WELCOME

# Modernisierung des Entwicklungsprozesses – ein Projektbericht

**Markus Heinisch**

Mai 2013

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

# AGENDA

1. Ausgangslage
2. Bisheriger Build- und Deploy-Prozess
3. Release-Strategie
4. SCM
5. Build-Prozess


2012 © Trivadis

trivadis
makes IT easier.

# Projekt

- Aufgabe
  - Unterstützung Umsetzung der Release-/Build-Deploymentstrategie
- Termin
  - Zeitperiode vom 03.12.2012 – 30.06.2013
- Consultants
  - Markus Heinisch, Principal Consultant, München
  - Mischa Kölliker, Principal Consultant, Zürich
- Kunde
  - IT-Abteilung eines Versicherungskonzerns
  - Ca. 5 Desktop- und Web-Anwendungen als Produkte
  - Technologieumfeld: Java, Eclipse RCP, DB2



2012 © Trivadis



trivadis
makes IT easier.

# Ausgangslage

- 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


2012 © Trivadis

trivadis
makes IT easier.

# AGENDA

1. Ausgangslage
2. Bisheriger Build- und Deploy-Prozess
3. Release-Strategie
4. SCM
5. Build-Prozess
6. Ausblick


2012 © Trivadis

trivadis
makes IT easier.

# Build- und Deploy-Prozess

- Komplexer Build- und Deploy-Prozess
  - Tool Automated Build Studio
  - Proprietär
  - 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
  - Sonst wird nicht gebaut!
- Build- und Deploy-Prozess ist fehleranfällig
  - Build-Vorgänge brechen auf Grund von Fehlern häufig ab


2012 © Trivadis

trivadis
makes IT easier.

# Build- und Deploy-Prozess

- Monolithischer Build
- Kein automatisches Dependency Management
- Keine Wiederverwendung von versionierten und zentral gelagerten vorgebauten Artefakten
  - → Sehr lange Build-Zeiten
- → Führt im Fehlerfalle zu grossen Verzögerungen bei der Bereitstellung einer Version


2012 © Trivadis

trivadis
makes IT easier.

# SCM

- Sourcecode-Verwaltung ClearCase
  - Wird in den Build-Prozeduren ungewöhnlich viel involviert
  - Als Daten-Sicherung 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


2012 © Trivadis

trivadis
makes IT easier.

# 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


2012 © Trivadis

# Ziele

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


2012 © Trivadis

trivadis
makes IT easier.

# AGENDA

1. Ausgangslage
2. Bisheriger Build- und Deploy-Prozess
3. Release-Strategie
4. SCM
5. Build-Prozess
6. Ausblick


2012 © Trivadis

trivadis
makes IT easier.

# Vorgehen

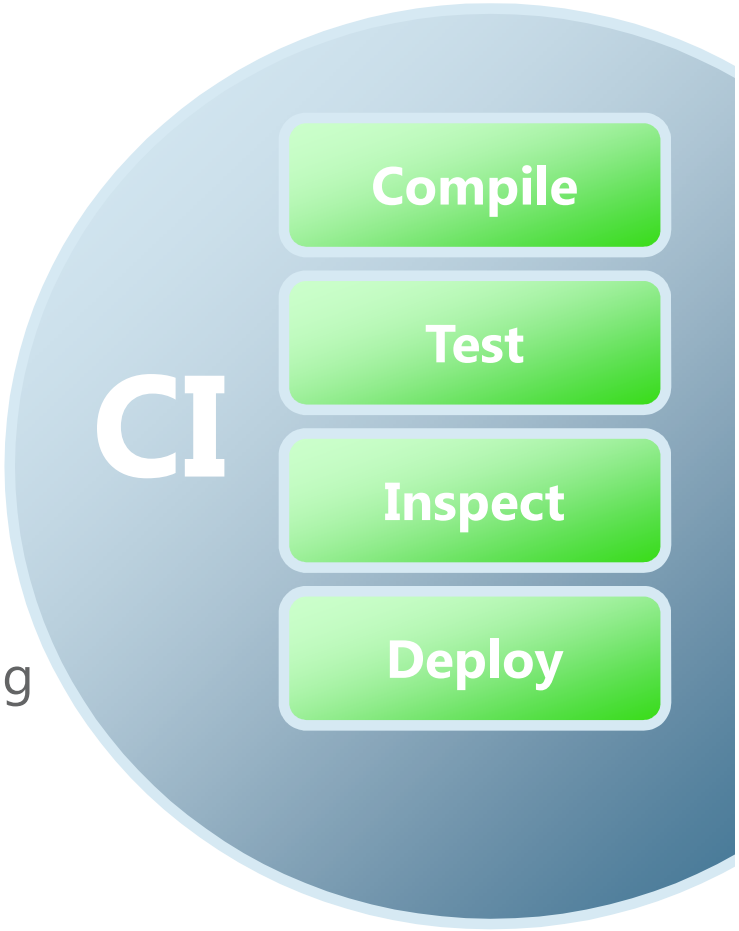
- ✓ Konzeption des Build- und Deployment-Prozesses
  - Sprich CI-Prozess → Nächste Folie
- ✓ Umsetzen der Konzepte in einem PoC
  - Implementierung
  - Einführung und Schulung bei drei Mitarbeitern
  - Etablierung und Feedback
- Umsetzen der Konzepte bei allen Produkten


2012 © Trivadis

trivadis
makes IT easier.

# 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*“





2012 © Trivadis

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


2012 © Trivadis

trivadis
makes IT easier.

# 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.


2012 © Trivadis

trivadis
makes IT easier.

# 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


2012 © Trivadis

trivadis
makes IT easier.

# 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


2012 © Trivadis

trivadis
makes IT easier.

# 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


2012 © Trivadis

trivadis
makes IT easier.

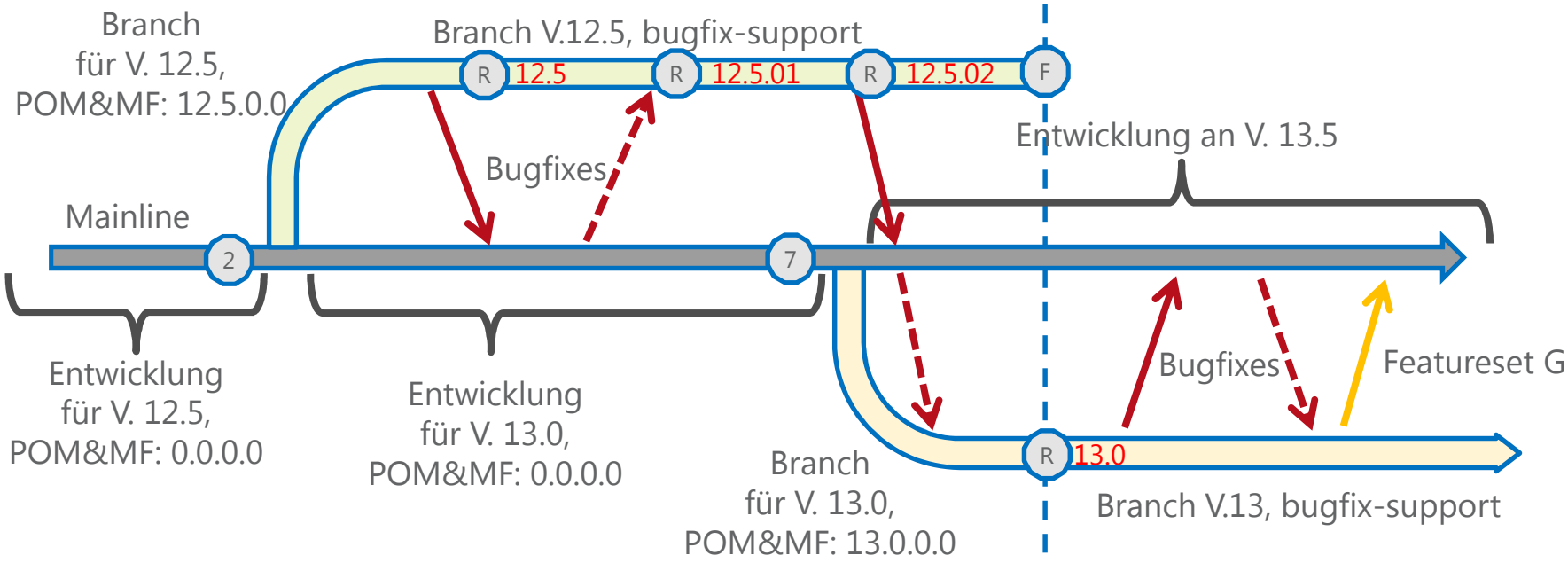
# Release-Strategie

- 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


2012 © Trivadis

trivadis
makes IT easier.

# Release-Strategie




- (2) Tag für einen Build, der einen Release-Status Promotion erhält
- (R) Tag für einen Build, der zu einem Release/Rollout führt
- (F) Tag für einen **finalen** Abschluss des Branch
- Featureset G, notwendige gesetzliche Änderungen
- Merge von Bugfixes
- Merge von Bugfixes aus Mainline
- Zeitpunkt der Umstellung auf neuen Release beim Kunden


2012 © Trivadis


trivadis
makes IT easier.

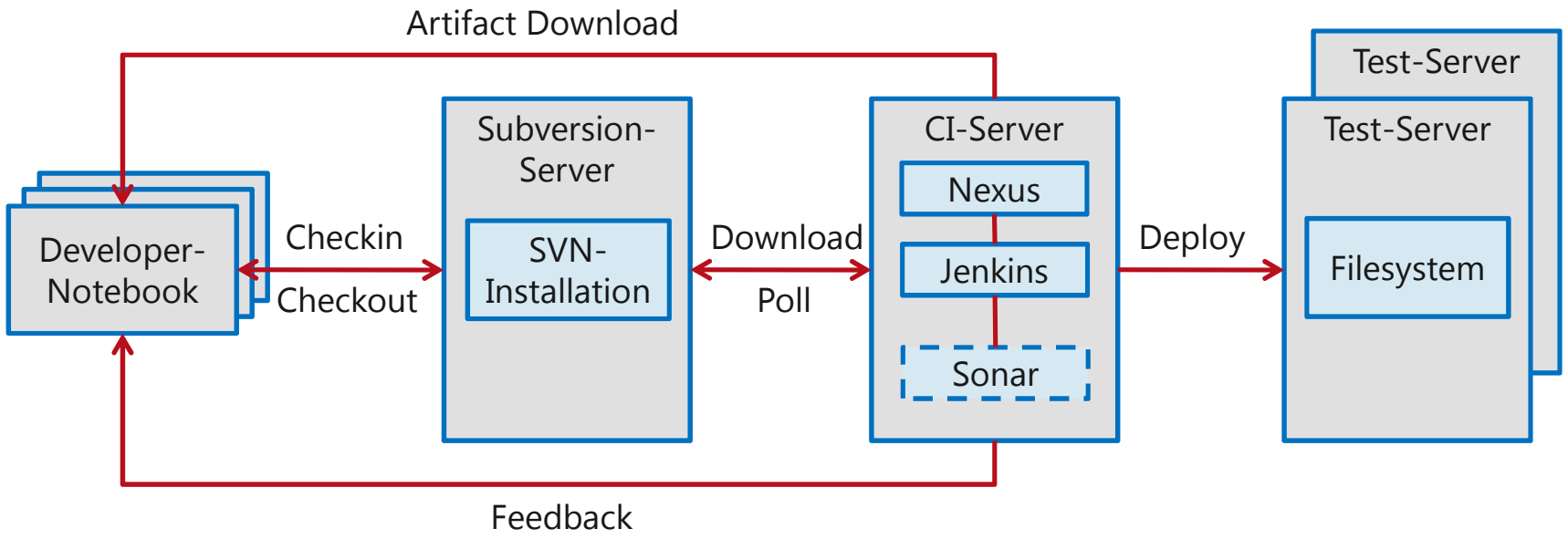
# AGENDA

1. Ausgangslage
2. Bisheriger Build- und Deploy-Prozess
3. Release-Strategie
4. SCM
5. Build-Prozess
6. Ausblick




2012 © Trivadis

# CI-Infrastruktur






2012 © Trivadis



trivadis
makes IT easier.

# Subversion

- Im SCM wird der Sourcecode für mehrere zum Teil von einander abhängiger Produkte verwaltet
- Frage 1
  - (a) Ein Repository für alle Produkte?
  - (b) Pro Produkt ein Repository?

```
http://sr02329:3080/abc/Produkt_A/...
                         /Produkt_B/...
                         /Produkt_C/...
```

```
http://sr02329:3080/abc_Produkt_A/...
http://sr02329:3080/abc_Produkt_B/...
http://sr02329:3080/abc_Produkt_C/...
```

- Unterschiede zwischen A&B
  - Berechtigungen
    Werden pro Repository verwaltet, es können Read/Write-Rechte pro Pfad vergeben werden
  - Branching und Merging
    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


2012 © Trivadis

trivadis
makes IT easier.

# 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
```


2012 © Trivadis

# 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
- Entscheidung für Variante 2

```
…/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
         …
```

# AGENDA

1. Ausgangslage
2. Bisheriger Build- und Deploy-Prozess
3. Release-Strategie
4. SCM
5. Build-Prozess
6. Ausblick



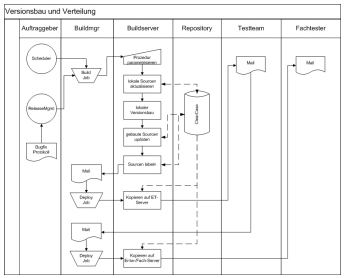


2012 © Trivadis



trivadis
makes IT easier.

# Build-Process (old) I

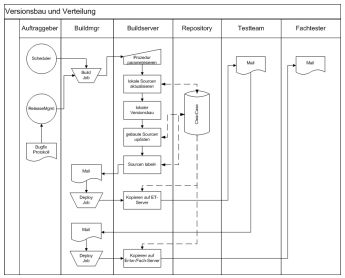



- Build- und Deployment-Prozess in drei Phasen
- Prebuild
  - Variablen initialisieren
  - ClearCase-View anlegen und Sourcen laden
  - Setzen der Variablen
  - Setzen und Speichern der Versionsnummern in der Datei versionsnummern.ini
  - Update der Versionsnummer in der Datei version.isc
  - Aktualisieren der Versionsnummer in der Datei manifest.mf
  - CRCs erstellen
  - Sourcen für den Build lokal kopieren
  - Löschen der Artefaktverzeichnisse
  - Kern Binaries kopieren
  - Build-Eclipse lokal kopieren


2012 © Trivadis

trivadis
makes IT easier.

# Build-Process (old) II & III




- Build Repository-Product-Build:
  - Ausführen des Antscripts in Eclipse
  - Fehlerbehandlung
  - Kontrolle, ob Platform-Plugins für das aktuelle Repository gebaut wurden
  - Kontrolle, ob etwas deployed wurde
- Postbuild
  - Repository und Deploy versionieren
  - File-Images auf Testserver verteilen
  - Mail an die Teams verschicken
  - Labeln aller VOBs incl. update der Versionsnummer in der versionsnummern.ini
  - Export Log


2012 © Trivadis

trivadis
makes IT easier.

# Jenkins



- 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


2012 © Trivadis

trivadis
makes IT easier.

# Jenkins



- Logische Darstellung:
  1. Gewöhnlicher Build nach SCM Checkin
     - Download der aktualisierten Sourcen aus dem SCM (Subversion)
     - Maven Aufruf zum Bau der Sourcen
  2. Manuelle Build-Promotion
     - Deployment der Artefakte in lokales Maven Repository (Nexus)
  3. Deployment auf Zielsysteme
     - 5 verschiedene Test- und Übergabesysteme


2012 © Trivadis

trivadis
makes IT easier.

# Gewöhnlicher Build I



**Source-Code-Management**

- ○ Git
- ○ Keines
- ◉ Subversion

Module

Repository URL: http://s 3080/CS/trunk/

Lokales Modulverzeichnis (optional): .

Repository depth option: infinity

Ignore externals option: ☐

Weiteres Modul hinzufügen...

Check-out Strategy: Use 'svn update' as much as possible

Use 'svn update' whenever possible, making the build faster. But this causes the artifacts from the previous build to remain when a new build starts.

Repository Browser: (Auto)

Erweitert...

**Build**

Stamm-POM: pom.xml

Goals und Optionen: clean install

Erweitert...

**Post Steps**

○ Run only if build succeeds ◉ Run only if build succeeds or is unstable ○ Run regardless of build result

Should the post-build steps run only for successful builds, etc.



2012 © Trivadis



trivadis
makes IT easier.

# Gewöhnlicher Build II



**Build-Einstellungen**

Suche im Arbeitsbereich nach offenen Punkten

Zu untersuchende Dateien: **/*.java

Angabe einer ANT Fileset includes Anweisung, die den Pfad der zu untersuchenden Dateien bestimmt, z.B. **/*.java. Es können auch mehrere durch Komma getrennte Anweisungen definiert werden, z.B. **/*.c, **/*.h Als Ausgangsverzeichnis für diese Anweisung wird der Arbeitsbereich verwendet. Falls kein Wert eingetragen wird, dann wird die Vorgabe **/*.java benutzt.

Zu ignorierende Dateien:

Angabe einer ANT Fileset excludes Anweisung, die die Dateien angibt, die beim Scannen nicht berücksichtigt werden sollen, z.B. Dateien von Fremdbibliotheken. Als Ausgangsverzeichnis für diese Anweisung wird der Arbeitsbereich verwendet.

Kennzeichnung offener Punkte

| Hohe Priorität | Normale Priorität | Niedrige Priorität | Groß-/Kleinschreibung ignorieren |
|---|---|---|---|
| FIXME | TODO | | ☑ |

Konfiguriere die Texte, nach denen in den angegebenen Dateien gesucht werden soll. Für jede Priorität kann eine durch Kommas getrennte Liste von Texten definiert werden, z.B. TODO, FIXME, o.a. Die Groß- bzw. Kleinschreibung kann dabei optional ignoriert werden.

Erweitert...

**Post-Build-Aktionen**

**Suche nach Compiler Warnungen**

Konsole durchsuchen

Parser: Maven

Passt keiner der vorhandenen Parser, so kann in der System Konfiguration ein eigener Parser entwickelt werden, der genau auf die eigenen Anforderungen abgestimmt ist.

Löschen

Hinzufügen

Diese Parser werden zum Analysieren der Konsolenausgabe verwendet. Falls kein Parser ausgewählt wird, wird die Konsolenausgabe nicht untersucht.

Dateien durchsuchen: Hinzufügen

Dateien aus dem Arbeitsbereich, die mit einem festdefiniertem Parser durchsucht werden sollen.

Erweitert...

Löschen



2012 © Trivadis



trivadis
makes IT easier.

# Build-Promotion



- Ein Promotion bedeutet technisch, dass das gebaute Artefakt auf den internen Nexus hochgeladen wird
- Promotion wir manuell auf einem vorhandenen Build angestossen
- Promotion-Vorgang kann wiederholt
- Allerdings kann ein Release mit der gleichen Versionsnummer nicht ein zweites Mal erfolgreich auf dem Nexus hochgeladen werden, da grundsätzlich bei Maven gilt, dass ein "deployes" Release-Artefakt mit einer Version nicht mehr geändert werden soll
  - Policy kann in Nexus geändert werden


2012 © Trivadis

trivadis
makes IT easier.

# Build-Promotion



Promote builds when...

Promotion process

Name: Deploy to Artifact Repository

Icon: Gold star

Restrict where this promotion process can be run

**Criteria**

Only when manually approved

Approvers

Approval Parameters

**Text Parameter**

Name: PEX_VERSION

Default Value: 13.0.0.0

Description: Version- oder Release-Nummer der PEX-Applikation, die in Nexus abgelegt werden soll, Beispiel Release: 13.5.0.5

Löschen

Add Parameter

Promote immediately once the build is complete

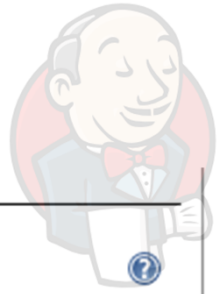
Promote immediately once the build is complete based on build parameters

When the following downstream projects build successfully

When the following upstream promotions are promoted

vadis asier.

# Build-Promotion



**Actions**

**Maven Goals aufrufen**

Maven Version: Maven 3.0.3

Goals: deploy:deploy-file

POM:

Eigenschaften ("properties"):

```
url=http://                    .ch:8080/nexus/content/repositories/releases/
file=pex_application/target/PEx-Application_V$PEX_VERSION.zip
repositoryId=PEX-nexus
groupId=          .pex
artifactId=PExApplication
version=$PEX_VERSION
packaging=zip
```

JVM-Optionen:

Privates Maven-Repository verwenden

Settings file: Use default maven settings

Global Settings file: Use default maven global settings

Löschen

Add action

Add another promotion process



2012 © Trivadis

# Deployment



- Das Deployment des erzeugten Produktes wird wie bisher als ausgepackte ZIP-Datei auf einem Test-Server zur Verfügung gestellt
  - Schnittstellen zu nachfolgenden Test- oder Rollout-Teams bleiben unverändert
- Ablauf eines Deployments (von 5):
  - Voraussetzung: Im internen Repository liegt das zu verteilende Artefakt als ZIP Datei
    - Beispiel: Datei PExApplication_V13.0.0.1.zip in Repo `http://abc.xyz.ch:8080/nexus/content/groups/public/`
  - Ein CI Job lädt die ZIP Datei aus dem Repository in lokales Arbeitsverzeichnis
  - ZIP Datei wird in ein temporäres Verzeichnis ausgepackt
  - Ausgepackte Dateien werden auf den vereinbarten Windows-Share kopiert
  - ZIP Datei u. temporäre Verzeichnis wird gelöscht
  - Optional: Email-Notifikation nach erfolgreichem Deployment aufs Share
    - Email-Struktur und Inhalt wie bisher

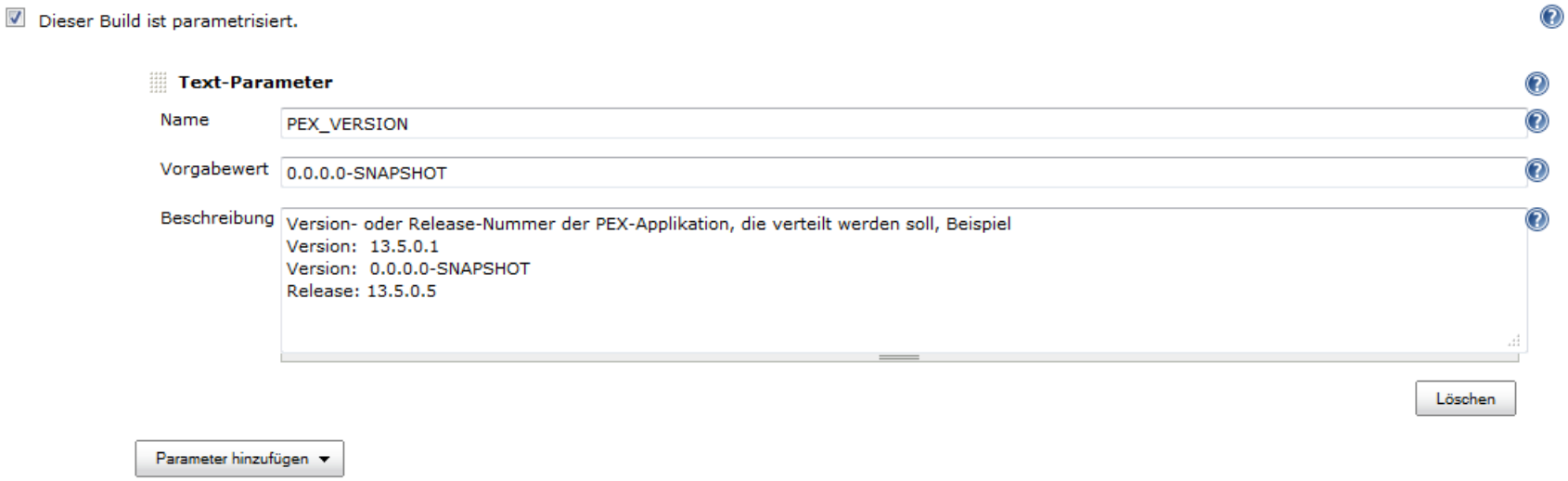

2012 © Trivadis

trivadis
makes IT easier.

# Deployment



- Angabe der Version fürs Deployment (parametrisierter Build)







2012 © Trivadis

# Deployment

**Buildverfahren**

**Maven Goals aufrufen**

Maven Version: (Standard)

Goals: -U org.apache.maven.plugins:maven-dependency-plugin:2.7:get

Erweitert...

Löschen

**Windows Batch Datei ausführen**

Kommando:

```
rmdir /S /Q temp
"C:\Program Files\VMware\VMware Tools\unzip" -o PExApplication_V%PEX_VERSION%.zip -d temp
del /Q PExApplication_V%PEX_VERSION%.zip
```

Liste der verfügbaren Umgebungsvariablen

Löschen

**Windows Batch Datei ausführen**

Kommando:

```
xcopy /Q /E /R /Y /I D:\java-dist\jre_1.7.0_09\jre\*.* temp\bin\jre
```

Liste der verfügbaren Umgebungsvariablen

Löschen



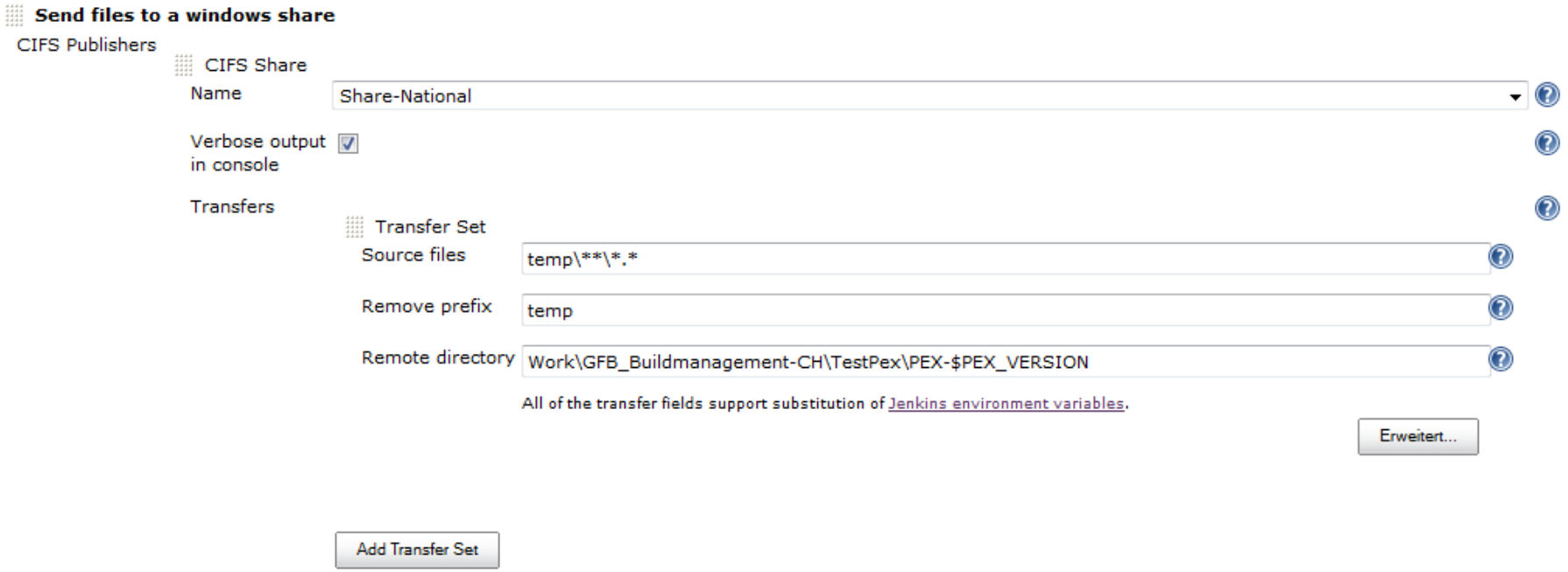
2012 © Trivadis



trivadis
makes IT easier.

# Deployment








2012 © Trivadis



trivadis
makes IT easier.

# AGENDA

1. Ausgangslage
2. Bisheriger Build- und Deploy-Prozess
3. Release-Strategie
4. SCM
5. Build-Prozess
6. Ausblick





2012 © Trivadis

# Ausblick

- PoC mit einem Produkt umgesetzt
  - Es folgen 3 weitere Produkte
- Schulung aller Mitarbeiter bei Überführung des nächsten Produktes auf den neuen Prozess
- Einführen/Vertiefen des Test-Gedankens

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


2012 © Trivadis

trivadis
makes IT easier.

# THANK YOU.

Company

Name

Address
City

Tel. +...
Fax +...

info@trivadis.com
www.trivadis.com

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



2012 © Trivadis



trivadis
makes IT easier.

# Ziele

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

- Zeitgerechte Bereitstellung aller geplanten Versionen in guter Qualität
- Durchlaufzeiten der Versions-Build-Vorgängen deutlich reduzieren
- Implementierung eines sauberen Versionsmanagement Konzeptes
- Entwicklungsubgebung sollte getrennt werden von der Runtimeumgebung.
  - Vermeidung von unterschiedlichen Buildsysteme vom Entwickler und Buildserver


2012 © Trivadis

trivadis
makes IT easier.

# Branch by Feature

- Für große Teams gedacht
  - Die an mehreren Features arbeiten wollen
  - Die Mainline in einem Release-Status halten wollen
- Jedes Feature wird dabei in einem eigenen Branch entwickelt
  - Wenn das Feature entwickelt ist und alle Tests erfolgreich durchlaufen hat, dann werden die Änderungen des Feature-Branch in die Mainline übertragen
- Grundsätzliche widerspricht diese Verfahren dem CI-Prozess, da die „Integration“ der einzelnen Branches zu einen kompletten System nicht mehr täglich (mehrfach) stattfindet, sondern erst nach vielen Tagen


2012 © Trivadis

trivadis
makes IT easier.

# Branch by Team

- Grundidee ist, dass die Mainline immer „releasable“ ist
  - Es wird ein Branch pro Team gezogen
  - Jedes Team arbeitet an mehreren Features in seinem Branch
- Der Branch wird in die Mainline gezogen, wenn er wieder stabil ist und ein oder mehrere Features komplett sind
  - Jede Änderung an der Mainline wird unverzüglich in jeden Branch gezogen
  - Je öfters Teams einen Merge in die Mainline durchführen, desto öfters müssen die Änderungen in alle Branches gezogen werden
- Vorteil diese Verfahrens gegenüber „Branch by Feature“ ist, dass es weniger Branches geben wird


2012 © Trivadis

trivadis
makes IT easier.

# Branch by Team

- Nachteil dieses Verfahrens ist es, dass die „unit of work“ sich auf einen kompletten Branch bezieht und nicht auf den zuletzt geänderten Sourcecode
  - Einzelne Änderung (auch ein einzelner Bugfix) kann nicht in die Mainline gezogen werden, Folge
    - Integration mit der Mainline verzögert sich in Richtung eines „Integration Milestone“, was fundamental gegen CI spricht
  - Aus der Praxis
    - Merging von größeren und verzögerten Soucecode-Änderungen aufwendig und risikoreich
    - Teams nicht konsequent den Merge aus dem Branch in die Mainline und von der Mainline in alle Branches vollziehen
      - Damit divergiert der Sourcecode der Branches relativ schnell und das Merging wird komplex und risikoreich
      - Von einem Gesamtsoftwarestand kann man dann nicht mehr sprechen


2012 © Trivadis

trivadis
makes IT easier.

# Branch by Team

- Dieses Verfahren hat einige Anforderung, damit es funktioniert
  - Jede Änderung der Mainline muss zeitnah (täglich) in jeden Branch übertragen werden
  - Branches dürfen nur kurz existieren (z.B. wenige Tage, niemals über ein Iteration hinaus)
  - Die Anzahl der aktiven Branches darf die Anzahl der zu entwickelnden Features nicht überschreiten. Es wird erst dann ein neuer Branch gezogen, wenn ein Feature in Mainline erfolgreich übertragen wurde
  - Ein Merge aus einem Branch darf nur durchgeführt werden, wenn das Feature erfolgreich getestet wurde.
  - Refactorings müssen sofort auf die Mainline übertragen werden, damit Merge-Konflikte möglichst gering bleiben
  - Die technische Leitung der Entwicklung ist verantwortlich, dass die Mainline releasable bleibt


2012 © Trivadis

trivadis
makes IT easier.