# Teamarbeit

Prof. Sven Apel

Universität des Saarlandes

Basierend auf Material von Andreas Zeller und Norbert Siegmund

# Problem Tracking

Ein Anwender (oder ein Entwickler) hat ein Problem. Wie kann  der Entwickler das Problem reproduzieren, um es zu beheben?

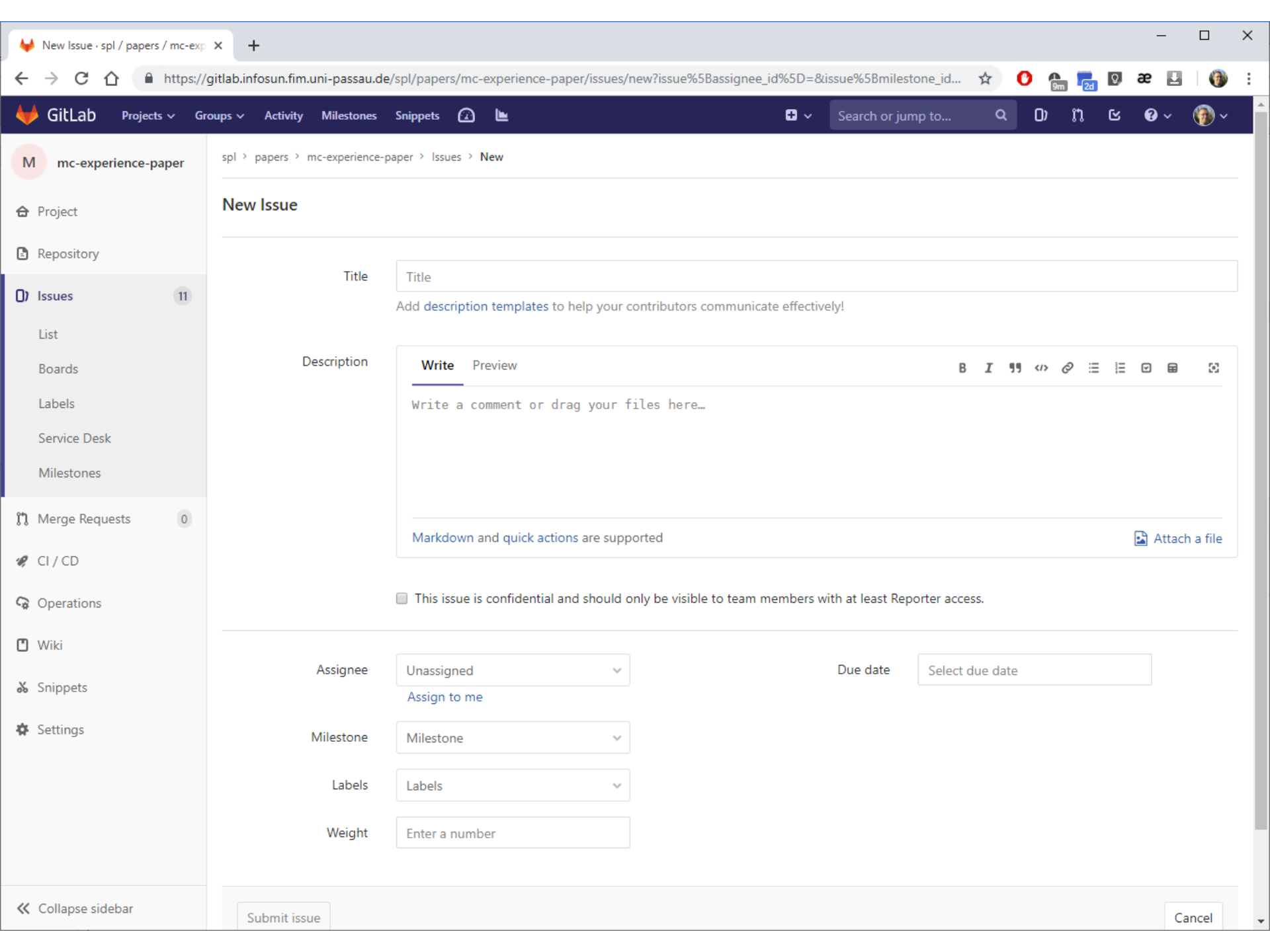
Lösung: alle *relevanten Informationen* über das Problem  erheben.

# Relevante Informationen

Explizite *Richtlinien* aufsetzen, was erhoben wird

- Produkt-Version
- Einsatzumgebung (z.B. Betriebssystem)
- Problem-Geschichte
- Beschreibung des *erwarteten* Verhaltens
- Beschreibung des *beobachteten* Verhaltens (typischerweise mit Verweis auf das Pflichtenheft)
- Wünschenswert: *Testfall*, der das Problem automatisch reproduziert

Diese Informationen enden in einem *Problembericht* (problem report, bug report)


New Issue · spl / papers / mc-exp
https://gitlab.infosun.fim.uni-passau.de/spl/papers/mc-experience-paper/issues/new?issue%5Bassignee_id%5D=&issue%5Bmilestone_id...
GitLab
Projects
Groups
Activity
Milestones
Snippets
Search or jump to...
M
mc-experience-paper
Project
Repository
Issues 11
List
Boards
Labels
Service Desk
Milestones
Merge Requests 0
CI / CD
Operations
Wiki
Snippets
Settings
Collapse sidebar
spl › papers › mc-experience-paper › Issues › New
New Issue
Title
Title
Add description templates to help your contributors communicate effectively!
Description
Write
Preview
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
Weight
Enter a number
Submit issue
Cancel

# Probleme identifizieren

Jedes Problem hat einen eindeutigen *Bezeichner* (auch Ticket number, PR number oder CR number – von PR = problem report, CR = change request).

Entwickler können sich darauf beziehen in

- E-Mails
- Änderungsmeldungen
- Status-Berichten

# Schwere des Problems

Blocker: Blocks development and/or testing work. This highest level of severity is also known as Showstopper.

Critical: Crashes, loss of data, severe memory leak.

Major: Major loss of function.

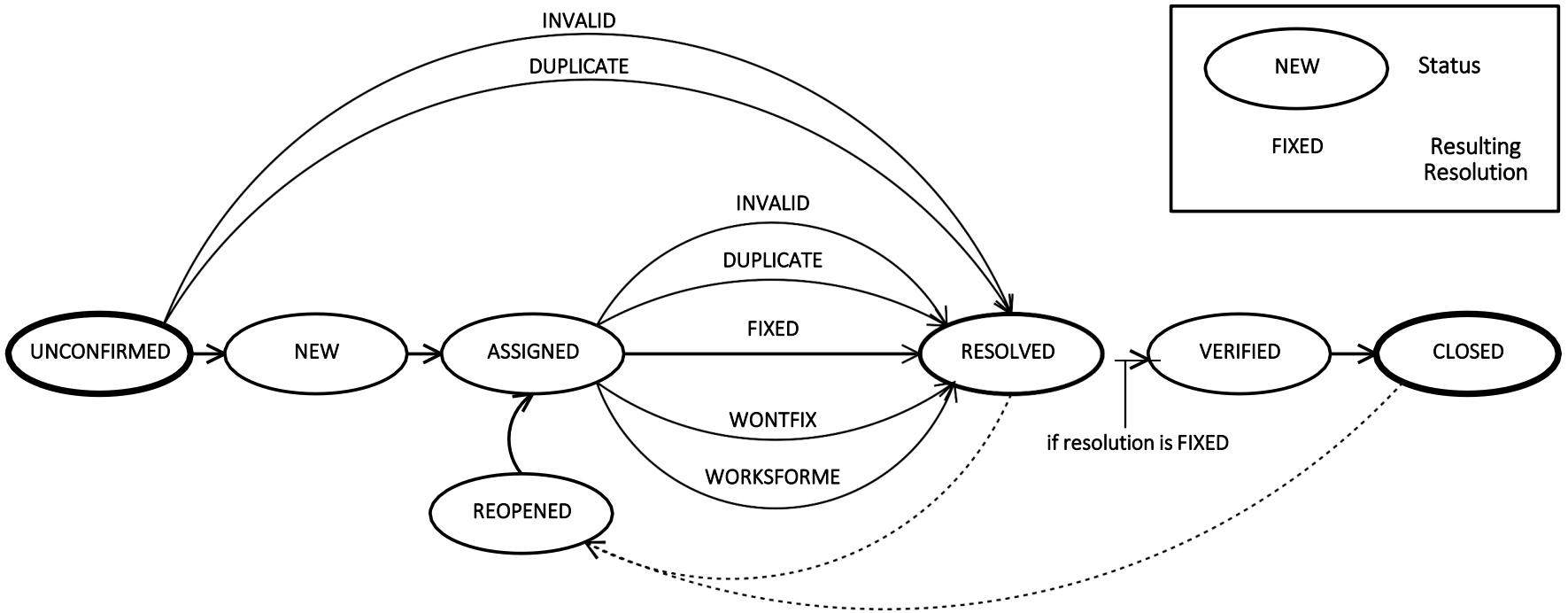
Normal: This is the “standard” problem.

Minor: Minor loss of function, or other problem where an easy workaround is present.

Trivial: Cosmetic problem like misspelled words or misaligned text.

Enhancement: Request for enhancement.

# Problem-Lebenszyklus am Beispiel Mozilla



# Problem-Status und Auflösung

FIXED: This problem has been fixed and tested.

INVALID: The problem described is not a problem.

WONTFIX: The problem described is a problem which will never be fixed.

LATER: The problem will be fixed in a later version.

REMIND: Like LATER, but might still be fixed earlier.

DUPLICATE: The problem is a duplicate of an existing one.

WORKSFORME: All attempts at reproducing this problem were futile. If more information appears later, the problem will be re-assigned.

# Vorrang eines Problems

Der *Vorrang* (Priority) bestimmt, in welcher Reihenfolge Probleme bearbeitet werden.

Der Vorrang ist das wichtigste Mittel, um die Entwicklung zu steuern!

In der Praxis gibt es eigene Komitees (aus 3–5 Entwicklern/Testern/Managern), die einzelnen Problemen einen Vorrang einräumen.

Nicht Spielprobleme, sondern *Kernprobleme* lösen!

# Problembezogene Teamarbeit

Grundidee: Das *Team arbeitet*, bis alle Probleme gelöst sind.

Das Projekt beginnt mit dem Ursprungsproblem:

*Das Produkt ist nicht da!*

Man teilt dieses Problem auf und weist Aufgaben zu.

Probleme können nicht nur im Produkt auftreten, sondern auch in allen Dokumenten – also von Beginn an.

Der Status *aller Probleme* wird über die Problem-Datenbank verwaltet (und ist jederzeit einsehbar).

Ist das letzte Problem gelöst, ist die Arbeit beendet :–)

*Und nun: Viel Spaß bei der Teamarbeit!*