Qualität in 3 Wochen!?

Von der Qualitätssicherung in der agilen Welt

www.cswob.de

CREATIVE SOFTWARE
WORKBENCH

IT Solution Architect , Entwickler, Coach

> 20 Jahre Projekterfahrung

> 10 Jahre agile Projekte

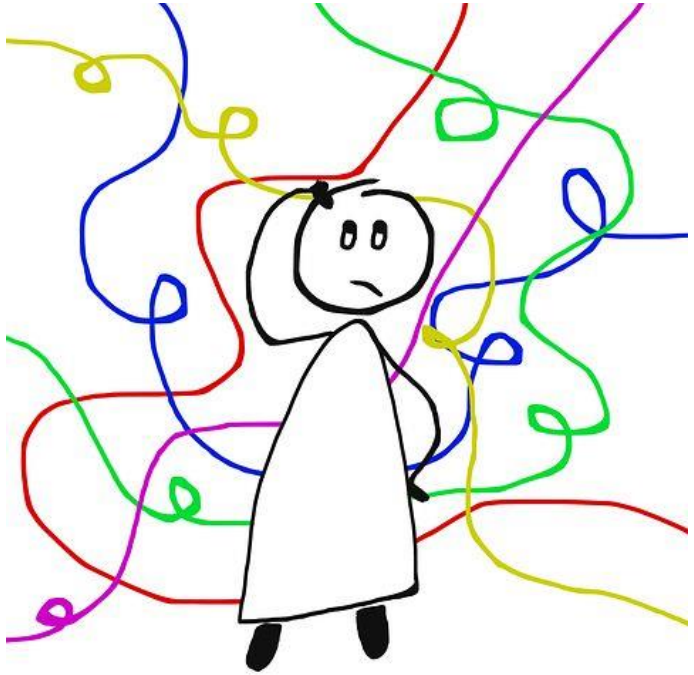
DAX - Startup

Wolfgang Pleus – **PLEUS** Consulting

www.pleus.net

Zentrale Frage

Wie lässt sich das **richtige Produkt** in **hoher Qualität** und **kurzer Zeit** herstellen?



Klassischer Ansatz

Lastenheft
Anforderungen

Pflichtenheft
Konzept
Spezifikation

Produkt

Testpläne

Idee

Analyse → Design → Entwicklung → Test → Integration/ Release

Kunde

Auftrag-
geber

Analysten

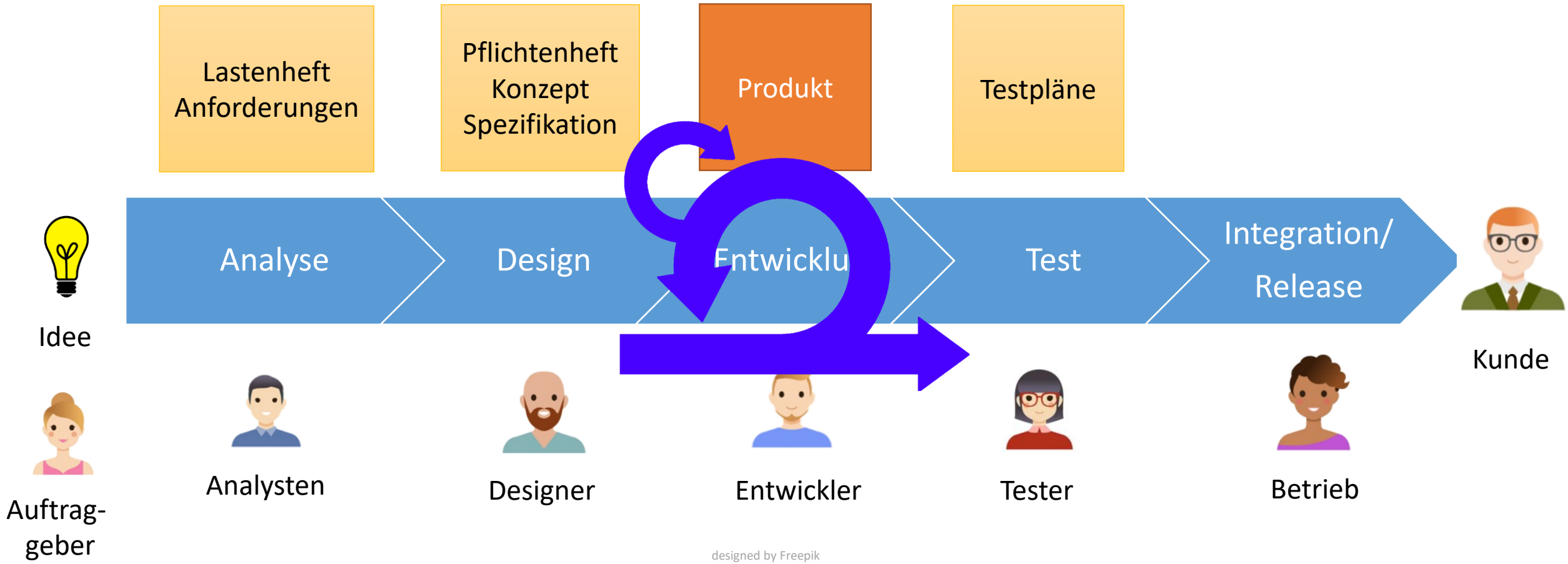
Designer

Entwickler

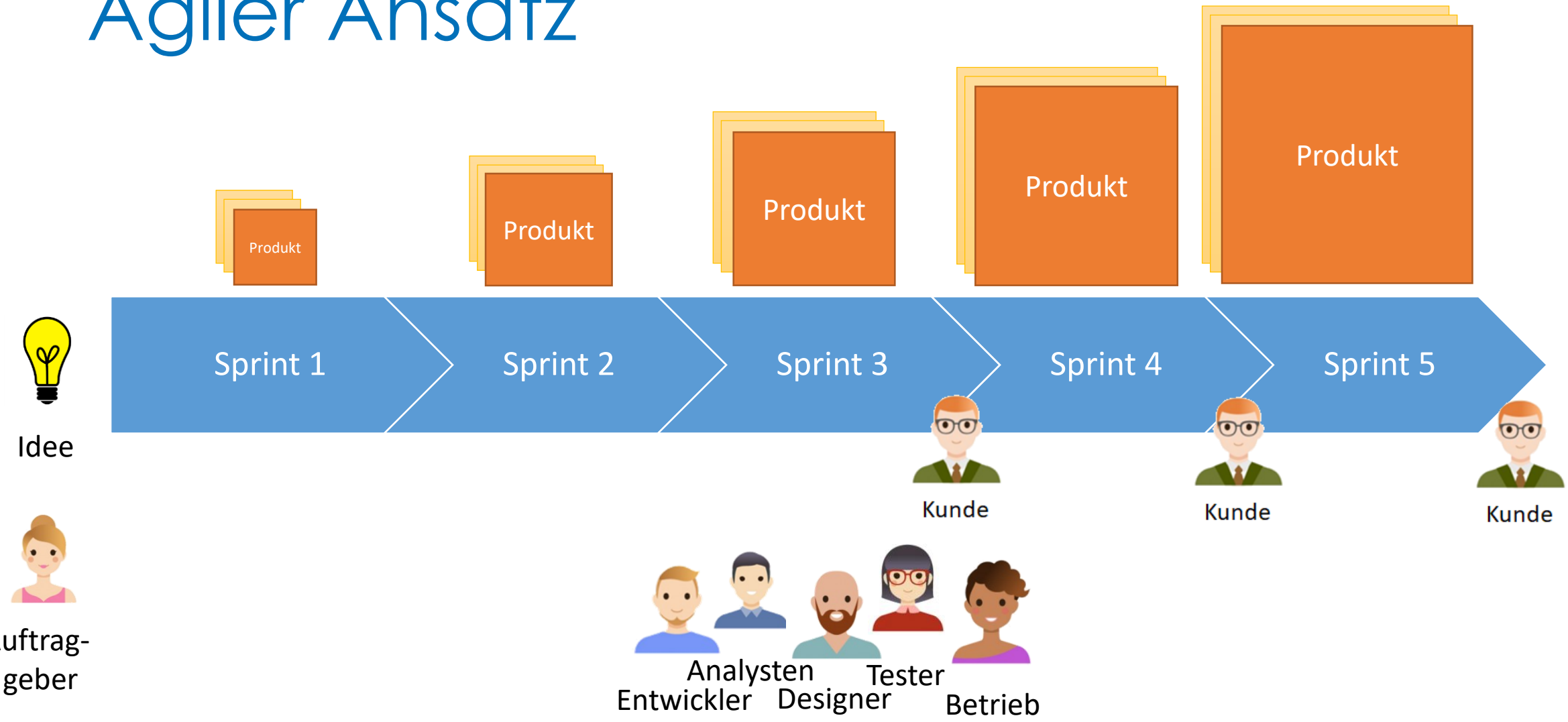
Tester

Betrieb

Klassischagiler Ansatz



Agiler Ansatz



Agiles Mindset

- Das Team will Qualität, Qualität als Teamverantwortung
- Ständiges Streben nach Verbesserung (Kaizen)
- Alle sind Tester (Crossfunctional)
- Feedback durch Reviews und Retros

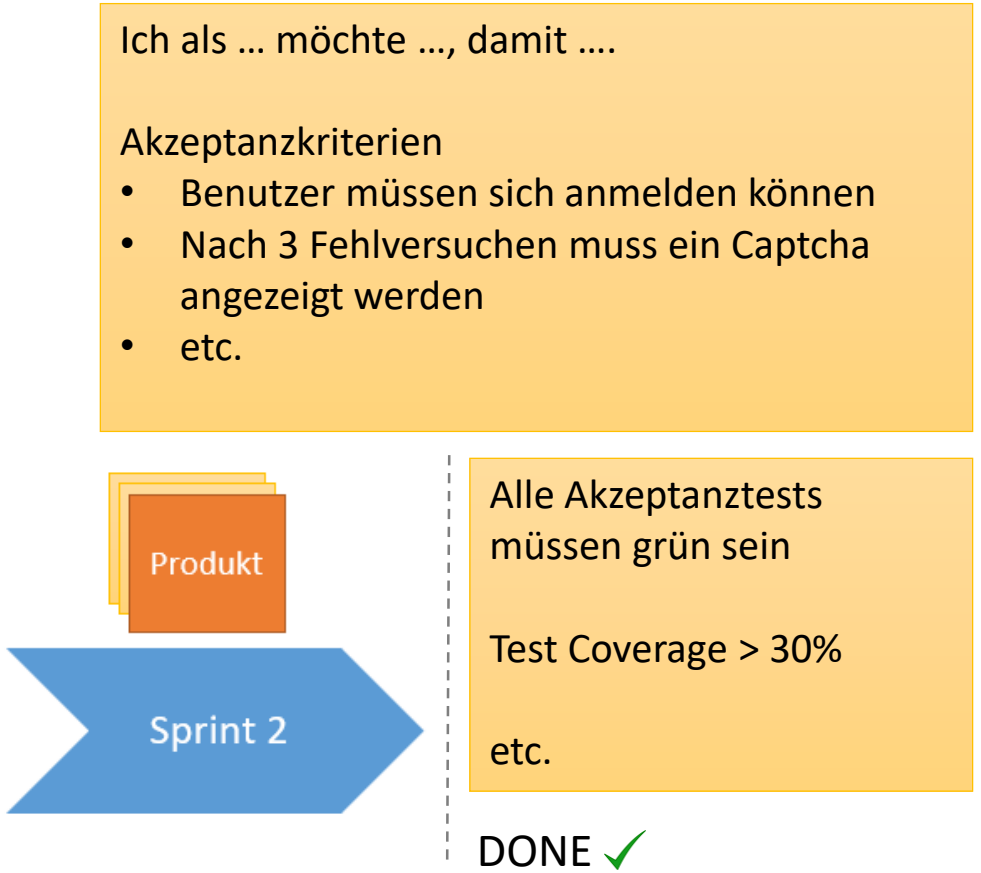
Definierte Qualität durch Scrum

User Stories

- Sind testbar
- Enthalten Akzeptanzkriterien

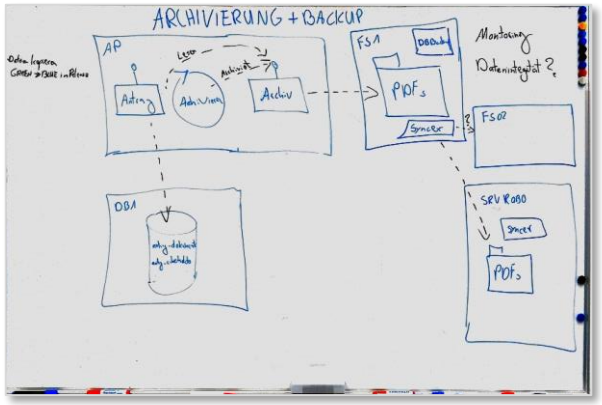
Definition of Done

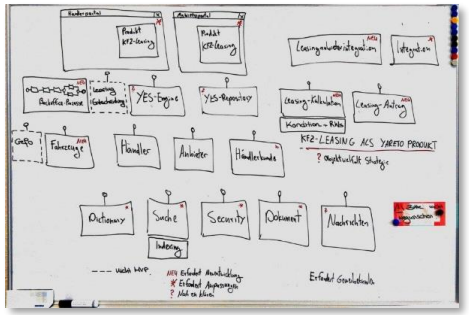
- Regelt, wann eine Story fertig ist
- Verbindlich für alle User Stories

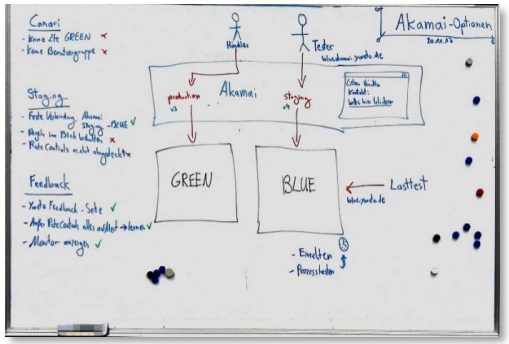


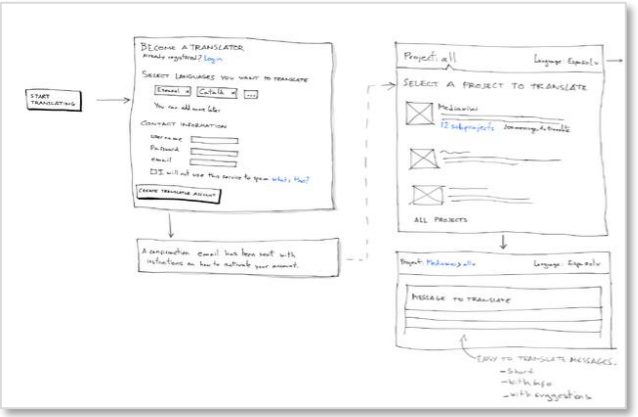
Bauen wir das Richtige?

- Transparenz, Überprüfung, Anpassung – Scrum Guide
- Feedbackschleifen durch Reviews und Retrospektiven
- Ständige Verbesserung (Kaizen)
- Direkte Kommunikation erzeugt Verständnis
- Visualisierung und Prototypen vermeiden Missverständnisse



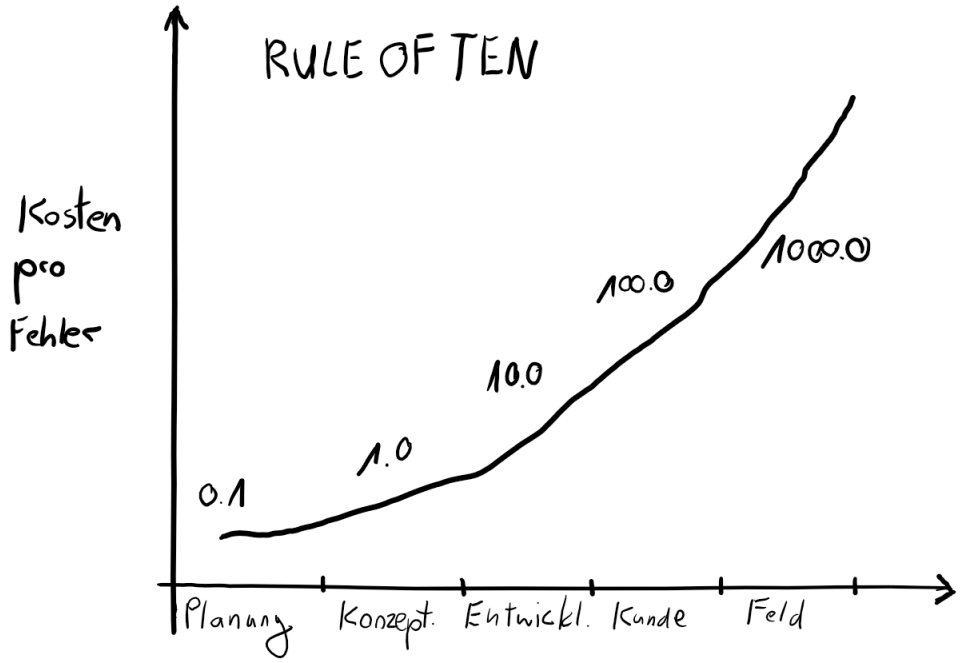






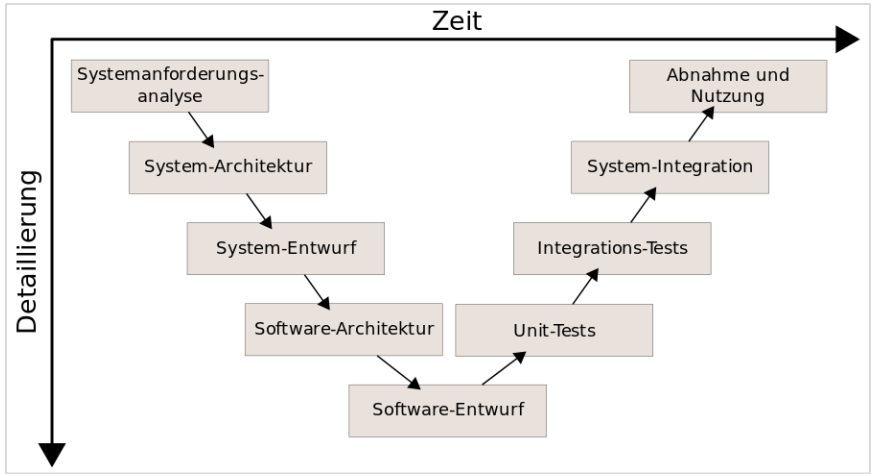
Umgang mit Fehlern

- Konstruktive Fehlerkultur
  - *„Super, dass du den Fehler gefunden hast!“*
- Rule of Ten
- Bugtracker für alle
- Bug vs. Change
  - Fehlfunktion
  - Fehlende Funktion
  - UX-Design
  - Softwaredesign



Teststufen

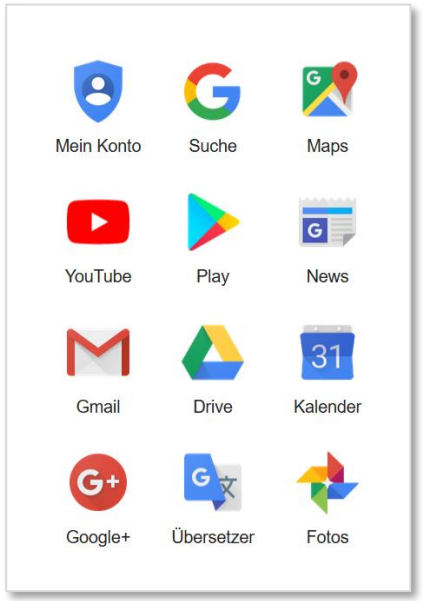
- Komponententests – Einzelne Komponenten
- Integrationstest – Zusammenspiel von Komponenten
- System – Das Gesamtsystem
- Akzeptanztest - Abnahme
- Last & Performancetest - Stabilität



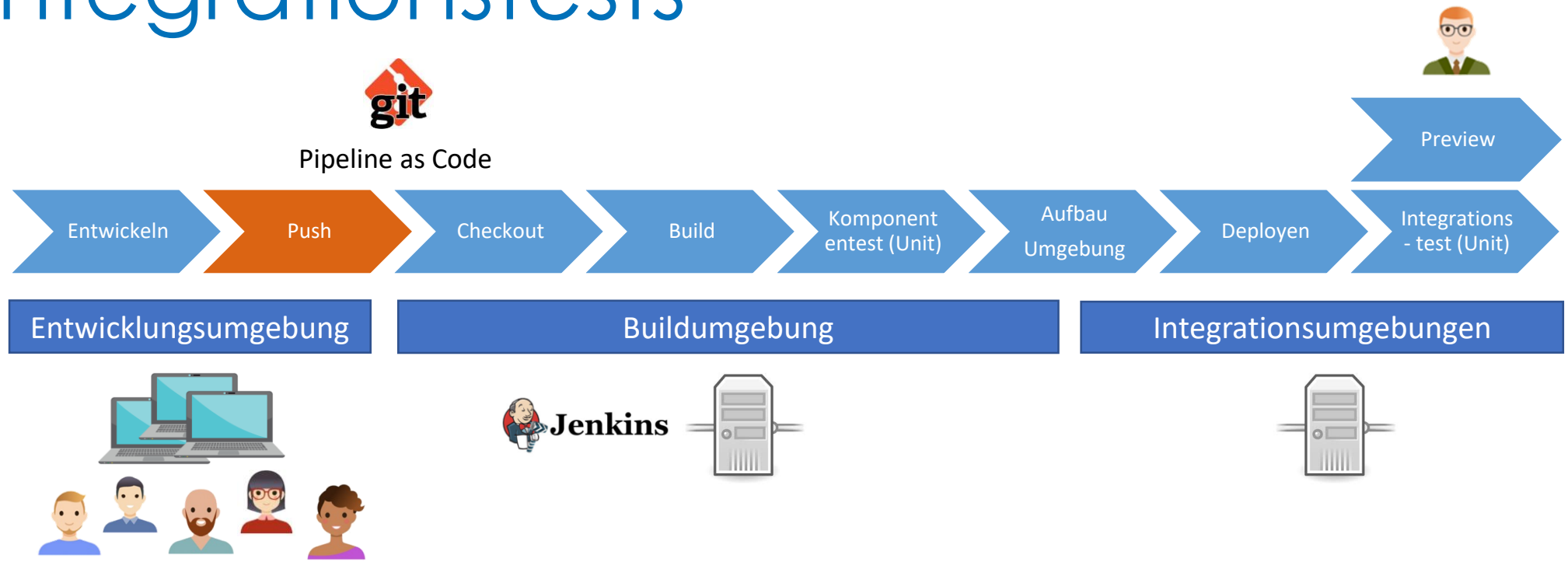
V-Modell

Komponententests

- Modulare Architektur reduziert Nebenwirkungen
- Backend
  - Microservices (kein shared code)
  - Contract First Design
  - Testdriven Development
  - JUnit
- Frontends
  - Angular Services
  - Anwendung pro Produkt (kein shared code)
  - Karma (Runner), Jasmine (BDD)
- Automatisierungsgrad 100%
- Buildserver ist Pflicht



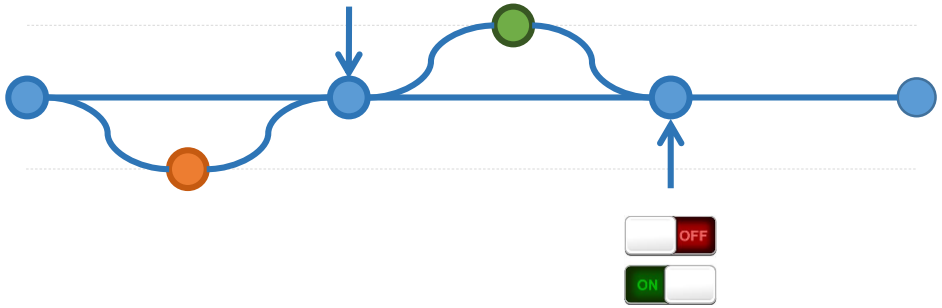
Integrationstests

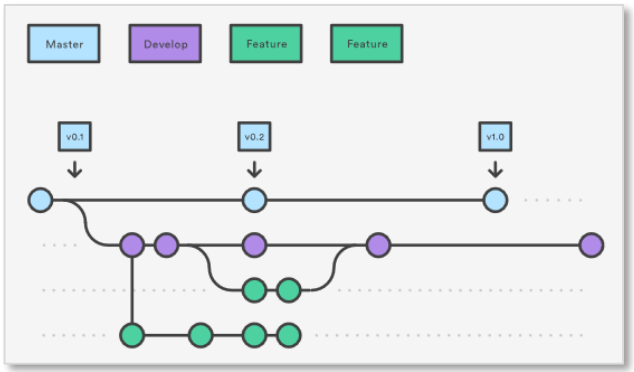


- Produktionsnahe Integrationsumgebung empfohlen
  - Alternativ Mocks und Consumer Driven Contracts
- Bereitstellung mehrmals täglich
- Kurze Feedbackschleifen
- Automatisierungsgrad 100%

Bereitstellen von Änderungen

- Oft pushen und integrieren
- Langlebige Featurebranches führen zu Integrationsverschleppung
- Entwickeln entlang stabiler Stände
- Feature Toggles für unfertige Features



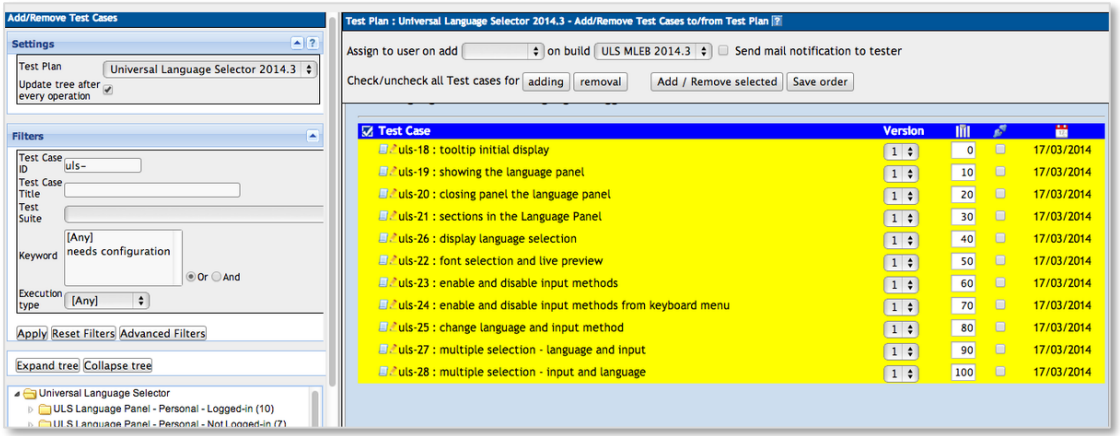


Gitflow ?

Akzeptanztests (strukturiert)

- User Story definiert (Anzahl der) Akzeptanzkriterien
- Strukturierte Tests durch Testmanagementtool
- Fachlich organisierter Testkatalog
- Jeder kann testen, aber …
- … nie das eigene Feature testen

Sprint Board



Testlink

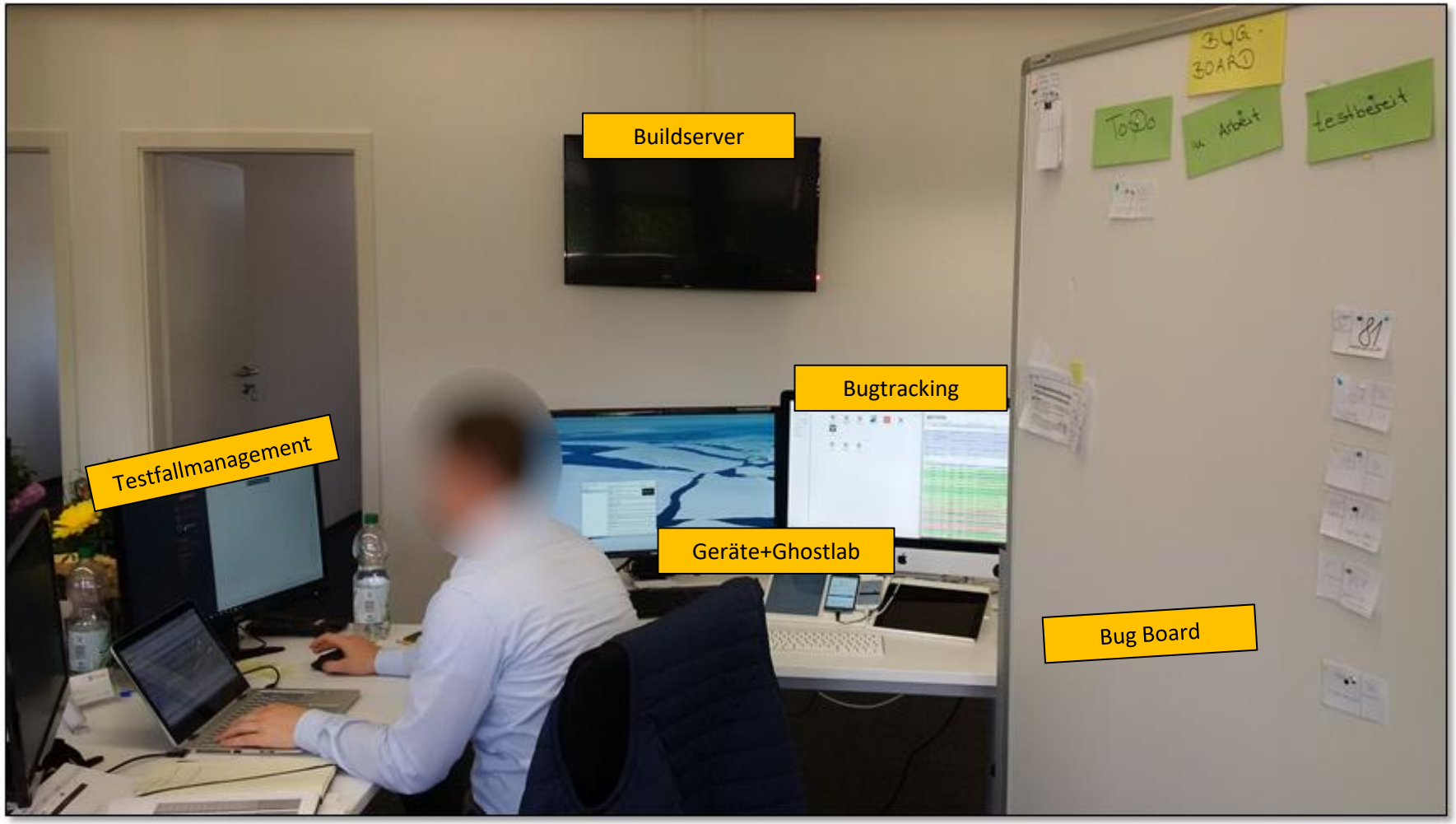
Exploratives Testen (unstrukturiert)

- Testen wie Touristen
  - z.B. Money Tour, Saboteur Tour, Intellectual Tour, Back-alley Tour
- User Story definiert die Routen als Akzeptanzkriterien
- Den Zufall nutzen

Automatisierung von Akzeptanztests

- Automatisierungsarten
  - Capture / Replay (z.B. Selenium, Appium)
  - Multi device (z.B. Ghostlab)
- Achtung: Kosten / Nutzen im Blick behalten
- Menschen sind kreativ und fehlertolerant
- Lean Prinzip No 1 Eliminate Waste
  - Das Maß ist der Anwender nicht die Metrik
  - Richtige Balance finden
- z.B. Smoketests automatisieren

Testlabor



Regressionstest

- 100% in kurzer Zeit schwierig
- Risikobasierte Regression

Codefreeze

GIT Commit Hook

Commit

User Story: 342
Risikoklasse: [0,1,2]
Testempfehlung: …

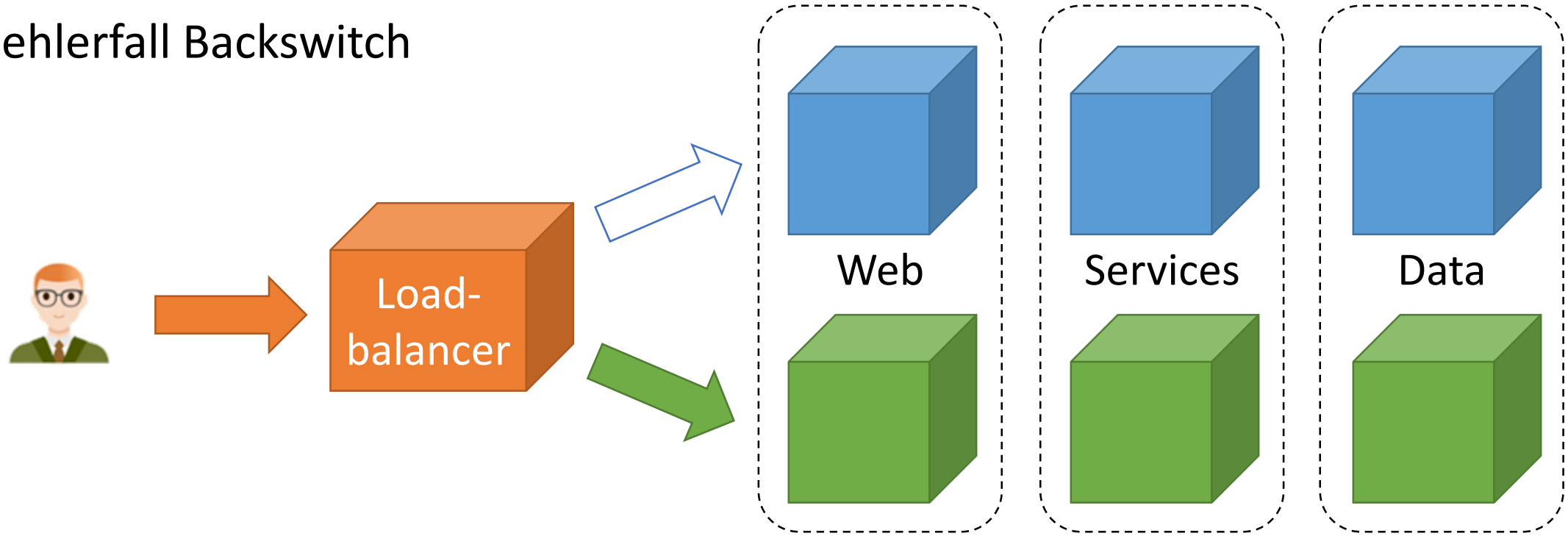
```
git log --pretty=format:"%s - %an (%cI)"
RELEASE30..RELEASE31 | grep -v "Merge" |
grep "RISK:1" | sort> risk1.txt
```

Risikoreport

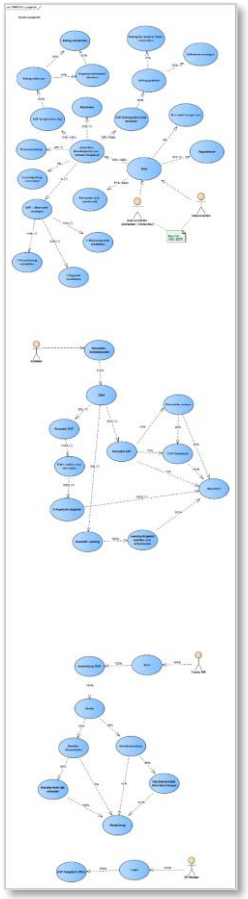
Testplan

Systemtest

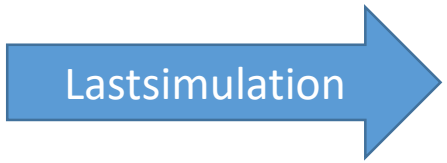
- BLUE / GREEN Deployment, Trash your Servers
- Cloud als Enabler (IaaS oder PaaS)
- Smoketests auf GREEN
- Im Fehlerfall Backswitch

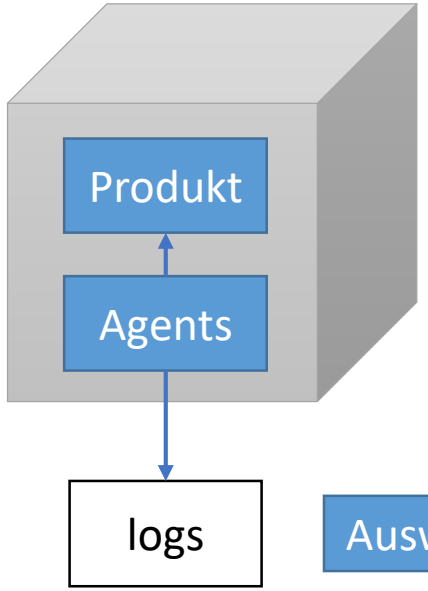


Last & Performance Tests

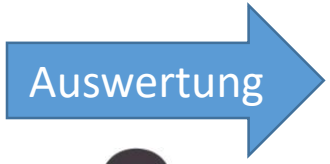


Lastprofil



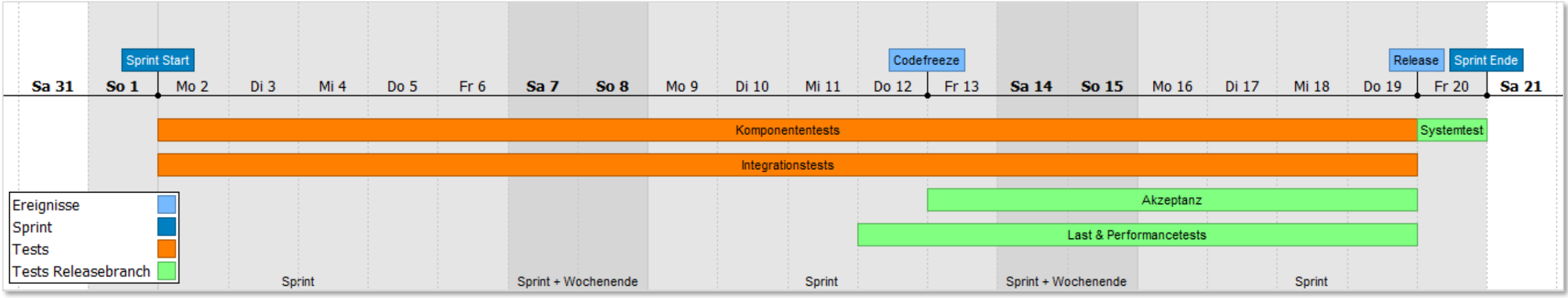


- Simulation
- Realistische Interaktionsszenarien
- Profilschärfung mit Produktivdaten
- Ausführung automatisiert
- Erfordert manuelle Auswertung



QS - Mikroprozess

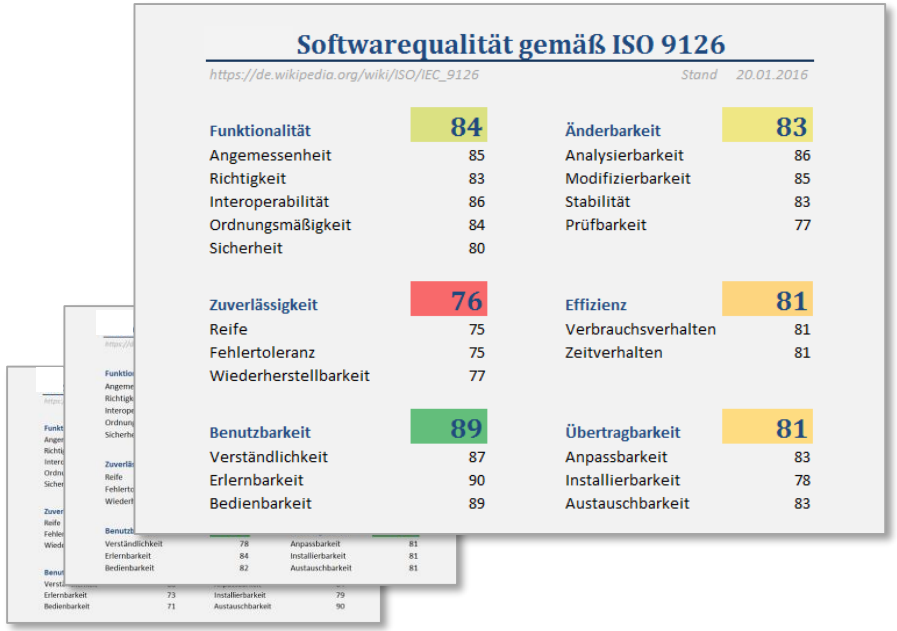
- Scrum / Kanban sagen dazu nichts
- Individuell je Projekt

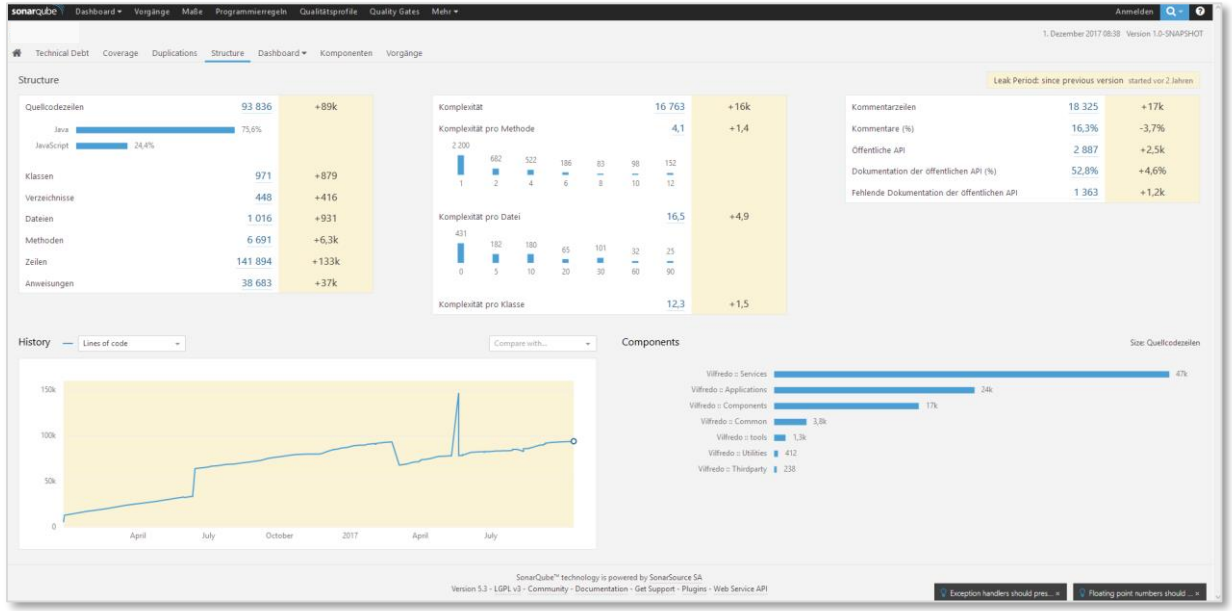


Beispiel 3 Wochen Sprint

Codequalität

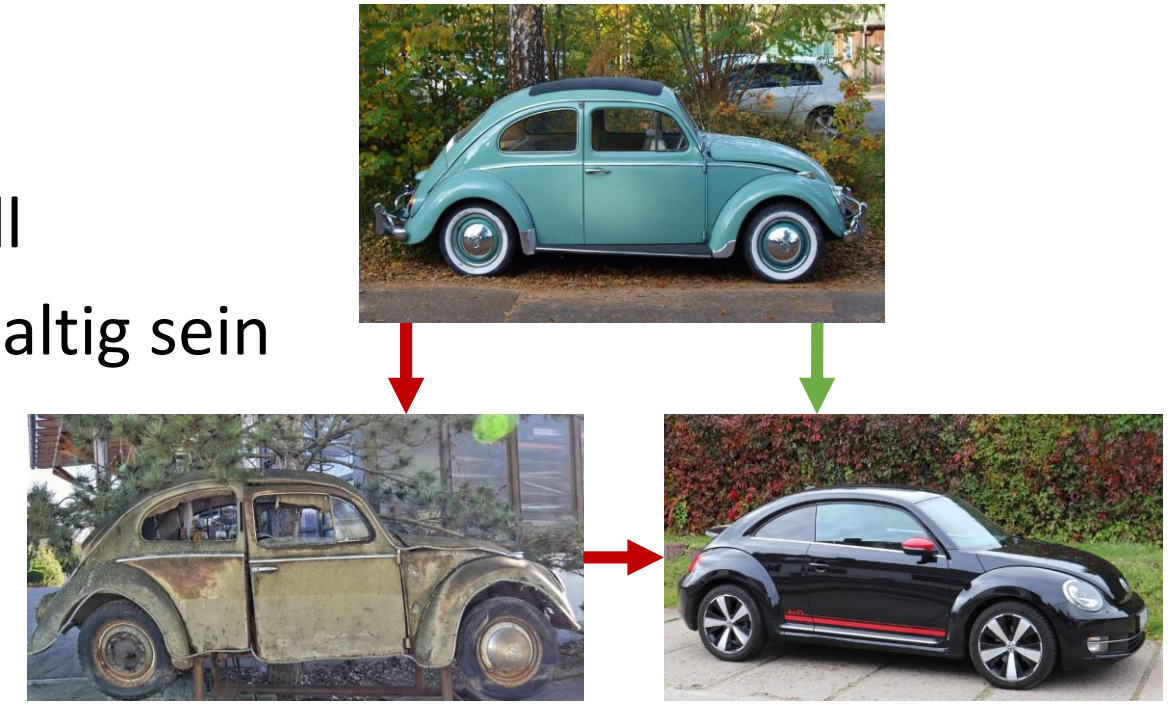
- Automatische Metriken mit SonarQube
- Regelmäßige teambasierte ISO-Bewertung





Refactoring

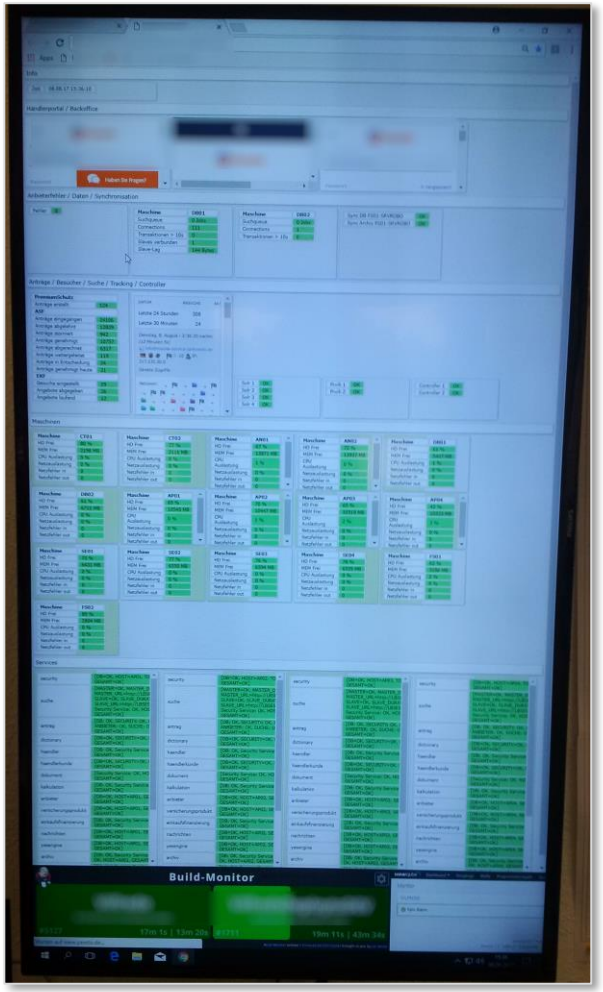
- Refactoring ist Qualitätssicherung
- Technologien entwickeln sich schnell
- Software soll evolvierbar und nachhaltig sein
- Business Value ?
- Ungewohnt



„Ich als Product Owner möchte das die Anwendung modularisiert wird, damit sie langfristig wartbar bleibt und die Investitionen geschützt werden.“

Maximale Transparenz

- Visualisierung nutzen
- Fehler sofort sichtbar machen
- Schnelle Reaktionen ermöglichen



Was, wenn der Druck zunimmt?

„Müssen wir wirklich soviel testen?“

- Kanban Serviceklassen erleichtern die Kommunikation
- Qualitätssicherung abhängig von der Serviceklasse
- Serviceklasse je Sprint oder User Story

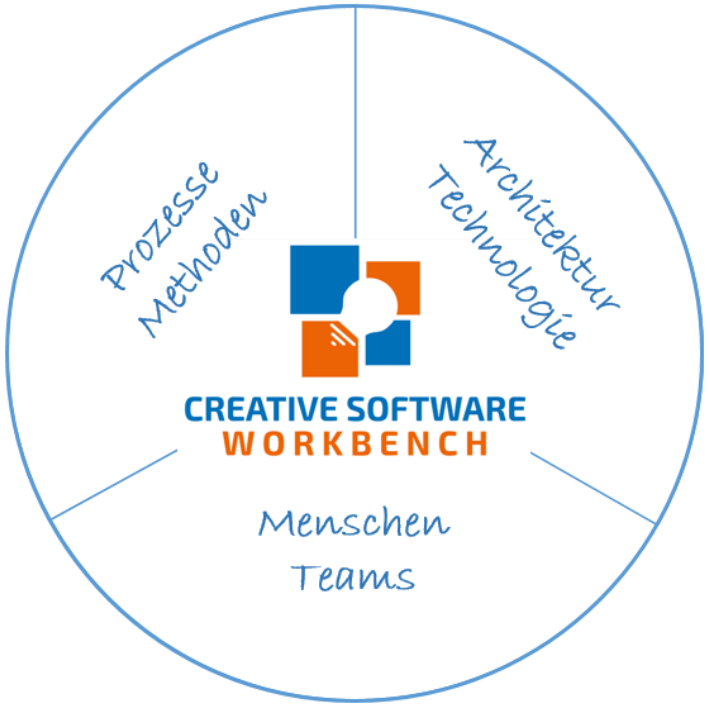
Serviceklasse (Qualität)	QS-Level	Risiken	Auswirkungen
A	High	Sehr niedrige Wahrscheinlichkeit für Nachbesserungen Geringst mögliche Prozesskosten (Rule of ten) Minimale Verzögerung der Endkundenlieferung Maximale Kapazität für die Entwicklung neuer Funktionen (Bindung durch Fehlerbehebung)	Maximale Kosten für Testerstellung, Durchführung und Dokumentation Maximale Lieferzeit für ein Inkrement
B	Medium	Niedrige Wahrscheinlichkeit für Nachbesserungen Niedrig Prozesskosten (Rule of ten) Geringe Verzögerung der Endkundenlieferung Erhöhte Kapazität für die Entwicklung neuer Funktionen (Bindung durch Fehlerbehebung)	Hohe Kosten für Testerstellung, Durchführung und Dokumentation Lange Lieferzeit für ein Inkrement
C	Low	Hohe Wahrscheinlichkeit für Nachbesserungen Hohe Prozesskosten (Rule of ten) Lange Verzögerung der Endkundenlieferung Verminderte Kapazität für die Entwicklung neuer Funktionen (Bindung durch Fehlerbehebung)	Geringe Kosten für Testerstellung, Durchführung und Dokumentation Kurze Lieferzeit für ein Inkrement
D	None	Sehr hohe Wahrscheinlichkeit für Nachbesserungen Sehr hohe Prozesskosten (Rule of ten) Sehr lange Verzögerung der Endkundenlieferung Stark verminderte Kapazität für die Entwicklung neuer Funktionen (Bindung durch Fehlerbehebung)	Keine Kosten für Testerstellung, Durchführung und Dokumentation Minimale Lieferzeit für ein Inkrement

Fazit

Erfolg ist interdisziplinär

- Agile Teams testen auf allen Stufen
- Qualitätsbewusstsein ist in den Teams verankert
- Modulare Softwarearchitektur reduziert das Risiko
- Technische Voraussetzungen sind wichtig (CI-Pipeline, Cloud)
- Hoher Automatisierungsgrad ist erforderlich

= Qualität in Rekordzeit



Fragen ?

Vielen Dank

www.pleus.net

wolfgang.pleus@pleus.net

Download Folien www.pleus.net/blog

Creative Software Workbench www.cswob.de