sepp.med
Qualität sichert Erfolg

Automatisierte Validierung einer Continuous Integration Toolkette

Ein Erfahrungsbericht

Beatrix Forster

Über uns

- Über 35 Jahre Erfahrung im industriellen Umfeld
  - Medizintechnik
  - Pharmazie
  - Automotive
  - Automatisierung
  - Avionik/Defense
  - Finance
  - Öffentlicher Dienst
- Expertise:

  komplexe und sicherheitskritische Systeme

Unser Serviceportfolio

Engineering

Softwareentwicklung

- Requirements Engineering
- Architekturentwicklung
- Design
- Implementierung
- Integrationsmanagement
- Performance-Optimierung
- Agile Entwicklung/SCRUM

Test

- Testmanagement
- Testdesign, -implementierung
- Testdurchführung
- Testautomatisierung
- Management der Infrastruktur
- Security Testing
- .modellzentrierterTest

Beratung

- Prozesse/Reifegradmodelle
- Konformität zu Normen und Standards
- Methoden
- Werkzeugunterstützung
- Zertifizierung und Validierung
- eigenes Validierungs-Framework

Training

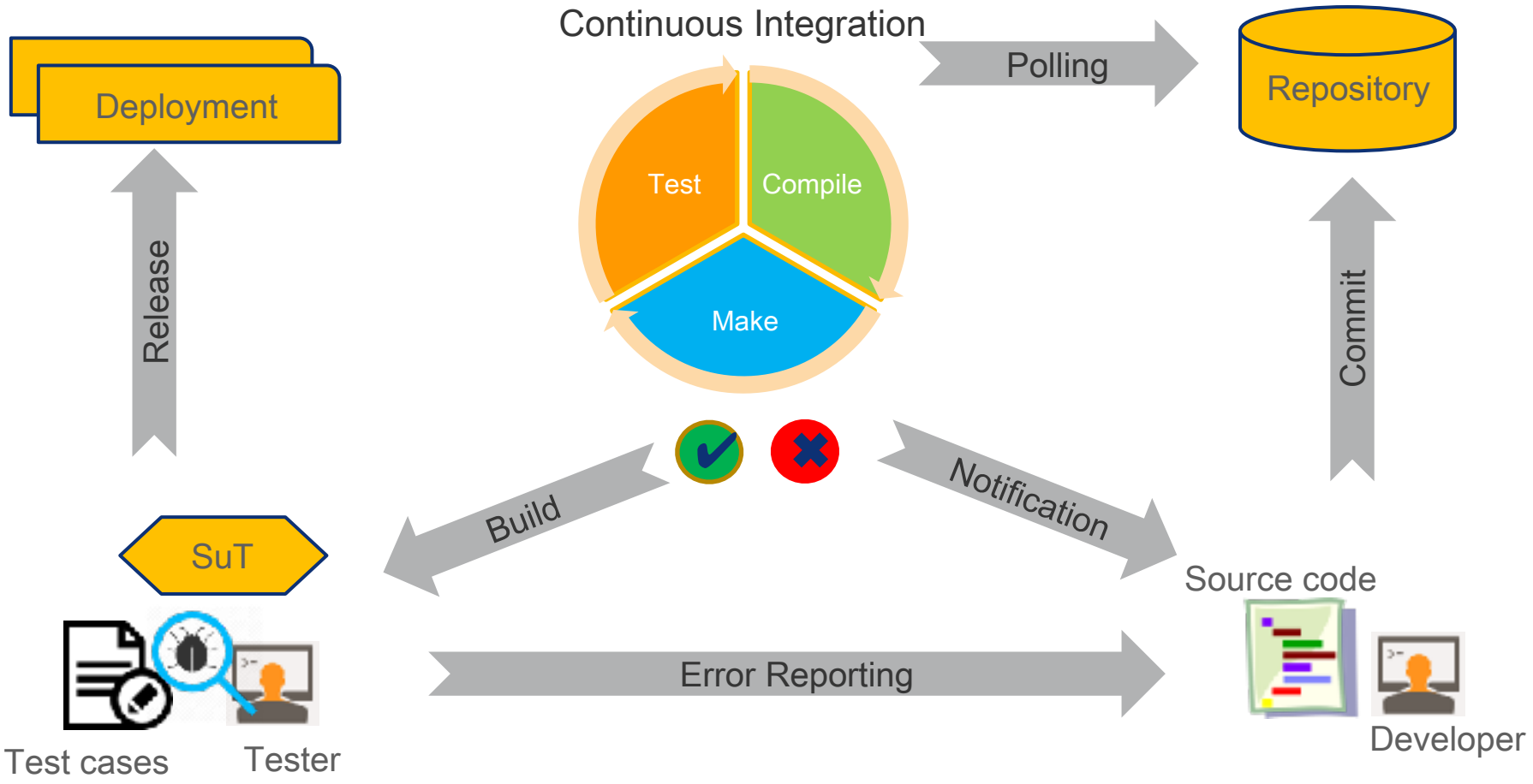
- ASQF® CPPM
- IREB® CPRE
- ISTQB® Certified Tester
- ISTQB® CTFL - Agile Extension
- ISTQB® CTFL - MBT
- ICPMSB® CPMS
- ASQF® CPIoT

- Anforderungs-management
- Risiko-management
- Änderungs-management
- Konfigurations-management
- Projekt-management

Das Projekt

- Aufgabe
  - Validierung einer Continuous Integration Toolkette
- Branche Medizintechnik
  - Konform zu FDA Guidelines und ISO/IEC 13485
- Umfang
  - 9 Personenmonate
  - 1 Requirements-Ingenieur, 2 Testspezialisten
- Vorgehensweise
  - Risiko- und Workflow-basiert
  - Einsatz von automatisierten Tests
- Ergebnis
  - Es wurden Abweichungen festgestellt

Continuous Integration – was ist das?



Validierung – Vorgehensweise

- Schritt 1: Analyse der Prozesse und Werkzeuge
- Schritt 2: Spezifizieren der Anforderungen
- Schritt 3: Testkonzept
- Schritt 4: Modellbasierte Testfallerstellung
- Schritt 5: Testautomatisierung
- Schritt 6: Testdurchführung und Dokumentation
- Schritt 7: Bewertung der Testergebnisse

Analyse der Tools

Jenkins Server – steuert den Prozess

Build-Umgebung

- Compiler (4 Tools)
- Make-Tool (4 Tools)
- Checksum Generator

Test-Umgebung

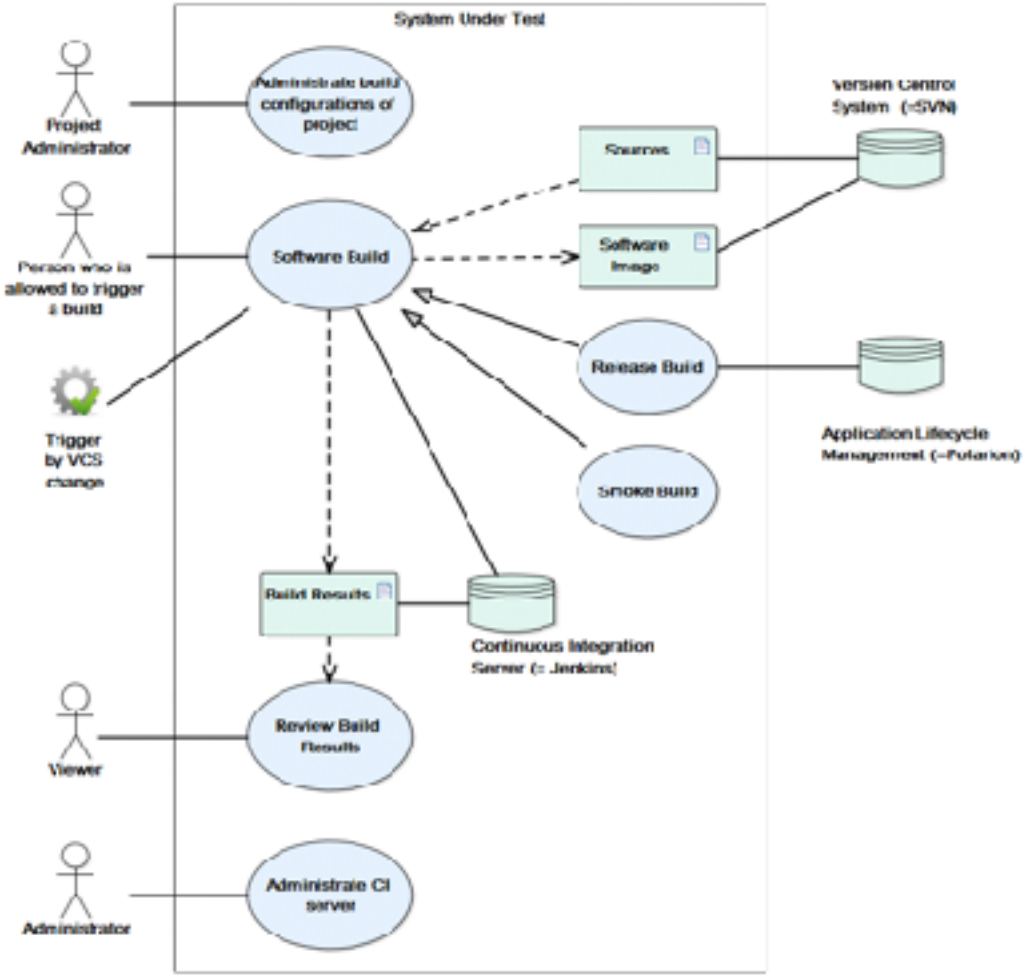
- Statische Code-Analyse (2 Tools)
- Unit-Test Framework
- Memory Leakage Detection
- Code Coverage Analyzer

Sonstige

- Dokumenten-Generatoren (2 Tools)
- 1 Hilfswerkzeug zur Ausführung von Windows-Tools unter Linux

Analyse der Prozesse

- SOPs und Guidelines
- Stakeholder Interviews
- Modellierung der Use Cases
- Review durch Stakeholder und Process-Owner

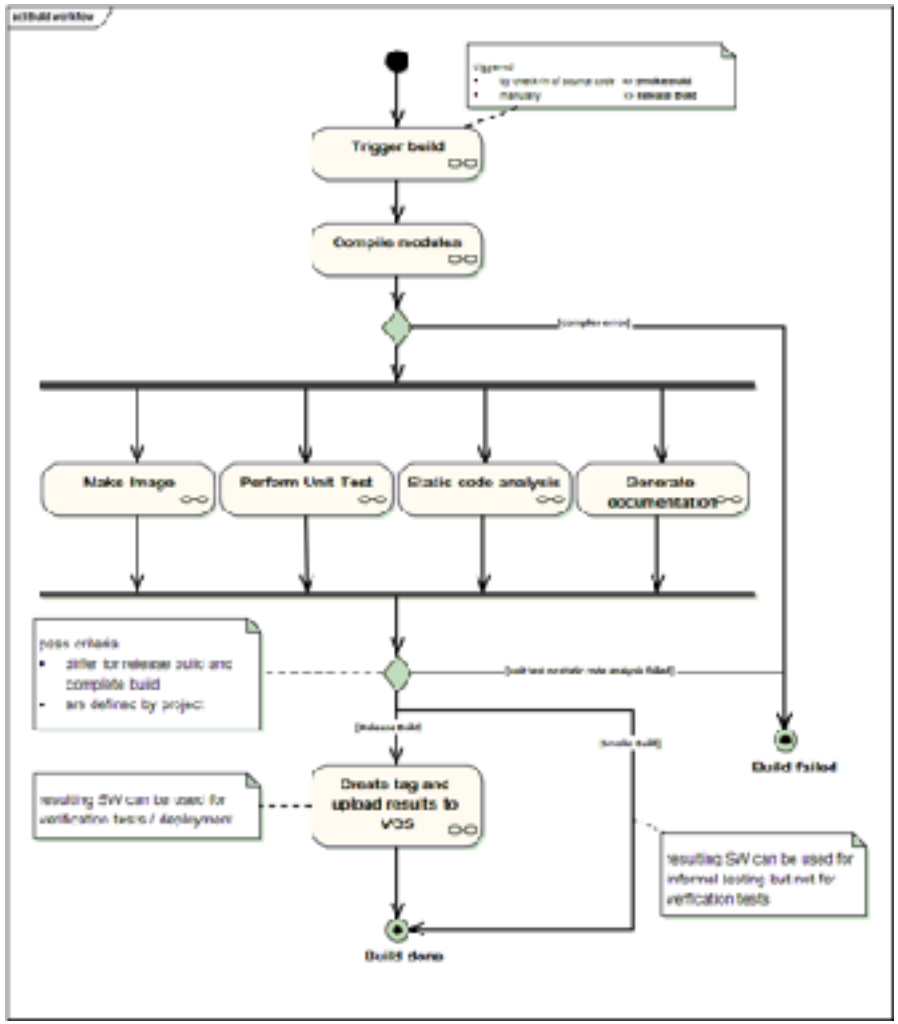


Validierung – Vorgehensweise

- Schritt 1: Analyse der Prozesse und Werkzeuge
- **Schritt 2: Spezifizieren der Anforderungen**
- Schritt 3: Testkonzept
- Schritt 4: Modellbasierte Testfallerstellung
- Schritt 5: Testautomatisierung
- Schritt 6: Testdurchführung und Dokumentation
- Schritt 7: Bewertung der Testergebnisse

Spezifizieren der Anforderungen

- Funktionale Anforderungen aus Workflow-Modellen ableiten



Spezifizieren der Anforderungen

- Risikoanalyse
  - Einfluss auf die Produktqualität
  - Sichtbarkeit von Fehlern
    - z.B. unabsichtliches Ändern der Konfiguration

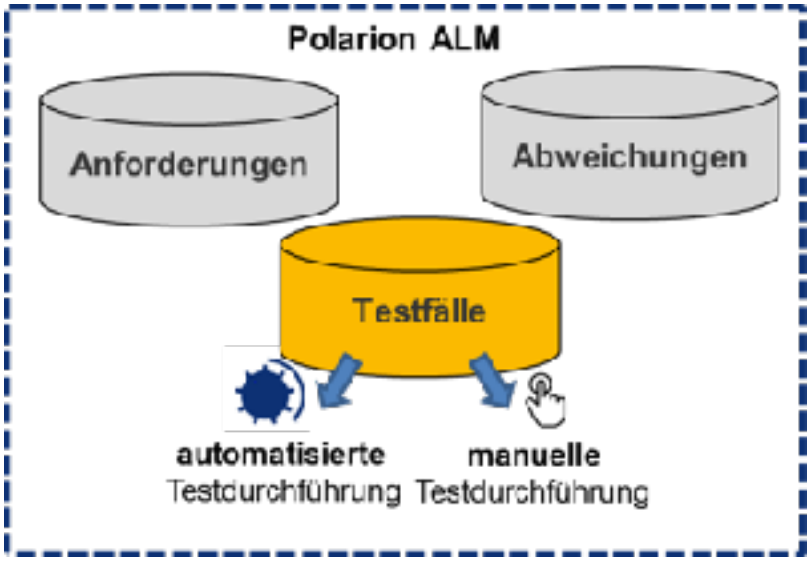
- Nichtfunktionale Anforderungen
  - Beispiele
    - Max. Dauer von Builds
    - Volumen des Source-Codes
    - Tolerable Ausfallzeiten => Dauer des Restore
    - Portabilität
  - aus Stakeholder-Interviews und Risikoanalyse

Validierung – Vorgehensweise

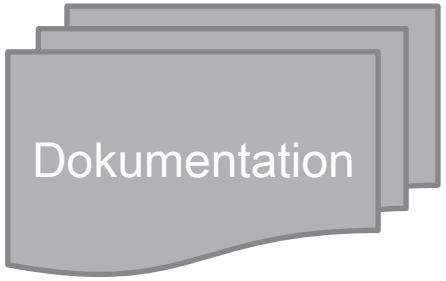
- Schritt 1: Analyse der Prozesse und Werkzeuge
- Schritt 2: Spezifizieren der Anforderungen
- **Schritt 3: Testkonzept**
- Schritt 4: Modellbasierte Testfallerstellung
- Schritt 5: Testautomatisierung
- Schritt 6: Testdurchführung und Dokumentation
- Schritt 7: Bewertung der Testergebnisse

Testkonzept

- Automatisierte Tests wo möglich und sinnvoll
- Manuelle Tests soweit erforderlich
- Alle Tests im selben Repository



Generierte Dokumente:
- Anforderungsspezifikation
- Testspezifikation



Vorlagenbasierte Dokumente:
- Validierungsplan
- Validierungsbericht

Testautomatisierung - Motivation

- Art der Tests
  - Auslesen und Verifizieren von Log-Dateien wäre manuell weder effektiv noch effizient
- Anzahl der Tests
  - Sämtliche Regeln nach Misra C++:2008
  - Sehr viele identische Testabläufe mit unterschiedlichen Daten
- Wiederholbarkeit von Tests
  - Zahlreiche Regressionen, da die Toolkette noch verändert wurde
  - Sichere und schnelle Prüfung bei projektbedingtem Update der Konfiguration der Continuous Integration

Testkonzept

Manueller Test

- Role-based workflows
- Installation Qualification
- Performance Qualification
- Backup- / Restore

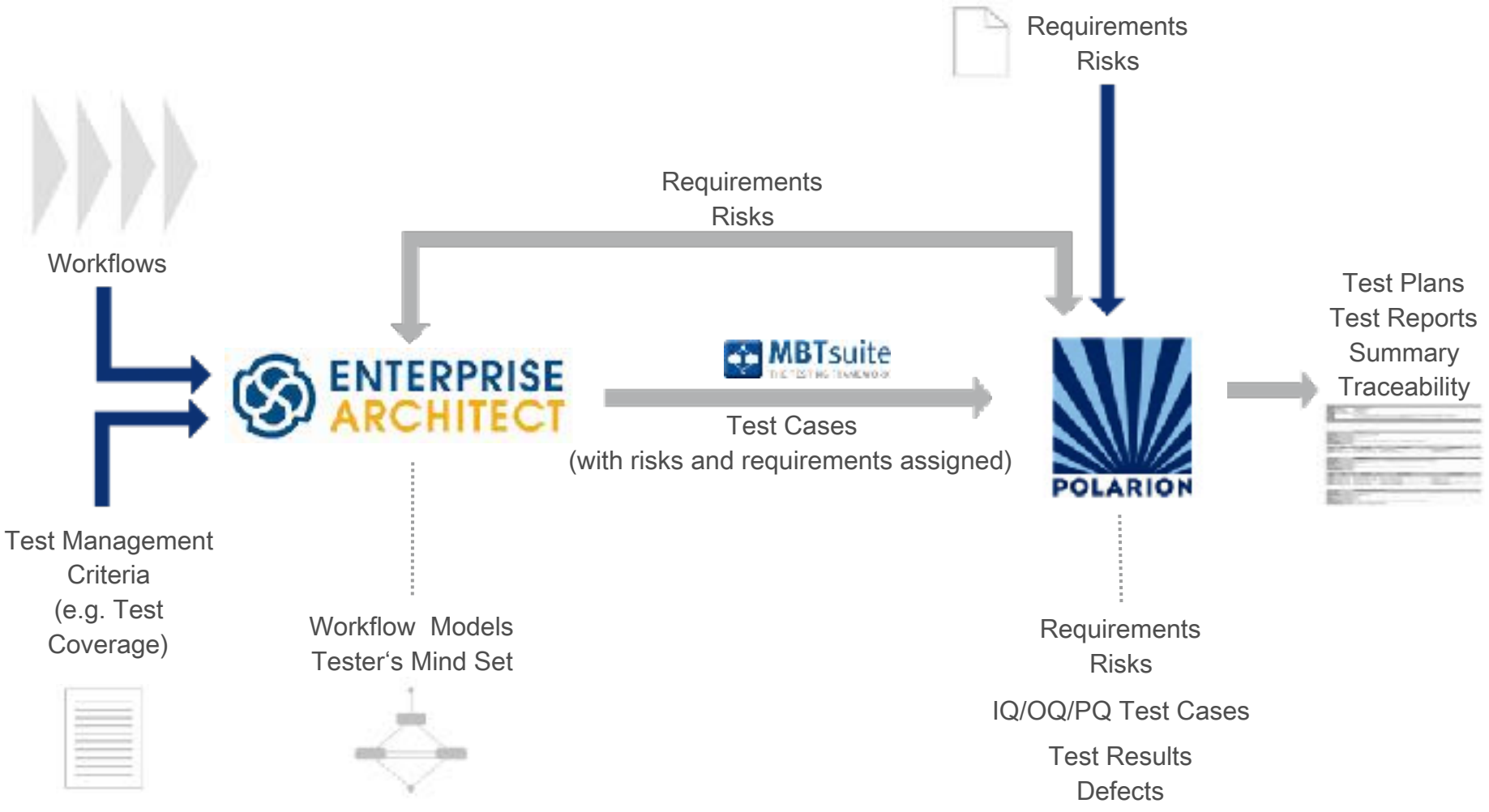
Automatischer Test

- Test data driven tests
  - Compiler
  - Static Code Analyzer
  - Unit Test
  - SW Image Creation

Validierung – Vorgehensweise

- Schritt 1: Analyse der Prozesse und Werkzeuge
- Schritt 2: Spezifizieren der Anforderungen
- Schritt 3: Testkonzept
- **Schritt 4: Modellbasierte Testfallerstellung**
- Schritt 5: Testautomatisierung
- Schritt 6: Testdurchführung und Dokumentation
- Schritt 7: Bewertung der Testergebnisse

modellbasierte Testfallerstellung



Modellbasierte Testfallerstellung

- Workflow-orientiert
  - anhand des geplanten Prozesses
  - anhand der Rollen im Prozess
- Risiko-basiert
  - verstärkte Prüfung da
    - wo Einfluss auf Produktqualität besteht
    - wo Fehler nicht offensichtlich sind
- Systematisch & Dokumentiert
  - was wird wie getestet
  - was wird nicht getestet (Nachweis, das etwas nicht vergessen wurde)

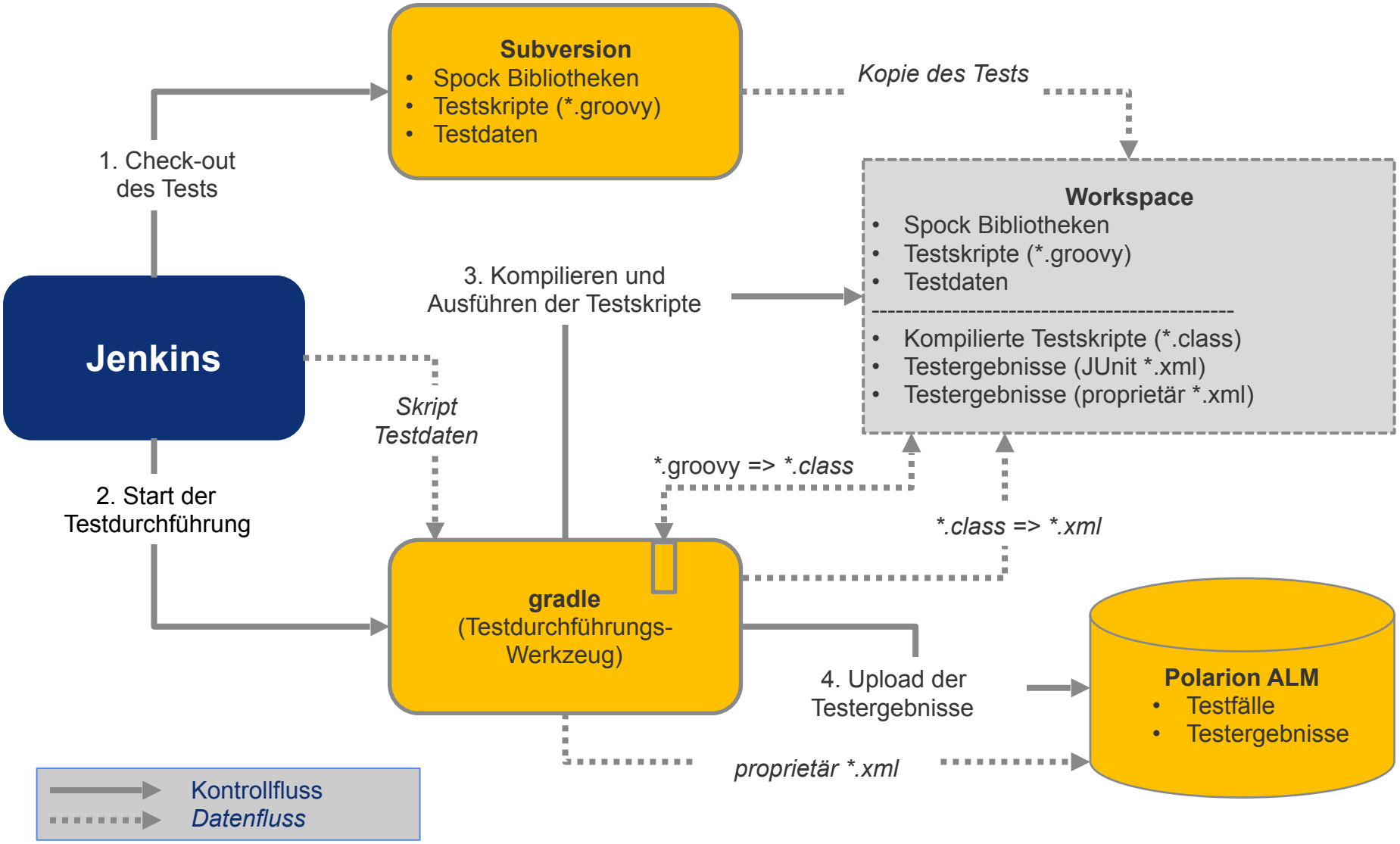
Validierung – Vorgehensweise

- Schritt 1: Analyse der Prozesse und Werkzeuge
- Schritt 2: Spezifizieren der Anforderungen
- Schritt 3: Testkonzept
- Schritt 4: Modellbasierte Testfallerstellung
- **Schritt 5: Testautomatisierung**
- Schritt 6: Testdurchführung und Dokumentation
- Schritt 7: Bewertung der Testergebnisse

Testautomatisierung - Aufbau

- Testumgebung strikt getrennt von Produktiv-Umgebung

- Separater Jenkins-Server
  - Zu testende Werkzeuge
- Test Repository in Subversion
  - Testframework (Spock) Bibliotheken und Testskripte
  - Testdaten
    - Projekt 1 analog zu Product-Projekt
    - Projekt …
  - Zusätzliche Informationen
    - z.B. source code für Misra rules


Subversion
• Spock Bibliotheken
• Testskripte (*.groovy)
• Testdaten
Kopie des Tests
1. Check-out des Tests
Workspace
• Spock Bibliotheken
• Testskripte (*.groovy)
• Testdaten

• Kompilierte Testskripte (*.class)
• Testergebnisse (JUnit *.xml)
• Testergebnisse (proprietär *.xml)
Jenkins
3. Kompilieren und Ausführen der Testskripte
Skript
Testdaten
*.groovy => *.class
*.class => *.xml
2. Start der Testdurchführung
gradle
(Testdurchführungs-Werkzeug)
4. Upload der Testergebnisse
Polarion ALM
• Testfälle
• Testergebnisse
proprietär *.xml
Kontrollfluss
Datenfluss

Testautomatisierung mit Spock und .groovy

- Spock Framework
  - Ansatz gemäß Behaviour Driven Development
    - given: Vorbedingung
    - expect: erwartetes Ergebnis
    - where: zu prüfende Regel

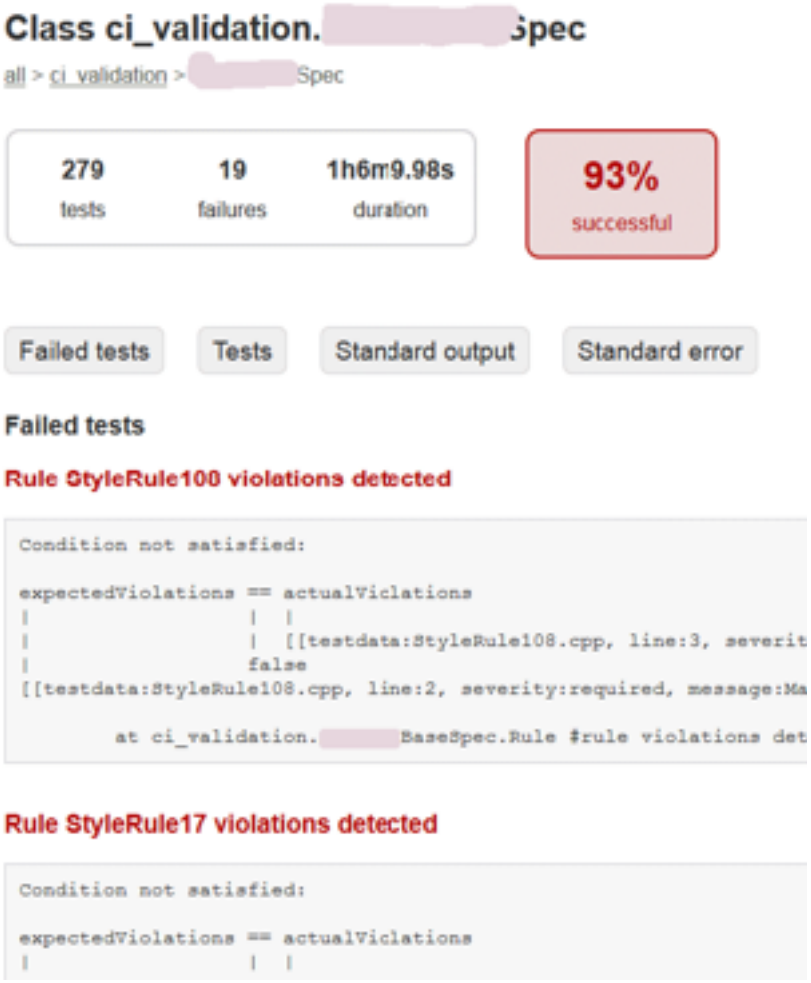
```
/*
*    For each misra rule, check if its expected violations are found in Code Analizer database
*/
@Unroll
def "Misra Rule #rule violations detected"() {

    def expectedViolations = expectedStyleViolations.find {it.id == rule }.violations.sort(a,b  > compareViolation(a,b)}
    def actualViolations    getActualViolationsForRule(rule).sort(a,b  > compareViolation(a,b)}

    expect: "#rule is detected"
        expectedViolations == actualViolations

    where:
        rule << expectedStyleViolations.findAll{it.id}

}
```

Validierung – Vorgehensweise

- Schritt 1: Analyse der Prozesse und Werkzeuge
- Schritt 2: Spezifizieren der Anforderungen
- Schritt 3: Testkonzept
- Schritt 4: Modellbasierte Testfallerstellung
- Schritt 5: Testautomatisierung
- **Schritt 6: Testdurchführung und Dokumentation**
- Schritt 7: Bewertung der Testergebnisse

Testdurchführung

- 41 Testfälle insgesamt
- 31 Manuelle Tests
  - 20 pass, 10 failed, 1 open
  - Testdauer 25 h
  - Minor deviations
- 10 Automatisierte Tests
  - 8 pass, 2 failed
  - Testdauer 1h 10 min
  - Major deviations



Validierung – Vorgehensweise

- Schritt 1: Analyse der Prozesse und Werkzeuge
- Schritt 2: Spezifizieren der Anforderungen
- Schritt 3: Testkonzept
- Schritt 4: Modellbasierte Testfallerstellung
- Schritt 5: Testautomatisierung
- Schritt 6: Testdurchführung und Dokumentation
- Schritt 7: Bewertung der Testergebnisse

Schritt 7: Bewertung der Testergebnisse

- Gravierende Abweichungen
  - Einzelne Verstöße gegen Misra C++:2008 wurden vom Static Code Analyzer nicht richtig erkannt
  - Continuous Integration V1.0 wurde zur Erstellung des Produktcodes nicht freigegeben

- Korrektur vom Toolhersteller
  - Erneute Validierung ergab nur Minor Deviations
  - Continuos Integration V2.0 validiert und freigegeben

Fazit

- Fehler in einer Toolkette können durch Validierung erkannt und vermieden werden
- Testautomatisierung ermöglicht
  - umfassenden Test mit großen Datenmengen
  - effiziente Revalidierung
- Das Konzept der Testautomatisierung konnte in einem Folgeprojekt übernommen werden
- Unsere Use-case- und Workflow-Modelle waren Basis für Verbesserungen in der Toolkette

Vielen Dank für Ihre Aufmerksamkeit!

Noch Fragen?

Antworten auch an unserem Stand!

sepp.med
Qualität sichert Erfolg

sepp.med gmbh
Gewerbering 9
91341 Röttenbach
Tel: +49 9195 931-0
Fax: +49 9195 931-300
www.seppmed.de

Beatrix Forster
Tel: +49 9195 931-0