Modernisierung des Entwicklungsprozesses

- ein Projektbericht

Markus Heinisch

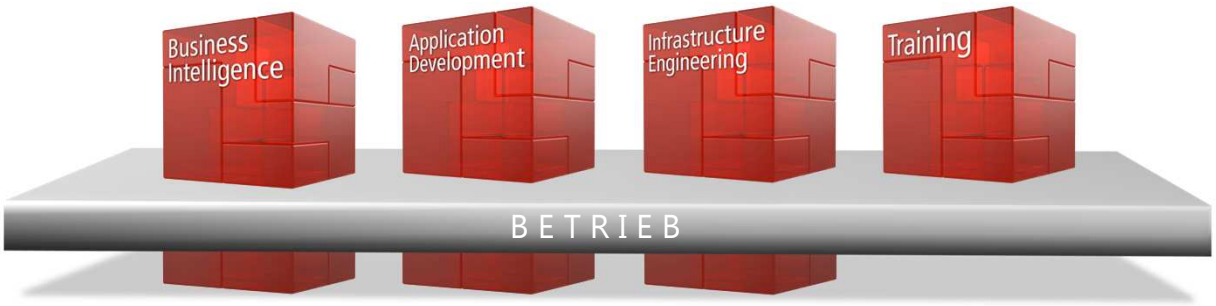
BASEL BERN BRUGG LAUSANNE ZÜRICH DÜSSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MÜNCHEN STUTTGART WIEN

trivadis
makes IT easier.

Unser Unternehmen

Trivadis ist führend bei der IT-Beratung, der Systemintegration, dem Solution-Engineering und der Erbringung von IT-Services mit Fokussierung auf ORACLE und Microsoft Technologien im D-A-CH-Raum.

Unsere Leistungen erbringen wir aus den strategischen Geschäftsfeldern:



Trivadis Services übernimmt den korrespondierenden Betrieb Ihrer IT Systeme.

Mit über 600 IT- und Fachexperten bei Ihnen vor Ort

12 Trivadis Niederlassungen mit über 600 Mitarbeitenden

200 Service Level Agreements

Mehr als 4'000 Trainingsteilnehmer

Forschungs- und Entwicklungs-budget: CHF 5.0 / EUR 4 Mio.

Finanziell unabhängig und nachhaltig profitabel

Erfahrung aus mehr als 1'900 Projekten pro Jahr bei über 800 Kunden

Stand 12/2012

trivadis
makes IT easier.

AGENDA

1. Ausgangslage
2. Release-Strategie
3. Build-Prozess
4. Fazit

trivadis
makes IT easier.

Ausgangslage

Projekt

- Aufgabe
  - Umsetzung einer zu erarbeitenden Release-/Build-Deploymentstrategie
- Kunde
  - IT-Abteilung eines Versicherungskonzerns
  - Ca. 5 Desktop- und Web-Anwendungen als Produkte
  - Technologieumfeld: Java, Eclipse RCP, DB2

Startpunkt

- Geänderte Rahmenbedingungen
  - Anzahl von Projekten steigt
  - Höhere Komplexität durch mehr Geschäftsfälle
  - Gestiegenen Anforderungen an „time to market“ als vor 5-10 Jahren
- Risiko
  Neue Versionen nicht zum gewünschten Termin bereitstellen zu können

- Problemfeld Build- und Deploy-System identifiziert
  - Verursacht sehr hohe Wartungskosten
  - Nicht stabil und flexibel genug für heutigen Anforderungen
  - Grosser Impact auf die Produktivität der einzelnen Entwickler

Build- und Deploy-Prozess

- Komplexer Build- und Deploy-Prozess mit Tool Automated Build Studio
  - Buildprozedur eines Produkts kann aus 50 und mehr manuell konfigurierten Build-Schritten bestehen
- Builds finden jeden Montag Morgen statt
  - Es wird immer ein Release gebaut
  - Monolithischer Build
- Kein automatiertes Dependency Management
  - Keine Wiederverwendung von versionierten und zentral gelagerten vorgebauten Artefakten
  - Sehr lange Build-Zeiten
- Build- und Deploy-Prozess ist fehleranfällig
  - Führt im Fehlerfalle zu grossen Verzögerungen bei der Bereitstellung einer Version

SCM

- Sourcecode-Verwaltung ClearCase
  - Wird in den Build-Prozeduren ungewöhnlich viel involviert
    - Zur Datensicherung für die zu einer Version gebauten binären Artefakte
  - Komplexen Konfiguration
    → sehr anfällig für fehlerhafte und unvollständige lokale Arbeitskopie (View)
  - Download der lokalen Arbeitskopie dauert sehr lange
  - Kein (Community) Support
  - Technische Limits bei windows path length
  - Neue Mitarbeiter benötigen Training

Eclipse RCP

- Equinox OSGi basierter Desktop Client
  - Bau per Eclipse PDE (Plug-in Development Environment) basierten Buildmechanismus und durch zusätzliche Buildskripte
  - Build nicht einfach zu erweitern und oft sehr schwierig zu debuggen
  - Fehleranalyse ist im unübersichtlichen Eclipse PDE Log jeweils sehr aufwendig
  - Verwendung einer Eclipse-Installation auf Build-Server
  - Zentraler Build != lokaler Build, vom Eclipse-Classpath negativ beeinflussten Build
- → Fehler in der Komponenten-Dependency-Konfiguration werden übersehen
  → Kompilierungsfehler erst spät während des zentralen Builds erkannt

Ziele

- Höherer Automatisierungsgrad
  - Mehr Zeit zur Erfüllung der fachlichen und zeitlichen Kundenanforderungen
- Standardisierung der Build-Umgebung und Prozesse
- State of the art: Veraltete und ungeeignete Lösungen sollen durch etablierte Technologien abgelöst werden

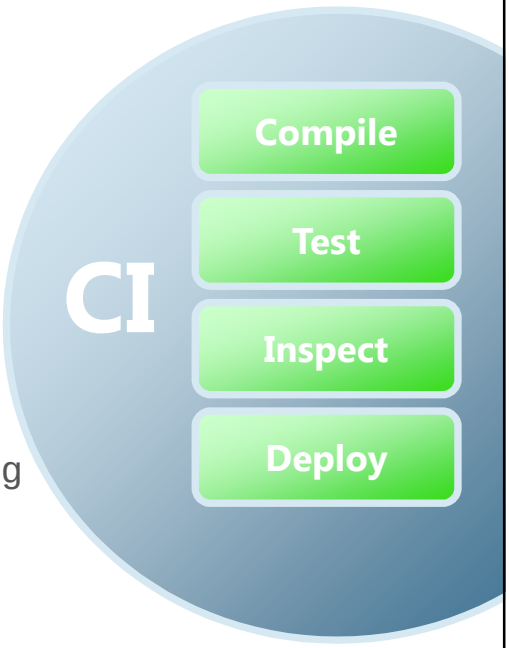
Release-Strategie

Vorgehen

- Konzeption des Build- und Deployment-Prozesses
  - Sprich CI-Prozess → Nächste Folie
- Umsetzen der Konzepte in einem PoC
  - Implementierung
  - Einführung und Schulung bei drei Mitarbeitern
  - Etablierung und Feedback
- Umsetzen der Konzepte bei allen Produkten
  - Produkt für Produkt
  - Parallelbetrieb von ClearCase und Automated Build Studio als Fallback
  - ClearCase bleibt ca. 1 Jahr im Readonly-Modus zugreifbar

Was ist Continuous Integration (CI)?

- Typisches Risiko ohne CI
  - Integration erst zum Ende hin durchgeführt
  - Dauer und Ausgang der Integration ungewiss
  - Manuelles Testen
- CI als Werkzeug und Prozess in Software-entwicklungsprojekten um
  - Risiken zu minimieren
  - Qualität zu steigern
- CI propagiert häufigere Integration als Lösung
  - Häufigere Integration vermeidet nicht nur das Durchleiden der „Integrationshölle“, sondern führt zu weiteren wichtigen Vorteilen
  - „*One check-in a day keeps the tickets away*“



trivadis
makes IT easier.

Release-Strategie

Unter dem Thema Release Strategie wird zunächst der Bereich Branching und Merging (B&M) definiert. Grundsätzlich existiert unabhängig von den gewählten Tools eine Reihe von B&M-Verfahren.

- Develop on Mainline
- Branch for Release
- Branch by Feature
- Branch by Team

Develop on Mainline

- „Develop on Mainline“ bedeutet, dass die Entwickler ihren Sourcecodeänderungen nahezu ausschließlich auf der Mainline (trunk) durchführen
- Vorteile:
  - Damit ist sichergestellt, dass Continuous Integration stattfindet
  - Änderungen sind für alle anderen Entwickler direkt verfügbar
  - Vermeidet das Merging von Sourcecode zu definierten Projektständen
- Nachteile:
  - Nicht jeder Build wird erfolgreich sein, wie die Erfahrung zeigt
  - Vermeiden von Branches kann bedeuten, dass die Entwicklung etwas höheren Aufwand beim Entwickeln von Features haben kann
- Branches machen allerdings Sinn unter der Voraussetzung, dass sie nicht mehr in die Mainline gezogen (merge) werden, wie beispielsweise bei einem Release.

Develop on Mainline

- Typische Frage bei diesem Verfahren:
  Wie können größere oder komplexere Änderungen durchgeführt werden?
- Antwort:
  Aufgabe in kleine Teilaufgaben zu zerlegen und Schritt für Schritt einzuführen, um bestehenden Code nicht zu brechen
  (Unit-Tests helfen ☺)
  - Verstecken von neuen Features bis Featureentwicklung fertig ist.
  - Serie von inkrementellen Änderungen in kleinen Schritten
  - „Branch by Abstraction“ bei größeren Änderungen bedeutet, das ein abstrakter Layer über die Teile des Codes gelegt werden, die geändert werden sollen, damit die Implementierung schlussendlich ausgetauscht werden kann.
  - Lose Kopplung von Komponenten des Systems, damit Abhängigkeiten gering sind und einzelne Komponenten leichter geändert werden können

Branch for Release

- „Branch for Release“ bedeutet, dass man kurz vor einem Release eine Branch für einen Release durchführt. Dieses Vorgehen komplettiert die „Develop on Mainline“ Strategie
- Anforderung
  - Ein Team entwickelt neue Features
  - Ein zweites Team will auf dem Release Bugfixes durchführen ohne dass dadurch neue Features in den Release kommen
- Wenn ein Branch erzeugt ist, dann wird im Wesentlichen der Code nur noch getestet und das Release validiert
  - Während in der Mainline an Features weiterentwickelt wird
- Minimiert den sogenannten „Code Freeze“, während der Zeit kein Code im SCM geändert werden darf

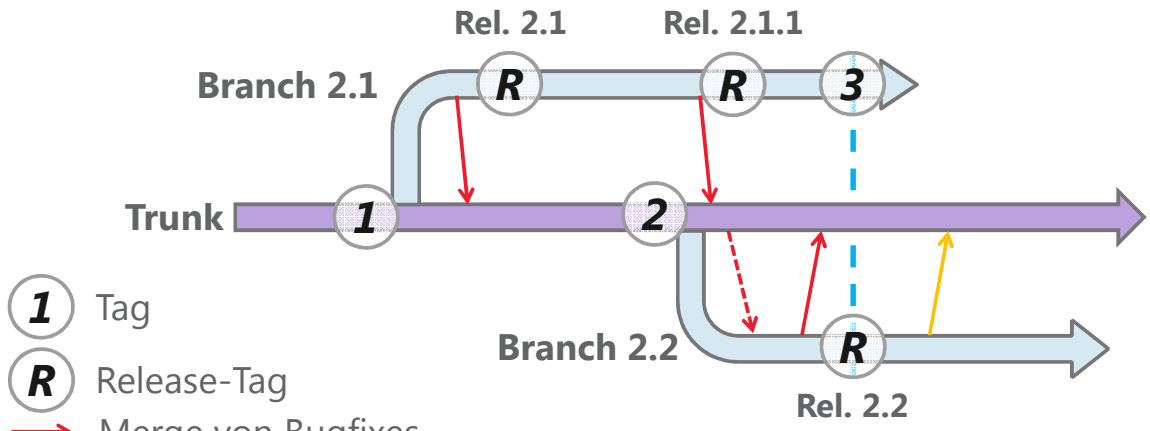
Branch for Release

- Vorgehen:
  - Features werden auf der Mainline entwickelt
  - Ein Branch wird gezogen, wenn der Feature-Code „releaseable“ ist und neue Features entwickelt werden sollen
  - Nur kritische Bugfixes werden im Release-Branch entwickelt und sofort in die Maineline übernommen (merging)
- Dieses Verfahren eignet sich weniger für große Teams, weil es auf Grund der Größe schwierig wird festzulegen, wann ein Set von Features fertig ist und ein Release-Branch gezogen werden sollte

Release-Strategie

- Anforderung:
  Entwicklung muss zwei parallele Releases unterstützen
  - Support des Anwenderkreis des SW-Produktes unterstützt die letzten beiden Versionen
- Für die Aufgaben in der Abteilung ist das Verfahren „Develop on Mainline“ und „Branch for Release“ vorgesehen. Gründe:
  - Optimale Unterstützung des CI-Prozesses
  - Vergleichsweise einfache Handhabung in der Praxis
  - Es sollen maximal zwei Releases der Software in der Entwicklung sein
  - Nahe am heutigen B&M Verfahren
  - Höhere Planungssicherheit zu offiziellen Release-Terminen

Release-Strategie



1 Tag

R Release-Tag

Merge von Bugfixes

Merge von Bugfixes aus Trunk

Zeitpunkt der Umstellung auf neuen Release

Featureset G, notwendige gesetzliche Änderungen

trivadis makes IT easier.

Subversion

- Frage 1
  - (a) Ein Repository für alle Produkte?
  - (b) Pro Produkt ein Repository?

```
http://svn.company.com:3080/abc/Produkt_A/...
                                /Produkt_B/...
                                /Produkt_C/...
```

```
http://svn.company.com:3080/abc_Produkt_A/...
http://svn.company.com:3080/abc_Produkt_B/...
http://svn.company.com:3080/abc_Produkt_C/...
```

- Unterschiede zwischen A&B
  - Berechtigungen werden pro Repository verwaltet, es können Read/Write-Rechte pro Pfad vergeben werden
  - Branching und Merging bezieht sich immer auf ein Repository
  - Revisionnumber-Verwaltung pro Repository

Subversion, Frage 1

- Entscheidung für Variante A
  - Geringen administrativen Aufwand (außerhalb und innerhalb der Abteilung)
  - Flexibilität bei Branching und Merging
  - Produkte besitzen fachlich und technisch gemeinsame Komponenten

Subversion, Frage 2

- Frage 2:
  Wie ist die Struktur innerhalb des Repositories?

Variante (1)

```
…/abc/
    /ProjektA/
              /trunk
              /branches
              /tags
    /ProjektB/
              /trunk
              /branches
              /tags
    /ProjektC/
              /trunk
              /branches
              /tags
```

Variante (2)

```
…/abc/
    /trunk/
          /ProjektA
          /ProjektB
          /ProjektC
    /branches/
             /ProjektA
             /ProjektB
             /ProjektC
    /tags/
         /ProjektA
         /ProjektB
         /ProjektC
```

Subversion, Frage 2

- Unterschiede zwischen Variante 1&2
  - Technisch keine Unterschiede fürs SCM
  - Variante 1 trennt die Produkte sehr anschaulich
  - Variante 2 betont die SCM Eigenschaften wie Tags und Branches
  - Aus Entwicklersicht:
    - Variante 1: Check-out aller Sourcen eine Produktes über einen Link
    - Variante 2: Check-out nach „Typ“ trunk, tag oder branch
  - Weiterentwicklung mit Sub-Produkten bei Variante 2 einfacher

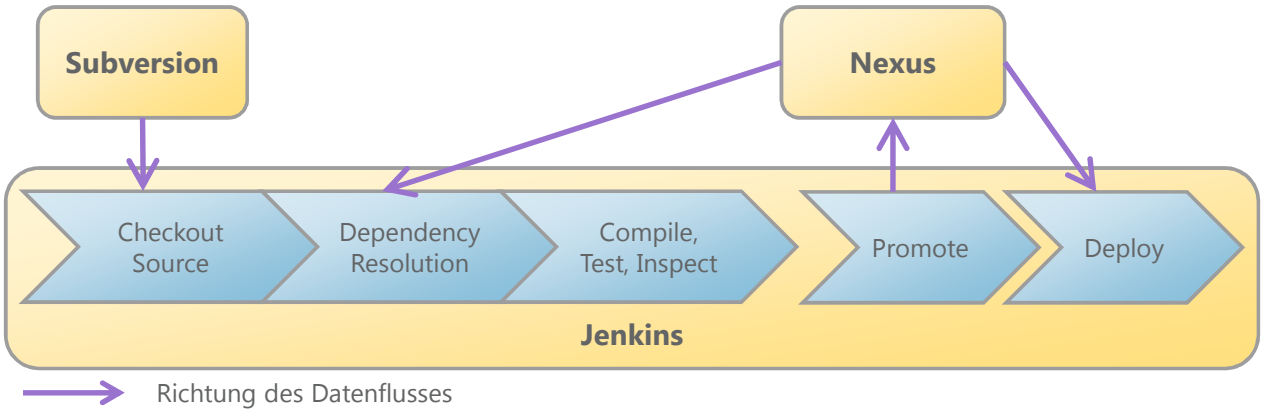
```
…/abc/
    /ProjektA/trunk/cmp1/
                        /trunk
                        /branches
                        /tags
                    /cmp2/
                        /trunk
                        /branches
                        /tags
              /branches
              /tags
    /ProjektB/…
```

```
…/abc/
    /trunk/
          /ProjektA/cmp1
                   /cmp2
          …
    /branches/
             /ProjektA/cmp1
                      /cmp2
             …
    /tags/
         /cmp1
         /cmp2
         …
```

Entscheidung für Variante 2

Build-Prozess

Neuer CI-Build-Prozess



- Gewöhnlicher Build nach Subversion Checkin
- Manuelle Build-Promotion bei ausgewählten Builds
- Deployment auf Zielsysteme in Absprache mit Test-Team
- CI-Prozess für alle 5 Produkte nahezu identisch

Unterschiede zum alten Build-Prozess

- Der neue Build-Prozess ist etwas anderes konstruiert, die wesentlichen Unterschiede sind:
  - Statt einem Build-Prozess wird das Produkt mittels mehreren verketteten Builds gebaut
    - Umfangreiche Produkte werden modularisiert
  - Jeden Tag (bzw. nach jedem Checkin ins SCM) wird gebaut
    → nicht jeder Build führt zu einem neuen «Release"
  - Typische ClearCase-Schritte entfallen
  - Kopieren von Sourcen und Eclipse-Distribution in ein gesondertes Arbeitsverzeichnis entfällt

Manuelle Build-Schritte

- Promotion
  - Ein Promotion in dem Projekt bedeutet technisch, dass das gebaute Artefakt auf den internen Nexus hochgeladen wird
  - Artefakt steht als Dependency für weitere Builds zur Verfügung
  - Promotion-Vorgang kann wiederholt werden
- Deployment
  - Mit Angabe einer Versionsnummer wird ein Artefakt (Produkt) aus Nexus geladen
  - Das Deployment des Produktes wird wie bisher als ausgepackte ZIP-Datei auf einem Test-Server zur Verfügung gestellt
  - Schnittstellen zu nachfolgenden Test- oder Rollout-Teams bleiben unverändert

trivadis
makes IT easier.

Fazit

Fazit

- Hindernisse
  - Sehr großer Aufwand, PDE-Build in Maven vollständig zu integrieren
    - Dependency-Management
    - U.a. Versionsnummer (OSGi <> Maven)
    - Release-Plugin

- Erfolgsfaktoren
  - Schrittweise Einführung, Parallelbetrieb
  - Geringe Einarbeitung der Entwickler

  - ➔ Mehr Agilität
  - ➔ Höhere Qualität
  - ➔ Größere Zuverlässigkeit

Fragen und Antworten...

Markus Heinisch

Principal Consultant

Tel.: +49 89 99275930

Markus.Heinisch@trivadis.com

BASEL BERN BRUGG LAUSANNE ZÜRICH DÜSSELDORF FRANKFURT A.M. FREIBURG I.BR. HAMBURG MÜNCHEN STUTTGART WIEN

trivadis
makes IT easier.

Trivadis an der DOAG

Ebene 3 - gleich neben der Rolltreppe

Wir freuen uns auf Ihren Besuch.

Denn mit Trivadis gewinnen Sie immer.