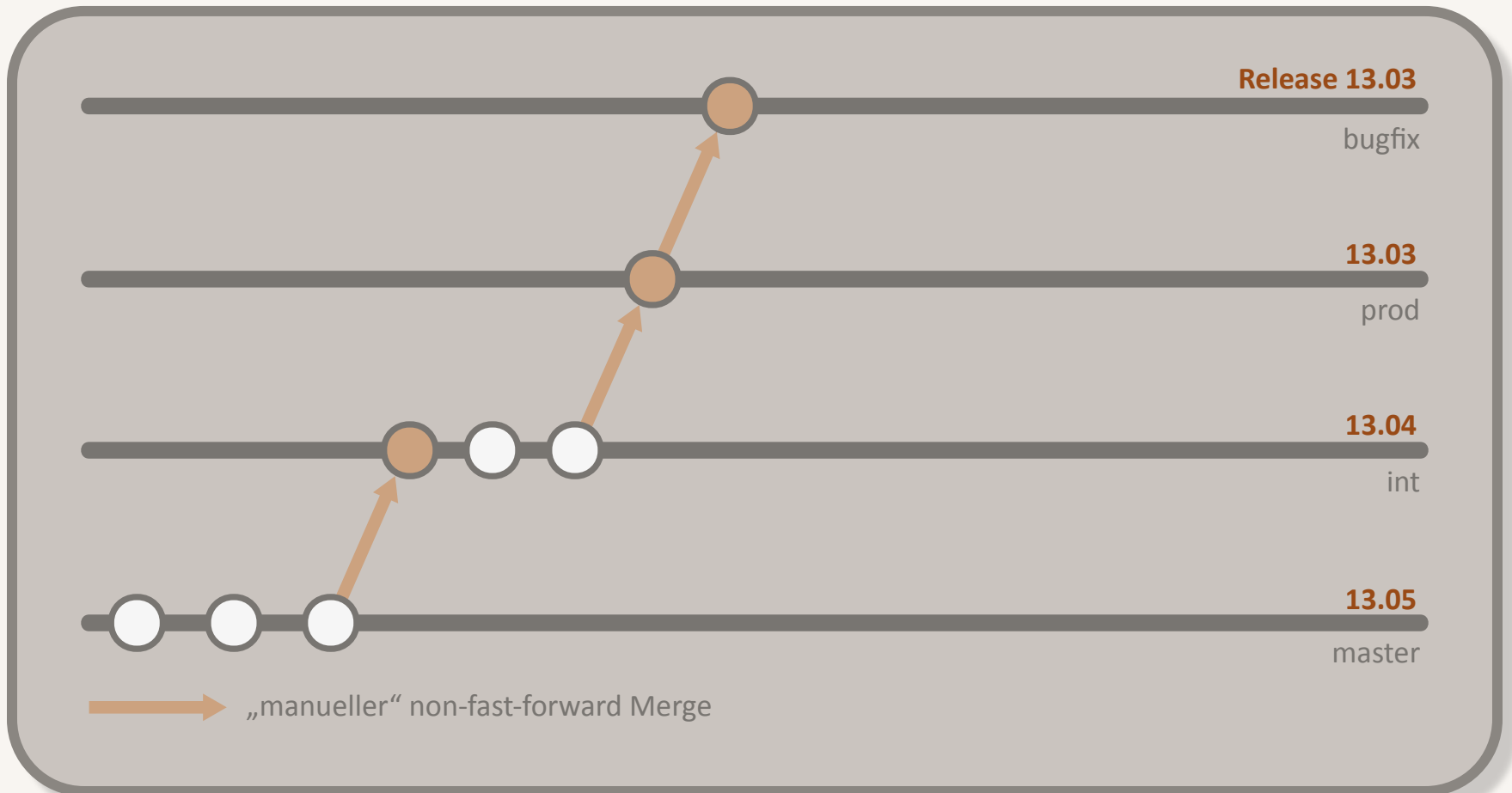
BRANCH MODELL „TK GIT FLOW“

Drei aktive Releases zeitgleich



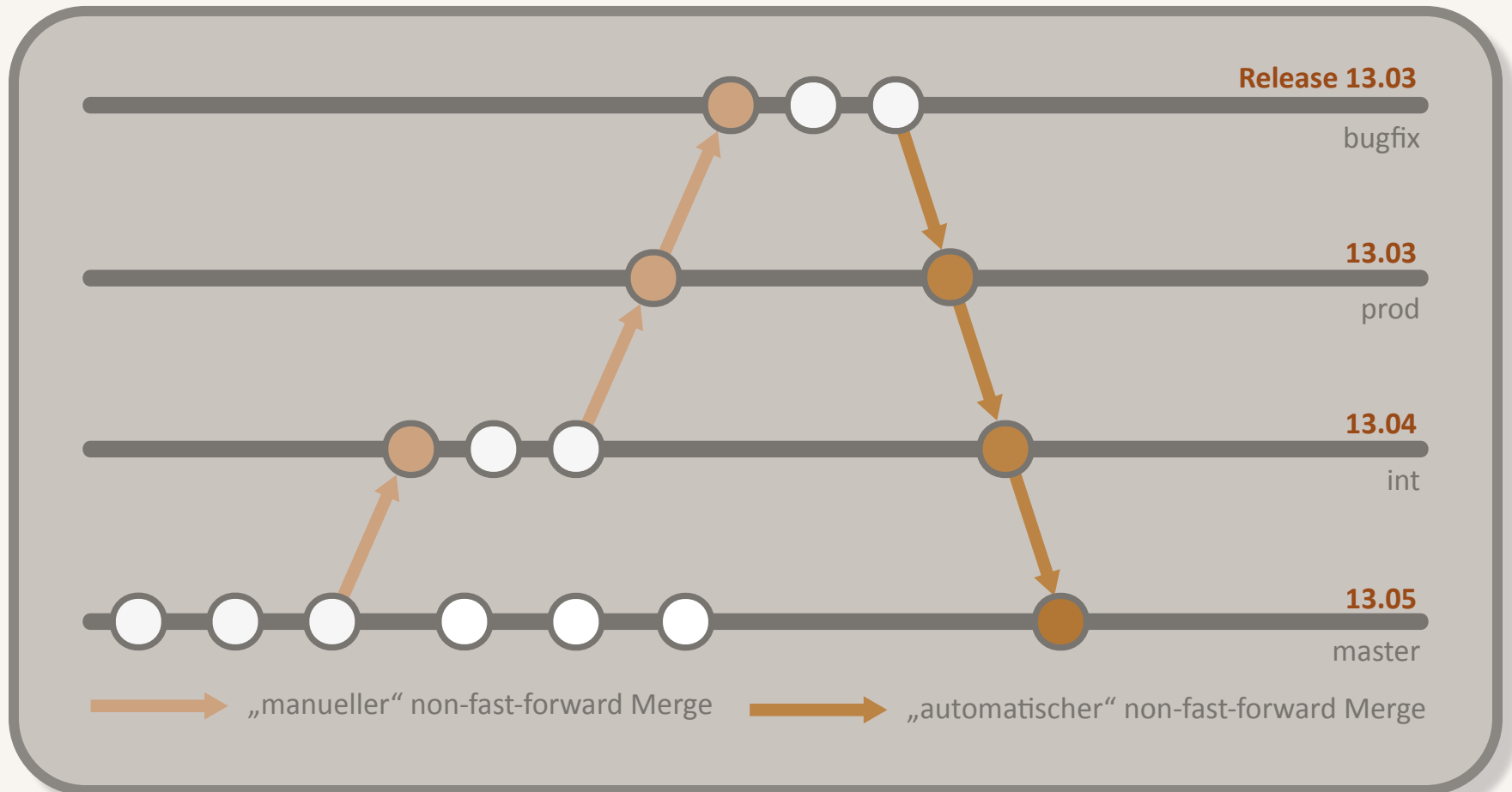
BRANCH MODELL „TK GIT FLOW“

Drei aktive Releases zeitgleich



BRANCH MODELL „TK GIT FLOW“

Drei aktive Releases zeitgleich



Entwicklungsprozess mit Git

TK ENTWICKLUNGSPROZESS

Keine Code-Änderung ohne Change

- Changes müssen freigegeben werden
- Changes müssen isoliert entwickelt werden
- Jeder Change benötigt Review

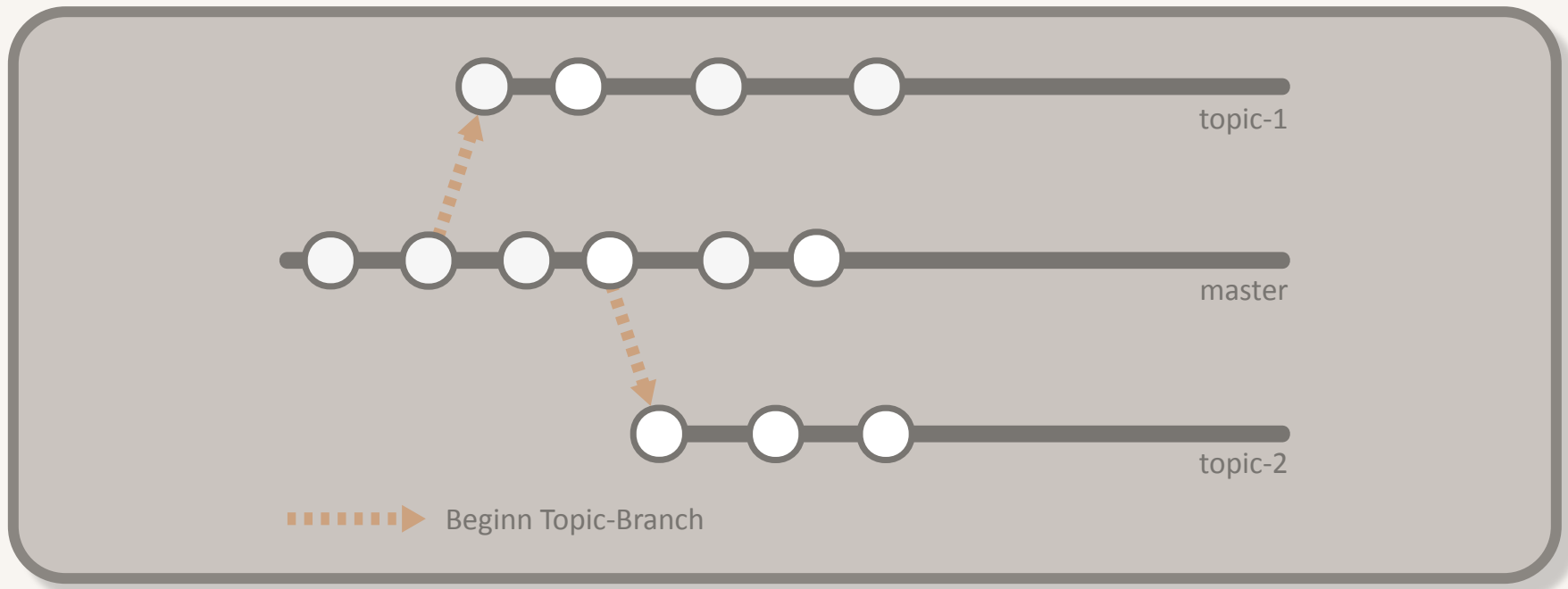
Synergy

- Integrierte Change Verwaltung
- Freigaben über Change Zustand
- Einsammeln über Query

EINEN CHANGE BEARBEITEN 1

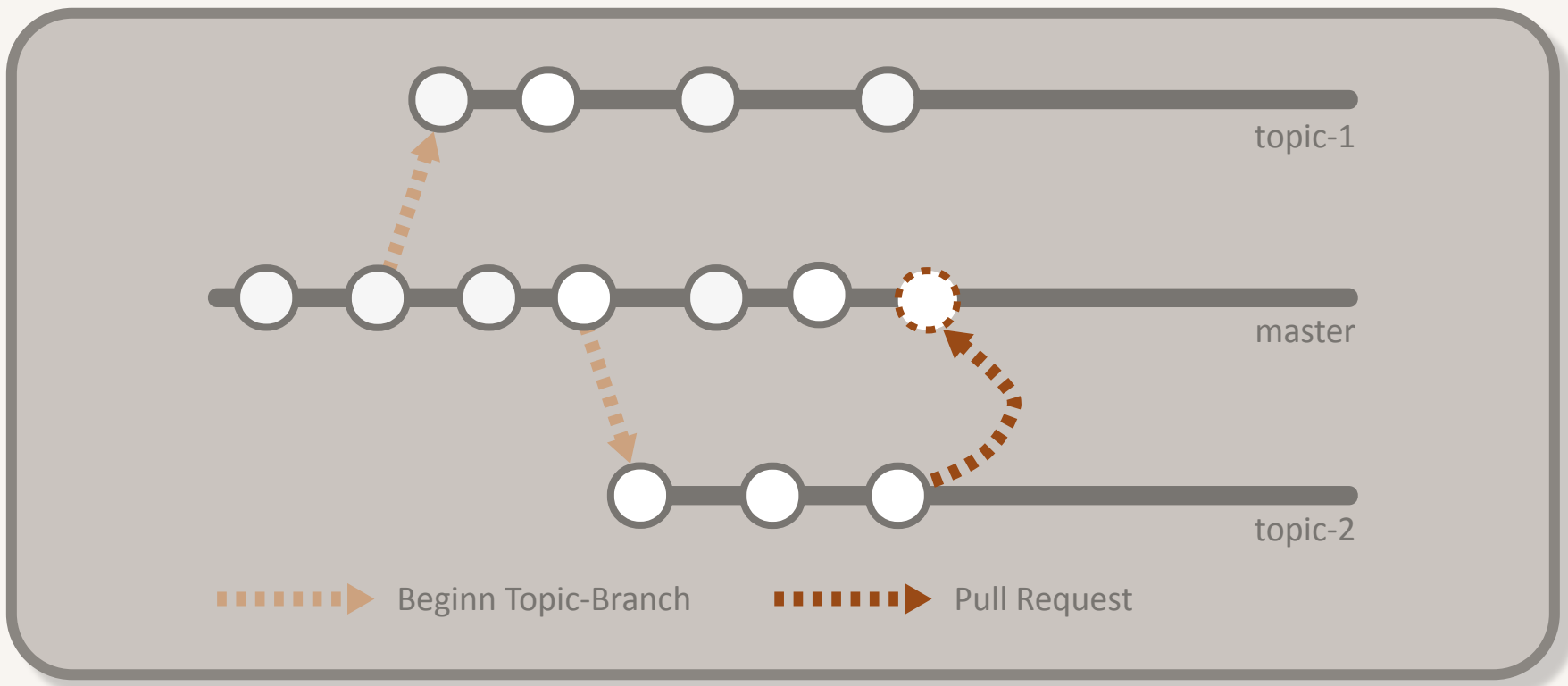
Change anlegen und aktivieren

- Verwaltung der Changes in Atlassian Jira
- Change wird auf Topic-Branch entwickelt



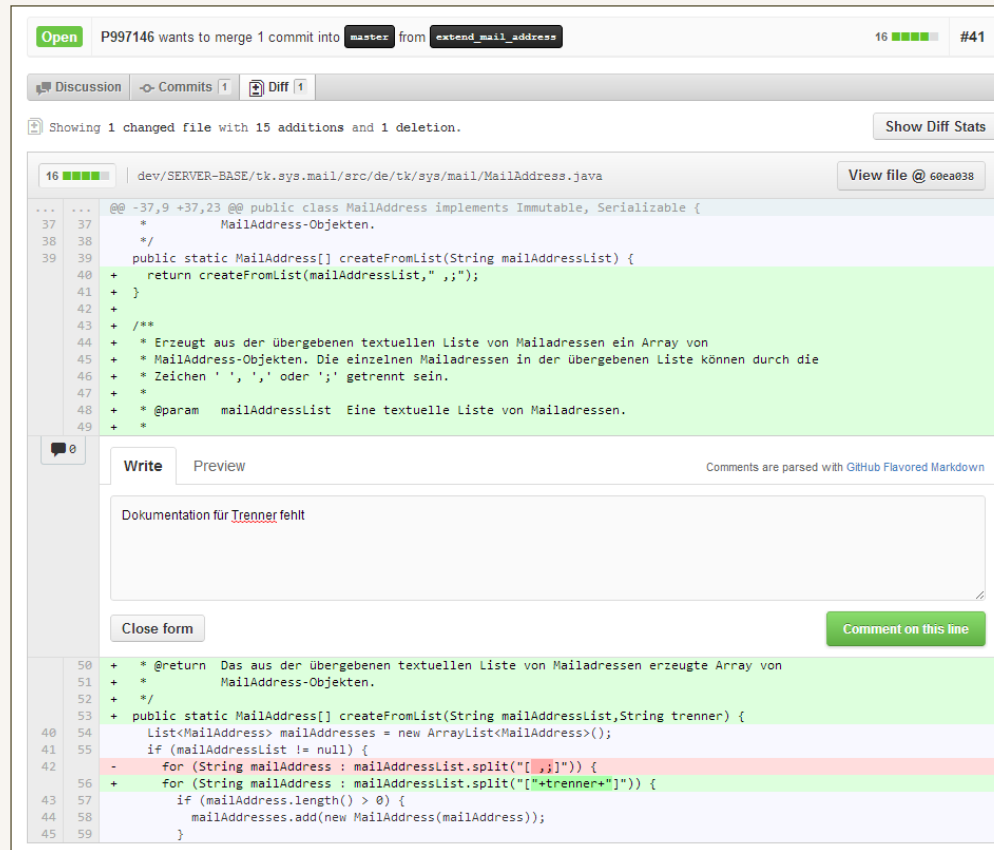
EINEN CHANGE BEARBEITEN 2

Change abgeben: Pull Request



EINEN CHANGE BEARBEITEN 3

Pull Requests mit GitHub



The screenshot displays a GitHub Pull Request interface. At the top, it indicates that user P997146 wants to merge 1 commit into the `master` branch from the `extend_mail_address` branch. The pull request is identified by the number #41. Below this, there are tabs for Discussion, Commits, and Diff. The Diff view shows 1 changed file with 15 additions and 1 deletion. The file path is `dev/SERVER-BASE/tk.sys.mail/src/de/tk/sys/mail/MailAddress.java`. The diff shows changes to the `createFromList` method, including a new parameter `trenner` and updated documentation. A comment box is visible below the diff, containing the text "Dokumentation für Trenner fehlt".

```
... @@ -37,9 +37,23 @@ public class MailAddress implements Immutable, Serializable {
37 37     *   MailAddress-Objekten.
38 38     */
39 39     public static MailAddress[] createFromList(String mailAddressList) {
40 +   return createFromList(mailAddressList, ",");
41 + }
42 +
43 + /**
44 +  * Erzeugt aus der übergebenen textuellen Liste von Mailadressen ein Array von
45 +  * MailAddress-Objekten. Die einzelnen Mailadressen in der übergebenen Liste können durch die
46 +  * Zeichen ' ', ',' oder ';' getrennt sein.
47 +  *
48 +  * @param mailAddressList Eine textuelle Liste von Mailadressen.
49 +  *
50 +  * @return Das aus der übergebenen textuellen Liste von Mailadressen erzeugte Array von
51 +  *   MailAddress-Objekten.
52 +  */
53 + public static MailAddress[] createFromList(String mailAddressList, String trenner) {
54     List<MailAddress> mailAddresses = new ArrayList<MailAddress>();
55     if (mailAddressList != null) {
56 -     for (String mailAddress : mailAddressList.split("[,;]")) {
57 +     for (String mailAddress : mailAddressList.split("[*+trenner+]")) {
58         if (mailAddress.length() > 0) {
59             mailAddresses.add(new MailAddress(mailAddress));
60         }
61     }
62 }
```

- Automatischer Build, QS, Tests

EINEN CHANGE BEARBEITEN 4

Freigabe des Changes

- Freigabe über den Jira-Vorgang
- Beim Deployment werden Freigaben überprüft

```
git log int..prod | grep Jira-Id
```

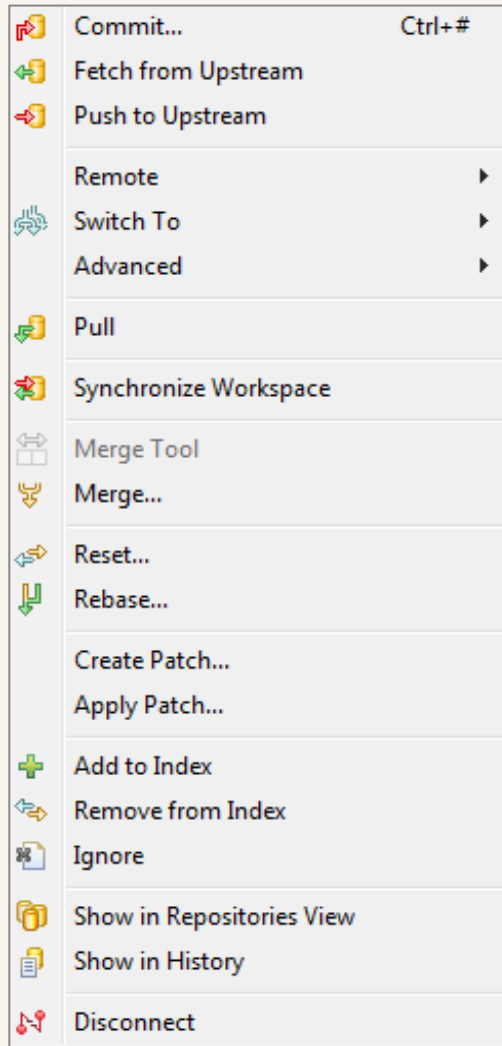
Bei Nicht-Freigabe eines Changes

- Änderungen müssen ausgebaut werden
- Git Reverse Commit

CONCEPT PEOPLE IT-TALK

Tooling

TOOLING FÜR ENTWICKLER



Git-Integration in Eclipse IDE

Egit – Git Team Provider

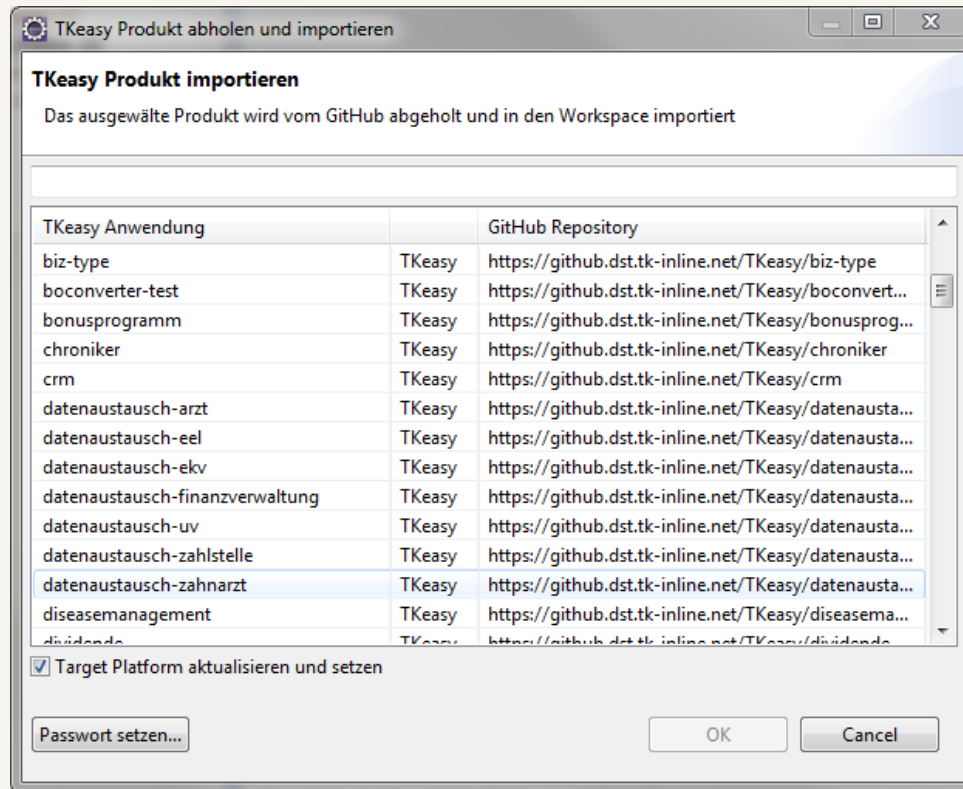
- Sehr „technisch“
- Kein Workflow
- Fehlerhafte Merges

Mylyn – Taskverwaltung

- Topic-Branch aktivieren nicht implementiert

TOOLING FÜR ENTWICKLER

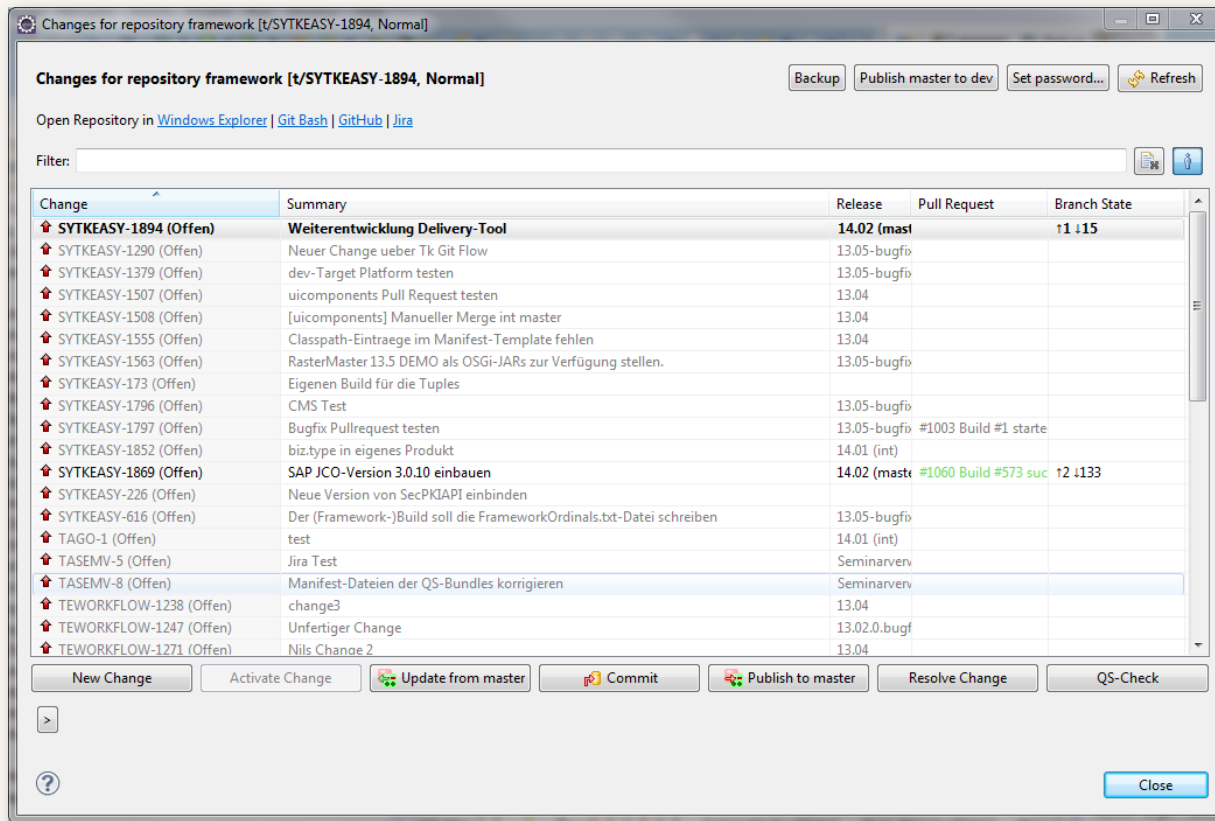
TK Git Flow – Importieren eines Produkts



- Authentifizieren und Klonen

TOOLING FÜR ENTWICKLER

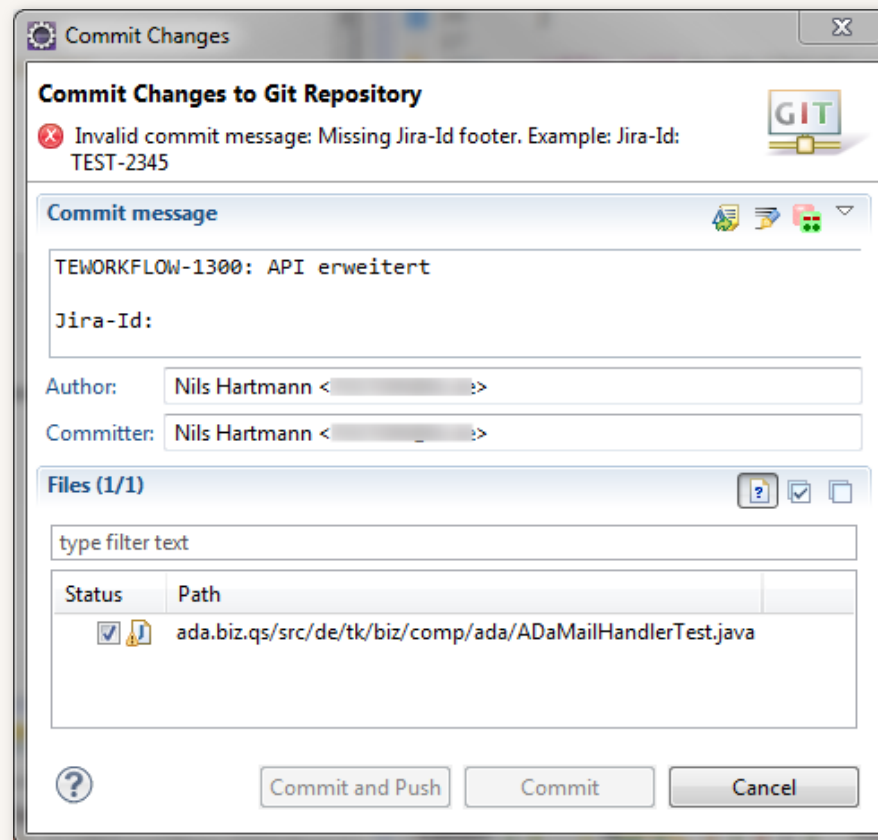
TK Git Flow – Anlegen, Beenden von Changes



- Branches anlegen und wechseln

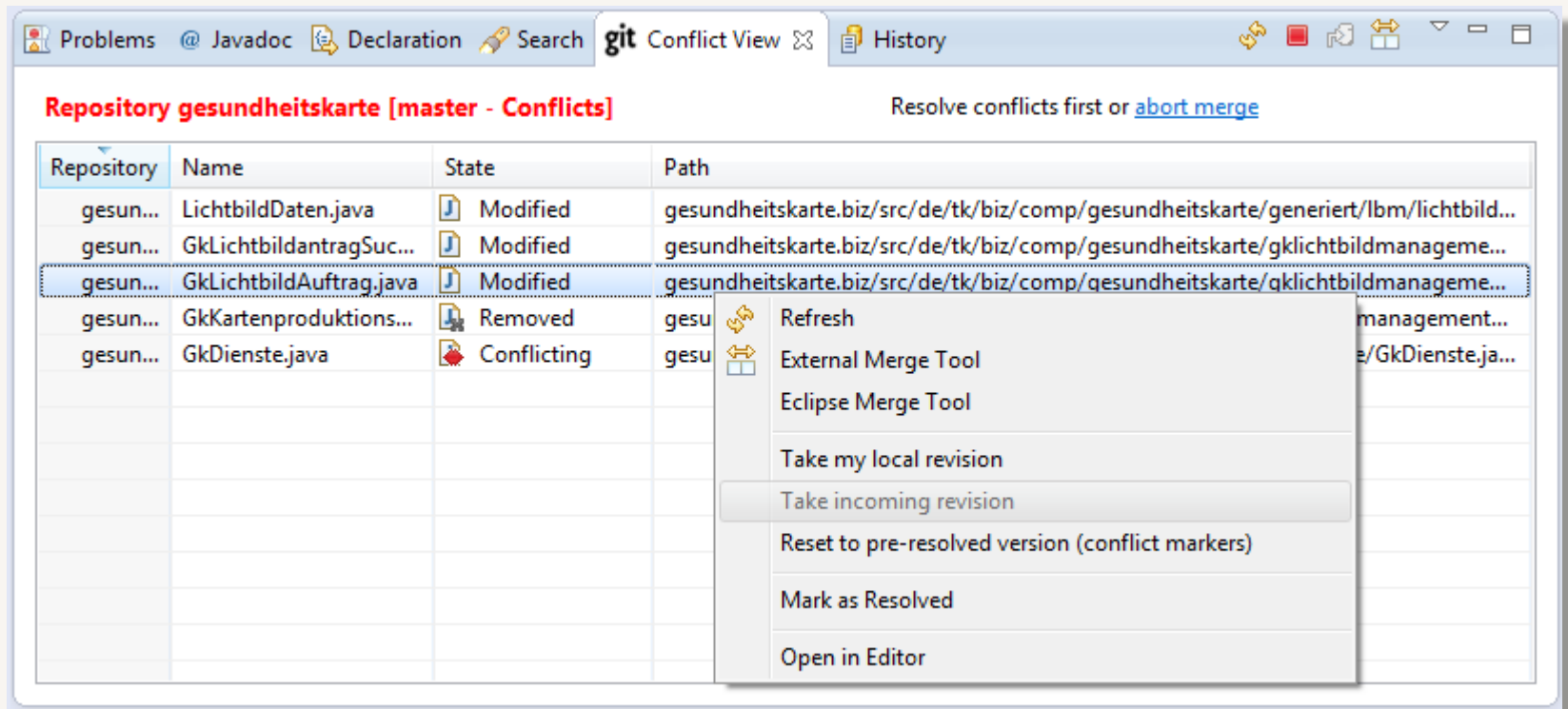
TOOLING FÜR ENTWICKLER

TK Git Flow – Jira-Git-Zuordnung



TOOLING FÜR ENTWICKLER

TK Git Flow – Merge-Konflikt-Support



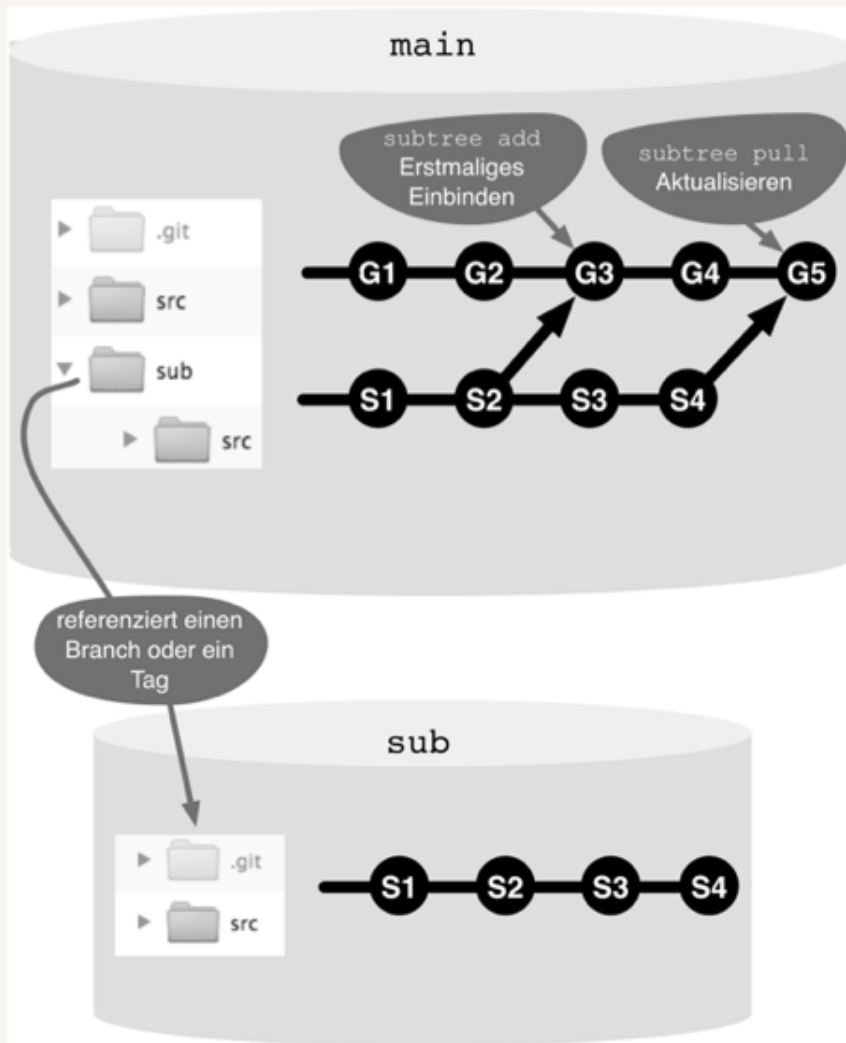
- CGit für Merge-Operationen

TOOLING FÜR BUILDMANAGEMENT

- Vollständiges Build benötigt alle Produkte und dauert lange
- Vor der Integration eines Topic-Branches soll ein Build inklusive Tests durchgeführt werden
- Schnelle Builds einzelner Produkte notwendig
 - Jedes Produkt kann separat mit den letzten Versionen abhängiger Produkte gebaut werden

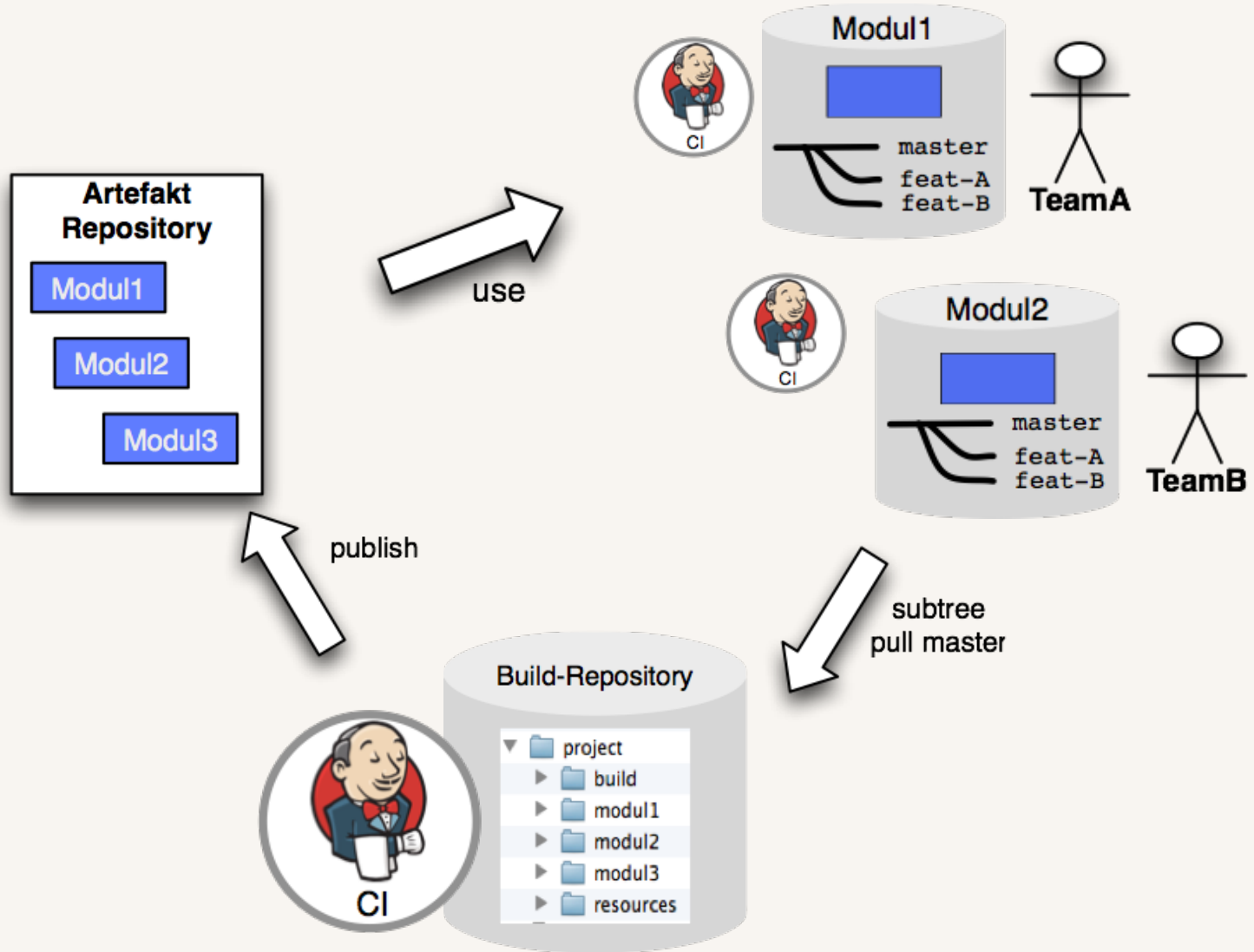
GIT SUBTREE

Zusammenführen von Repositories



- Vollständiges Build benötigt alle Produkte
- Git-Subtree wird für die Integration aller Produkte in ein Repository benutzt

PRODUKT-BUILD



TOOLING FÜR CHANGEMANAGEMENT

Pflege von 120 Git-Repositories & Jira-Projekten

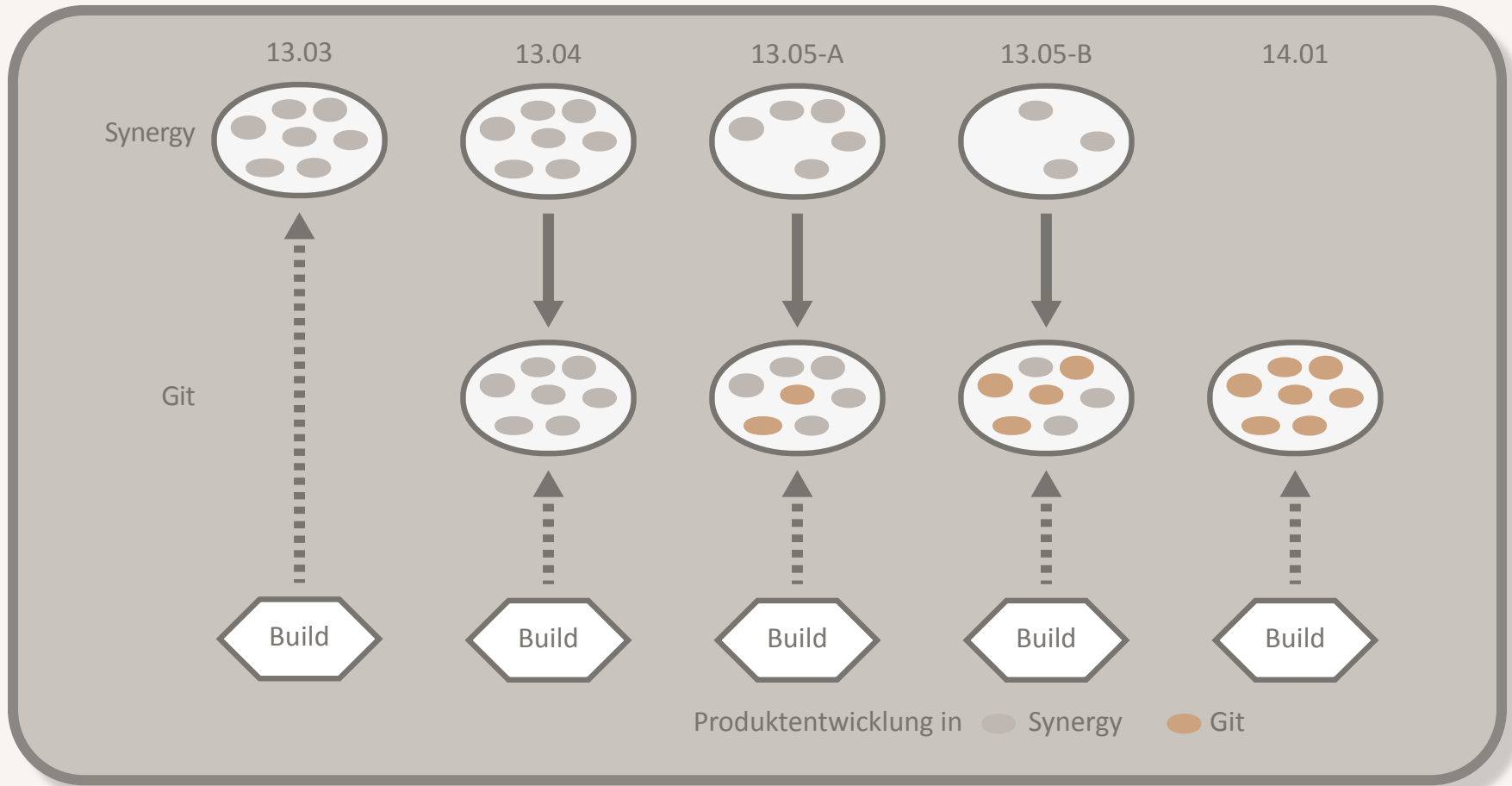
- Automatische Merges
- Taggen von Code-Ständen (Installierte Stände)
- Abgleich von Git und Jira
- Erzeugen und Pflegen von Jenkins Jobs

- Java, Python, Git Bash
- REST-APIs

Einführung & Fazit

MIGRATIONSPFAD

Kontinuierliche Übernahme



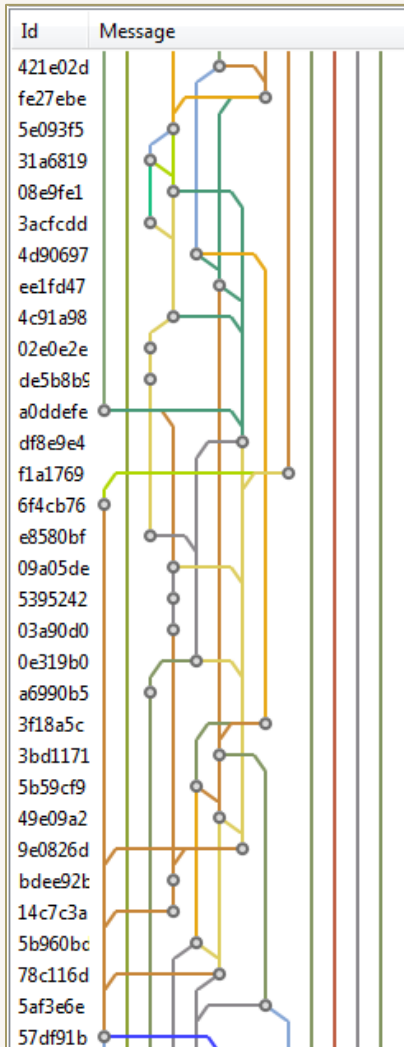
EINFÜHRUNG

- Beschränkung auf essentielle Git-Features
- Frühzeitige Ausbildung interner Experten
- Entwicklerschulungen kurz vor der Umstellung

- Verzicht auf Übernahme der gesamten Historie

Kein Big Bang

PROBLEME: VIELE NEUE KONZEPTE



Dezentrales Arbeiten

- Lokale vs Remote-Branched
- Commit vs Push

Komplexe Historie

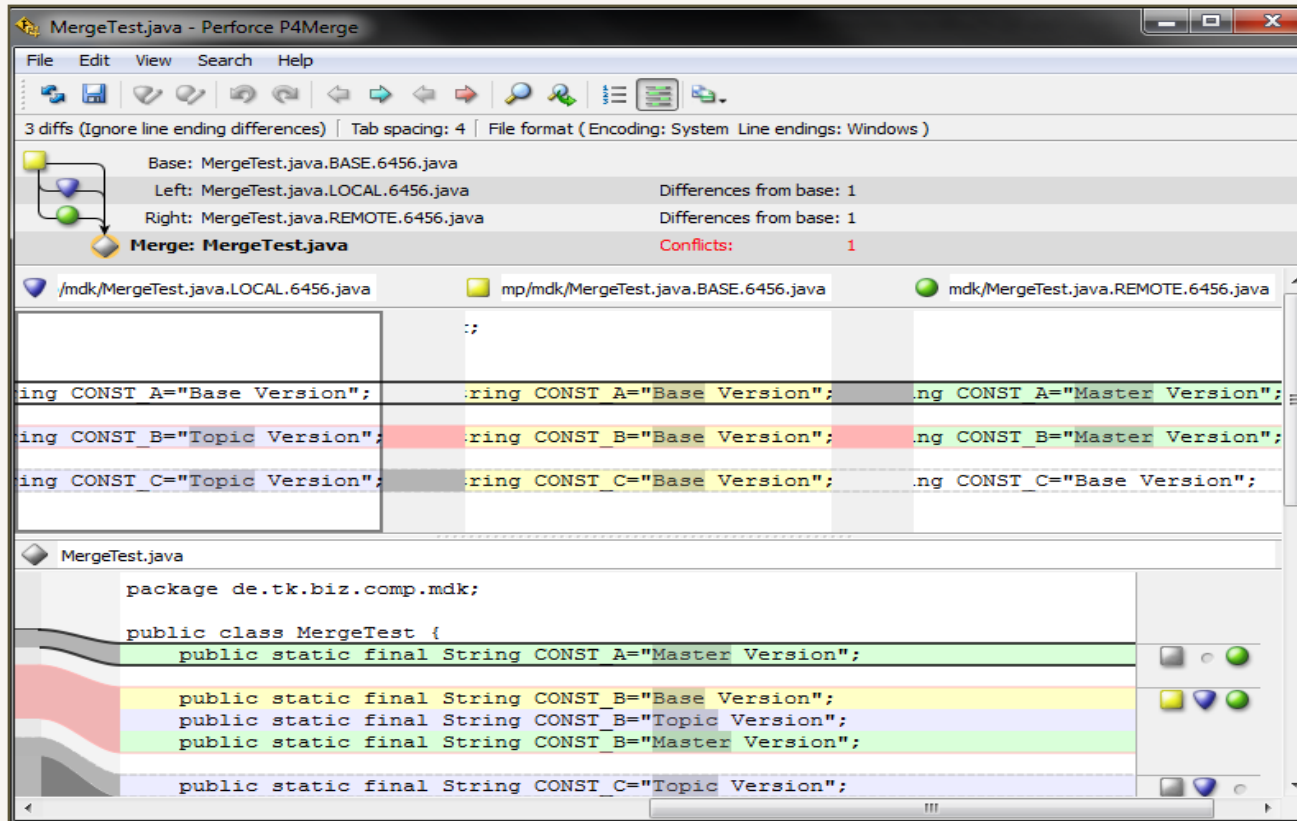
- „Wo ist mein Change?“

Wenig integrierter Tool-Stack

- Jira vs Git vs Jenkins...
- Viele potentielle Fehlerquellen

PROBLEME: MERGE-KONFLIKTE

Arbeiten mit (P4) Merge ungewohnt



- Synergy: pessimistische Sperren

FAZIT

- **Umstellung hat gut geklappt**
 - Einführung länger als geplant
 - Abstimmung der Prozesse aufwendiger als Umstellung der Tools
 - Praktische Umstellung hat gut funktioniert
- **Toollandschaft hat sich verbessert**
 - Weniger Bugs
 - Mehr Auswahl

Wenn man mit Git arbeiten will, muss man sich auf „Git-Philosophie“ einlassen

VIELEN DANK!

Fragen ?