Statische Prüfung in der Praxis

- Es ist nicht sinnvoll, alle vorhandenen Prüfregeln anzuwenden, die ein Werkzeug bietet *Zu viele Warnungen, die nicht mit den Qualitätszielen des Projekts zu tun haben*
- Es müssen daher für das Unternehmen/das Projekt hilfreiche Prüfregeln identifiziert werden
- Für jede Prüfregel ist eine passende Warnstufe festzulegen (z.B. Warning oder Error)
  - Für jede Warnstufe ist eine passende Folge festzulegen (z.B. Build nicht möglich)
- Die statischen Prüfungen müssen in den Entwicklungsprozess eingebunden werden (Wer, Was, Wann, Wie, Womit)

  Qualitätssicherungsplan
- Die Werkzeuge sind entsprechend zu konfigurieren

- Einen Spezialfall stellt die Softwaremessung dar
- Der Wert einer einmaligen Messung ist beschränkt
- Das Ziel ist daher die Einrichtung einer andauernden (kontinuierlichen) Messung (nur so können die Messwerte die Grundlage für eine Steuerung oder Vorhersage sein)
- Es ist eine Messdatenbank aufzubauen
- Vermessung des Prozesses
  - Produktivität
  - Ressourceneinsatz
  - Termintreue
- Vermessung des Produkts
  - Quantität
  - Qualität
  - Komplexität

Erfordert eine kontinuierliche Erfassung der Aufwände

- Die Mitarbeiter stehen einer kontinuierlichen Messung in der Regel sehr skeptisch gegenüber
  - Sie befürchten eine Bewertung ihrer persönlichen Produktivität und die Bewertung der Qualität ihrer Arbeitsergebnisse
  - Sie zweifeln die Aussagekraft der/einiger Metriken an (evtl. auch zu Recht)

- Der Aufbau einer kontinuierlichen Messung ohne die Beteiligung der Mitarbeiter ist daher nicht ratsam

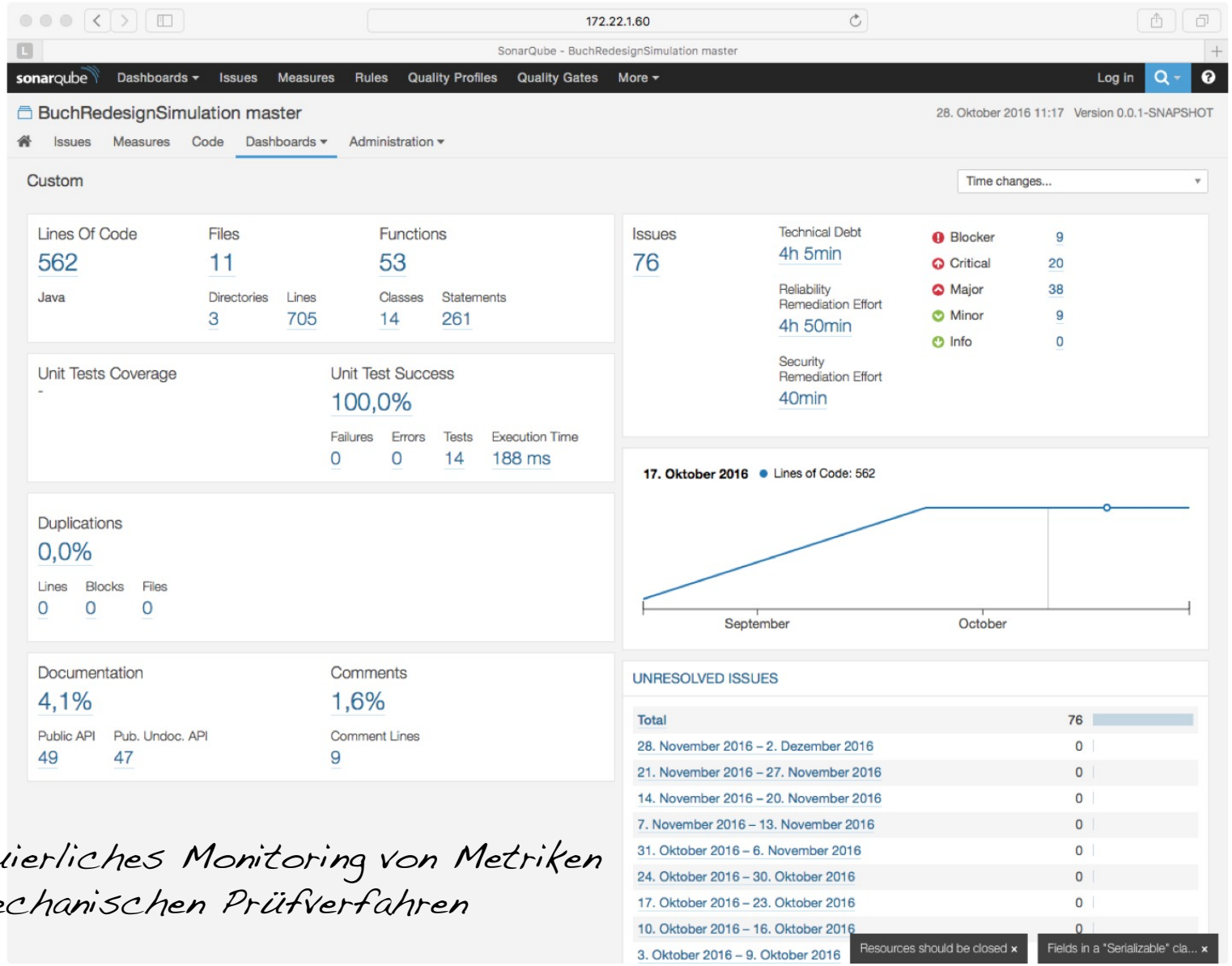
- Beteiligung der Mitarbeiter
  - Vorschlag von Metriken
  - Anpassung von Metriken
- Transparenz
  - Veröffentlichung der Messmethoden
  - Jeder Mitarbeiter hat Zugriff auf die Messdaten

- Automatisierung mit Maven
  - Konventionsanalyse: Checkstyle

```
<plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-checkstyle-plugin</artifactId>
   <version>3.1.0</version>
   <configuration>
      <configLocation>
         google_checks.xml
      </configLocation>
   </configuration>
</plugin>
```

  - Statische Codeanalyse (auf Bytecode-Ebene): SpotBugs (FindBugs)

```
<plugin>
   <groupId>com.github.spotbugs</groupId>
   <artifactId>spotbugs-maven-plugin</artifactId>
   <version>3.1.12.2</version>
</plugin>
```



Kontinuierliches Monitoring von Metriken und mechanischen Prüfverfahren

Kontinuierliche Integration (*Continuous Integration*)

- Kontinuierliche Integration (CI) ist eine Softwareentwicklungspraktik
  - Neu Artefakte / Komponenten werden häufig (täglich) integriert
  - Integration erfolgt durch einen vollautomatischen Build-Prozess

- Ziele von CI
  - Reduzierung von Risiken *primär*
    - Fehler bei der Integration können leichter/schneller lokalisiert werden
    - Eine lauffähige Produktversion kann jederzeit ausgeliefert werden
    - Nach einer Fehlerkorrektur kann schnell eine korrigierte Programmversion zur Verfügung gestellt werden
    - Probleme mit Nicht-funktionalen Anforderungen fallen frühzeitig auf

    Abhängig von den Maßnahmen im Qualitätssicherungsplan
  - Verbesserung der Produktqualität
    - Statische und dynamische Prüfungen werden regelmäßig (automatisiert) durchgeführt
    - Abweichungen von Soll-Werte werden frühzeitig erkannt

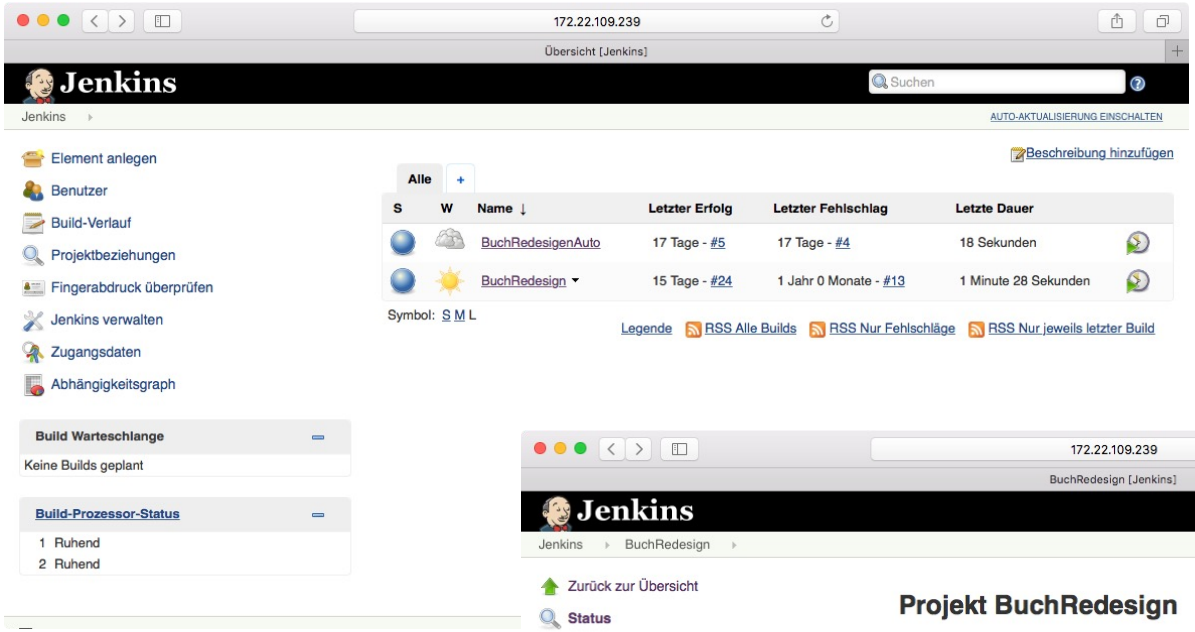
- Transparenz / Übersicht
  - Kennzahlen
  - Berichtswesen
  - Historie

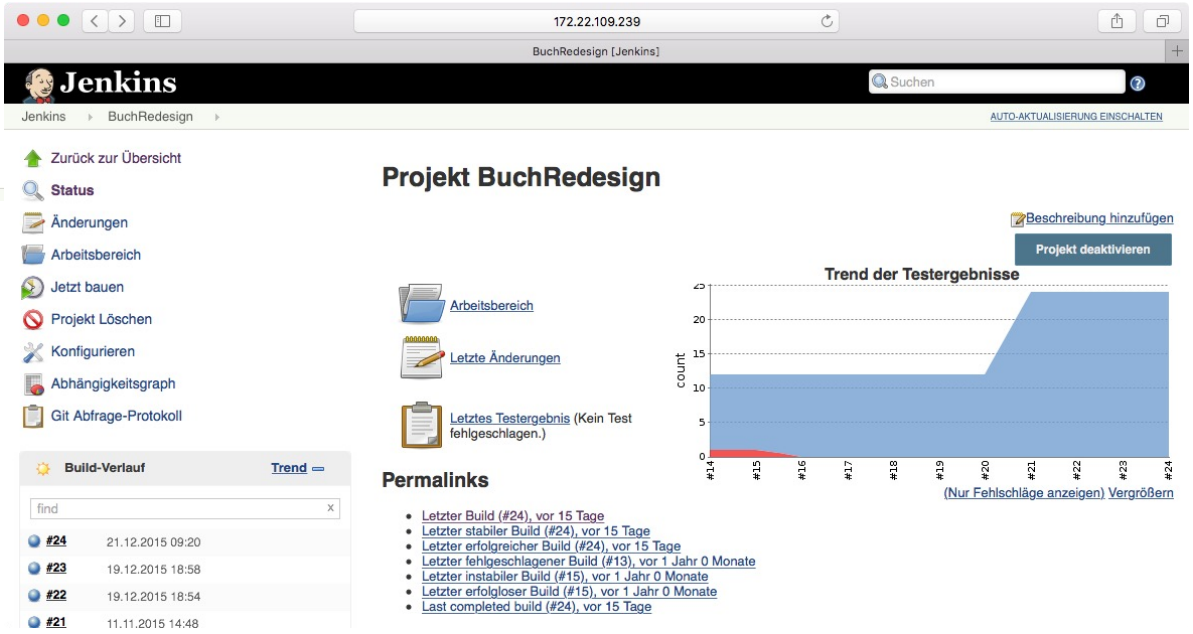
– CI-Server

unabhängig von der Entwicklungsumgebung

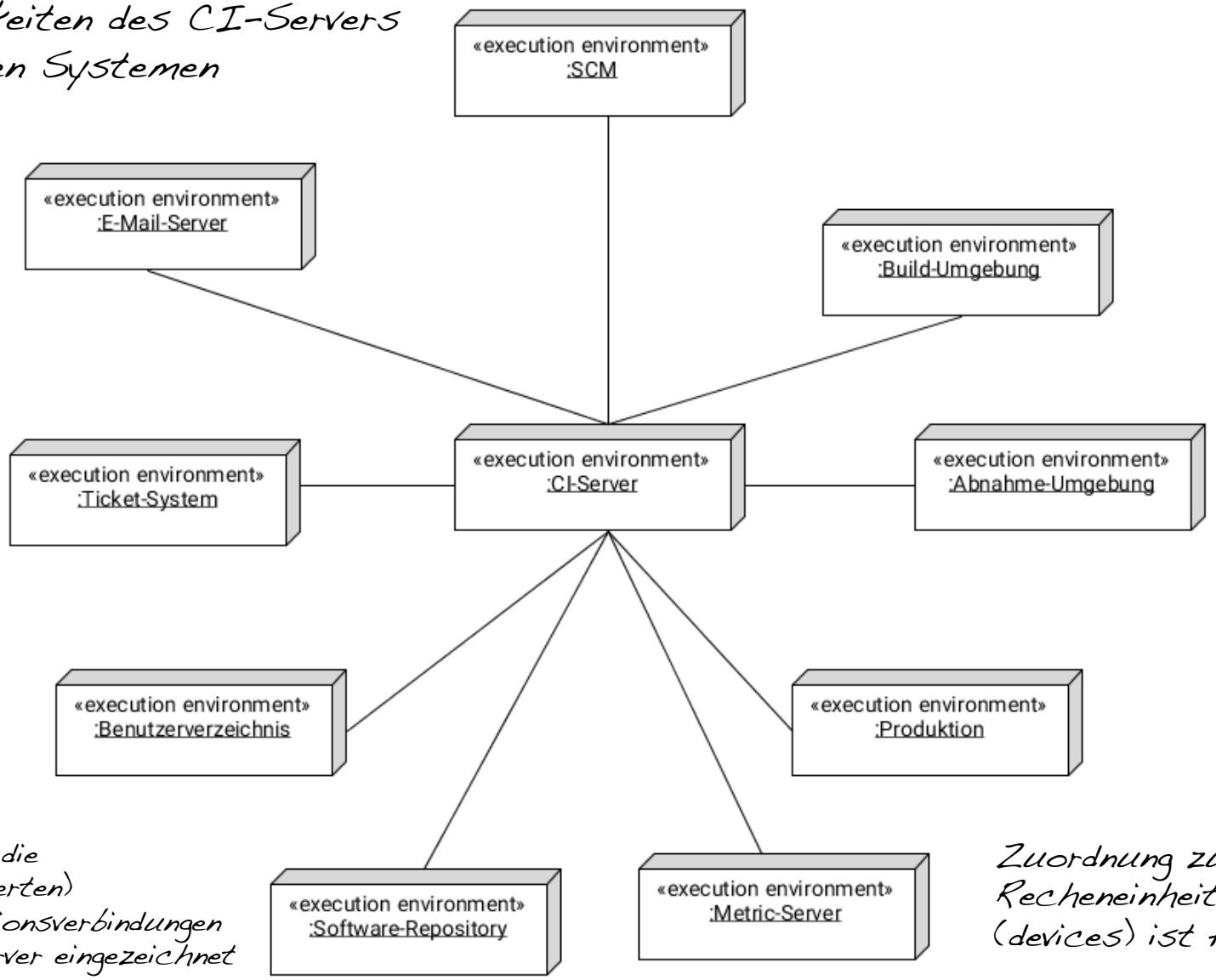
Laufzeitumgebung zur Durchführung des kompletten Integrationsprozesses

- Bereitstellung einer Oberfläche (häufig Web-Anwendung)
  - Konfiguration eines Build-Prozesses
  - Build-Status anzeigen
  - Historie von Builds anzeigen
- Auslösen eines Build-Prozesses
  a) Manuell
  b) Ereignisgesteuert (Änderungen im SCM)
  c) Zeitgesteuert (z.B. Nightly-Builds)


172.22.109.239
Übersicht [Jenkins]
Jenkins
Suchen
Jenkins
AUTO-AKTUALISIERUNG EINSCHALTEN
Element anlegen
Benutzer
Build-Verlauf
Projektbeziehungen
Fingerabdruck überprüfen
Jenkins verwalten
Zugangsdaten
Abhängigkeitsgraph
Beschreibung hinzufügen
Alle
S
W
Name ↓
Letzter Erfolg
Letzter Fehlschlag
Letzte Dauer
BuchRedesigenAuto
17 Tage - #5
17 Tage - #4
18 Sekunden
BuchRedesign
15 Tage - #24
1 Jahr 0 Monate - #13
1 Minute 28 Sekunden
Symbol: S M L
Legende
RSS Alle Builds
RSS Nur Fehlschläge
RSS Nur jeweils letzter Build
Build Warteschlange
Keine Builds geplant
Build-Prozessor-Status
1 Ruhend
2 Ruhend


172.22.109.239
BuchRedesign [Jenkins]
Jenkins
Suchen
Jenkins
BuchRedesign
AUTO-AKTUALISIERUNG EINSCHALTEN
Zurück zur Übersicht
Status
Änderungen
Arbeitsbereich
Jetzt bauen
Projekt Löschen
Konfigurieren
Abhängigkeitsgraph
Git Abfrage-Protokoll
Projekt BuchRedesign
Beschreibung hinzufügen
Projekt deaktivieren
Arbeitsbereich
Letzte Änderungen
Letztes Testergebnis (Kein Test fehlgeschlagen.)
Trend der Testergebnisse
count
#14
#15
#16
#17
#18
#19
#20
#21
#22
#23
#24
(Nur Fehlschläge anzeigen) Vergrößern
Permalinks
Letzter Build (#24), vor 15 Tage
Letzter stabiler Build (#24), vor 15 Tage
Letzter erfolgreicher Build (#24), vor 15 Tage
Letzter fehlgeschlagener Build (#13), vor 1 Jahr 0 Monate
Letzter instabiler Build (#15), vor 1 Jahr 0 Monate
Letzter erfolgloser Build (#15), vor 1 Jahr 0 Monate
Last completed build (#24), vor 15 Tage
Build-Verlauf
Trend
find
#24 21.12.2015 09:20
#23 19.12.2015 18:58
#22 19.12.2015 18:54
#21 11.11.2015 14:48

Abhängigkeiten des CI-Servers zu anderen Systemen



Es sind nur die (unspezifizierten) Kommunikationsverbindungen zum CI-Server eingezeichnet

Zuordnung zu Recheneinheiten (devices) ist flexibel

- Build-Prozess umfasst
  - Vorbereitende Maßnahmen
    - Löschen und Anlegen von Verzeichnissen
    - Auschecken von Artefakten aus dem SCM
    - Bereitstellen von Abhängigkeiten (z.B. Bibliotheken)
  - Übersetzen
    - Compilieren und Binden
  - Statische Prüfungen
  - Dynamische Prüfungen
    - Unit- und Integrationstests
    - Glass-Box-Test (Anweisungs- und Zweigüberdeckung)
  - Paketierung
  - Erzeugung von Berichten und Dokumentation
  - Bereitstellung / Verteilung

- CI-Server nutzt verschiedene Werkzeuge (häufig über Build-Tool)
  - Build-Tools (*Ant*, *Maven*)
  - Compiler / Linker (*javac*, *gcc*, *csc*)
  - Statische Analysewerkzeuge (*PMD*, *Checkstyle*, *FindBugs*)
  - Unit-Test-Runner (*JUnit*)
  - Testberichte (*Surefire*)
  - Glass-Box-Test (*Cobertura*)

- CI-Praktiken
  - Gemeinsame Codebasis im SCM
  - Automatisierte Builds
  - Automatisierte Durchführung von dynamischen und statischen Prüfungen
  - Automatisches Berichtswesen
  - Häufige Integration
  - Schnelle Build-Zyklen
  - Einfacher Zugriff auf Build-Ergebnisse und Berichte
  - Automatische Verteilung (und Bereitstellung der Umgebungen) -> Continuous Deliviery

Konfigurationsmanagement

- Softwaresysteme lassen sich durch folgende Eigenschaften charakterisieren
  - Bestehen aus zahlreichen Artefakten
  - Besitzen zahlreiche Abhängigkeiten (z.B. zu anderen Systemen)
  - Unterliegen laufend Änderungen (funktionale Erweiterungen, Fehlerbehebung, ...)
  - Der „Bau“ eines Releases kann ein komplexer Vorgang sein
  - Können sich in unterschiedlichen Versionen im Einsatz befinden
  - Werden in einem Team entwickelt

- Damit ergeben sich negative Auswirkungen auf die Qualität und Produktivität durch
  - Unkontrollierte Aktivitäten
  - Willkürliche Änderungen
  - Mangelnde Transparenz
  - Fehlende Nachvollziehbarkeit
  - unzureichende Automatisierung

- Konfigurationsmanagement
  - Ein sich änderndes Softwaresystem wird mit Hilfe von Richtlinien, Prozessen und Werkzeugen aktiv verwaltet
  - Das Konfigurationsmanagement ist eine konstruktive Maßnahme der Qualitätssicherung (Prozessqualität)

- Ein Konfigurationsmanagement stellt sicher, dass
  - alte Versionsstände wieder hergestellt werden können
  - jede Änderung nachvollzogen werden kann (Wer, Wann, Was, Warum)
  - nachvollzogen werden kann, welches Release wohin ausgeliefert wurde
  - der Zustand eines Änderungsauftrags stets eingesehen werden kann
  - die Teamarbeit koordiniert erfolgen kann
  - keine falsche Version ausgeliefert wird
  - ein hoher Automatisierungsgrad bei einem Auslieferungsprozess erreicht wird

– Das Konfigurationsmanagement umfasst vier miteinander verbundene Bereiche und erfordert geeignete Prozesse und den Einsatz unterschiedlicher Werkzeuge

hier: pragmatisch, mit Open-Source Werkzeugen