Leveraging pre-commit hooks for context-sensitive checklists: a case study

Tobias Baum

FG Software Engineering
Leibniz Universität Hannover

19.03.2015

Agenda

1. Einführung
2. Das Tool „TortoiseChecklist“
3. Die Fallstudie

Agenda

Ausgangshypothese

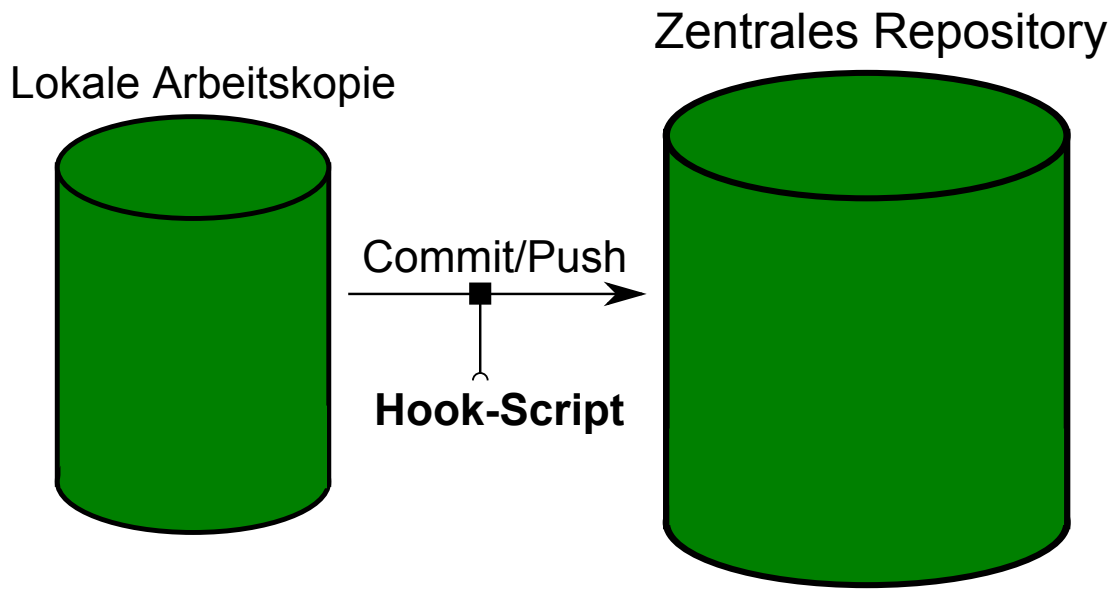
- Software muss wirtschaftlich und in hoher Qualität entwickelt werden
- Dafür hat es sich als sinnvoll erwiesen ...
  - verschiedene sich ergänzende Techniken zu kombinieren
  - frühes Feedback zu geben
  - Fehler zu finden, bevor sie Folgeaufwände verursachen
  - Prüfungen so in den Entwicklungsprozess zu integrieren, dass sie nicht vergessen und auch nicht als Last empfunden werden

- Hypothese: Prüfungen an Pre-commit Hooks stellen eine sinnvolle Ergänzung im Vorrat möglicher Techniken dar

Pre-commit Hooks

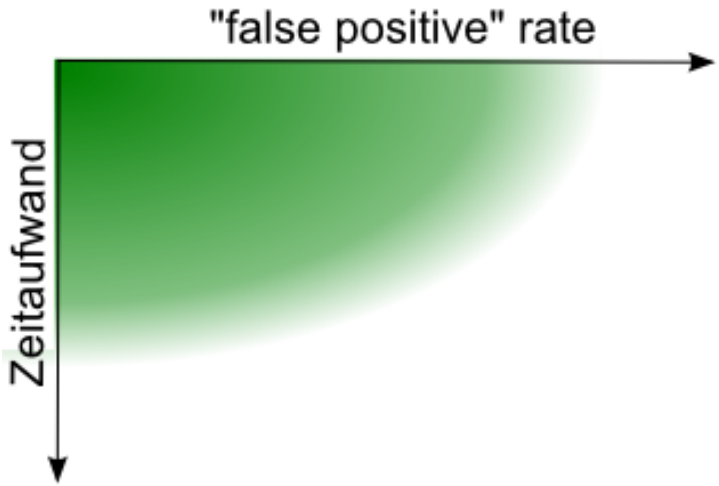
- Pre-commit Hooks sind ausführbare Programme/Skripte, die beim Commiten in ein Versionsverwaltungssystem ausgeführt werden
- Sie können den Commit verhindern

⇒ Pre-commit Hooks ermöglichen in den Entwicklungsfluss integrierte Prüfungen



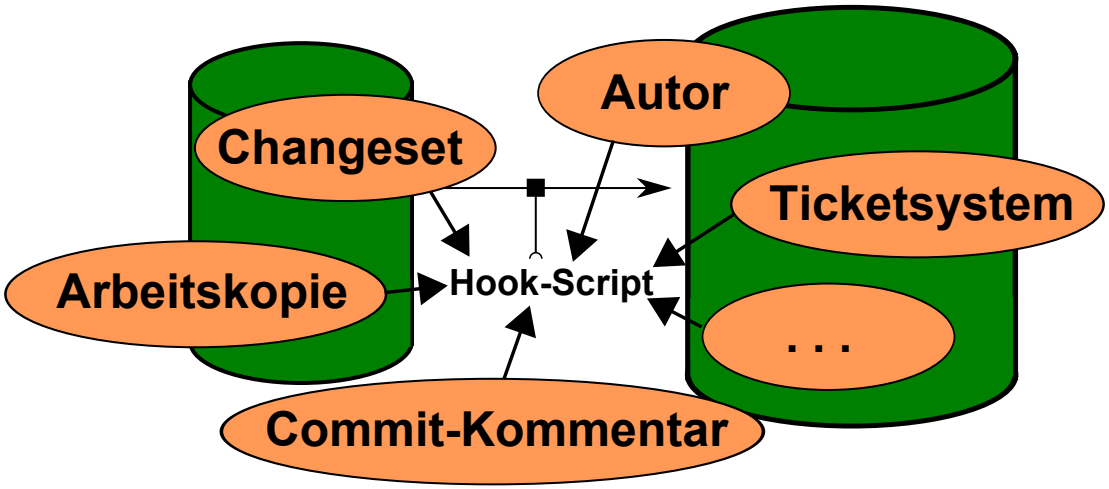
Geeignete Prüfungen

- Ob eine Prüfung geeignet ist ergibt sich aus Nutzen und Aufwand
- Nutzen
  - „true positives“/Commit
- Aufwand
  - Zeitaufwand/Commit
  - „false positives“/Commit



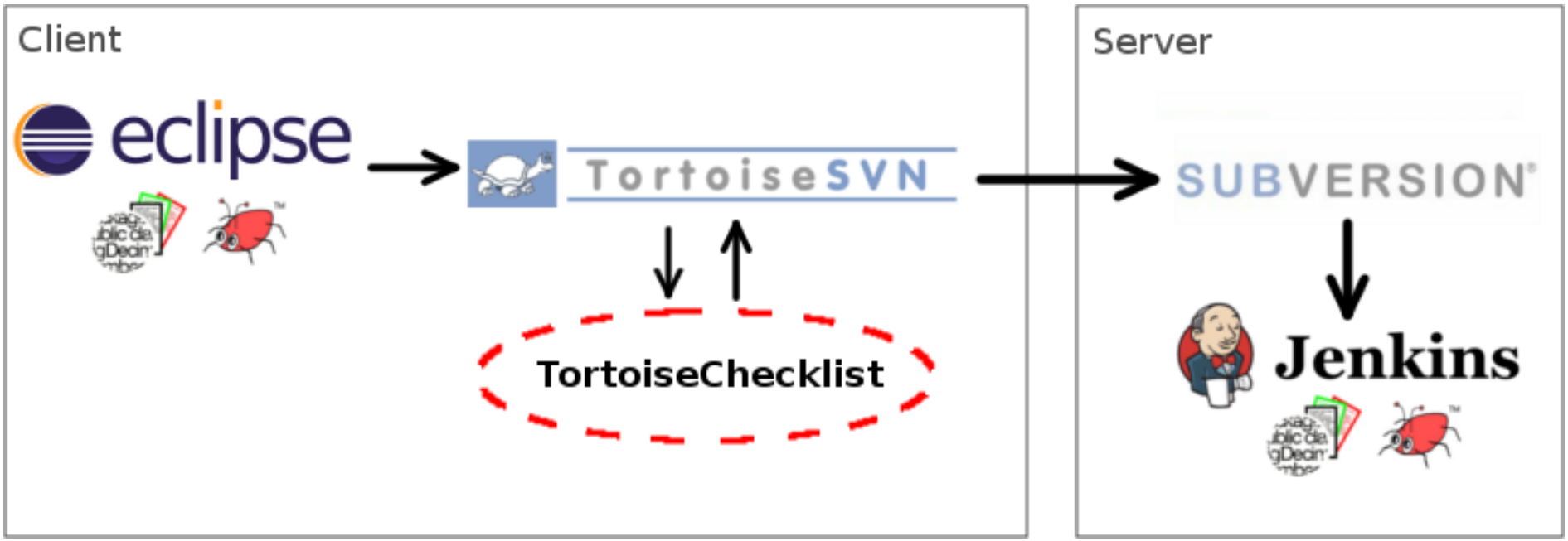
Server vs Client

- Pre-commit Hooks gibt es in zwei Varianten
- server-seitig
  - zentrale Aktivierung, wird immer ausgeführt
- client-seitig
  - vom Client umgehbar
  - Benutzerinteraktion möglich
  - mehr Kontext-Information verfügbar, die den Anteil von „false positives“ senken kann

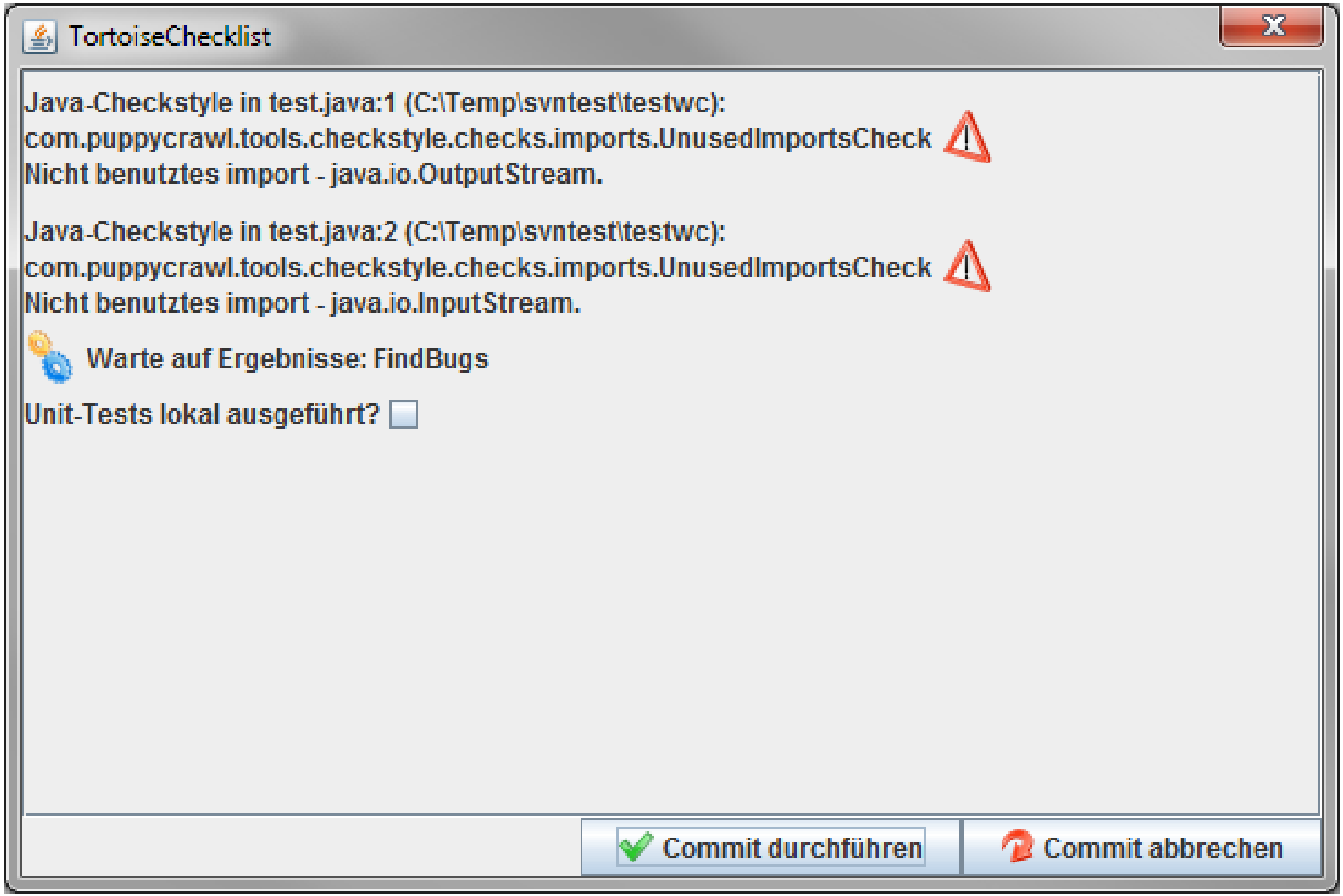


Agenda

Studienkontext – Teil 1: Technik

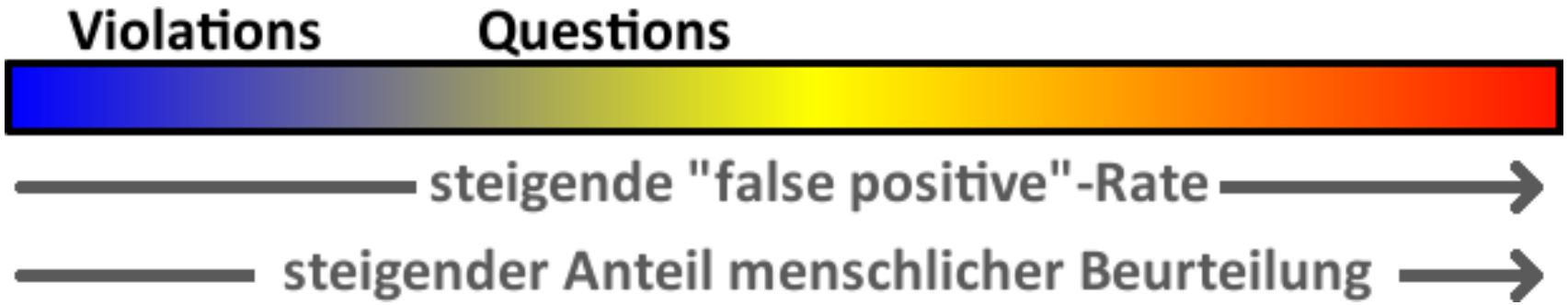


TortoiseChecklist in Aktion



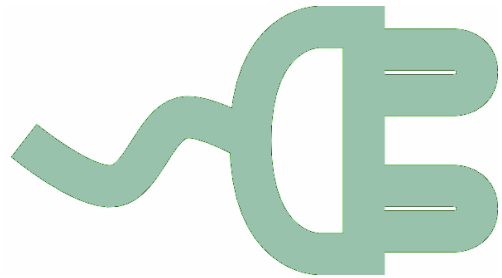
Questions vs Violations

- TortoiseChecklist unterscheidet zwei Arten von Checklisteneinträgen
- „Violations“
  - mit hoher Sicherheit Fehler/Probleme
  - entspricht statischer Analyse mit wenig „false positives“
- „Questions“
  - Prüf-Hinweise für den Menschen
  - entspricht klassischen Checklisten-Fragen und statischer Analyse mit mehr „false positives“



Plugins

- TortoiseChecklist basiert auf Plugins
  - Erweitern der Konfigurations-DSL
  - Durchführen von Prüfungen
- vorhandene Plugins zur Anbindung von
  - Checkstyle
  - FindBugs
  - JIRA
  - firmenspezifischer statischer Analyse (z. B. Prüfung von XML Schema)



Agenda

Studienkontext – Teil 2

- Produkt
  - Standardsoftware für Outputmanagement
  - ca. 580.000 LOC Java-Code (ohne Testcode u. ä.)
  - Entwicklung seit ca. 7 Jahren
- Entwicklungsteam
  - 13 Entwickler
  - ein Standort
- Prozess
  - Agile Entwicklung (Scrum-ban)
  - Nutzung statischer Analyse in Eclipse + auf CI-Server
  - hohe CI-Instabilität (36 % problembehaftete Builds vor Tool-Einführung)

Forschungsfragestellungen

- Wirkung:

  RQ1 Erhöht sich der Anteil stabiler CI-Builds?

- Nebenwirkungen:

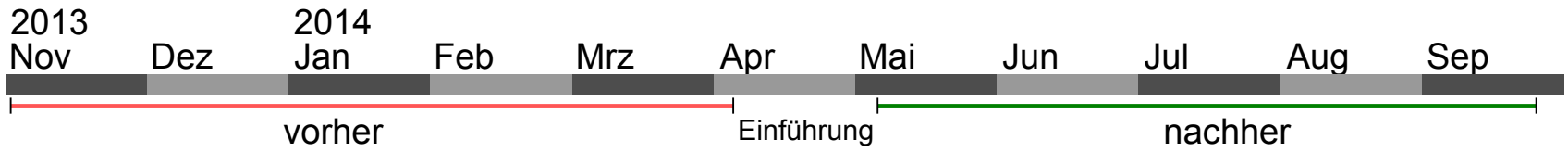
  RQ2 Wird weniger häufig commitet?

  RQ3 Kosten die Prüfungen übermäßig viel Zeit?

  RQ4 Wird die Freiwilligkeit der Prüfungen ausgenutzt? Wie oft werden sie übersprungen?

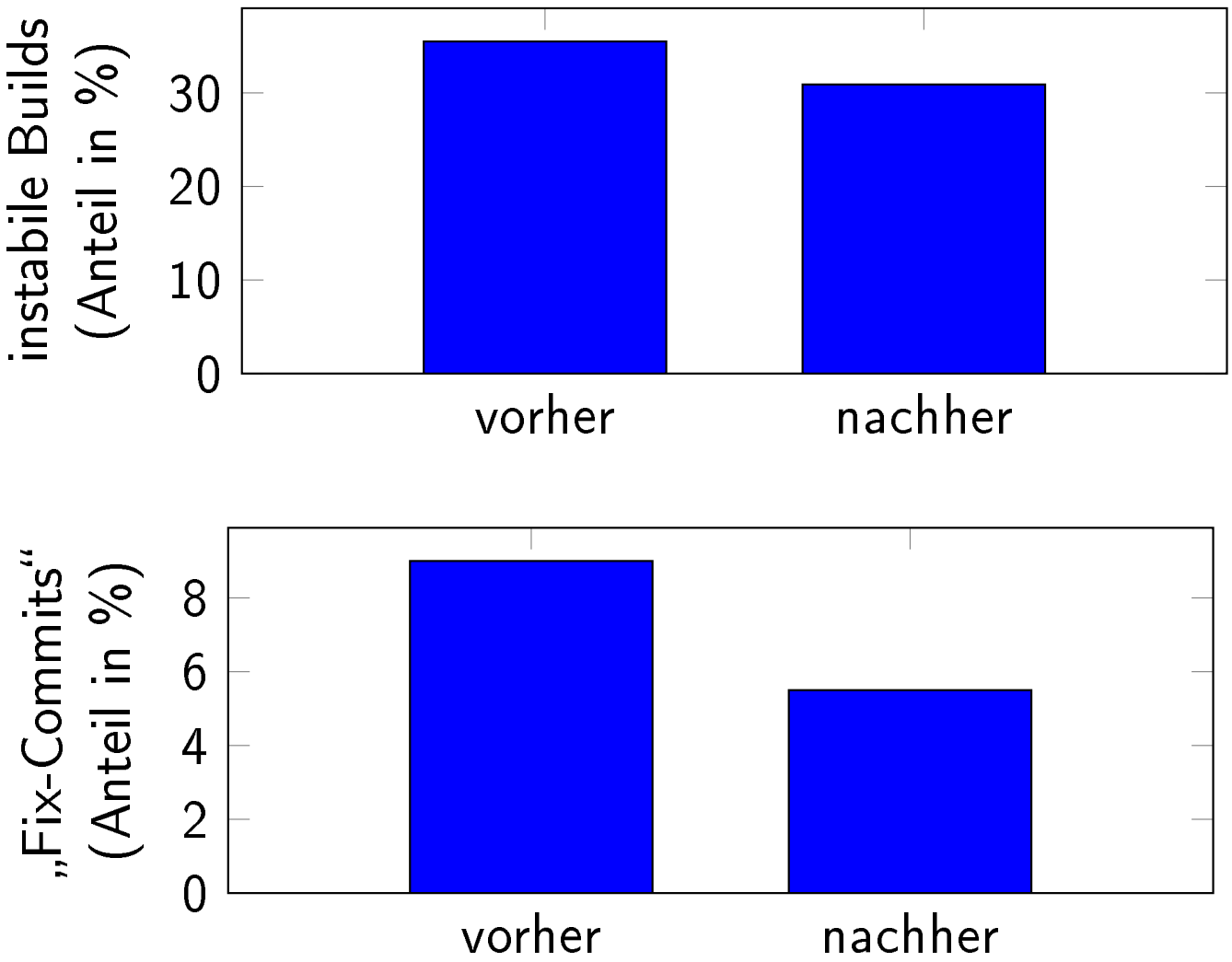
Durchführung der Studie

- zentrale Aktivierung von TortoiseChecklist nach Testphase
  - ca. 90 % der Checkstyle-Checks nutzerübergreifend aktiviert
  - FindBugs-Prüfung nur nutzerspezifisch
  - auch weitere Fragen nur auf Nutzerinitiative
- Betrachtung eines Zeitraums vor und nach Einführung



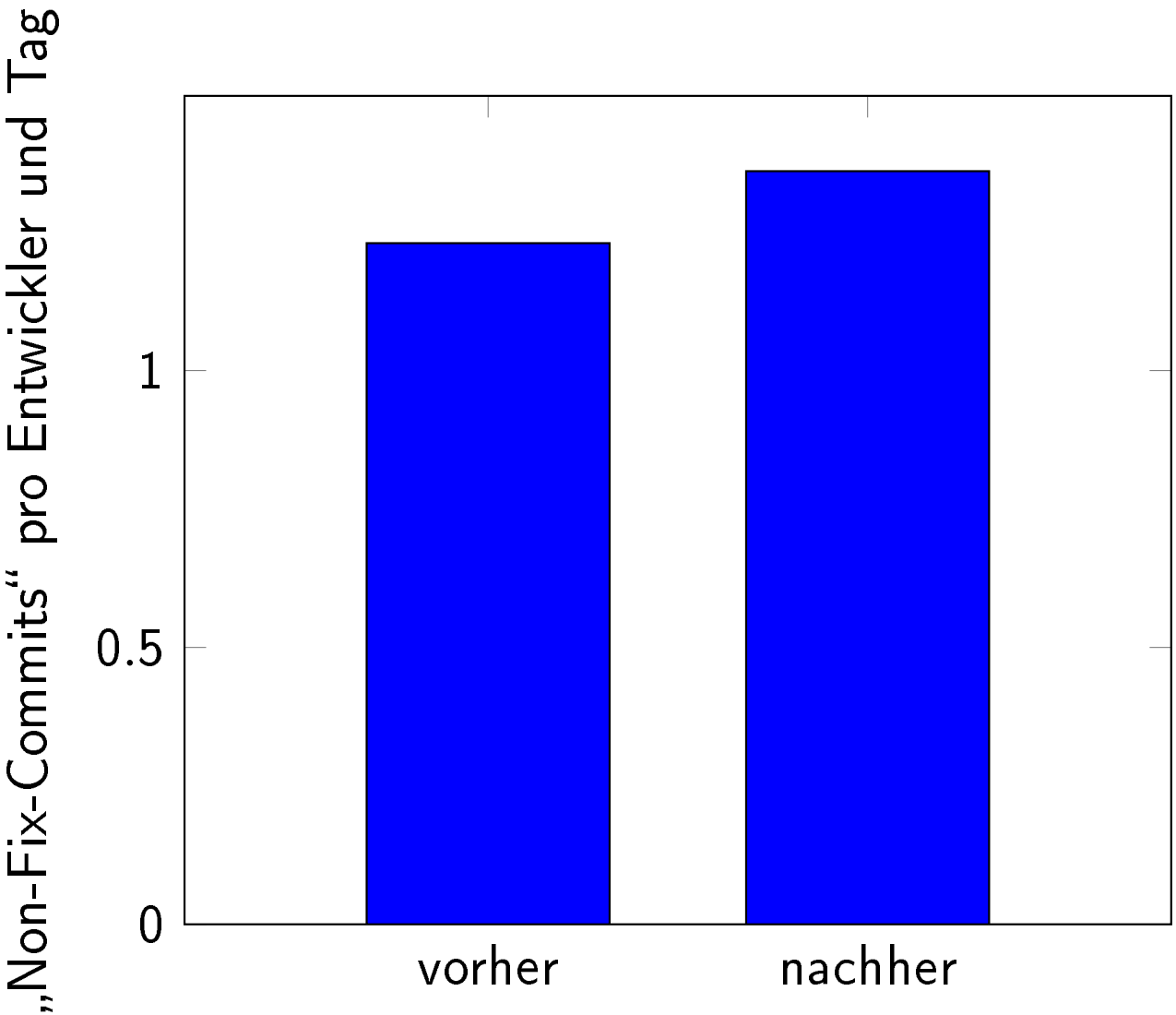
- Bestimmung der Kennzahlen durch Log-Analyse (SVN, Jenkins, TortoiseChecklist)
- bei Commits Unterscheidung zwischen „Fix“ und „Non-Fix“

RQ1: Veränderung der Build-Stabilität

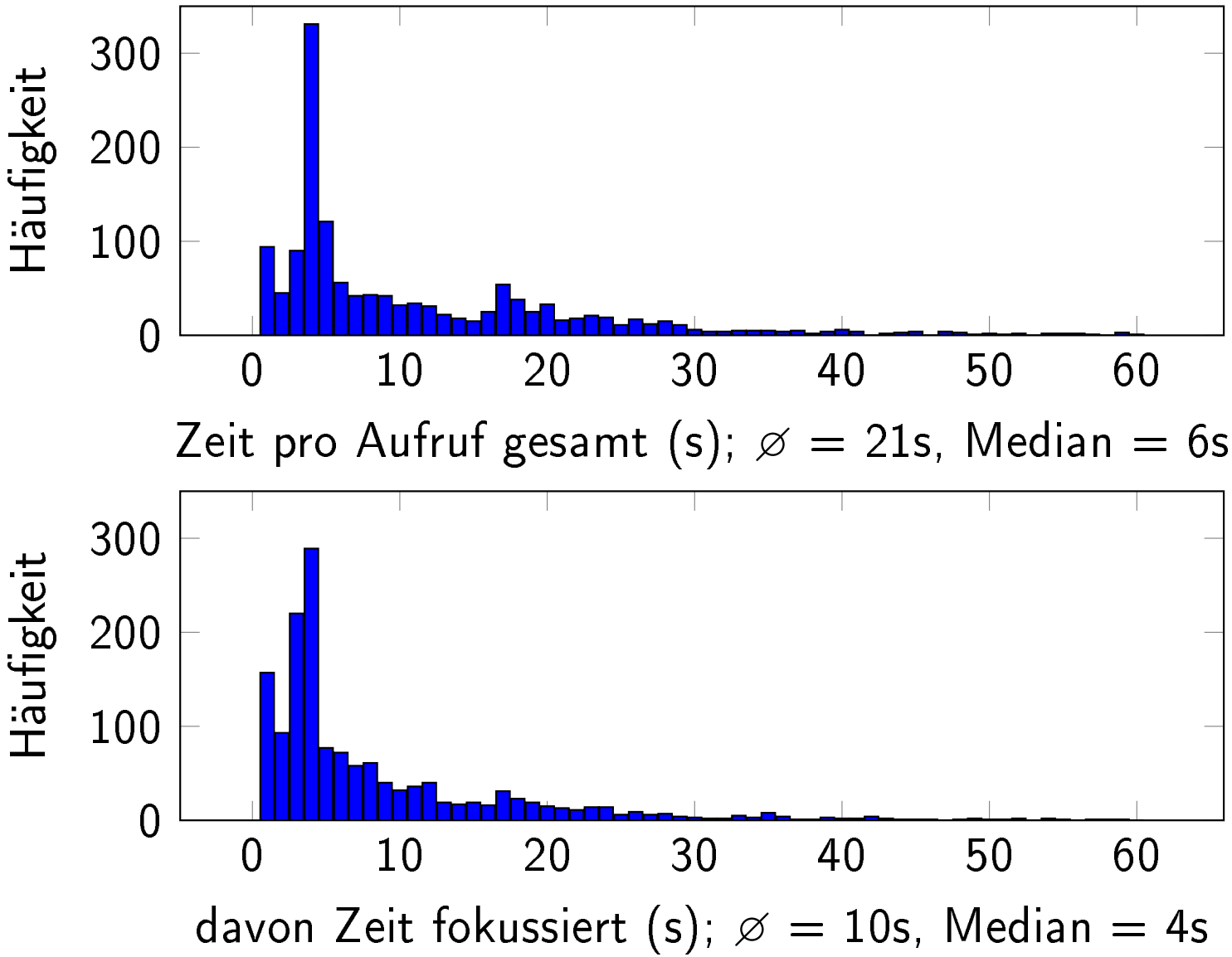


beide Verbesserungen statistisch signifikant mit Signifikanzniveau 1%

RQ2: Veränderung der Commit-Häufigkeit

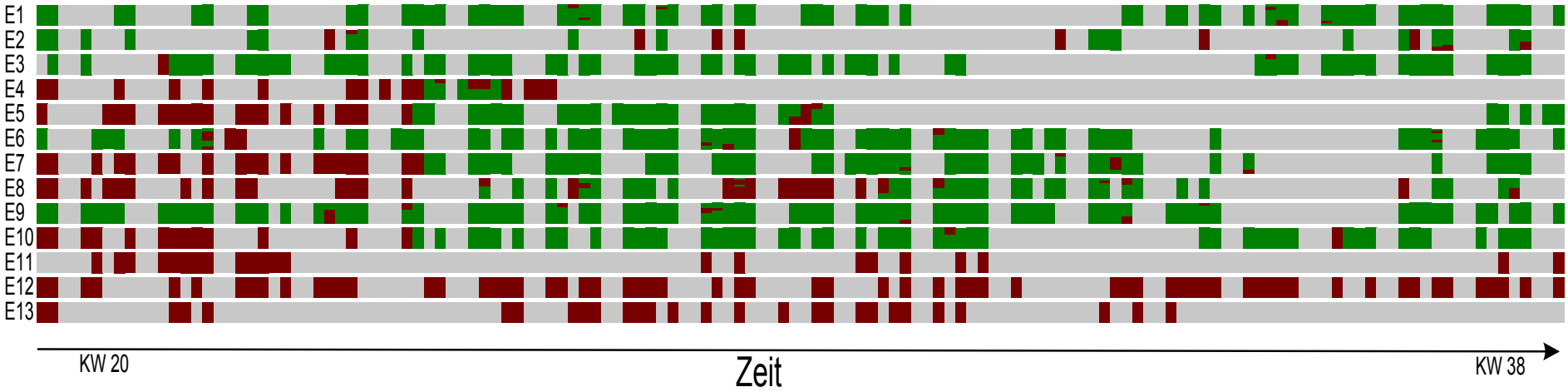


RQ3: Zeitbedarf der Prüfungen



RQ4: Wird häufig übersprungen?

- Tool wurde für 69 % der Commits verwendet (1377 von 1987)
- Entwickler zerfallen grob in drei Gruppen: Nutzung „sofort“, „nach einiger Zeit“ und „nie“



Studie 2: Nutzerbefragung

- Ergänzende Studie: Befragung der betroffenen Entwickler (März 2015)
- Ergebnis: Tool wird durchgehend als hilfreich und kaum störend wahrgenommen

Studie 2: Nutzerbefragung

- Ergänzende Studie: Befragung der betroffenen Entwickler (März 2015)
- Ergebnis: Tool wird durchgehend als hilfreich und kaum störend wahrgenommen
- Einige der negativen Anmerkungen
  - wenn man vorher schon sorgfältig gearbeitet hat, gibt es kaum Zusatznutzen
  - man verlässt sich drauf und wird deshalb überrascht von Dingen, die trotzdem durchrutschen
  - manchmal etwas nervig
  - DSL ist nicht ausreichend dokumentiert
  - manchmal gibt es Probleme mit der FindBugs-Integration

Studie 2: Nutzerbefragung

- Ergänzende Studie: Befragung der betroffenen Entwickler (März 2015)
- Ergebnis: Tool wird durchgehend als hilfreich und kaum störend wahrgenommen
- Einige der negativen Anmerkungen
  - wenn man vorher schon sorgfältig gearbeitet hat, gibt es kaum Zusatznutzen
  - man verlässt sich drauf und wird deshalb überrascht von Dingen, die trotzdem durchrutschen
  - manchmal etwas nervig
  - DSL ist nicht ausreichend dokumentiert
  - manchmal gibt es Probleme mit der FindBugs-Integration
- Einige der positiven Anmerkungen
  - man erhält schnelles Feedback
  - findet viele Kleinigkeiten
  - schützt vor Wiederholung von Fehlern und Vergesslichkeit
  - relativ schnell

Fazit und Ausblick

- Ergebnisse
  - Prüfungen an Client-seitigen Pre-commit Hooks bieten Potential für Verbesserungen im Entwicklungsprozess und ergänzen bestehende Techniken
  - In einer industriellen Fallstudie konnte mit kontextsensitiven Checklisten an Pre-commit Hooks die CI-Stabilität gesteigert werden
  - Die Fallstudie sowie eine nachgelagerte Umfrage belegen die Praktikabilität und Akzeptanz der Technik
- Offene Punkte
  - Lassen sich positive Effekte über die CI-Stabilität hinaus nachweisen?
  - Wie kann Pflege und Verbesserung der Checklisten verstetigt werden?
- Mehr Infos
  - https://github.com/tobiasbaum/TortoiseChecklist
  - tobias.baum@inf.uni-hannover.de