



# Build-Management mit marktüblichen Tools



Björn Feustel  
Steffen Schluff

Orientation in Objects GmbH

Weinheimer Str. 68  
68309 Mannheim

[www.oio.de](http://www.oio.de)  
[info@oio.de](mailto:info@oio.de)

Version: 2.0

## Gliederung

- Einleitung
- Issue-Tracker und IDE
- SCM und CI-Server
- Release Management
- Continuous Delivery
- Zusammenfassung und Ausblick

- Einleitung
- Issue-Tracker und IDE
- SCM und CI-Server
- Release Management
- Continuous Delivery
- Zusammenfassung und Ausblick

- Ein guter Entwicklungsprozess ist einfach, flexibel und praxisorientiert, d.h.:
  - Reibungslose Arbeit im Team
  - Schnelle Entwicklungszyklen
  - Inhärenter Qualitätsanspruch
  - Gute Planung und Steuerung
- Eine Build-Infrastruktur muss das unterstützen, z. B. durch:
  - Bereitstellen gemeinsamer, integrierter Entwicklungswerkzeuge
  - Automatisieren von wiederkehrenden Prozessen
  - Vorgeben und Prüfen von Konventionen, z.B. Metriken
  - Vereinfachen der Projektkontrolle
- Und wen betrifft es?

- Rollenbegriffe sind abhängig von Projektgröße / -struktur, Organisation
  - Developer, Architect
  - Tester / QA
  - Release Engineer & Manager
  - Product & Project Manager
  - Product Owner
  - Scrum Master
- Im Kontext der Präsentation
  - Team
    - Entwickler, Spezialisten
    - Ändert den Sourcecode
    - Erstellt Tests/sichert die Qualität
    - Kennt (und verbessert) den Build-Prozess
  - Controller
    - Scrum Master
    - Pflegt und optimiert das Projekt
    - Überwacht und steuert den Projektfortschritt
  - Stakeholder
    - Product Owner und Interessenten
    - Bestimmen die Ziele und Prioritäten



## Gliederung

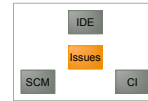
- Einleitung
- **Issue-Tracker und IDE**
- SCM und CI-Server
- Release Management
- Continuous Delivery
- Zusammenfassung und Ausblick

## Issue-Tracker

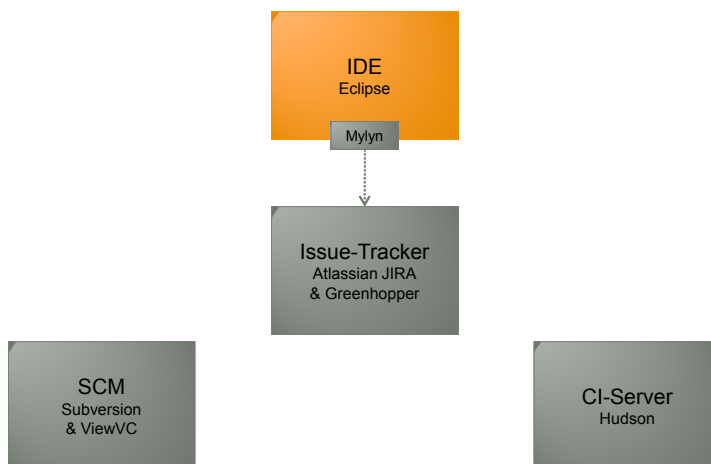


## Issue-Tracker – Synopsis

- Aufgabe
  - Erfasst **alle** Änderungen und Aktivitäten
    - Bug Tracking vs. Issue Management vs. SCRUM
  - Ermöglicht die Projektplanung
    - Features, Versionen, Fix-Termine, Kapazität
  - Gibt verbindliche Auskunft über den Projekt(zu)stand
    - Nächste Aufgaben, Versionsfortschritt, Arbeitsauslastung
  - Entkopplung der Entwicklung von ablenkenden Prozessen
    - Requirements Management, Change Management
- Rollen und Verwendung
  - Alle: Ermitteln und Pflege des Projektstatus
  - Alle: Projektplanung
  - Team: Bereitstellen des Arbeitskontexts (Mylyn / Eclipse)
- Produkte
  - **Atlassian JIRA**, Bugzilla, Roundup, FogBugz, Trac

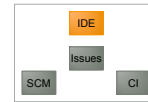


## IDE – Integrated Development Environment



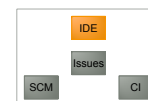
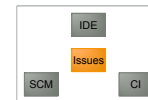
## IDE – Synopsis

- Aufgabe
  - Zentrales Arbeitswerkzeug der Entwickler
  - Maximierung der Entwicklerproduktivität
- Rollen und Verwendung
  - Team: Entwicklers Habitat
  - Team: Allgemeiner Zugriff auf SCM (Subversive)
  - Team: Kontextbezogener Zugriff auf Issue-Tracker (Mylyn)
- Produkte
  - **Eclipse**, NetBeans, IntelliJ IDEA

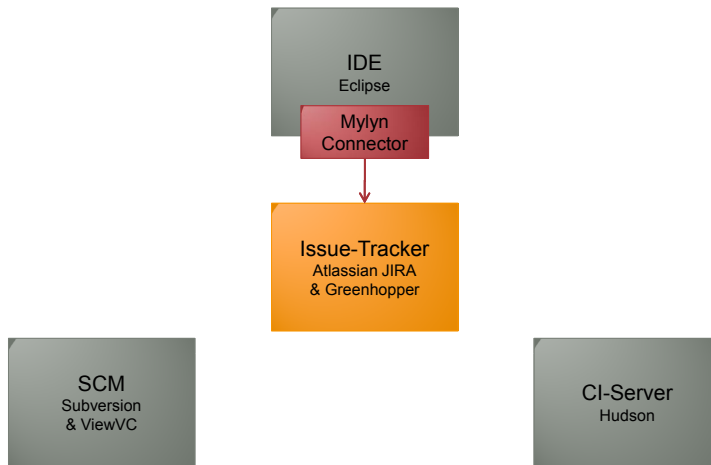


## Demonstration

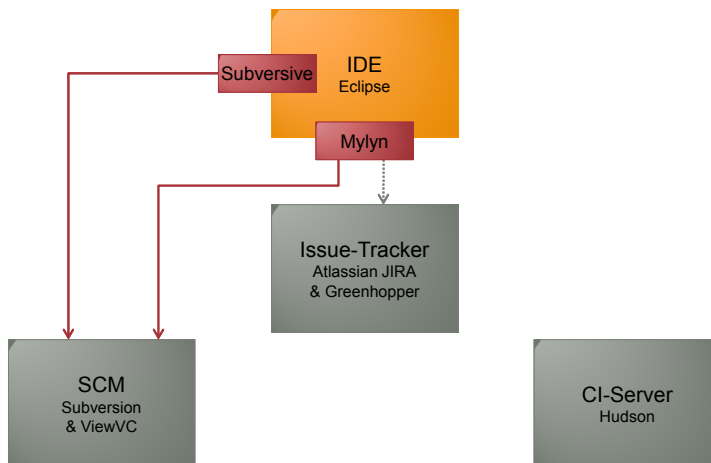
- Issue Tracker:
  - Organisation der Issues / Release-Notes
  - Anbindung an IDE per Mylyn (Atlassian IDE Connector)
- IDE:
  - SVN Integration
  - Changesets verwalten mit Mylyn



## Issue-Tracker – Das Build-System wächst...



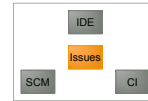
## IDE – Das Build-System wächst...



## Issue-Tracker – Best Practices & Konventionen

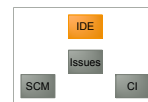
- Nachvollziehbarkeit / Reproduzierbarkeit
  - Arbeiten immer im Kontext eines Issues
  - Issues nach Versionen erfassen
- Aktualität
  - Issues immer auf Personen zuordnen
  - Änderungen unmittelbar dokumentieren
  - Medienbruch für den Entwickler vermeiden (z. B. mit Mylyn)
  - Organisation der Issues optimieren (z.B. mit Greenhopper)
- Als Single Point of Truth etablieren
  - Berührungspunkte bei allen Beteiligten abbauen
- *Aber: Individuals and interactions over processes and tools*

(Agile Manifesto)



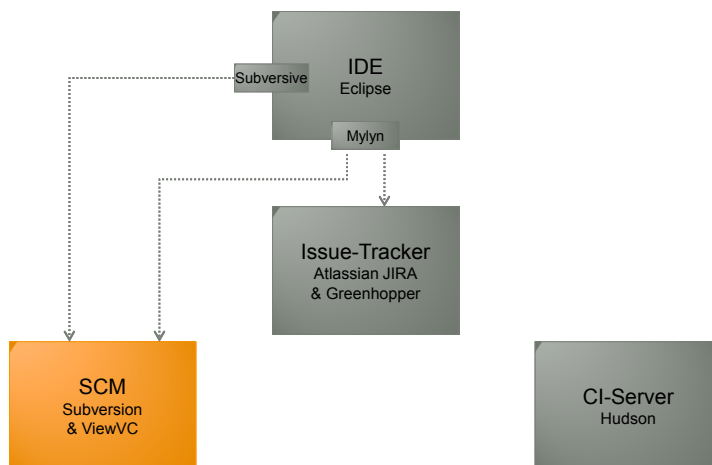
## IDE – Best Practices & Konventionen

- Projektweite Vorgaben für alle
  - Die gleiche IDE (Produkt, Plugins)
  - Das gleiche Vorgehen (Handling der Issues)
  - Die gleichen Einstellungen (Code Formatter, Code Syntax, ...)
- Vermeiden von Tool-Brüchen
  - Integrierter SVN Client
  - Integriertes Deployment in lokale Testserver (pre-tested Commit)
- Optimieren des Arbeitsflusses
  - Task/Context Management (z. B. Mylyn / Eclipse oder Cube'n / NetBeans)
  - Automatische Prozesse (z. B. „Save Actions“ / Eclipse)
- *Aber: Build-Prozess muss außerhalb der IDE funktionieren*



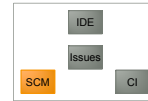
- Einleitung
- Issue-Tracker und IDE
- **SCM und CI-Server**
- Release Management
- Continuous Delivery
- Zusammenfassung und Ausblick

## SCM – Software Configuration Management

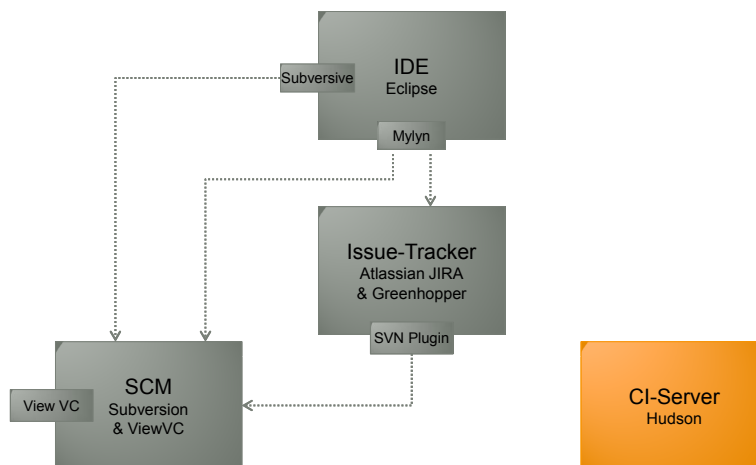


## SCM – Synopsis

- Aufgabe
  - Verwaltung sämtlicher Quellartefakte
    - Sourcen, Konfiguration, Dokumentation
  - Zusammenarbeit im Team ermöglichen
  - Versionsverwaltung / Baselining
- Rollen und Verwendung
  - Alle: Zugriff auf alle Artefakte und Dokumentation (ViewVC / SVN)
  - Alle: Nachvollziehen von Änderungen (JIRA Subversion Plugin)
  - Team: Grundlage für parallele Entwicklung (Branch/Merge)
- Produkte
  - **SVN**, Git, Perforce, Mercurial, CVS

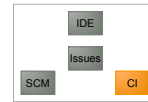


## CI-Server – Continuous Integration Server



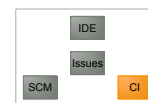
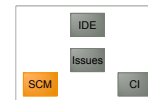
## CI-Server – Synopsis

- Aufgabe
  - Qualitätssicherung
    - Automatische Integration
    - Ausführen von Tests, Erstellen von Reports
    - Qualitätshistorie und Trends aufzeigen
  - Gewährleisten der Reproduzierbarkeit
    - Ausführen von Referenz-Builds
    - Automatisches Markieren im SCM
  - Automatisches Erstellen und Ausliefern des Produktes
    - Instanzieren der Deployment Pipeline
- Rollen und Verwendung
  - Team: Integrations- und Qualitätsfeedback
- Produkte
  - **Hudson**, Jenkins, Bamboo, TeamCity, Go, CruiseControl

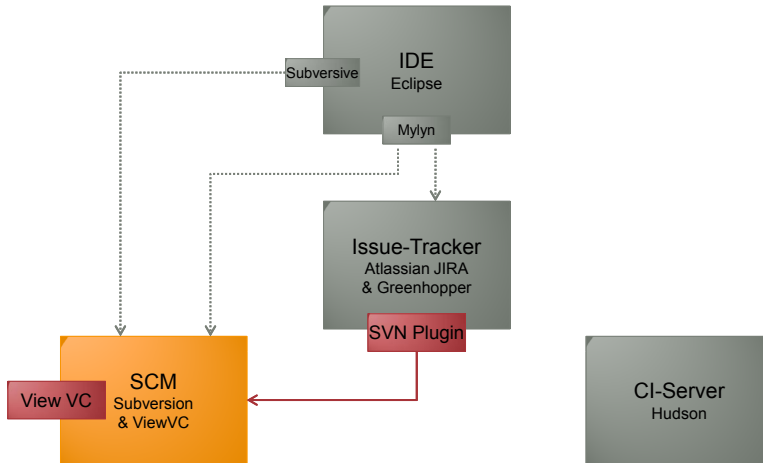


## Demonstration

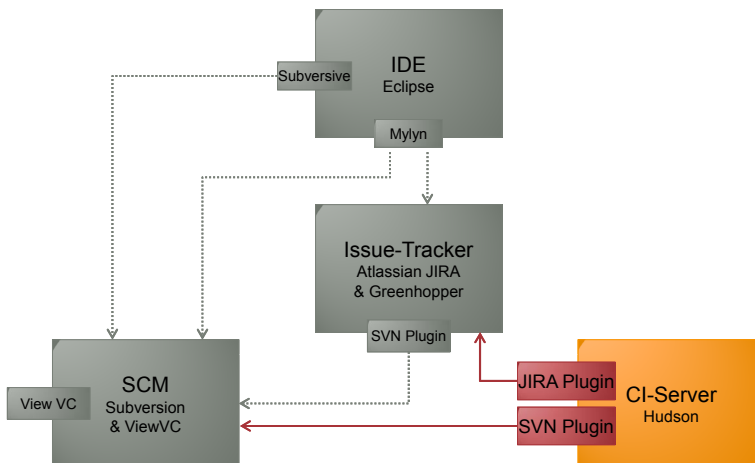
- SCM:
  - Nachvollziehbarkeit in JIRA
  - Repository Zugriff mit ViewVC
- CI-Server:
  - SVN Anbindung
  - JIRA Plugin für Hudson



## SCM – Das Build-System wächst...

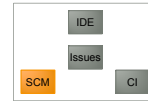


## CI-Server – Das Build-System wächst...



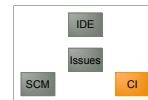
## SCM – Best Practices & Konventionen

- Optimieren des Projektflusses
  - Tooling beherrschen (Merging)
  - Häufige Check-ins & Merges (aber: Head stabil halten)
  - Check-in immer auf ein Issue bezogen (z.B. SVN-Hook)
  - Atomare Check-ins mit aussagekräftigen Kommentaren
- Mechanismen zur Projektverfolgung nutzen
  - Zugriff für alle ermöglichen: ViewVC, Tortoise
  - Benachrichtigungen (z.B. automatischer Mailversand oder RSS-Feeds)
- Konsistenz, Vollständigkeit und Ordnung wahren
  - Jede Version der Software ist aus dem SCM reproduzierbar
  - Zentrales Repository bei DVCS verwenden
  - Alte Daten löschen
- Aber: Bei SCM gibt es kein „aber“!

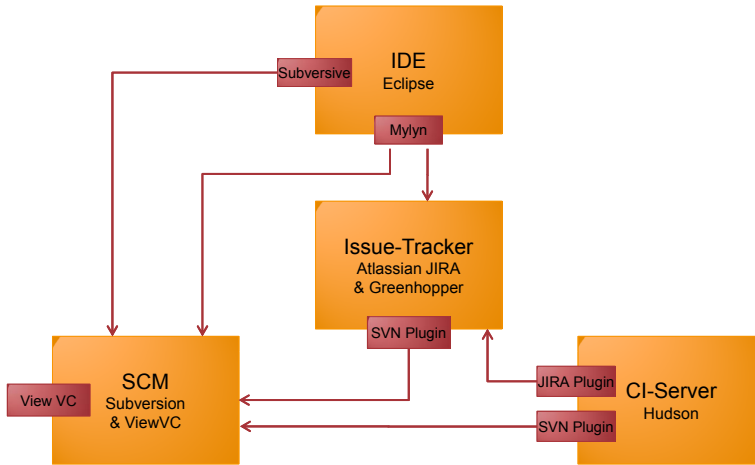


## CI-Server – Best Practices & Konventionen

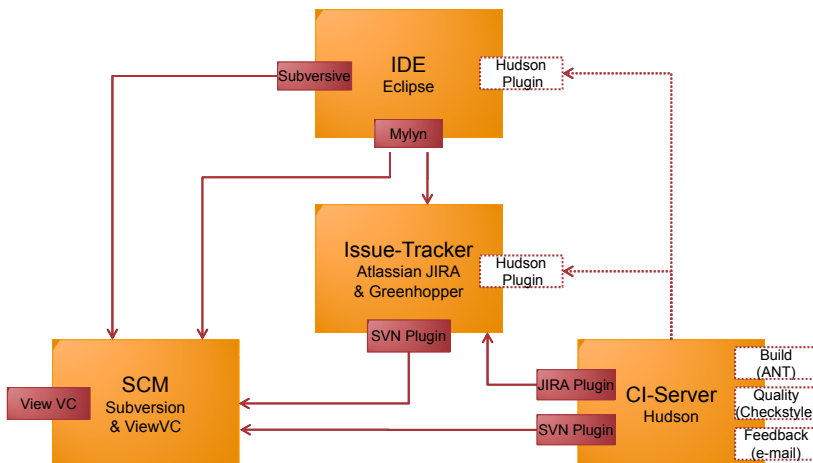
- Optimieren des Projektflusses
  - Abwarten des CI-Laufs, ggf. sofort claimen/reparieren
  - Tests lokal ausführen vor dem Check-in (Pre-Commit Test)
  - Don't commit on a broken build
  - Quantität & Qualität der Tests muss stimmen
- Optimieren des Arbeitsflusses
  - Testlaufzeiten niedrig halten (Test-Optimierung, Staffelung, Parallelisierung)
  - Status für alle sichtbar machen
- Feedback nutzen
  - Benachrichtigungen bei Fehlern (Mail, IM, IDE-Plugin)
  - Code Qualität (Metriken) auswerten, Trends beobachten
- Aber: CI-Server ist nur so gut wie man ihn gut sein lässt



## Et voilà – die Build-Infrastruktur



## Und hier der Nachschlag

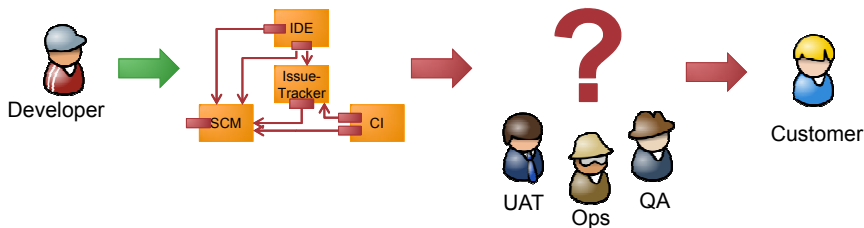




- Einleitung
- Issue-Tracker und IDE
- SCM und CI-Server
- **Release Management**
- Continuous Delivery
- Zusammenfassung und Ausblick

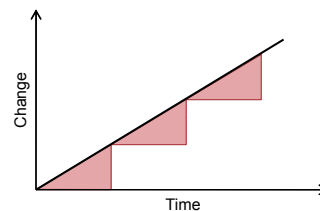
## Continuous Integration – und wie weiter?

- Die Build-Infrastruktur steht, ...
- ...der CI Server brummt und zeigt grün...
- ...und der Kunde wartet.
  
- Erfolgreiches Commit != Auslieferung in Produktion
- CI ist fokussiert auf Entwicklung
- ...nicht manuelles Testen oder Produktivsetzung



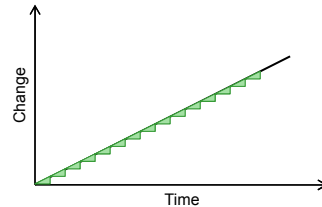
## Kraftakt Release?

- Klassisches Release: Big Bang
  - Manuell
  - Aufwändig
  - Fehleranfällig
  - Wenig planbar
  
- Die Folgen
  - Es wird zu selten released
  - Releases sind personengebunden
  - Releases bedeuten Frust
  - Releases = Veränderung = Risiko = Furcht
  
- Alternativen?



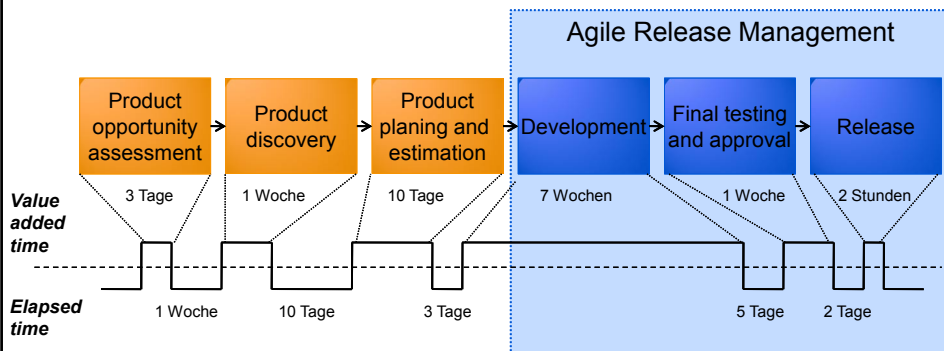
## Agiles Release Management als Lösung

- “If it hurts, do it more often!”
- Culture, Automation, Measurement and Sharing (CAMS)
  - Cross-functional Teams: enge Zusammenarbeit Entwicklung & Betrieb
  - Hoher Grad an Automatisierung
  - Monitoring des Status
  - Informationsverteilung und Offenheit
- Agiles Release Management reduziert das Projektrisiko



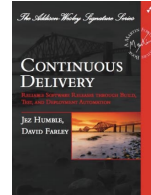
## Agiles Release Management als Teil des Value Streams

- Value Stream Map: Wie lange braucht eine Idee bis zu ihrer Produktivsetzung? „Concept to Cash“



(Nach „Continuous Delivery“/J. Humble, D. Farley)

- Continuous Delivery
- “Continuous Delivery is not Continuous Integration. Continuous Delivery is being in the position to ship your product whenever you want, day or night.” (Neal Ford)
- Wurzeln
  - “Deployment Pipeline” (2004/2005)
  - “Continuous Deployment” (2009)
- Gleichnamiges Buch von Jez Humble & David Farley



- Einleitung
- Issue-Tracker und IDE
- SCM und CI-Server
- Release Management
- **Continuous Delivery**
- Zusammenfassung und Ausblick

## Continuous Delivery – Kerngedanken

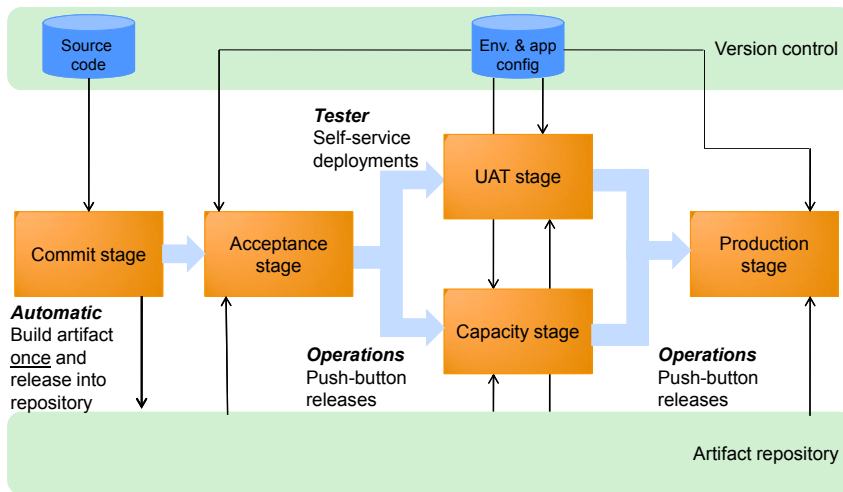
- Create a Repeatable, Reliable Process for Releasing Software
- If It Hurts, Do It More Frequently, and Bring the Pain Forward
- Everybody Is Responsible for the Delivery Process
- Keep Everything in Version Control
- Automate Almost Everything
- Done Means Released

(Nach „Continuous Delivery“/J. Humble, D. Farley)

## Continuous Delivery – Zentrale Konzepte

- Die **Deployment Pipeline**
  - Technisch-konzeptuelle Basis des Release Prozesses
  - Macht Status der Produktentwicklung sichtbar
  - Liefert Feedback zu jeder Änderung
- Die Pipeline besteht aus einer Folge von **Stages**
  - Commit Stage als zentrales Eingangs-Gate
  - Typische Stages: UAT, Performance Tests, Production Deployment
  - Stages verbunden durch Trigger (automatisch oder manuell)
- **Jobs** sind die Bausteine der Stages
  - „Unit of Work“
  - Bestehen aus Tasks wie Build, Deploy, Copy, Test, ...

## Continuous Delivery – Deployment Pipeline



(Nach „Continuous Delivery“/J. Humble, D. Farley)

## Continuous Delivery – Prinzipien (1)

- Fortlaufende Optimierung
  - In Verantwortung von Developern und Operations
- Artefakte
  - Werden **einmal** gebaut und im Artefakt Repository verwaltet...
  - und allen Stages zur Verfügung gestellt
  - Ziel ist identisches Deployment in allen Umgebungen
  - Umgebungsspezifika durch eigene Konfigurationen
- Configuration-Management
  - Basis für einmalig erstellte Artefakte
  - Umfasst Software und Infrastruktur
  - Konfigurationen werden versioniert

## Continuous Delivery – Prinzipien (2)

- Automatisierung
  - So umfangreich wie möglich
  - Prägung durch Developer und Operations
  - Umfasst auch alle Aspekte der Infrastruktur (inklusive OS)
- Tests
  - Basis für Automatisierung und Pipeline Processing
  - Geben Sicherheit für erfolgreiche Änderungen
- Monitoring
  - Basis für fortlaufende Optimierung
  - Ermöglicht Feedback für Operations (vgl. Code-Metriken für Developer)

## Continuous Delivery – Tooling

- Continuous Delivery Implementierungen sind individuell
  - Art und Komplexität des Produktes (Web-App vs. Standalone)
  - Komplexität des Release Prozesses
  - Technischer Rahmen (Programmiersprache, OS, Browser)
- Es gibt nicht *das* Continuous Delivery Tool
- Bestehendes Java-Tooling als Basis
  - CI Server: Go, Hudson, Bamboo, ...
  - Artefakt Repos: Build-in CI-Server, Maven, ...
  - Config Management: ESCAPE, Puppet, Chef, ...
  - Monitoring: Nagios, OpenNMS, CI-Server Reports, Logging, ...

- Tool Beispiele „Continuous Delivery“
  - Go
  - Bamboo



- Mit lauffähigem Pipeline Skelett starten und...
- ...inkrementell fortwährend automatisieren
  - Größte Hürden zuerst (nach Fehleranfälligkeit oder Aufwand)
- „Feature Branches considered harmful“
  - Besser „Branch by Abstraction“ oder „Feature Toggles“
- Environments so identisch wie möglich halten
  - Dadurch wenig divergierende Konfiguration
- Bewährte Release und Deployment Muster verwenden
  - Blue/Green Deployment
  - Canary Releasing

- Einleitung
- Issue-Tracker und IDE
- SCM und CI-Server
- Release Management
- Continuous Delivery
- **Zusammenfassung und Ausblick**

- Projektkontrolle basierend auf Issues ermöglicht...
  - ...eine hohe Informationsdichte und –vernetzung in allen Tools
  - Projektstatus ist nicht nur für den Controller wichtig (Wallboards)
- Softwareentwicklung endet beim Kunden...
  - ...und nicht auf dem CI-Server
  - „Done means Released“
- Automatisierung der Kernprozesse ist notwendig...
  - ...und muss alle Bereiche adressieren
  - Interdisziplinäres Teamwork ist unabdingbar
- Entsprechende Tools existieren...
  - ...in unterschiedlichem Reifegrad
  - Es kann kein „One Size Fits All“ geben

## If you remember one thing

“Computers are designed to do simple repetitive tasks.  
The second you have humans doing repetitive tasks,  
all the computers get together late at night and laugh  
at you”

*Neal Ford*

## Mehr von OIO zum Thema...

- Atlassian Jira - Issue Tracker
  - <http://www.oio.de/software-factory/tools-agile-software-entwicklung/jira/atlassian-jira-agil-preise-euro.htm>
- Schulung: Jira - Fachliche Administration
  - <http://www.oio.de/seminar/methodik-prozess-management-soft-skills/seminar-training-atlassian-jira-schulung.htm>
- Schulung: Versionsverwaltung mit Subversion
  - <http://www.oio.de/subversion-svn-schulung.htm>
- Schulung: Das Buildtool Apache Maven
  - <http://www.oio.de/maven-schulung.htm>

## Mehr von OIO zum Thema...

- Artikel: Java Concurrency Utilities
  - <http://www.oio.de/public/java/concurrency/concurrency-utils.htm>
- Artikel: Mylin (ehem. Mylar)
  - <http://www.oio.de/eclipse-mylar-artikel.htm>
- Beratung zu: Open Source Tools
  - <http://www.oio.de/beratung-consulting/open-source-software/tools/>

## Ihr Sprecher

Steffen Schluff

*Trainer, Berater, Entwickler*



**Schwerpunkte**  
*Open Source Tooling  
Build Management  
Refactoring*

Björn Feustel

*Trainer, Berater, Entwickler*



**Schwerpunkte**  
*Build- und Konfigurationsmanagement  
Systemarchitekturen  
Requirements-Engineering*



Orientation in Objects GmbH

Weinheimer Str. 68  
68309 Mannheim

[www.oio.de](http://www.oio.de)  
[info@oio.de](mailto:info@oio.de)



# Vielen Dank für ihre Aufmerksamkeit !

Orientation in Objects GmbH

Weinheimer Str. 68  
68309 Mannheim

[www.oio.de](http://www.oio.de)  
[info@oio.de](mailto:info@oio.de)

**Pause**



<http://www.publicdomainpictures.net/>