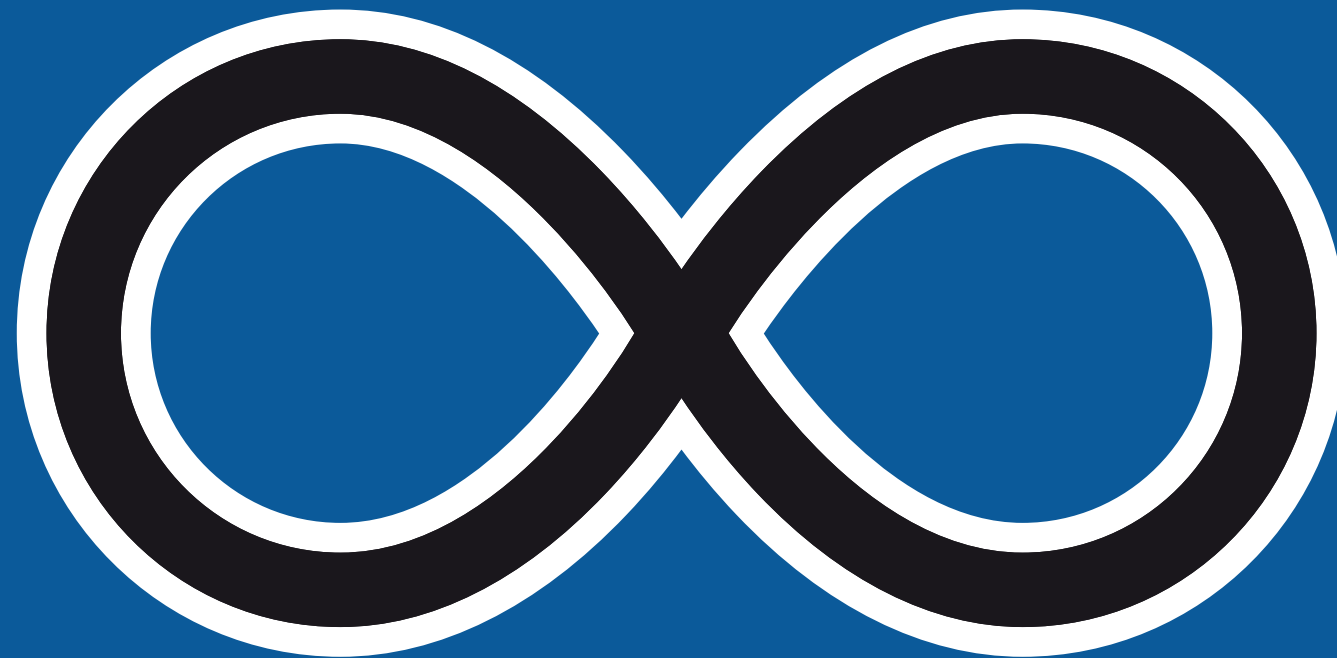


# SOFTWAREENTWICKLUNG

IM TEAM MIT OPEN-SOURCE-WERKZEUGEN

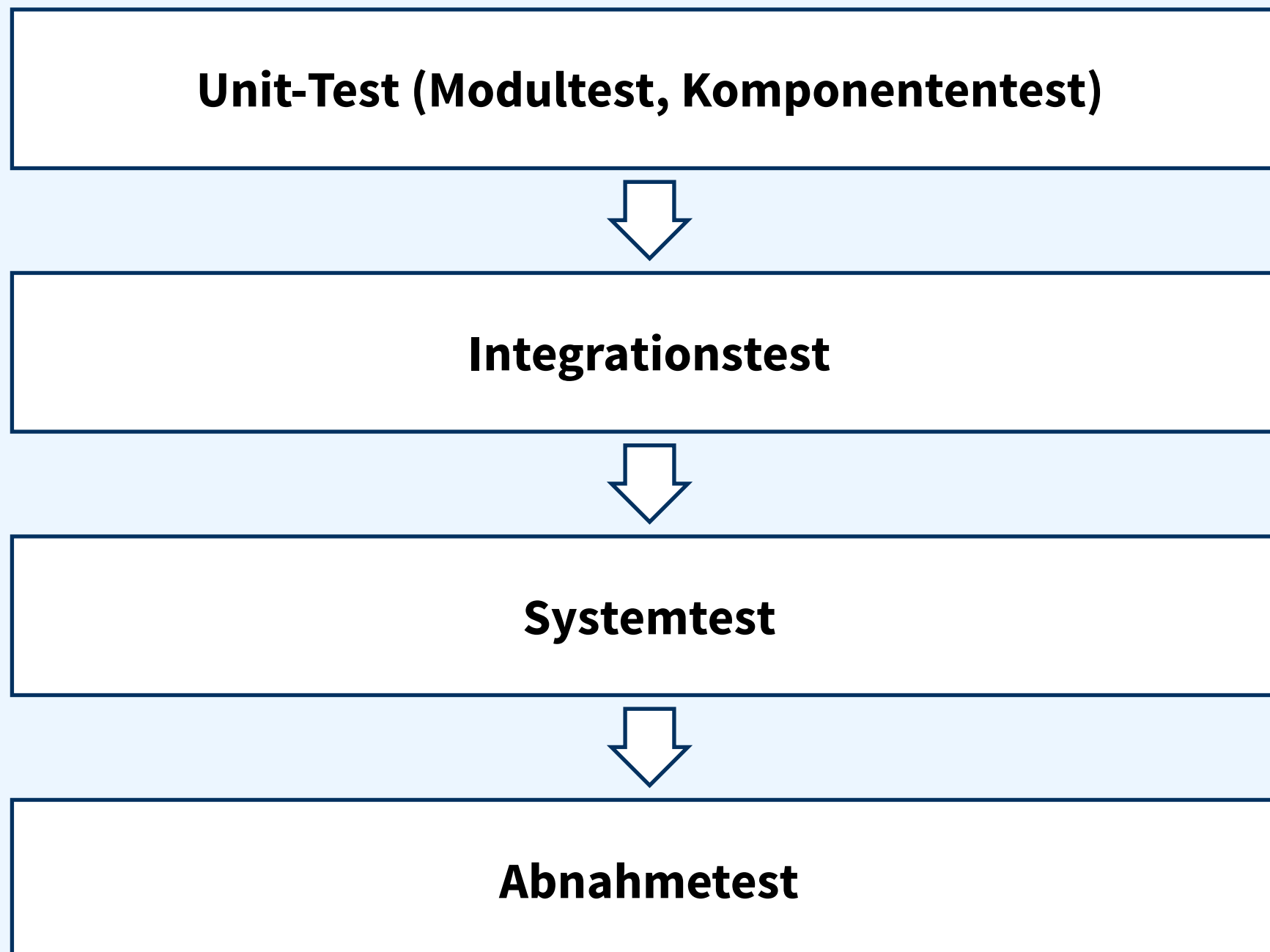
08 - Fehlermanagement



# WIEDERHOLUNG

# Prüfebenen

Tests lassen sich nach der Prüfebene in verschiedene *Teststufen* einteilen:

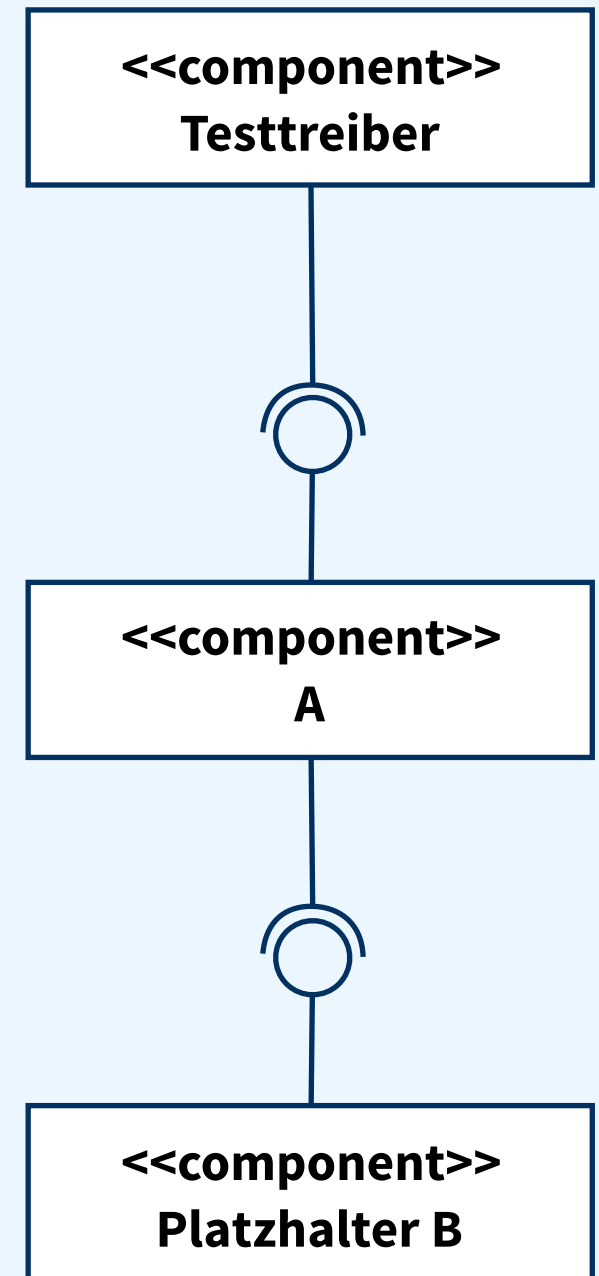


# Software-Integration

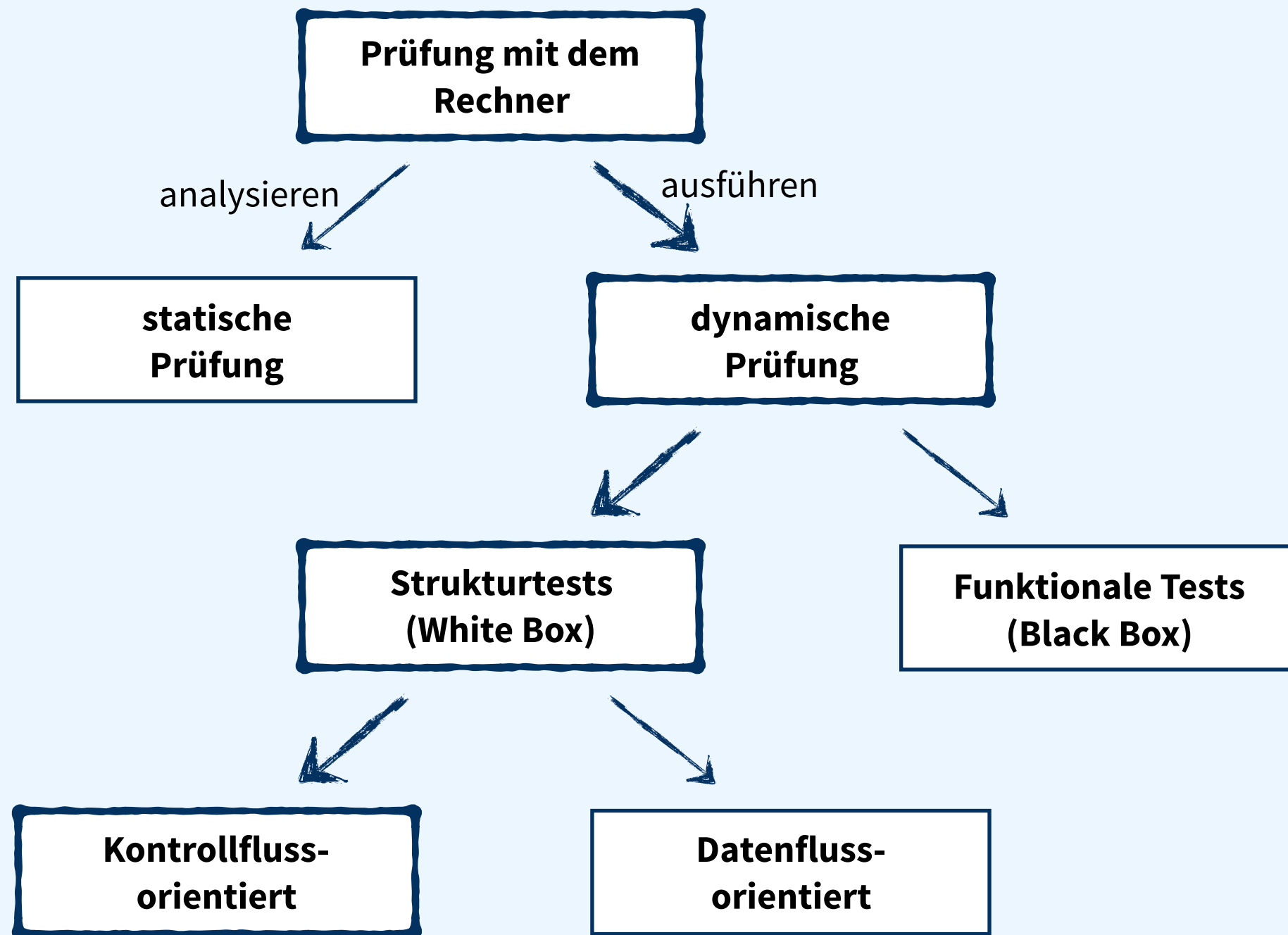
- Nachdem einzelne Module fertiggestellt und getestet wurden, müssen sie zu einem lauffähigen System zusammengebaut werden
- Der Aufwand für die Integration darf nicht unterschätzt werden, da Fehler an den Schnittstellen auftreten können
- Der Grund für diese Fehler ist häufig eine unvollständige oder inkonsistente Spezifikation
- Bezüglich des *Ablaufs* unterscheidet man:
  - ▶ die Integration in einem Schritt  
(Big-Bang-Integration → hohes Risiko!)
  - ▶ die Inkrementelle Integration  
(Top-Down, Bottom-Up, Outside-In, kontinuierlich)

# Integrationstest - Aufbau

- Wenn Komponente A getestet werden soll, wird ggf. ein *Testtreiber* benötigt
  - versorgt die Schnittstelle der Komponente A mit Testdaten
- Falls die Komponente A Dienste einer Komponente B nutzt, die noch nicht integriert ist, wird ein *Platzhalter* (*stub*) für B benötigt
  - vertritt die fehlende Komponente und liefert entweder
    - » konstante Werte, oder
    - » simuliert das Verhalten der späteren Komponente in Ausschnitten



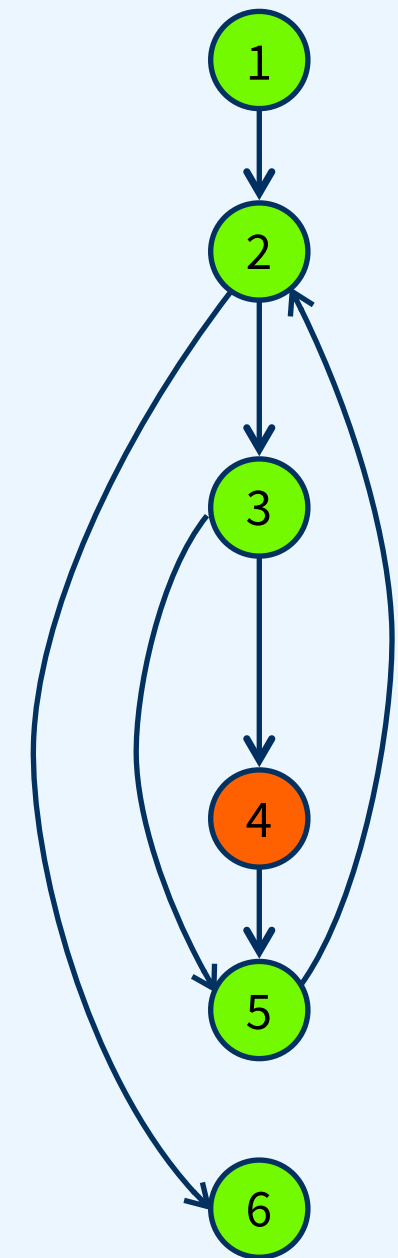
# Klassifizierung der Testverfahren



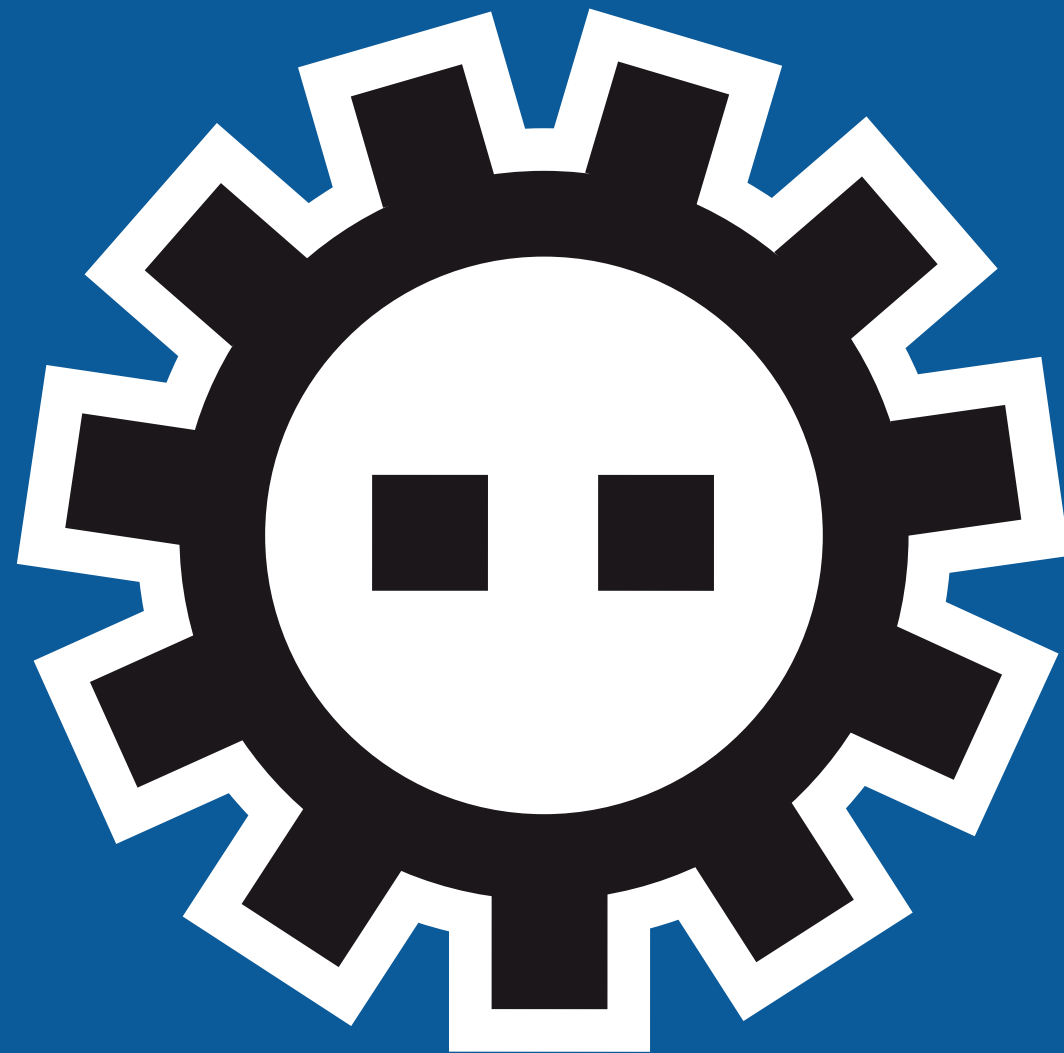
# Anweisungsüberdeckung

- Jede Anweisung im Programm muss im Test mindestens einmal ausgeführt werden (100% Anweisungsüberdeckung)
- Kennzahl  $C_0$  = Anzahl der überdeckten Anweisungen / Gesamtzahl der Anweisungen
- Eine Anweisungsüberdeckung  $C_0 > 80\%$  sollte angestrebt werden
  - ▶  $C_0 = 100\%$  ist unrealistisch, da der Aufwand zur Erstellung von Testfällen für die letzten Prozentpunkte stark ansteigt

**Beispiel:**  
eingabe = "XYZ"



$$C_0 = 5 / 6 = \mathbf{83\%}$$



# MOTIVATION



# Fehlermanagement

- Fehlermanagement ist eigentlich Änderungsmanagement
- Änderungen müssen auf zwei Ebenen verwaltet werden:
  - ▶ existierende Fehler müssen behoben werden
  - ▶ neue Funktionen müssen geplant werden
- Konzepte und Werkzeuge helfen uns bei der Umsetzung
- »Nichts ist so beständig wie der Wandel«  
*Heraklit, 500 v. Chr.*
- »Ein Grashalm wächst auch nicht schneller, wenn man daran zieht.«  
*Chinesisches Sprichwort*

# Motivation

Warum entstehen Änderungen am geplanten Vorgehen?

- Produkt zeigt Fehlverhalten
- Kunde wünscht andere Funktionen
- Probleme bei der geplanten technologischen Umsetzung
- Marktumfeld ändert sich

Warum benötigen wir dafür Werkzeuge?

- Änderungen sollen nicht ad hoc umgesetzt werden
- Änderungen sollen nachvollziehbar dokumentiert sein

# Übliche Softwarefehler

## Arithmetische Fehler:

- Division durch 0  $\rightarrow 20 / 0$
- Arithmetischer Überlauf  $\rightarrow 32768 + 1$
- Rundungsfehler  $\rightarrow 0,00001 + 100000 = ?$

## Logische Fehler:

- Endlosschleifen und endlose Rekursion  $\rightarrow \text{while (true) ...}$
- Off-by-one-Error (OBOE)  $\rightarrow \text{int zahlen[10]; zahlen[10] = 1;}$

# Übliche Softwarefehler

## Syntax-Fehler

- Verwendung des falschen Operators → `x=5` im Vergleich zu `x==5`

## Ressourcen-Fehler

- Null Dereferenzierung → `String x = null; x.length();`
- Ressourcen-Lecks, wenn begrenzte Ressourcen (Speicher, Dateien) nicht wieder freigegeben werden
- Buffer-Overflow, wenn über die Grenzen eines allokierten Bereichs geschrieben wird
- Exzessive, aber logisch korrekte Rekursion kann zu einem Stack-Overflow führen

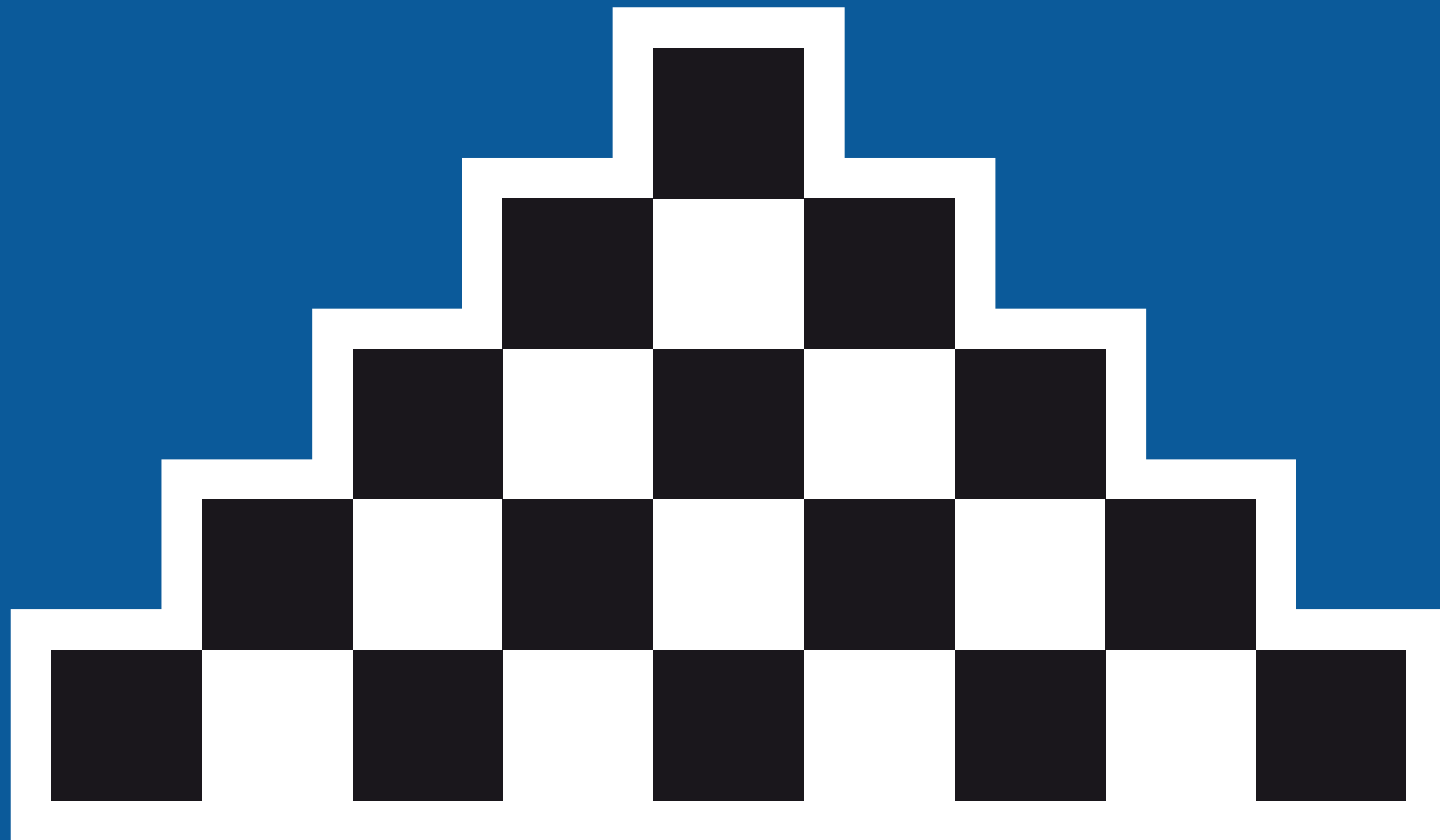
# Übliche Softwarefehler

## Multi-Threading-Fehler

- Deadlock, wenn zwei Threads gegenseitig auf die Beendigung warten
- Race-Condition, wenn Threads in anderer Reihenfolge ablaufen als vom Programmierer erwartet

## Teamwork-Fehler

- Von zwei gleichen Funktionen wird nur eine verändert → kann mithilfe von » Don't Repeat Yourself« (DRY) verhindert werden
- Kommentare sind veraltet oder passen nicht zum Quellcode
- Dokumentation ist veraltet und passt nicht mehr zum Produkt



# GRUNDLAGEN

# Fehlermanagement

Viele Namen für ähnliche Konzepte:

**Fehler**

**Defects**

**User  
Stories**

**Bugs**

**Änderungen**

**Changes**

**Issues**

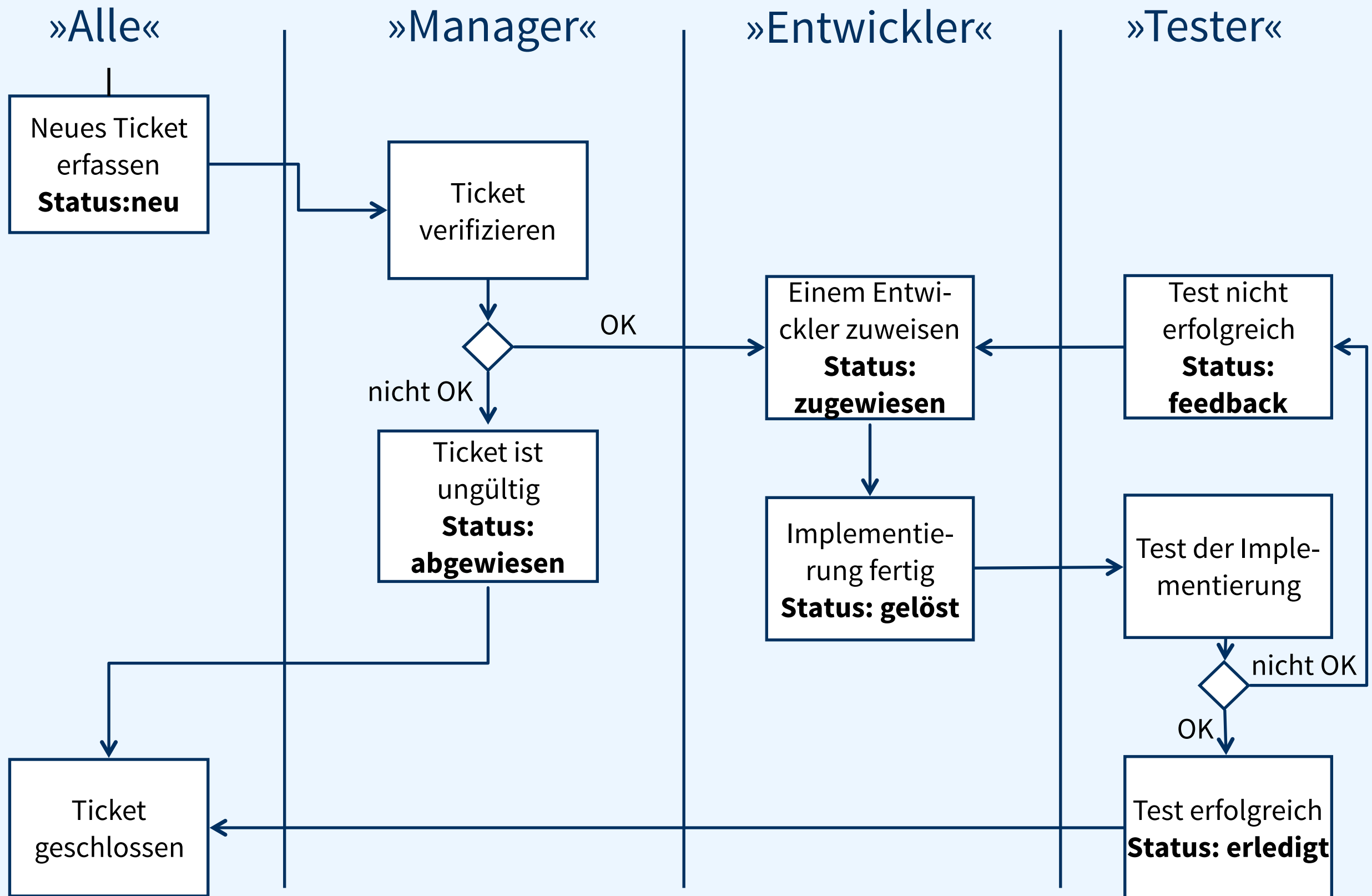
- Fehlerberichte werden von unterschiedlichen Nutzergruppen erstellt
- Jede Gruppe verwendet eigene Begriffe
- Gemeinsame Definition:  
Eine gewünschte Änderung des Softwareprodukts

# Fehlerberichte

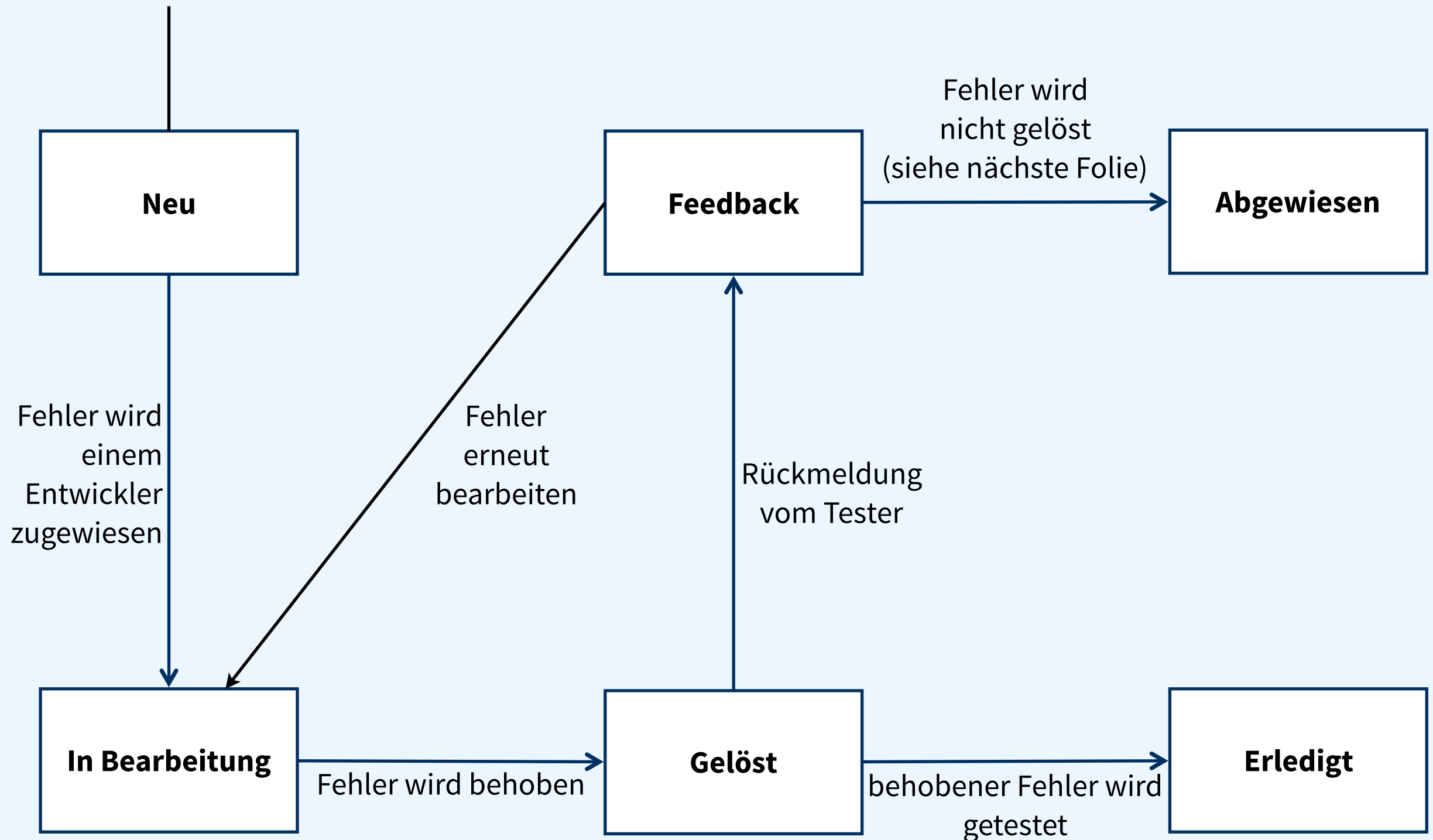
Quelle	Art der Fehlerberichte
Entwickler	Zur Dokumentation der gefundenen Fehler
Tester	Zur Weiterleitung der Fehler an die Entwickler
Interne Nutzer	Alpha-Test, »Eat Your Own Dog Food«
Auftraggeber	Um Änderungswünsche zu dokumentieren
Externe Nutzer	Beta-Test, Crash-Reports



# Lebenszyklus eines Fehlers



# Fehlerstatus



# Abgewiesene Fehler

Fehler können aus verschiedenen Gründen nicht bearbeitet werden:

- Deadline muss erreicht werden → Fehlerbehebung wird auf einen späteren Zeitpunkt verschoben
- Fehler ist in einer zukünftigen Version bereits behoben
- Komponente mit Fehler ist in einer zukünftigen Version nicht mehr vorhanden
- Änderungsaufwand zur Fehlerbehebung ist zu groß
- »It's not a bug, it is a feature!«

# Triage

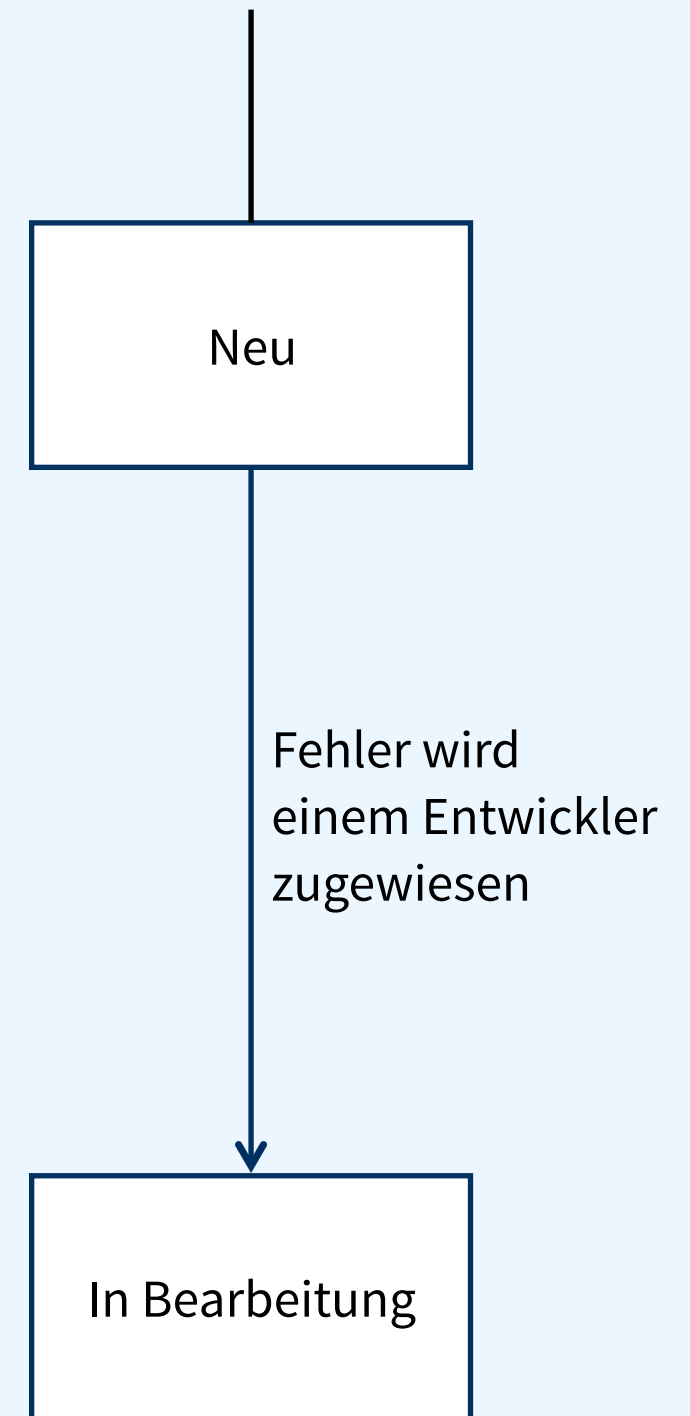
Medizinischer Begriff:

Priorisierung der Behandlung von Verletzten in Krisensituationen

- Ziel: mit begrenzten Ressourcen möglichst vielen Menschen helfen
- Ist es sinnvoll, die am schwersten verletzten Patienten als erstes zu behandeln?
- Übertragen auf die Softwareentwicklung:
  - ▶ Fehlerberichte kommen unkategorisiert herein
  - ▶ Fehlerberichte müssen gesichtet werden und ihrer Priorität gemäß bearbeitet werden
- Beispiel: Eclipse Projekt
  - ▶ 3500 Fehlerberichte im Quartal (durchschn. 29/Tag)
  - ▶ 39% wurden abgewiesen

# Triage: Aufgaben

- Die Triage ist die erste Aufgabe, wenn ein Fehlerbericht erstellt wurde
- Folgende Aspekte müssen geklärt werden:
  - ▶ Fehler nachstellen  
Reichen die Informationen im Fehlerbericht aus, den Fehler selbst nachzustellen?
  - ▶ Priorität festlegen  
Wie wichtig ist die Behebung des Fehlers?
  - ▶ Verantwortlichen Entwickler herausfinden  
Wer kann den Fehler beheben?
  - ▶ Bearbeitungsdauer festlegen  
Wie lange dauert die Behebung des Fehlers?



# Prioritäten

- Die Priorität wird basierend auf zwei Aspekten festgelegt:
  - Schweregrad des Fehlers
  - Wahrscheinlichkeit, dass der Fehler auftritt
- Die folgende Matrix gibt eine mögliche Einordnung, sie kann aber individuell anders aussehen:

		Wahrscheinlichkeit		
		selten	häufig	immer
Schwere- grad	Absturz	dringend	sofort	sofort
	Abbruch	niedrig	hoch	dringend
	Optik	niedrig	normal	dringend

# Gute Fehlerberichte

- Studie mit Umfrage unter Entwicklern und Nutzern der Open-Source-Projekte Apache, Eclipse und Mozilla
- 400 Teilnehmer, Voraussetzungen:
  - ▶ Entwickler haben mindestens 50 Fehlerberichte selbst bearbeitet
  - ▶ Nutzer haben mindestens 25 Fehlerberichte verfasst
- Unterschiedliche Sichtweise führt zu einer Diskrepanz zwischen...
  - ▶ ...dem, was der Nutzer berichtet
  - ▶ ...das, was der Entwickler benötigt

# Gute Fehlerberichte

Entwickler		Nutzer
Reproduzieren (97%)		Reproduzieren (98%)
Beobachtetes Verhalten (95%)		Beobachtetes Verhalten (96%)
Erwartetes Verhalten (89%)		Erwartetes Verhalten (94%)
Stack Traces (89%)		Produkt (94%)
Testfälle (85%)		Version (91%)
Zusammenfassung (81%)		Betriebssystem (90%)
Screenshots (75%)		Zusammenfassung (90%)
Version (75%)		Screenshots (60%)
Code Beispiele (68%)		Testfälle (56%)
Produkt (65%)		Stack Traces (50%)
Betriebssystem (63%)		Hardware (48%)
Hardware (32%)		Code Beispiele (36%)



# Gute Fehlerberichte

## Empfehlungen der Studie:

- Die Erstellung von Fehlerberichten besser strukturieren, für den Entwickler notwendige Angaben hervorheben
- Nutzer, die Fehlerberichte schreiben sollten belohnt werden (nicht nur durch das beheben des Fehlers)
  - ▶ z.B.: Nutzerreputation für jeden »guten« Fehlerbericht aufbauen
- Nutzer ermutigen, bestehende Fehlerberichte durch zusätzliche Angaben zu ergänzen (Screenshots, Stack Traces, etc...)
- Verschiedene Benutzeroberflächen für verschiedene Benutzergruppen
- Einfache Suchfunktion anbieten, um Duplikate zu verhindern



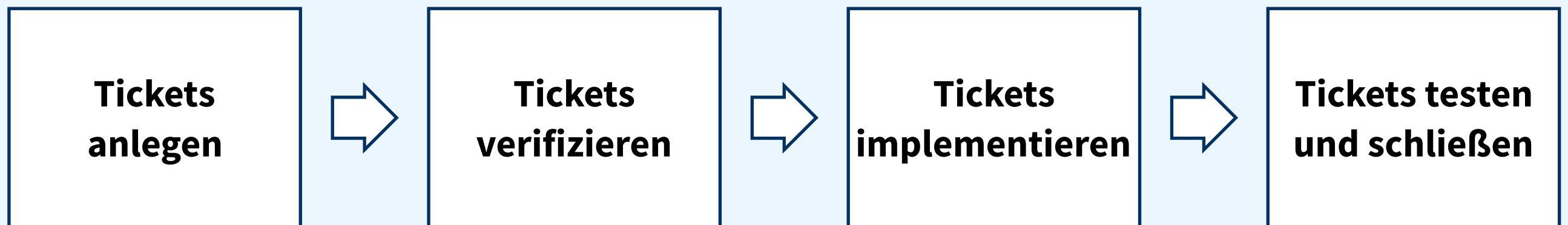
# WERKZEUGE

# Werkzeug Gitlab

- Wir betrachten das Open-Source-Werkzeug »Gitlab«
- Die Kernfunktionalität ist die Verwaltung von Git-Projekten
- Gitlab erweitert diese um ergänzende Aspekte:
  - ▶ Verwaltung von Fehlern  
Fehlerberichte werden mit Commits verknüpft
  - ▶ Projektplanung  
Aufwandsabschätzung von Fehlern und Änderungen ergeben einen Projektplan
  - ▶ Projekt-Wiki und -diskussionsforum  
Interne und externe Dokumentation des Projekts

# Gitlab Arbeitsablauf

Die Arbeit mit Gitlab entspricht dem »Lebenszyklus eines Fehlers«



Zuerst muss Gitlab allerdings noch initial konfiguriert werden

# Release-Planung

- In Gitlab können für die Planung der Software Zielversionen (Milestones) erstellt werden
- Wenn z.B. die Software aktuell die Version 1.4 hat, könnten folgenden Zielversionen definiert werden:
  - ▶ 1.4.1
  - ▶ 1.5
  - ▶ 2
- Bei der Triage des Fehlerberichts kann dann entschieden werden, in welcher Version dieser behoben sein soll
- Basierend auf dieser Zuordnung können automatische Auswertungen durchgeführt werden

# Roadmap/Changelog

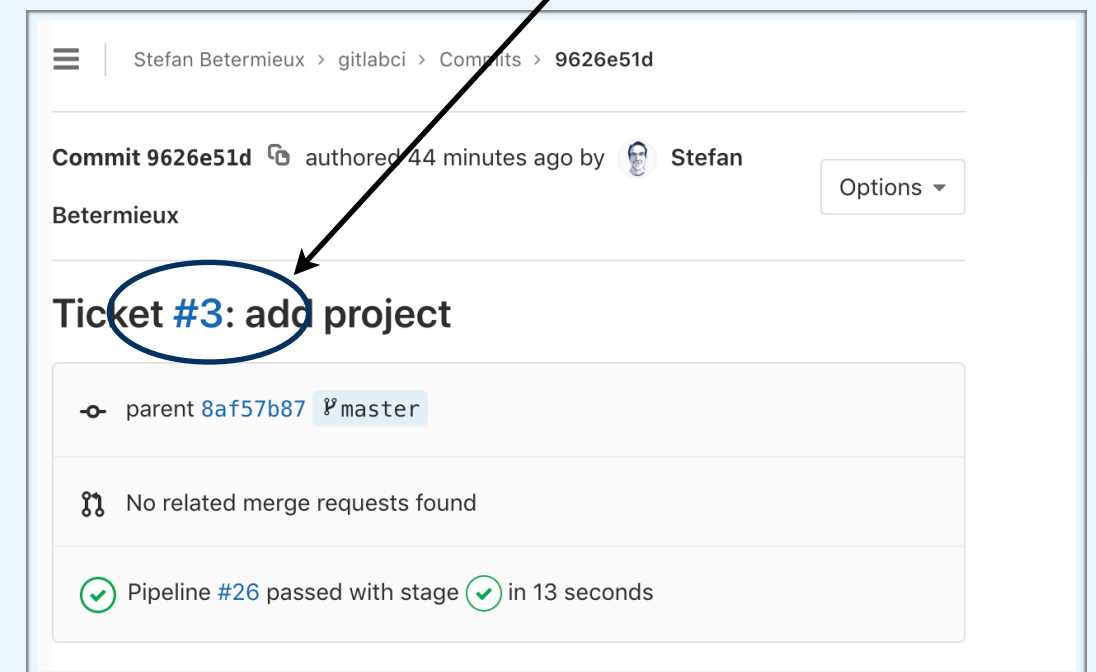
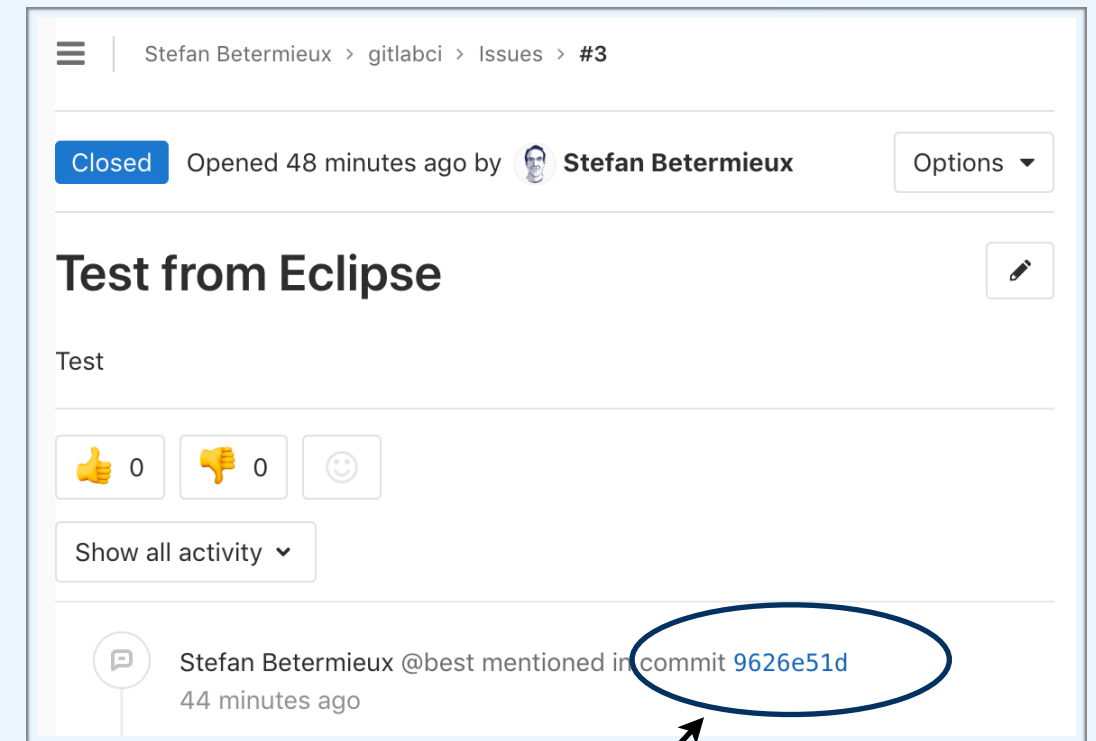
- Wenn Fehlerberichte korrekt den Zielversionen der Software zugeordnet sind, kann automatisch eine Roadmap erstellt werden
- Auflistung aller offenen/geschlossenen Fehlerberichte für eine Zielversion
- Verhältnis der Zahlen gibt Fertigstellungsgrad an
- Für bereits ausgelieferte Versionen sind die Fehlerberichte der Changelog (Fertigstellungsgrad 100%)

The screenshot shows the 'Milestones' page for the 'gitlabci' project by 'Stefan Betermieux'. At the top, there are filters for 'Open' (1), 'Closed' (0), and 'All' (1). Below these are search and sorting options: 'Filter by milestone name', 'Due soon', and a 'New milestone' button. The main content area displays a milestone named '1.0' with a progress bar indicating '33% complete'. It also shows '3 Issues · 0 Merge Requests' and a 'Close Milestone' button. The repository path 'Stefan Betermieux / gitlabci' is visible in the bottom left of the milestone card.

Milestone Name	Progress	Issues	Merge Requests	Completion Status
1.0	33% complete	3 Issues	0 Merge Requests	In Progress

# Versionsverwaltung

- Die Fehlerverwaltung sollte eng mit der Versionsverwaltung verknüpft sein:
  - ▶ aus einem Fehlerbericht sollten die Revisionen der dafür veränderten Quelldateien zugreifbar sein
  - ▶ bei jedem Commit in die Versionsverwaltung sollte eine Fehlernummer angegeben sein, der die Begründung für die Änderung enthält
- Einfache Realisierung: In der Commit-Nachricht die Fehlernummer mit Präfix # angeben

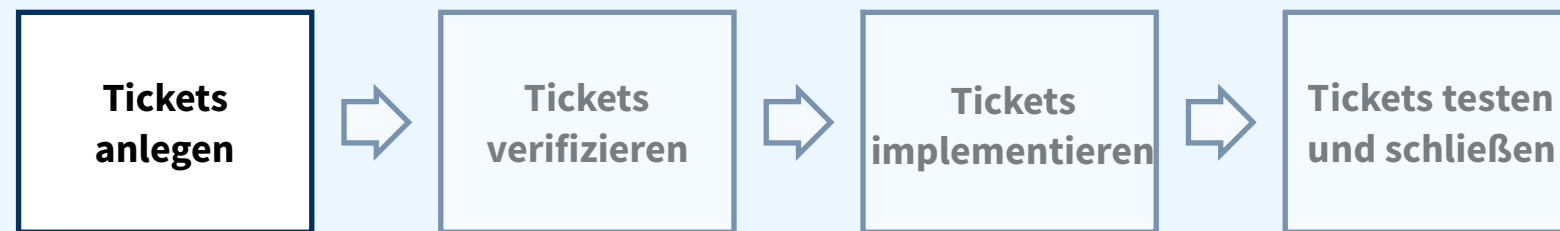


# Versionsverwaltung

- Durch die Angabe der Fehlernummer im Commit kann die Fehlerverwaltung die bidirektionale Verbindung herstellen
  - ▶ z.B.: "Webservice angebunden, behebt Fehler #2"
- Gitlab ist eine Web-Anwendung, kann also nur auf Browseranfragen reagieren
  - ▶ wie kann die Versionsverwaltung Gitlab mitteilen, dass eine neue Revision erstellt wurde?
- Lösung: Post-Commit-Hooks / Post-Receive-Hooks
  - ▶ kleines Shell-Skript, das von der Versionsverwaltung ausgeführt wird, wenn eine neue Revision erstellt wird
  - ▶ das Skript muss einen Web-Service auf Gitlab aufrufen



# Tickets anlegen



- Spielregeln definieren:
  - ▶ jeder darf Tickets erstellen
  - ▶ keine Änderung in der Versionsverwaltung ohne Ticket
  - ▶ Tickets sind keine Kommunikationsplattform
  - ▶ Tickets sind kein Wunschzettel
- Die Eingabefelder eines Tickets sollten möglichst präzise ausgefüllt werden

# Tickets anlegen

Title

Title

Add [description templates](#) to help your contributors communicate effectively!

Description

WritePreview

**B****I**

Write a comment or drag your files here...

Markdown and [quick actions](#) are supported

[Attach a file](#)

☐ This issue is confidential and should only be visible to team members with at least Reporter access.

Assignee

Unassigned

Assign to me

Due date

Select due date

Milestone

Milestone

Labels

Labels

# Tickets verifizieren



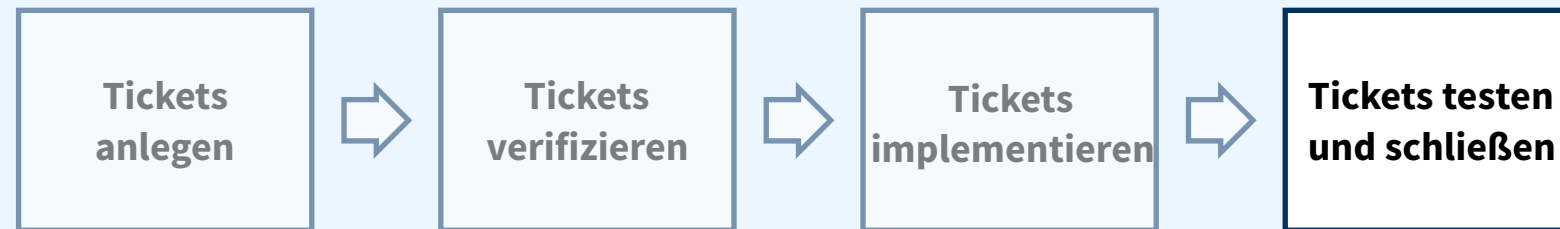
- Über die Suche alle neuen Tickets auflisten
- Formal auf Vollständigkeit prüfen
  - ▶ wenn z.B. die Beschreibung fehlt, Ticket abweisen
- Inhaltlich prüfen
  - ▶ erfordert Domänenwissen / technisches Wissen
- Wenn Ticket akzeptiert wird, die weiteren Felder ausfüllen (falls noch nicht vom Ersteller ausgefüllt)
  - ▶ zugewiesener Entwickler
  - ▶ geschätzte Bearbeitungsdauer
  - ▶ anvisierte Zielversion

# Tickets implementieren



- Entwickler lässt sich eine Liste »seiner« Tickets anzeigen
- Entwicklung erfolgt außerhalb von Gitlab
- Über Commit-Messages werden Revisionen mit Tickets verknüpft
- Wenn die Bearbeitung aus Sicht des Entwicklers abgeschlossen ist:
  - ▶ Status auf »gelöst« setzen
- Mit dem später vorgestellten Werkzeug »Mylyn« wird eine noch engere Verzahnung von Tickets und IDE vorgestellt

# Tickets testen und schließen



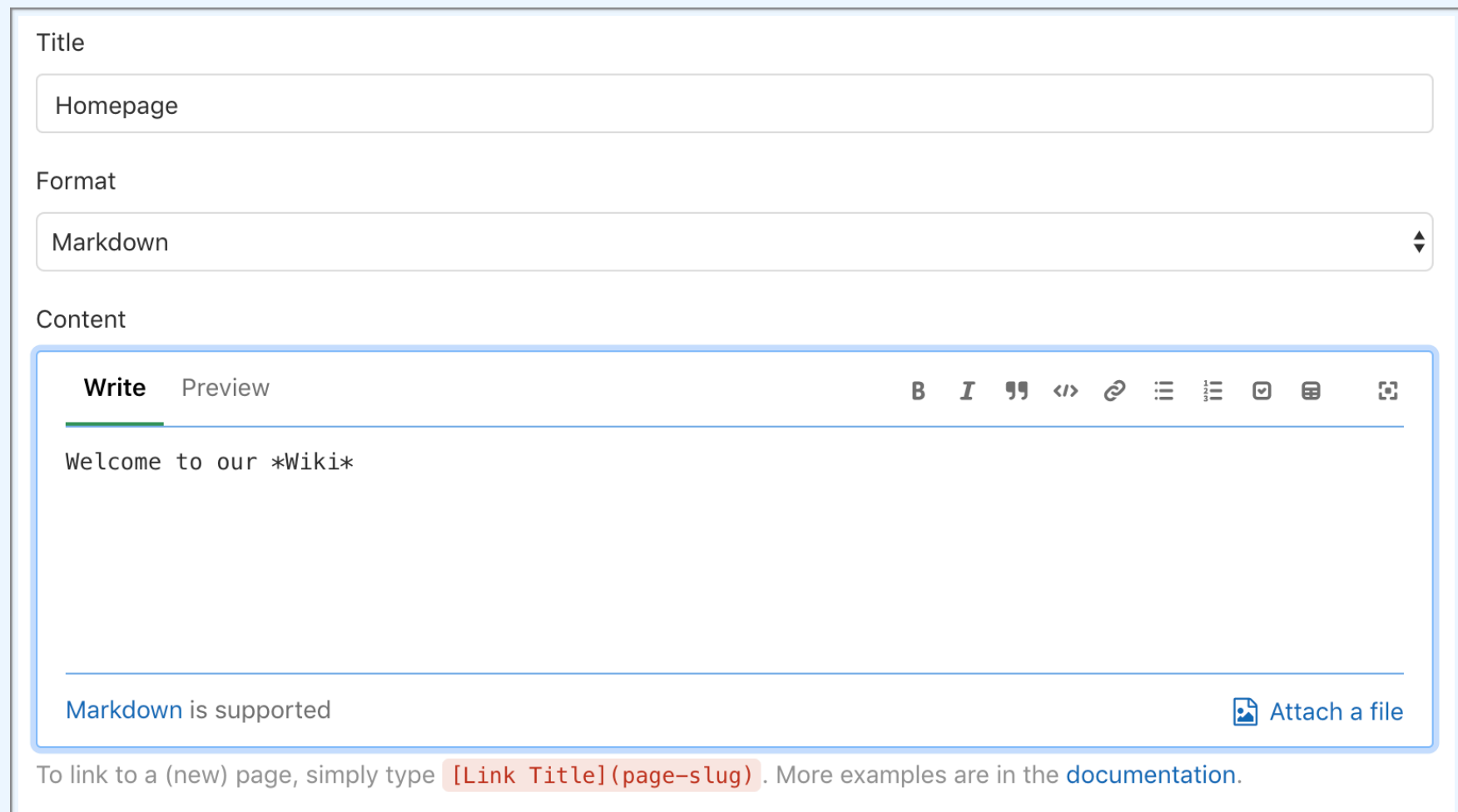
- Gelöste Aufgabe wird getestet und Lösung wird bewertet
- Getestet werden kann von:
  - ▶ Softwaretestern
  - ▶ Akzeptanztest durch Nutzer
  - ▶ Softwareentwickler, der die Aufgabe gelöst hat
- Getestet werden sollte eine zentrales Integrationssystem, nicht eine lokale Installation
- Wenn Test erfolgreich, kann das Ticket geschlossen werden

Gitlab

# Weitere Funktionalität

# Gitlab Wiki

- Einträge können mit der Markup-Sprache »Markdown« formatiert werden
  - ähnelt Wikitext
- Für kleine Projekte eine sinnvolle Möglichkeit, eine Projekthomepage zu pflegen



The screenshot shows the Gitlab Wiki edit interface. It has three main sections: 'Title', 'Format', and 'Content'. The 'Title' field contains 'Homepage'. The 'Format' dropdown is set to 'Markdown'. The 'Content' section has a 'Write' tab (active) and a 'Preview' tab. The 'Write' tab shows a rich text editor with a toolbar containing icons for bold, italic, quote, code, link, list, table, and image. The text area contains 'Welcome to our \*Wiki\*'. Below the text area, it says 'Markdown is supported' and 'Attach a file'. At the bottom, there is a note: 'To link to a (new) page, simply type [Link Title] (page-slug) . More examples are in the documentation.'

Title

Homepage

Format

Markdown

Content

**Write** Preview

Welcome to our \*Wiki\*

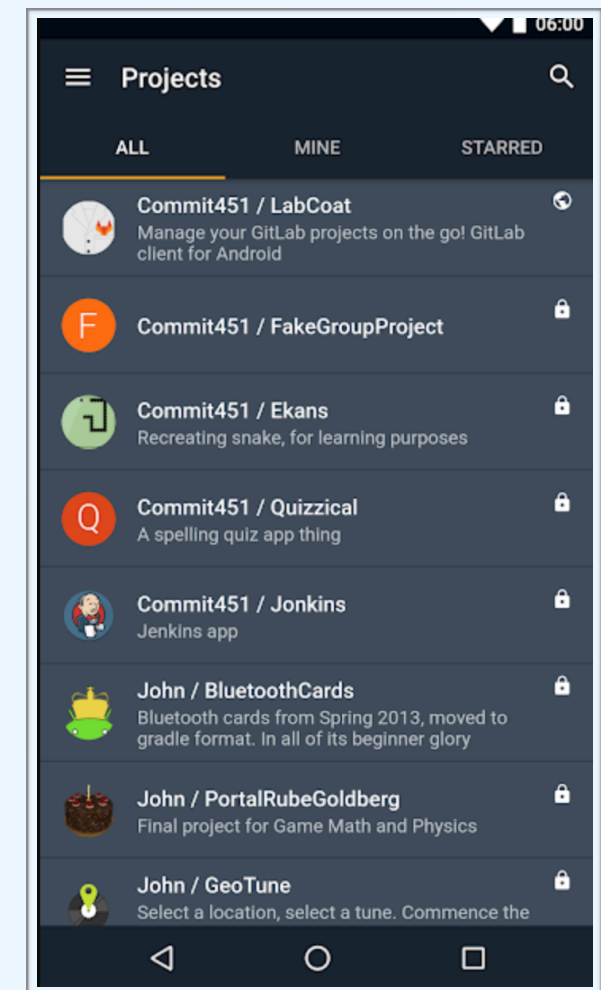
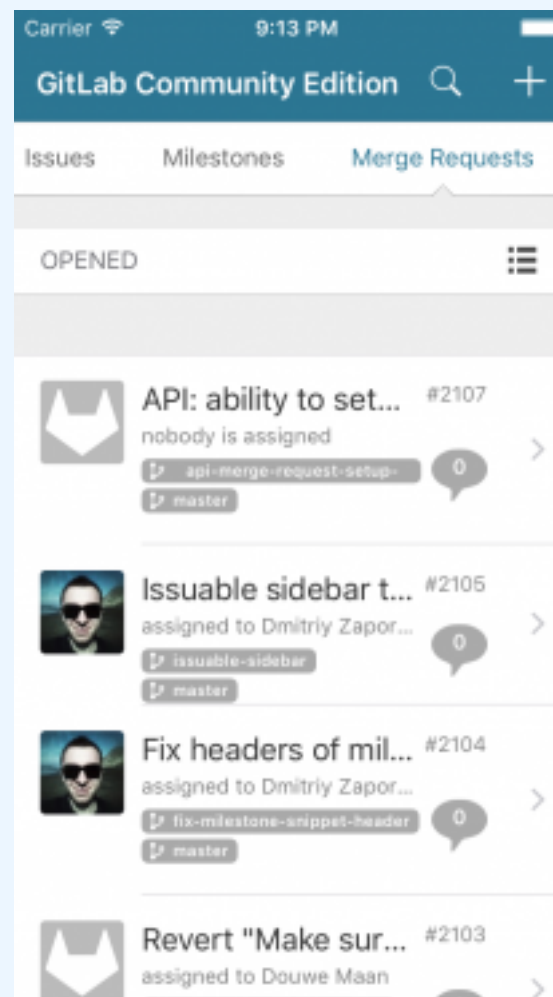
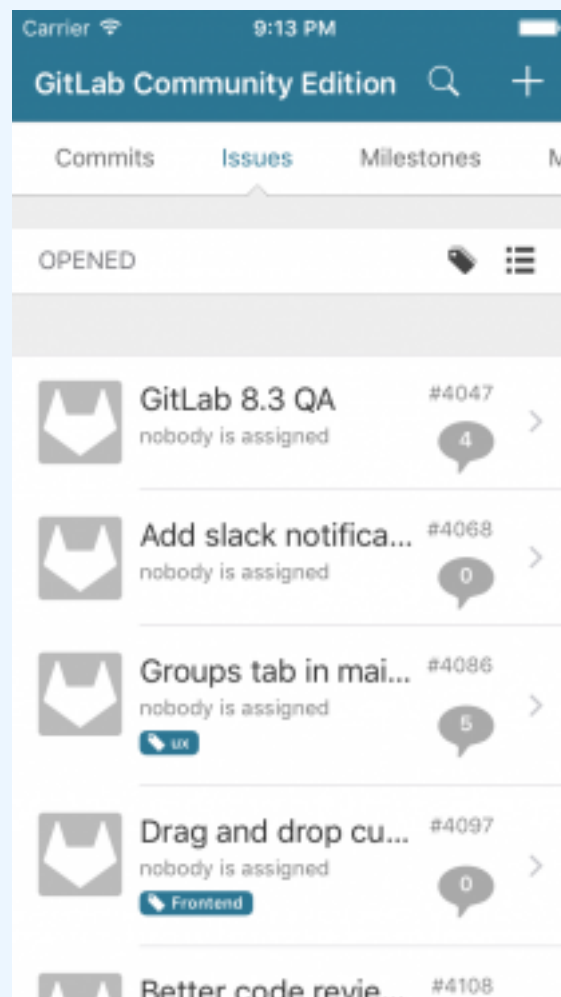
Markdown is supported

Attach a file

To link to a (new) page, simply type [Link Title] (page-slug) . More examples are in the documentation.

# Gitlab Mobil-App

- Für iPhone, Android und Windows Phone existieren Apps
- Greifen über die REST-Schnittstelle auf den Server zu





Eclipse und Mylyn

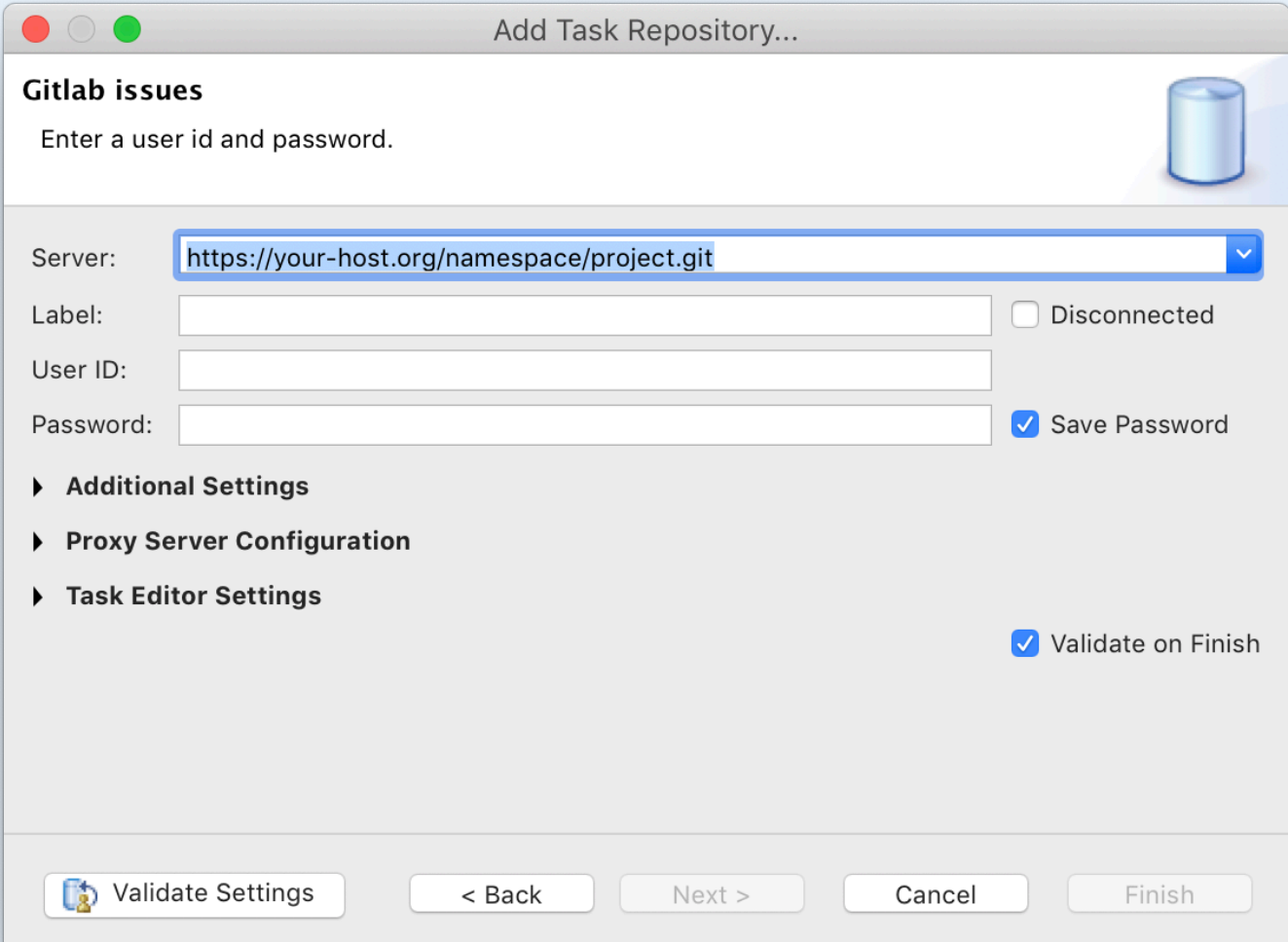
# **aufgabenbasierte IDE**

# aufgabenbasierte IDE

- Sobald in Gitlab ein Ticket einem Entwickler zugewiesen ist, muss dieser in einer IDE (z.B. Eclipse) die Änderung implementieren
  - ▶ hohe Distanz zwischen Webseite und IDE
- Open-Source-Werkzeug *Mylyn* holt die einem selbst zugewiesenen Tickets als Aufgaben in die IDE
- In der IDE sucht man sich den aktuell zu bearbeitenden Aufgabe aus und »startet« ihn:
  - ▶ Mylyn protokolliert, wie lange eine Aufgabe gestartet ist  
→ Berechnung der Bearbeitungsdauer
  - ▶ bei einem Commit wird automatisch die Ticket-Nummer der gestarteten Aufgabe in den Commit geschrieben

# Mylyn Plugin für Eclipse

- Eclipse installiert in der JavaEE Version Mylyn standardmäßig:
  - ▶ per Default kann nur Bugzilla angesprochen werden
  - ▶ für Gitlab muss ein Connector nachinstalliert werden:  
<http://pweingardt.github.com/mylyn-gitlab>
- Danach muss Gitlab als Mylyn-Verbindung angelegt werden:
  - ▶ View → Task Repositories
  - ▶ Add Task Repository...



**Add Task Repository...**

**Gitlab issues**  
Enter a user id and password.

Server:

Label:

User ID:

Password:

☐ Disconnected


☒ Save Password

▶ **Additional Settings**

▶ **Proxy Server Configuration**

▶ **Task Editor Settings**

☒ Validate on Finish

 Validate Settings   < Back   Next >   Cancel   Finish

# Mylyn: Aufgabenliste

Gitlab

Eclipse/Mylyn

Open 3 Closed 0 All 3

Search or filter results... Created date

**Test from Eclipse**  
#3 · opened 1 hour ago by Stefan Betermieux 1.0 updated just now

**Test unassigned from Gitlab**  
#2 · opened 1 hour ago by Stefan Betermieux 1.0 updated 20 minutes ago

**Test from Gitlab**  
#1 · opened 1 hour ago by Stefan Betermieux 1.0 updated 19 minutes ago

Task List

Find All Test from Ecl...

Alle Tickets [Webtech Redmine]

Assigned [Gitlab CI Testproject]

1: Test from Gitlab

**3: Test from Eclipse**

Meine Aufgaben [Webtech Redmine]

Unmatched [Webtech Redmine]

Unmatched [Gitlab CI Testproject]

Commit Changes

Commit Changes to Git Repository

Commit message

Ticket #3:

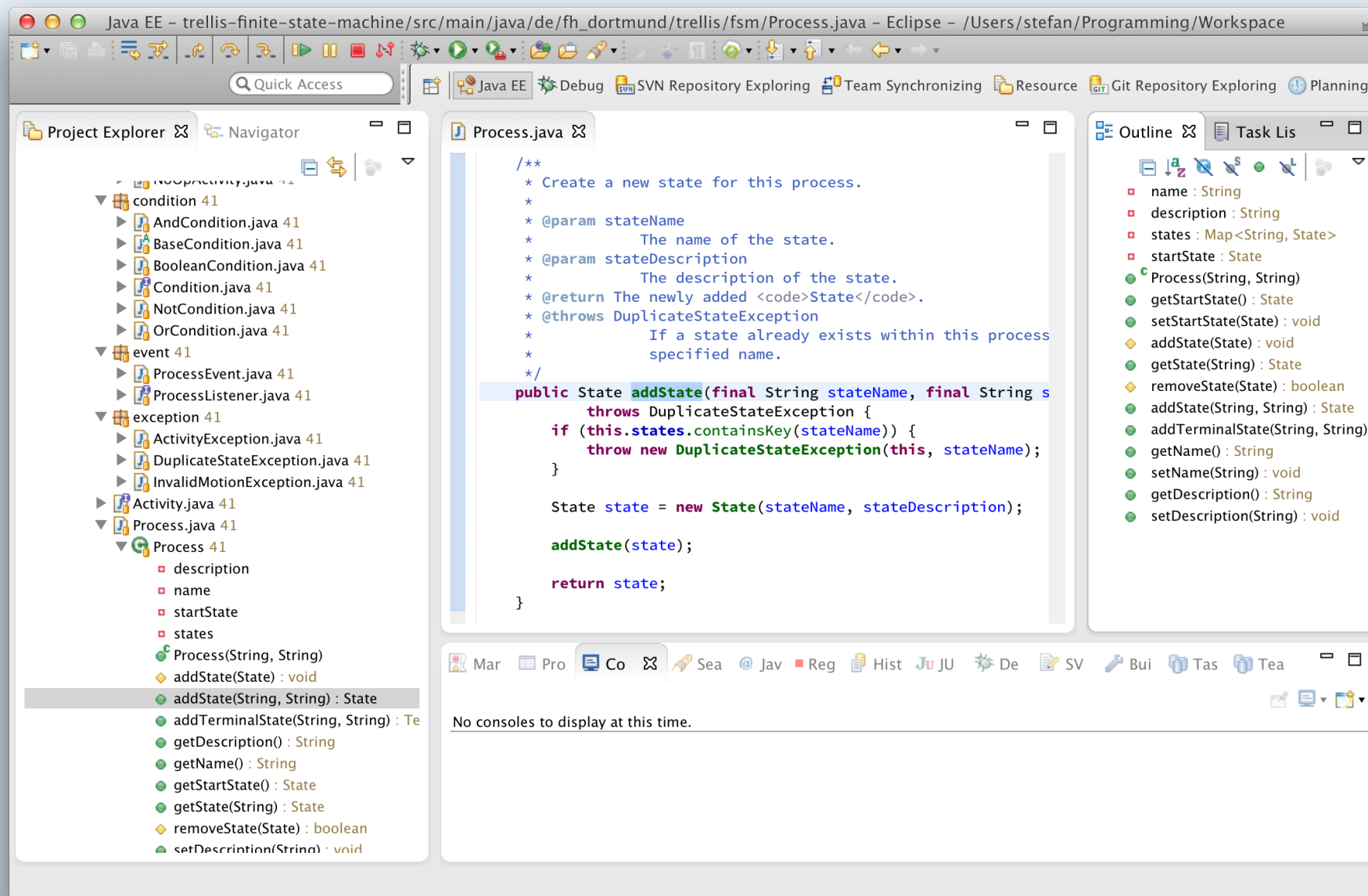
Author: Stefan Betermieux <stefan@betermieux.de>

Committer: Stefan Betermieux <stefan@betermieux.de>

Open [Git Staging](#) view Commit and Push Cancel Commit

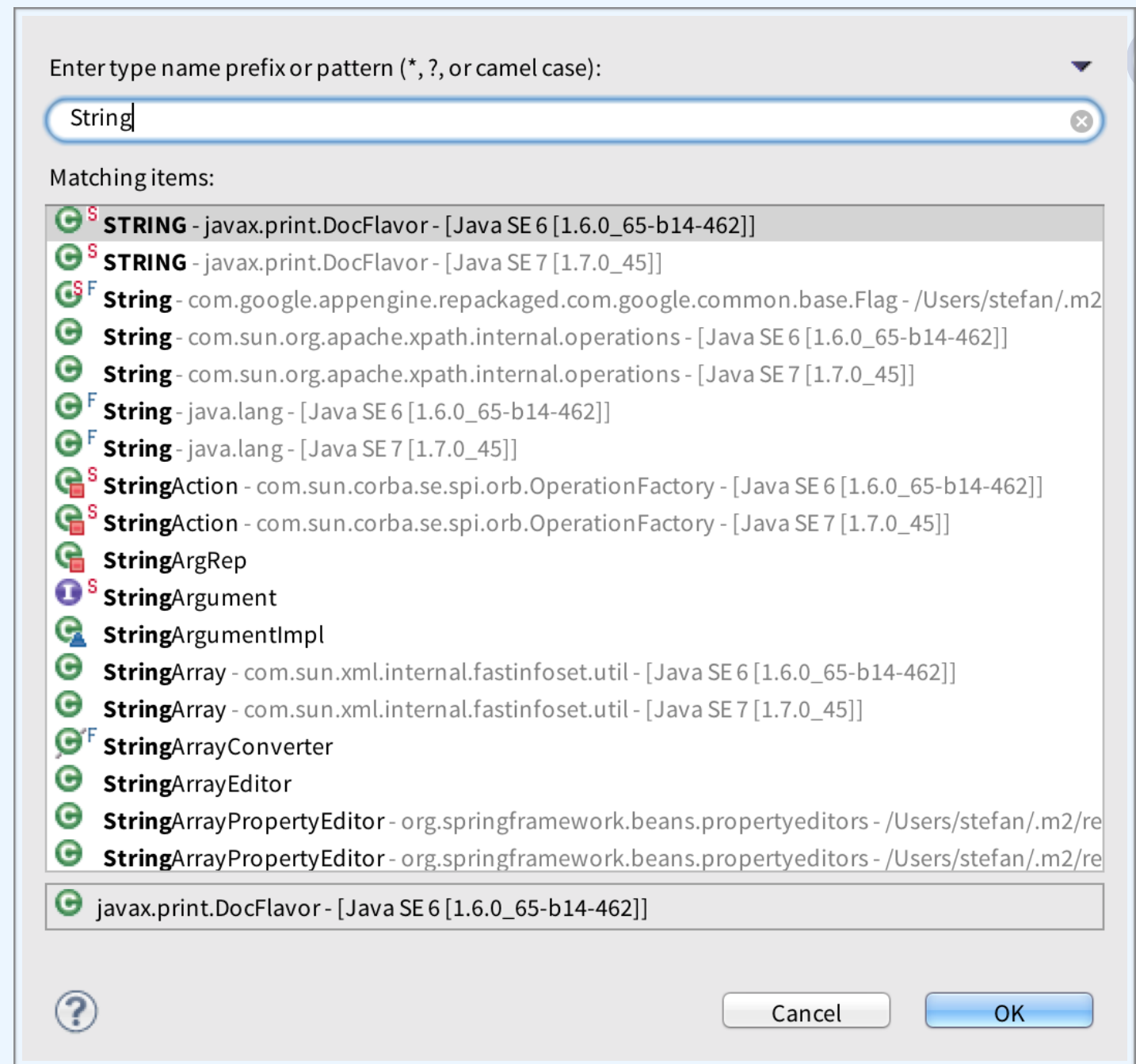
# Mylyn: Aufgabenfokus

- Häufig hat ein Entwickler in der IDE Zugriff auf hunderte Klassen
  - ▶ wie behält man dort den Überblick?



# Mylyn: Aufgabenfokus

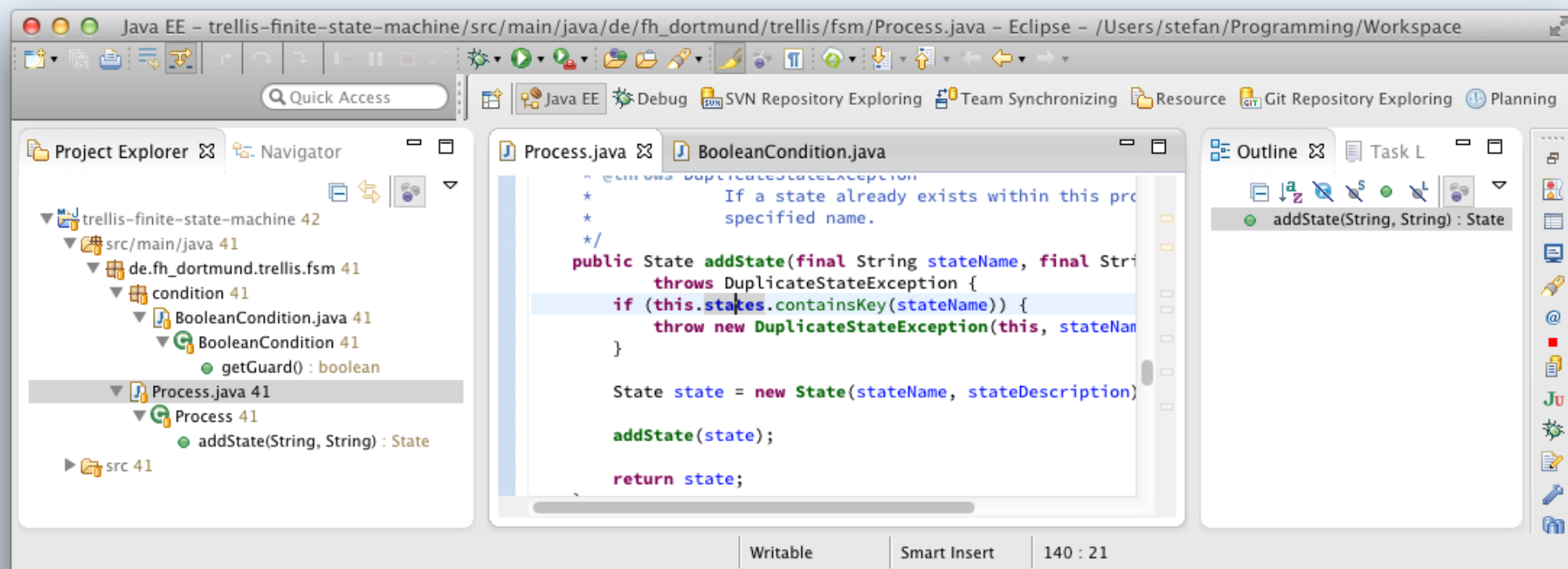
- Überblick durch:
  - ▶ wiederholte Suche
  - ▶ Scrollen
  - ▶ Navigieren im Baum
  - ▶ Working-Sets
- In großen Projekten verliert man viel Zeit mit diesen wiederkehrenden Aufgaben





# Mylyn: Aufgabenfokus

- Mylyn kann sich zu einer Aufgabe die gerade bearbeiteten Dateien merken
  - ▶ nur die verwendeten Dokumente werden angezeigt
  - ▶ der Zustand kann in einem Kontext gespeichert werden (sogar in Gitlab direkt als Anhang an dem Ticket)
  - ▶ in Eclipse kann zwischen den Aufgaben hin und zurück geschaltet werden, die Kontexte werden gespeichert und wieder hergestellt



# DANKE