

Dipl.-Ing. Frank Sell
System Engineer Unix

🌐: www.xpile.de
✉: fsell@xpil.de



Workflow von Entwicklungen

am Beispiel von ClearCase

Dokument: entwicklung_workflow

Erstellungsdatum: 19. Oktober 2004
Letzte Bearbeitung: 14. Januar 2014
Version: 86

© Copyright 1994-2014 – Xpile-Softwareentwicklung

Dieses Dokument wurde von Xpile-Softwareentwicklung erstellt und enthält vertrauliche Informationen. Alle Rechte der Nutzung dieses Dokuments sind ausschließlich Xpile-Softwareentwicklung vorbehalten. Die Anfertigung von Kopien, jegliche Vervielfältigung und Speicherung auf irgendein Datenmedium, die Weitergabe an Personen, die nicht Mitarbeiter der am Projekt beteiligten Unternehmen sind, oder die Benutzung in irgendeiner Weise ist ohne ausdrückliche schriftliche Genehmigung von Xpile-Softwareentwicklung verboten.

Inhaltsverzeichnis

1	Einleitung.....	4
2	Allgemeiner Workflow eines Software-Entwicklungsauftrages.....	5
2.1	Statusübergänge eines geplanten Entwicklungsauftrags.....	6
2.2	Notfallmaßnahmen.....	9
3	Synchronisation zwischen Kunde und Softwarelieferant.....	11
4	Sicherung der Qualität innerhalb der Versionsverwaltung.....	13
4.1	Verwalten von Config-Specs.....	13
5	Unterstützende Werkzeuge für Versionsverwaltungen.....	15
5.1	Konfiguration eines Projektes in der Versionsverwaltung.....	15
5.2	Tools zum Bearbeiten der Config-Specs.....	18
5.2.1	Editieren von Config-Specs.....	19
5.2.2	Sichern von Config-Specs.....	21
5.2.2.1	Config-Specs für Releasestände.....	21
5.2.2.2	Config-Specs für Produktionsstände.....	23
5.2.3	Erzeugen neuer Projekte.....	23
5.2.4	Weitere Hinweise zur Gestaltung von Config-Specs.....	24
5.2.4.1	Nutzung gemeinsamer Module.....	24
5.2.4.2	Paketierung der Projekte.....	25
5.2.5	Erzeugung eines Windows-Profiles für ClearCase-Clients.....	27
5.3	Sicherstellung des Zugriffs auf die Versionsverwaltung.....	27
5.3.1	Erzeugung einer Sicht auf die Versionsverwaltung.....	27
5.3.2	View auf ein neues Projekt konfigurieren.....	28
5.3.3	Vorhandene Views zur Arbeit nutzen.....	30
5.3.4	Einstellen der Sicht in einem View.....	31
5.3.5	Ermitteln der aktuellen Release- und Produktionsstände.....	32
5.3.6	Löschen einer Sicht auf die Versionsverwaltung.....	33
5.3.7	Kommunikation der Views in verschiedenen Terminals.....	35
5.4	Development-Tools.....	37
5.4.1	Start einer Entwicklung.....	37
5.4.1.1	Standartentwicklung.....	38
5.4.1.2	Hotfixentwicklung.....	40
5.4.2	Abschluss einer Entwicklung.....	41
5.4.3	Rebase eines neuen Releasestandes auf eine laufende Entwicklung.....	44
5.5	Deployment-Tools.....	50
5.5.1	Integration eines Entwicklungsauftrages.....	50
5.5.2	Erzeugen eines neuen Release-Standes.....	54
5.5.3	Auswertungen über den Releasestand.....	57
6	Zusammenfassung.....	60
7	Abkürzungsverzeichnis.....	61

1 Einleitung

Entwicklungstätigkeiten laufen nach fest vorgegebenen 'Fahrplänen' ab.

Am Anfang jeder Entwicklung steht der Auftrag. Ihm folgen die eigentliche Entwicklungstätigkeit, Tests und letztendlich die Produktionseinführung.

Die Qualitätssicherung der einzelnen Schritte ist in der heutigen Zeit kein 'nice to have' sondern ein unbedingtes 'must have'.

Alles muss nachvollziehbar dokumentiert und verwaltet werden.

Es gibt nichts fataleres, als z.B. einen Fehler direkt im produktiven System zu beseitigen, dieses aber nicht in der Versionsverwaltung zu dokumentieren.

Es muss zu jedem Zeitpunkt gewährleistet sein, dass die Versionsverwaltung den aktuellen bzw. den geplanten Produktionsstand widerspiegelt.

Dieses Dokument beschreibt den Workflow des Entwicklungsprozesses sowie Rollen und Übergabeschnittstellen der daran beteiligten Teams.

Weiterhin werden Werkzeuge, welche zur Unterstützung des Workflows unter Anwendung von ClearCase¹ als Versionsverwaltungssystem entwickelt wurden, vorgestellt.

Diese Tools sind für eine Arbeit in UNIX-Systemen mit Base ClearCase bestimmt.

Es werden ausschließlich die Schritte und die Anwendung der Tools innerhalb der Versionsverwaltung beschrieben.

Die Verwaltung der Entwicklungstätigkeiten (Fehler-Tracking-Tools wie z.B. ClearQuest², DDTS³, Bugzilla⁴ u.ä.) wird nicht betrachtet.

1 Die Rechte an Rational ClearCase werden von IBM gehalten

2 Die Rechte an Rational ClearQuest werden von IBM gehalten

3 Die Rechte an Rational DDTS werden von IBM gehalten

4 Bugzilla ist eine Open Source Entwicklung der The Mozilla Organization

2 Allgemeiner Workflow eines Software-Entwicklungsauftrages

An einer Entwicklung sind in der Regel unterschiedliche Teams beteiligt, zwischen denen eine klar abgegrenzte Aufgabenverteilung existiert:

1. Releasemanagement

Das wichtigste und damit allen anderen Teams übergeordnet ist das Releasemanagement-Team. Hier werden alle Entwicklungsaufträge koordiniert und verwaltet.

Das Team muss folgende Anforderungen abdecken:

- Erfassen der eingehenden Entwicklungsaufträge (Aufträge resultierend aus einer Störungsmeldung, aus Änderungen von Anforderungen, aus Änderungen von Schnittstellen oder aus neuen Anforderungen).
- regulative Aufgaben (Auftragserteilung, Kontrolle, Terminabstimmung, ...)
- Freigaben zur produktiven Installation
- Risikoabschätzungen
- Dokumentation aller Informationen

2. Developmentteam

In diesem Team erfolgt die Umsetzung des Entwicklungsauftrages:

- Entwicklung
- Modultests

3. Deploymentteam

Das Bindeglied zwischen Entwicklung, Test und produktiven Betrieb ist das Deploymentteam.

Es hat folgenden Aufgabenbereich:

- Management der Versionsverwaltungssysteme
- Bereitstellung von Installationspaketen
- Installation auf Test- und Produktivsystemen

4. Testteam

Diesem Team obliegt maßgeblich die Qualität der zu liefernden Entwicklungsergebnissen. Sie wird gesichert durch:

- System- und Integrationstests anhand der Spezifikationen
- Aussprechen von Freigabeempfehlungen oder Zurückweisung der Entwicklungsergebnisse

Aus Gründen der Qualitätssicherung und der Verantwortlichkeit sollte diese strikte Aufgabenabgrenzung auch wirklich eingehalten werden.

Natürlich ist es auch möglich, bei kleinen Projekten die Verantwortlichkeiten zusammen zu fassen. Trotz alledem sollten die Inhalte der Rollen qualitätsgerecht ausgefüllt werden.

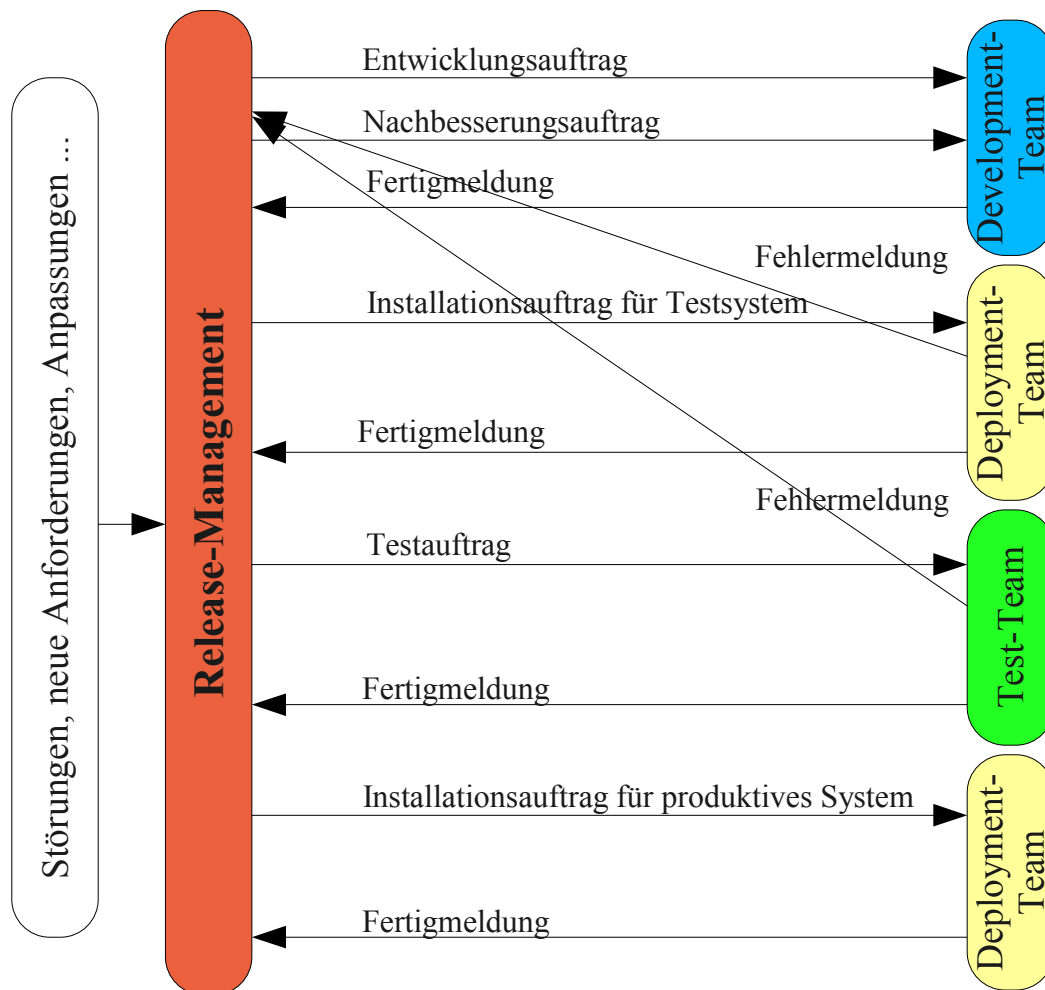


Bild 1: Workflow einer Entwicklung

2.1 Statusübergänge eines geplanten Entwicklungsauftrags

Bei geplanten Entwicklungsaufträgen wird in der Regel das volle Spektrum des Workflows ausgeschöpft.

Die Entwicklung wird komplett vom Releasemanagement gesteuert und auch überwacht. Verlässt der Entwicklungsauftrag ein Team, wird immer eine Fertigmeldung an das Releasemanagement abgegeben.

Der Entwicklungsauftrag ist beendet, sobald diese in einen Stand versetzt wird, welcher für eine Installation auf ein produktives System vorgesehen ist. Eventuelle Nachbesserungen werden dann nicht mehr entgegengenommen.

Sollten Fehler im produktiven Betrieb auftreten oder Änderungen notwendig werden, führen diese zwingend zu einem neuen Entwicklungsauftrag.

Nachfolgendes Bild verdeutlicht das Vorgehen.

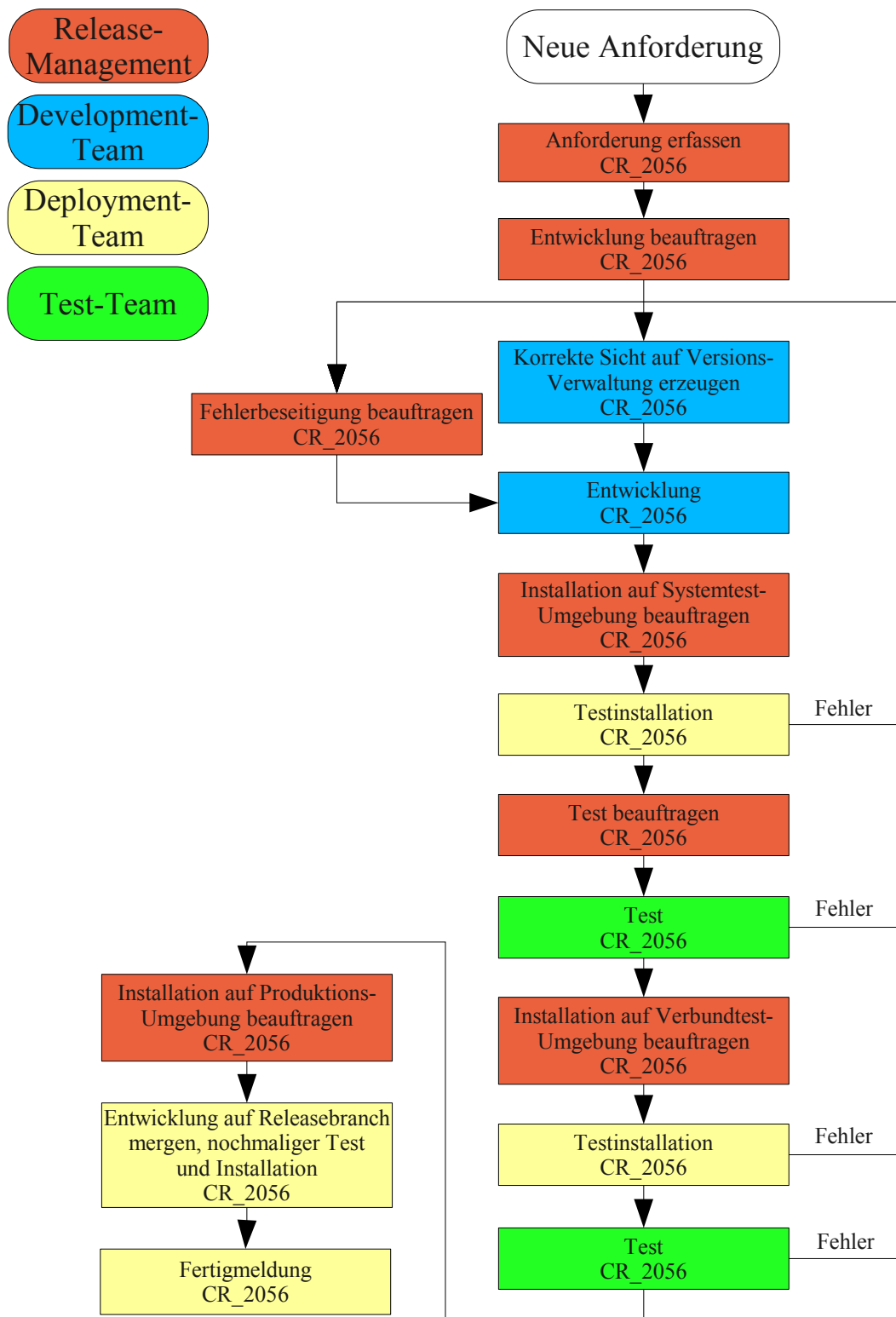


Bild 2: Verlauf einer geplanten Entwicklung

Je nach Größe und Komplexität des produktiven Systems ist es erforderlich verschiedene Tests mit unterschiedlichen Testinhalten auf unterschiedlichen Umgebungen durchzuführen.

Natürlich ist es auch möglich, die Tests für kleine und überschaubare Entwicklungsaufträge

einzuschränken oder im Rahmen einer Notmaßnahmen komplett auf die Tests zu verzichten. Diese Entscheidung liegt aber in jedem Fall in den Händen des Releasemanagements.

Im folgenden werden die einzelnen Schritte näher erläutert.

Anforderung erfassen:

Die Anforderung, welche von außen an das Releasemanagement herangetragen wird, muss als Entwicklungsauftrag eindeutig gekennzeichnet werden.

In der Regel handelt es sich um neue Anforderungen (Change-Request CR), Fehlermeldungen (Problem-Report PR) oder aber besonders schnell durchzuführende Notfallmaßnahmen (Hot-Fixes HOT).

Um eine Eindeutigkeit zu erreichen, werden die Anforderungen nummeriert (CR_2056, HOT_2057). Die Nummerierung sollte aus geeigneten Fehler-Tracking-Tools gewonnen werden.

Entwicklung beauftragen:

Der Entwicklungsauftrag wird mit allen Begleitinformationen und Terminforderungen an des Development-Team über stellt.

Korrekte Sicht auf Versionsverwaltung erzeugen:

Das Entwicklungsteam analysiert die Anforderung und muss in der Regel ein Mapping des Entwicklungsauftrages auf die zu bearbeitenden Projekte und deren Pakete durchführen (näheres dazu siehe weiter unten im Text).

Mit diesen Informationen werden Branch- und Labeltypen sowie eine Sicht (Config-Spec) auf das Versionsverwaltungssystem erzeugt und allen an der Entwicklung beteiligten Entwicklern zur Verfügung gestellt.

Das bedeutet: Jeder Entwicklungsauftrag wird in seinem eigenen Branch durchgeführt.

Dieser setzt auf dem zum Zeitpunkt des Entwicklungsbeginns gültigen Releasestand auf.

Entwicklung:

Was immer nötig ist und anschließend die Fertigmeldung an das Releasemanagement.

Die Fertigstellung ist erst vollzogen, wenn alle geänderten oder neu hinzugekommenen Files/Verzeichnisse gelabelt wurden. Es versteht sich von selbst, dass die Modultests korrekt durchgeführt und dokumentiert wurden.

Die Übergabeschnittstelle ist also das Entwickler-Label.

Testinstallationen beauftragen:

Das Releasemanagement beauftragt das Deployment-Team eine oder mehrere Installationen auf Testsysteme durchzuführen.

Testinstallation durchführen:

Das Deployment-Team muss als erstes den durch die Entwicklung übergebenen Stand sichern. Dieses geschieht durch einfaches Mergen aller Files, welche durch das Entwickler-Label gekennzeichnet sind, auf einen Test- oder Integrations-Branch.

Dieser Test- oder Integrationsbranch dient zur Installation auf die Testsysteme.

Fehler werden an das Releasemanagement kommuniziert, welches eine Nachbesserung der Entwicklung einfordert.

Nach Abschluss der Installation wird ebenfalls das Releasemanagement informiert, welches dann den Test beauftragt.

Test beauftragen:

Das Test-Team wird mit allen notwendigen Informationen versorgt.

Test:

Fehler oder Freigabeempfehlungen gehen wiederum an das Releasemanagement.

Installation auf produktivem System beauftragen:

Das Deployment-Team wird mit der Installation oder Auslieferung beauftragt.

Merge auf Release-Branch, finaler Test und Auslieferung:

Das Deployment-Team wird nach Freigabe des Entwicklungsauftrags alle betroffenen Dateien vom Test- oder Integrationsbranch auf den Releasebranch mergen.

Nach einem finalen Test wird ein neuer Releasestand definiert.

Dieser wird dann Basis neuer Entwicklungen

Anschließend kann die Auslieferung angestoßen werden.

Nach Abschluss aller Arbeiten wird das Releasemanagement wiederum informiert und der Entwicklungsauftrag wird geschlossen.

2.2 Notfallmaßnahmen

Notfälle betreffen einen kritischen Zustand im produktiven System, welche es erforderlich machen, sofort mit einer Entwicklungstätigkeit zu beginnen und diese unter Umständen auch ohne Test in die Produktion zu bringen.

Dieses bedeutet aber keinen Freibrief, am Versionsverwaltungssystem vorbei zu gehen.

Folgendes Bild zeigt den Workflow im Falle eines Hot-Fixes.

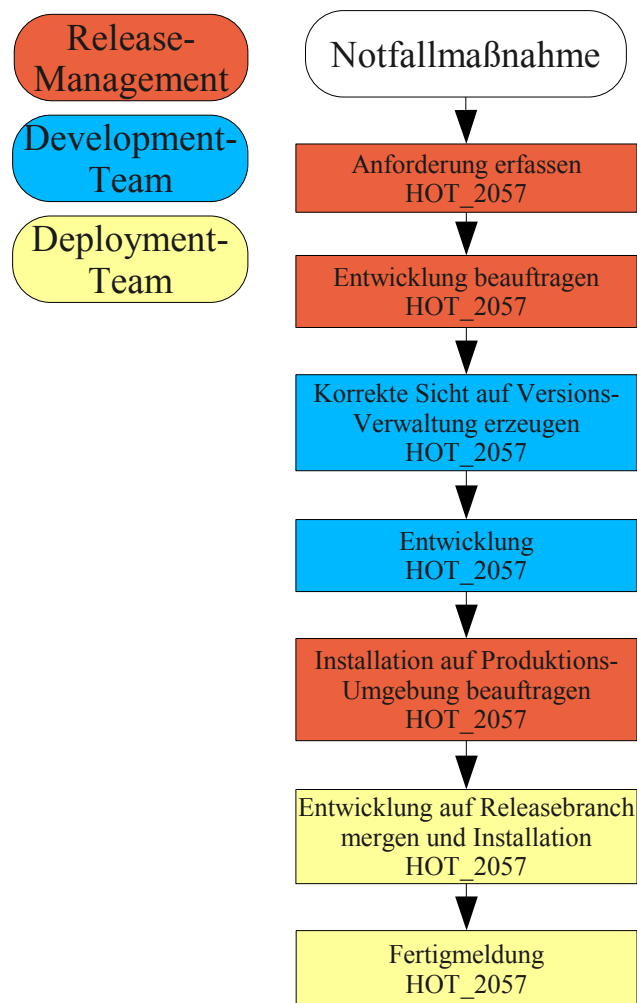


Bild 3: Verlauf einer Notfallmaßnahme

Damit die einzelnen Teams Hand in Hand arbeiten können, ist es von Vorteil, diese durch Tools zu unterstützen, welche die Arbeit mit dem Versionsmanagement-System vereinfachen bzw. automatisieren.

3 Synchronisation zwischen Kunde und Softwarelieferant

Bei der Bereitstellung von Software gibt es zwei grundsätzlich unterschiedliche Verfahren.

Der wohl am besten bekannte Fall ist die Entwicklung und Lieferung von Standard-Software.

Hier werden vom Softwarelieferanten in regelmäßigen Abständen Updates und Patches bereitgestellt und dem Kunden überlassen, ob er sie einspielt.

Wenn sie eingespielt werden, kann es vorkommen, dass sich Funktionalitäten ändern, ohne das der Kunde dieses wünscht.

Rückmeldungen über Fehler in der Software haben nur eine mittelbare Auswirkung auf den Bereitstellungsprozess.

Viel straffer ist aber Bindung zwischen Kunde und Lieferant bei kundenspezifischer Software.

In vielen Großprojekten ist es so, dass Aufträge für die Erstellung von Software an andere Firmen vergeben werden.

Der Lieferant wird also nach Fertigstellung an den Kunden liefern. Der Kunde seinerseits muss zur eigenen Absicherung diese Software einem ausgiebigen Test unterziehen.

Und hier setzen die Schwierigkeiten des Softwareherstellers ein.

Sein Releasestand der Software wird durch Erweiterungen und Fixes von Fehlern einen anderen Stand haben, als der beim Kunde im produktiven System eingesetzte Stand.

Aus Sicht des Softwareherstellers ergibt sich dann z.B. Folgendes Bild:

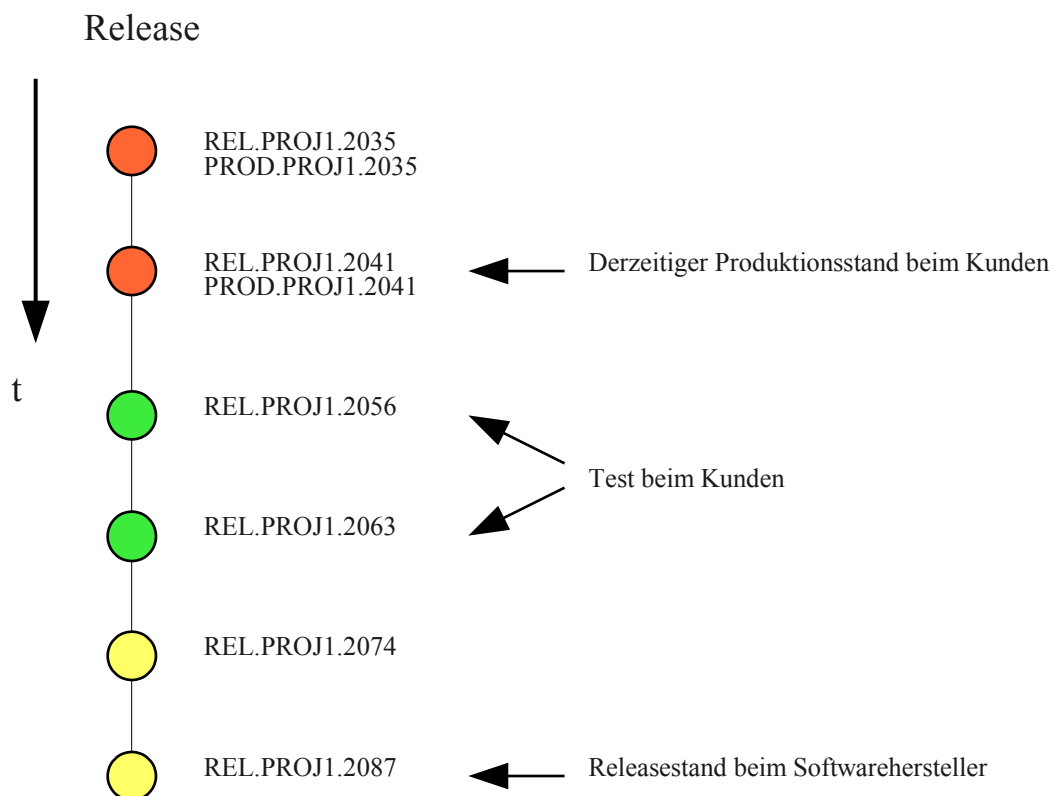


Bild 4: Releasestand aus Sicht des Softwareherstellers

Der Softwarehersteller wird bei einer normalen Entwicklung immer auf seinem aktuellen

Releasestand aufsetzen.

Sollten aber Fehler im produktiven System auftreten, muss schnell reagiert werden und ein Fix auf Basis des Produktionsstandes bereitgestellt werden.

Um diesen auch korrekt zu identifizieren, ist es unbedingt notwendig, dass der Kunde den Softwarehersteller über Inbetriebnahmen von getesteten Releases informiert.

Dieser muss dann innerhalb seiner Versionsverwaltung den entsprechenden Releasestand als produktiv kennzeichnen.

4 Sicherung der Qualität innerhalb der Versionsverwaltung

Für die qualitätsgerechte Arbeit innerhalb der Versionsverwaltung gilt es Regeln einzuhalten:

- Jeder Entwicklung zu einem Projekt erfolgt in einem eigenen Branch
- Entwicklungsbranches müssen zu einem Test zusammengefasst werden können. Diese Zusammenfassung hat wiederum einen gesonderten Branch
- Zurückführung von Entwicklungen nach ausgiebigen Tests erst unmittelbar vor Auslieferung an den Kunden.
- Versionierung und eindeutige Kennzeichnung der Config-Specs.
- Abbildung des aktuellen Releasestandes des Projektes.
- Abbildung des Produktionsstandes des Projektes.
- Abbildung des aktuellen Standes einer Testumgebung.

Um diese Regeln einhalten zu können bedarf einer klaren Absprache zwischen Lieferant und Kunde.

Termine und Inhalte von Lieferungen sind genau zu definieren.

4.1 Verwalten von Config-Specs

Die Config-Specs bei ClearCase bestimmen die Sicht auf die Versionsverwaltung. Ihnen kommt eine zentrale Bedeutung bei der Qualitätssicherung zu.

Sie müssen sicher verwaltet werden, damit eine Sicht auf die Versionsverwaltung jederzeit wieder eingestellt werden kann.

Jedes Projekt wird durch eine eigene Config-Spec dargestellt.

Folgende Anforderungen sind zu erfüllen:

- Ablage der Config-Specs an einer für alle Benutzer der Versionsverwaltung erreichbaren Stelle
- Versionierung und eindeutige Kennzeichnung der Config-Specs.

Im folgenden Beispiel ist ein Vorschlag für die Verwaltung von Config-Specs für drei unterschiedliche Projekte dargestellt.

Die Identifikation der Config-Specs erfolgt über Label. Jede Version der Config-Spec ist eindeutig gekennzeichnet und über Regel der Benennung Release-, Produktions- oder CR/PR-Config-Specs zuordenbar.

Durch Hyperlinks sind die Ursprünge der einzelnen Versionen kenntlich gemacht.

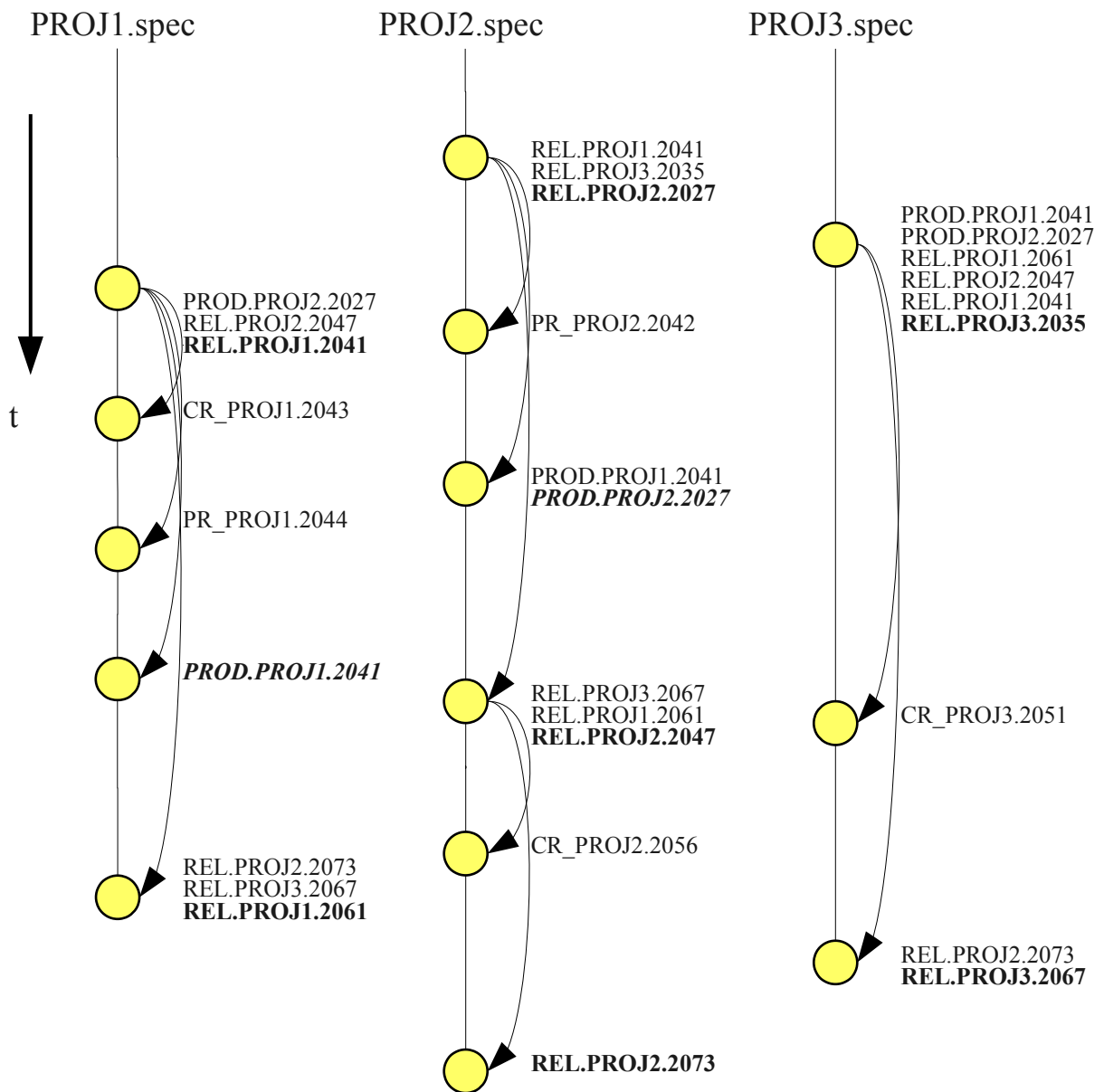


Bild 5: Zeitlicher Verlauf der Abbildung von Config-Specs

Die Label der jeweiligen Releasestände (fett dargestellt) und die Label der Produktionsstände (fett+kursiv dargestellt) werden zusätzlich auf die Release- bzw. Produktionsstände der anderen Projekte gelegt.

Dadurch ist es möglich den aktuellen Stand der Software über alle Projekte zu identifizieren.

Die Versionen zwischen Release- und dazugehörigen Produktionsständen sehen auf den ersten Blick unterschiedlich aus, sind es aber nicht. Die Inhalte dieser Versionen sind völlig identisch.

5 Unterstützende Werkzeuge für Versionsverwaltungen

Im Rahmen eines konkreten Projektes wurde eine Lösung für Base ClearCase auf Basis von UNIX mittels Shell- und Perl-Scripts erarbeitet.

Es wurde von Anfang an Wert auf eine einfache Konfigurierbarkeit gelegt.

Während der Implementation wurden Schnittstellen zu verschiedenen Versionsverwaltungen vorbereitet.

Es ist also möglich, durch einfache Konfiguration andere Versionsverwaltungen (z.B. CVS⁵, Subversion⁶) zu nutzen oder diese parallel zu ClearCase zu betreiben.

Ziel der Entwicklung war es, den Nutzern von ClearCase oder auch anderen Versionsverwaltungen die Arbeit mit einem minimalen Satz von Kommandos zu erleichtern.

Weiterhin muss sichergestellt werden, dass unterschiedliche Projekte unter Umständen nichts voneinander sehen dürfen. Dazu wurde ein Sicherheitskonzept auf Basis der UNIX-Gruppenrechte implementiert.

Alle nachfolgend beschriebenen Scripte haben generell folgende Aufrufoptionen:

-h	Ausgabe der Usage
-V	Ausgabe der Version
-x <1-9>	Debug-Level

Diese Optionen werden bei der Beschreibung der Tools nicht mehr mit angegeben.

Alle Scripte generieren ein Logfile der Form <username>_<script_name>_<datum>.log. Die Abspeicherung der Logfiles wird über die Umgebungsvariable LOGS gesteuert und ist per default auf \$HOME/logs eingestellt.

Der Debug-Level ist per default auf 1 eingestellt (Ausgabe von Start- / Stop-Meldungen sowie von Warnungen) und kann über die Umgebungsvariable DEBUG gesteuert werden. Die Option '**-x level**' überschreibt diese Einstellung.

Fehlermeldungen werden in jedem Fall protokolliert. Unabhängig vom eingestellten Debug-Level werden bis zu 500 Log-Zeilen vor dem aufgetretenen Fehler im Logfile ausgegeben. Somit ist der Programmfluss eindeutig nachvollziehbar und eine komfortable Fehlersuche möglich.

5.1 Konfiguration eines Projektes in der Versionsverwaltung

Die zur Arbeit notwendige Software ist im Verzeichnis /opt/EWU angesiedelt.

Eine Installation von Perl mit allen notwendigen Modulen wird im Verzeichnis /opt/EWU/perl bereitgestellt.

Die Tools befinden sich im Verzeichnis /opt/EWU/bin. Da die Perl- und Shellscripte auf den unterschiedlichsten UNIX-Derivaten lauffähig sind, bot es sich an, diese im ClearCase zu belassen. Sie werden also nicht gesondert installiert. Vielmehr wird ein View erzeugt gestartet dessen Config-Spec so eingestellt ist, dass immer die aktuelle Version der EWU-Tools zu sehen ist. Dieser View wird beim Systemstart im Zuge des Hochfahrens von ClearCase gestartet. Im Verzeichnis '/opt/EWU' existiert ein Softlink 'bin' welcher auf das richtige Verzeichnis im View

5 CVS ist ein Open Source Projekt

6 Subversion ist ein Open Source Projekt als Weiterentwicklung von CVS

zeigt.

Natürlich kann hier auch jedes andere geeignete Verzeichnis bzw. andere Perl-Installation genutzt werden.

Die Konfiguration der Projekte erfolgt über eine zentrale Steuerdatei:

```
/opt/EWU/bin/ewu.conf
```

Das folgende Beispiel zeigt den Aufbau diese Datei:

```
#####  
<COMMON>  
  
# Konfigurierte Projekte (Komma und/oder Space separiert)  
PROJEKTE           =    MDSCORE, IBMD, ADM, EWU  
  
# User, der die Tools 'integrate', 'release' und 'show_integration' ausfuehren darf  
INTEGRATION_USER   =    ccadm  
  
# eingesetzte Versionsverwaltungssysteme (Komma und/oder Space separiert)  
# Bsp.: VERSION_SYSTEMS = CC, CVS RCS  
VERSION_SYSTEMS    =    CC  
  
# Config-Spec-Verzeichnis und Versionssystem, in dem die Config-Specs gehalten werden  
CONFIG_SPEC_VOB     =    adm  
CONFIG_SPEC_DIR     =    ConfigSpecs  
CONFIG_SPEC_SYSTEM  =    CC  
  
# Verzeichnis in $HOME, in dem die Deployment_Specs bearbeitet werden  
CONFIG_SPEC_WORK_DIR =    .specs  
  
</COMMON>  
  
#####
```

Die Sektion COMMON beschreibt das allgemeine Verhalten der Tools.

In den folgenden Sektionen sind die Informationen zu den eingesetzten Versionsverwaltungen enthalten.

```
# Versionsverwaltungssystem  
# (ClearCase = CC, CVS = CVS, RCS = RCS)  
  
<CC>  
  
# Vobs  
VOB_HOME           =    /vobs  
  
# Views  
VIEW_HOME          =    /view  
VIEW_STORAGE       =    /ccexports_new/viewstrg  
SUPPORT_VIEW       =    support
```



```

# Templates fuer Windows-Profiles
# (Verzeichnis unterhalb von CONFIG_SPEC_VOB/CONFIG_SPEC_DIR)
WIN_TEMPLATE_DIR      =      .profile_templates

# Verzeichnis fuer Windows-Profiles
WIN_PROFILES          =      /net/clhost1/clearcase/win_profiles

</CC>

<CVS>

# Working-directory
VIEW_HOME              =      /home

</CVS>

```

```
#####
```

Für die eigentlichen Projekte sind die Informationen in den Projekt-Sektionen maßgeblich. Für jedes in der Sektion COMMON definiertes Projekt muss eine eigene Sektion angelegt werden.

```
#####
```

```

# PROJEKT-BESCHREIBUNGEN
#
# Benoetigte Informationen:
# ANWENDUNG, SYSTEM, WORK_BRANCH_EXPR, TEST_BRANCH_EXPR,
# PROD_LABEL_EXPR, RELEASE_LABEL_EXPR, REL_BRANCH, APP_HOME, APP_CFG,
# APP_GROUP, VOBS
#
# ANWENDUNG:                Name des Projektes
#
# SYSTEM:                   Versionsverwaltungssystem
#                           (ClearCase = CC, CVS = CVS, RCS = RCS)
#                           (notwendig)
#
# WORK_BRANCH_EXPR:         Branch- und Labeldefinitionen fuer CR's und PR's
#                           (notwendig)
#
# TEST_BRANCH_EXPR:         Branch- und Labeldefinitionen fuer Integrationen
#                           von CR's und PR's (notwendig)
#
# PROD_LABEL_EXPR:          Labeldefinition zur Kennzeichnung eines
#                           produktiven Standes (notwendig)
#
# RELEASE_LABEL_EXPR:       Labeldefinition zur Kennzeichnung des
#                           aktuellen Release-Standes (notwendig)
#
# REL_BRANCH:                Branchname, auf dem das produktive
#                           System gehalten wird (notwendig)
#
# APP_HOME:                 Home-Verzeichnis fuer das Projekt
#                           Wird fuer die Arbeit mit der EWU erweitert
#                           (notwendig)
#

```

```

#
# APP_CFG:           Verzeichnis+Datei fuer allgemeine
#                   Konfigurationen
#                   Diese Datei wird nach dem Setzen des Views
#                   eingesourced (optional)
#
# APP_GROUP:        Zu setzende Unix-Gruppe, um in diesem Projekt zu
#                   arbeiten (notwendig)
#
# VOBS:             Fuer das Projekt verwendete Repositories
#                   Es koennen mehrere Repositories angegeben werden
#                   (getrennt durch whitespace und/oder Komma)
#                   (notwendig)
#
# Fuer CVS-Anwendungen werden zusaetzlich benoetigt:
# CVS_ROOT, CVS_SERVER, CVS_ACCESS
#
# CVSROOT:          Repository-Wurzel
#
# CVS_SERVER:       Kann leer bleiben; Ansonsten der Server, auf dem
#                   sich das Repository befindet.
#
# CVS_ACCESS:       Zugriffsmethode auf das Repository
#                   (ext | pserver):
#                   Zugriff via pserver nur in Ausnahmefaelen
#
# CVS_SERVER        leer: direkter Zugriff auf locale Maschine
#                   ext: Zugriff via ssh.
#                   (Jeder User sollte sich ein Schluesselpaar ohne
#                   Password erzeugen)
#
#####

```

Hier noch ein konkretes Beispiel für das Projekt EWU.

```

# Projekt EWU
<ewu>
  ANWENDUNG           =   EWU
  SYSTEM              =   CC
  REL_BRANCH          =   main
  APP_HOME            =   adm/EWU
  APP_CFG             =   ewu.env
  APP_GROUP           =   ccac6
  VOBS                =   adm
  WORK_BRANCH_EXPR    =   (^cr-ewu-\d{4}-\d{3}$)|(^pr_ewu_\d{4,5}$)
  TEST_BRANCH_EXPR    =   ^int_ewu_\d{3}$
  PROD_LABEL_EXPR     =   ^PROD_EWU_
  RELEASE_LABEL_EXPR  =   ^REL_EWU_
</ewu>

```

5.2 Tools zum Bearbeiten der Config-Specs

Da die Sicht in der Versionsverwaltung (ClearCase) über eine Config-Spec gesteuert wird, ist der Verwaltung dieser ein besonderes Augenmerk zu schenken.

In den nachfolgende Beispielen sind bereits Funktionalitäten dargestellt bzw. Tools

angesprochen, welche erst weiter unten im Text beschrieben werden.

Die Config-Specs aller Projekte sind an zentrale Stelle abgelegt (konfiguriert in der Datei **ewu.conf**).

Tools, die den Umgang mit den Config-Specs ermöglichen, müssen folgendes realisieren:

- Ermitteln des aktuellen Release- bzw. Produktionsstandes eines Projektes bzw. über alle Projekte.
- Editieren der Config-Specs für ein Projekt:
 1. Ermitteln des korrekten Ausgangszustandes und aller dazugehörenden Includes
 2. Kopieren der Dateien in ein Verzeichnis, in dem ein Editieren möglich ist
 3. Mitführen notwendiger Informationen
- Sichern und Labeln der editierten Config-Spec für ein Projekt:
 1. Check ob sich relevante Änderungen ergeben haben
 2. Syntax-Check der neuen Config-Spec
 3. Sichern und Labeln der Dateien
- komfortables Erzeugen neuer Projekte

Der Speicherort und das System unter dem die Config-Specs verwaltet werden, ist in der Datei **ewu.conf** konfiguriert.

Die Bearbeitung der Config-Specs ist nur Mitglieder des Deployment-Teams erlaubt.

Da diese Tools eine gewisse Sonderstellung haben, werden sie auch unter den Rechten eines gesondert in der Datei **ewu.conf** definierten Integration-Users ausgeführt

Mitglieder des Deployment-Teams loggen sich mittels

login2merge

als Integration-User ein.

Hinter diesem Aufruf verbirgt sich ein ssh-Login als Integration-User.

Da das Password des Integration-Users nicht bekannt werden soll, muss für jeden berechtigten User ein Key hinterlegt werden. Die Schlüssel können ohne Password generiert werden. Damit entfällt die Password-Abfrage beim Login.

Verantwortlich für das Freischalten der Mitglieder für den Integration-User sind die Systemverwalter.

5.2.1 Editieren von Config-Specs

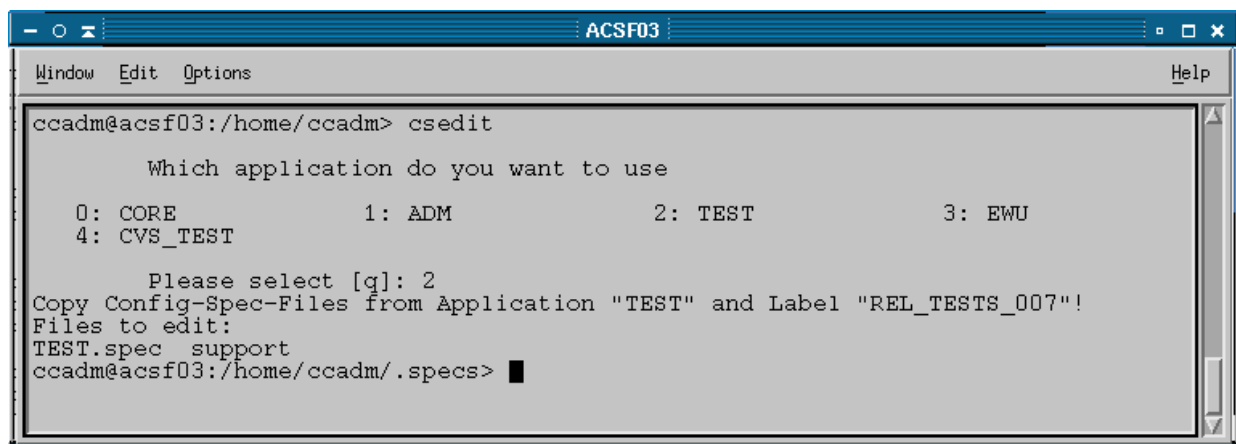
Neue Config-Specs werden für das jeweilige Projekt erzeugt.

Der Aufruf:

```
csedit [-a application] [-l label]  
-a application      Application, Config-Spec  
                     shall be edited for.  
-l label           Base-label  
                     (default last release)
```

ermittelt alle zum editieren notwendigen Informationen.

Beispiel:



```
ccadm@acsf03:/home/ccadm> csedit

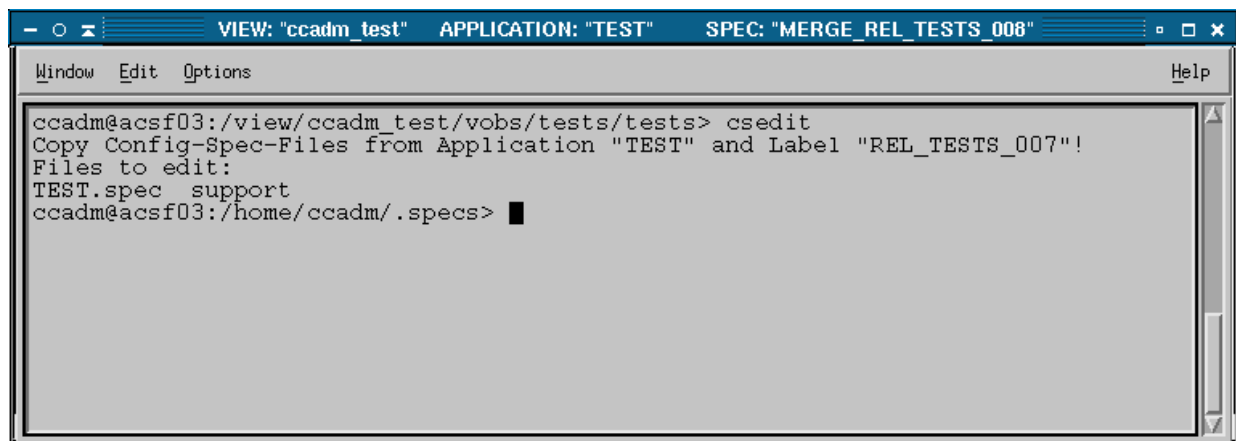
Which application do you want to use

0: CORE          1: ADM          2: TEST          3: EWU
4: CVS_TEST

Please select [q]: 2
Copy Config-Spec-Files from Application "TEST" and Label "REL_TESTS_007"!
Files to edit:
TEST.spec support
ccadm@acsf03:/home/ccadm/.specs>
```

Bild 6: Ermitteln der Informationen zum Editieren der Config-Spec

Im obigen Beispiel erfolgt der Aufruf ohne einen konfigurierten View zu setzen (siehe weiter unten: mkview/setview). Da aus diesem Grunde das Projekt nicht bekannt ist, wird es abgefragt. Ist ein View gesetzt, werden die Informationen des konfigurierten Projektes ermittelt:



```
VIEW: "ccadm_test" APPLICATION: "TEST" SPEC: "MERGE_REL_TESTS_008"
ccadm@acsf03:/view/ccadm_test/vobs/tests/tests> csedit

Copy Config-Spec-Files from Application "TEST" and Label "REL_TESTS_007"!
Files to edit:
TEST.spec support
ccadm@acsf03:/home/ccadm/.specs>
```

Bild 7: Ermitteln der Informationen zum Editieren der Config-Spec bei gesetztem View

Nach dem Ermitteln der Config-Spec-Files werden diese in das im File **ewu.conf** konfigurierte Verzeichnis kopiert und die Shell wechselt in dieses Verzeichnis.

Die Datei **support** hat eine besondere Funktion. Sie wird in alle Config-Specs inkludiert und sorgt dafür, dass aus allen Projekten heraus auf die gespeicherten Config-Specs zugegriffen werden kann. Diese Datei darf nicht editiert werden! Sollten Änderungen an dieser Datei notwendig sein, so müssen diese direkt im Vob gemacht werden.

Die Config-Spec für das Projekt 'TEST' ist recht einfach aufgebaut und hat z.B. folgenden Aufbau:

```
# allgemeines Paket zur Sicht auf die Config-Specs (do not edit!)
include support

# Paket TEST
# <test>
element /vobs/tests/...          REL_TESTS_007 -nocheckout
# </test>
```

Sie wird dahingehend editiert, dass im Vob 'tests' das neue Release, welches durch das Label REL_TESTS_008 dargestellt wird, sichtbar wird.

Das bedeutet also, REL_TESTS_007 gegen REL_TESTS_008 austauschen.

Beim Erzeugen von Config-Specs ist darauf zu achten, dass die Pfade des Vobs voll qualifiziert sind.

Dieses trägt zur Übersichtlichkeit bei und ermöglicht es auch, diese Config-Spec als Basis für ein Windows-Profil zu nutzen.

Windows-Profile für den ClearCase-Windows-Client sind hier nicht Bestandteil der Betrachtung. Es wird aber im folgenden ein Script zur Erzeugung dieser Profiles vorgestellt.

5.2.2 Sichern von Config-Specs

Nachdem die Config-Spec editiert wurde, wird sie mit dem Aufruf:

```
cssave [-l label]
-l label                Identification of new spec
```

gesichert.

Es werden vor dem Sichern folgende Tests durchgeführt:

- Falls es sich bei der Config-Spec um eine Release- oder Produktions-Spec handelt, wird geprüft, ob in der Zeit zwischen dem Aufruf **csedit** und **cssave** bereits eine neue Config-Spec eingestellt wurde. In diesem Fall, beendet sich das Tool mit einer Fehlermeldung
- Syntaxüberprüfung der Config-Spec. Es wird nur geprüft, ob ClearCase die neue Config-Spec syntaktisch akzeptiert.

Es gibt verschiedene Arten Config-Spec-Erstellung.

1. Config-Specs für die Entwicklungstätigkeit
Dieses soll hier nicht Gegenstand sein, da solche Specs von den weiter unten aufgeführten Tools automatisch erzeugt werden
2. Config-Specs für Releasestände
3. Config-Specs für Produktionsstände

5.2.2.1 Config-Specs für Releasestände

Dieses ist die eigentliche Bestimmung des hier beschriebenen Kommandos

Beispiel:

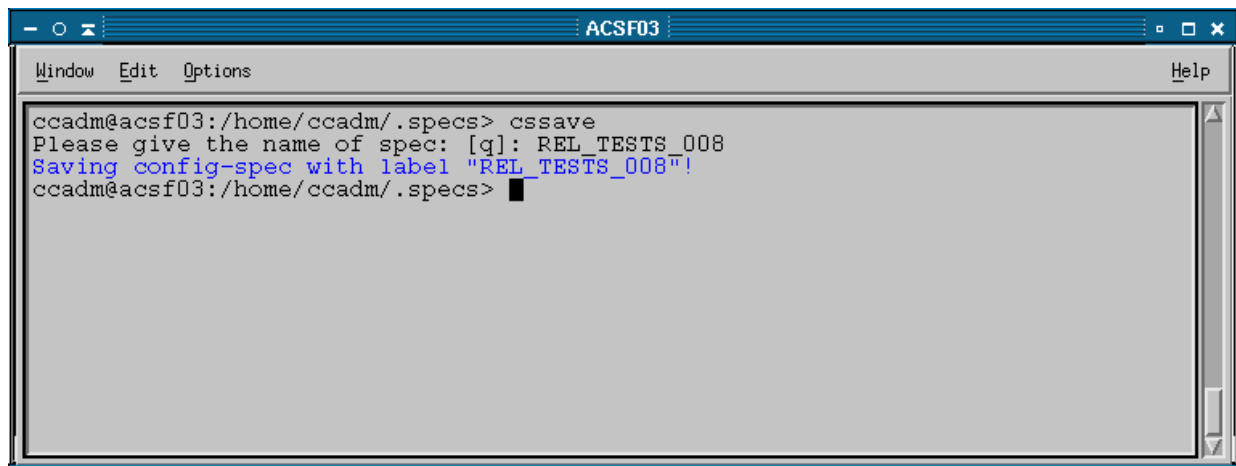


Bild 8: Sichern der Config-Spec


Natürlich ist es nicht zwingend notwendig, die Config-Specs so zu benennen wie die Label, welche das Release im Vob kennzeichnen. Es erleichtert aber die Zuordnung untereinander ungemein.

Die neue Config-Spec hat jetzt folgenden Aufbau:

```
# allgemeines Paket zur Sicht auf die Config-Specs (do not edit!)
include support

# Paket TEST
# <test>
element /vobs/tests/...          REL_TESTS_008 -nocheckout
# </test>
```

Beim Setzen der Config-Spec über die weiter unten beschriebenen Tools werden die Includes aufgelöst, so dass mit dem Befehl **cleartool catcs** die Config-Spec mit allen ihren Elementen angezeigt wird:



```
frank@acsf03:/view/frank_test/vobs/tests/tests> cleartool catcs

# SPEC-LABEL: REL_TESTS_008
# APP_ANW: TEST

# allgemeines Paket zur Sicht auf die Config-Specs (do not edit!)
# <support> BEGIN INCLUDE
element /vobs/adm/ConfigSpecs/... /main/LATEST
element -directory /vobs/adm /main/LATEST
# </support> END INCLUDE

# Paket TEST
# <test>
element /vobs/tests/... REL_TESTS_008 -nocheckout
# </test>

frank@acsf03:/view/frank_test/vobs/tests/tests>
```

Bild 9: Listing der Config-Spec über cleartool-Kommando

5.2.2.2 Config-Specs für Produktionsstände

Wie bereits oben beschrieben kann es eine Diskrepanz zwischen Softwarezulieferer und Kunde bezüglich der Softwarestände haben.

Es ist zwingend notwendig, wenn ein Kunde ein Release in die Produktion übernimmt, dieses beim Zulieferer als solches zu kennzeichnen.

Beispiel:

Der Kunde übernimmt das Release REL_TESTS_007 in die Produktion und kommuniziert dieses an Den Zulieferer.

Für diesen besteht folgender Handlungsbedarf:

- Aufruf von **csedit -I REL_TESTS_007**
- Aufruf von **cssave -I PROD_TESTS_007**

Damit ist dieses Release als neuer Produktionsstand gekennzeichnet und wird Basis eventuell notwendiger Hotfixes.

5.2.3 Erzeugen neuer Projekte

Natürlich muss es auch möglich sein, neue Projekte schnell und komfortabel einzurichten.

Dazu müssen allerdings einige Vorarbeiten geleistet werden:

- Eventuell Anlegen eines neuen Vobs für das Projekt
- Initiales Einrichten der Verzeichnisstruktur für dieses Projekt
- Labeln aller Verzeichnisse mit einem Start-Label
- Editieren der Datei **ewu.conf** (Einführen des neuen Projektes und definieren aller geforderten Angaben (siehe oben))

Jetzt kann mittels `csedit` eine gültige Config-Spec editierbar gemacht werden. Es ist völlig belanglos, welches Projekt dazu benutzt wird, Man sollte aber eines nutzen, das in etwa den neuen Gegebenheiten entspricht.

Folgende Schritte sind auszuführen:

- Aufruf von **csedit** für ein beliebiges Projekt
- Umbenennen der Datei `<PROJ>.spec` nach `<NEU_PROJ>.spec` (der Name der neuen Spec-Datei muss dem Namen des neuen Projektes entsprechen. z.B LIB.spec)
- Editieren der Spec-Datei
- Aufruf von **cssave** und Speicherung der Spec mit einem geeigneten Releaselabel

Damit ist das neue Projekt nutzbar

5.2.4 Weitere Hinweise zur Gestaltung von Config-Specs

5.2.4.1 Nutzung gemeinsamer Module

Bei parallel bearbeiteten Projekten ist es durchaus möglich, das mehrere Projekte auf Source-Code zugreift, der allgemein verwendbar ist.

Dieses könnten zum Beispiel Library- und Shell-Funktionen oder auch Perl-Module sein.

Es ist sinnvoll diesen Code in ein gesondertes Projekt zu legen und dort auch zu pflegen.

Um den anderen Projekten den Zugriff darauf zu geben, ist folgendermaßen vorzugehen:

- Projekt für die gemeinsamen Module wie oben beschrieben anlegen
- Wechsel in das Verzeichnis, in dem Config-Specs abgelegt werden (siehe **ewu.conf**). Dazu wird ein ClearCase-View mit einer Default-Config-Spec benutzt.
- Auschecken der Datei **'lock'** (damit wird verhindert, das während der Arbeiten neue Config-Specs über die Tools eingestellt werden können)
- Auschecken des Verzeichnisses
- Anlegen eines ClearCase-Links wie folgt:
z.B.: **cleartool ln LIB.spec lib**
- Verzeichnis wieder einchecken und uncheckout auf die Datei **'lock'**

Ab jetzt können in jedes Projekt die gemeinsamen Module einbezogen werden.

Beispiel:

- Aufruf von **csedit** für das Projekt 'TEST'
- Editieren der TEST.spec-Datei:

```
# allgemeines Paket zur Sicht auf die Config-Specs (do not edit!)
include support
```

```
# Library-Sourcen
include lib
```

```
# Paket TEST
# <test>
element /vobs/tests/...
# </test>
```

```
REL_TESTS_008 -nocheckout
```


- Aufruf von **cssave** und Speicherung der Spec mit einem geeigneten Releaselabel

Die Datei '**lib**' muss nicht im Editier-Verzeichnis vorhanden sein. Sie wird vom Tool **cssave** in der richtigen Version ermittelt und kopiert.

5.2.4.2 Paketierung der Projekte

Wenn Projekte einen gewissen Umfang überschreiten, kann es erforderlich sein, dieses in Pakete aufzuteilen.

Damit erreicht man ein gewisses Maß an Sicherheit in der Form, dass für die Bearbeitung eines PR/CR nur innerhalb des geforderten Paketes ein Auschecken der Sourcen erlaubt wird.

Um diese Funktionalität zu nutzen, muss die Release-Config-Spec ein anderes Aussehen haben und in der Datei **ewu.conf** muss der Wert des Tokens **WORK_BRANCH_EXPR** nach bestimmten Regel definiert werden.

Folgendes Beispiel soll den Sachverhalt näher erläutern:

Im Projekt TEST sollen zum Beispiel die Pakete LIB, DB und ETC eingeführt werden.

Diese Pakete entsprechen auch idealerweise auch unterschiedlichen Verzeichnissen.

Die Release-Config-Spec wird jetzt folgendermassen erstellt:

```
# SPEC-LABEL:          PROD_TESTS_008
# APP_ANW:             TEST

# allgemeines Paket zur Sicht auf die Config-Specs (do not edit!)
include support

# Paket LIB
# <lib>
element                /vobs/tests/lib_src/...    REL_TESTS_008 -nocheckout
# </lib>

# Paket DB
# <db>
element                /vobs/tests/db_src/...    REL_TESTS_008 -nocheckout
# </db>

# Paket ETC
# <etc>
element                /vobs/tests/etc/...      REL_TESTS_008 -nocheckout
# </etc>

# Paket TEST
# <test>
element                /vobs/tests/...         REL_TESTS_008 -nocheckout
# </test>
```

In der Datei ewu.conf muss nun noch die Regulare Expression in der Sektion TEST angepasst werden. Die Definition der PR's bzw. CR's muss dem Muster **XXX.YYY.ZZZ** entsprechen:

```
WORK_BRANCH_EXPR = (^cr-ewu\.[a-z]{2-4}\.d{4}-d{3}$)|(^pr_ewu\.[a-z]{2-4}\.d{4,5}$)
```

Eine Config-Spec für den cr-ewu.lib.2006-003 würde dann in etwa wie folgt aussehen:

```

.....
# allgemeines Paket zur Sicht auf die Config-Specs (do not edit!)
include support

# Paket LIB
# <lib>
element          /vobs/tests/lib_src/...  .../cr-ewu.lib.2006-003/LATEST
element          /vobs/tests/lib_src/...  REL_TESTS_008  -mkbranch cr-ewu.lib.2006-003
element          /vobs/tests/lib_src/...  /main/0        -mkbranch cr-ewu.lib.2006-003
# </lib>

# Paket DB
# <db>
element          /vobs/tests/db_src/...  REL_TESTS_008  -nocheckout
# </db>

# Paket ETC
# <etc>
element          /vobs/tests/etc/...    REL_TESTS_008  -nocheckout
# </etc>

# Paket TEST
# <test>
element          /vobs/tests/...        REL_TESTS_008  -nocheckout
# </test>

```

Mit dieser Config-Spec ist jetzt nur noch eine Bearbeitung im Verzeichnis .../lib_src möglich. Ist keine eindeutige Zuordnung des zweiten Tokens möglich (YYY), wird die die die Spec folgendes Aussehen annehmen:

```

.....
# allgemeines Paket zur Sicht auf die Config-Specs (do not edit!)
include support

# Paket LIB
# <lib>
element          /vobs/tests/lib_src/...  REL_TESTS_008  -nocheckout
# </lib>

# Paket DB
# <db>
element          /vobs/tests/db_src/...  REL_TESTS_008  -nocheckout
# </db>

# Paket ETC
# <etc>
element          /vobs/tests/etc/...    REL_TESTS_008  -nocheckout
# </etc>

# Paket TEST
# <test>
element          /vobs/tests/...        .../cr-ewu.bla.2006-003/LATEST
element          /vobs/tests/...        REL_TESTS_008  -mkbranch cr-ewu.bla.2006-003
element          /vobs/tests/...        /main/0        -mkbranch cr-ewu.bla.2006-003
# </test>

```

Natürlich ist es auch möglich, mit solchen Config-Specs alle Pakete gleichzeitig zu bearbeiten. Die der automatischen Erzeugung darf allerdings jedes Paket nur einmal aufgeführt werden und

es darf auch nur maximal ein unbekannter Token übergeben werden!

Die unten beschriebenen Tools unterstützen diese Mehrfachbranches nicht, sind aber jederzeit darauf anpassbar.

5.2.5 Erzeugung eines Windows-Profiles für ClearCase-Clients

Obwohl sich die hier vorgestellten Verfahren ausschließlich auf Base-ClearCase unter Unix beziehen, ist es doch möglich Config-Specs als Windows-Profile für den Windows-ClearCase-Client zur Verfügung zu stellen.

Voraussetzung dafür ist, dass auch diese Profiles an einer definierten Stelle abgelegt werden.

Um dann den Windows-Clients den Zugriff zu gewähren, muss dieses Verzeichnis via Samba⁷ freigegeben werden.

Ein neues Profile wird mit dem Kommando:

```
mk_ms_profile [-l spec]  
-l spec                    Config-spec-label
```

angelegt.

Jedes Projekt hat für die Profiles ein eigenes Unterverzeichnis. Die Verzeichnisse werden ebenfalls in der Datei **ewu.conf** konfiguriert.

5.3 Sicherstellung des Zugriffs auf die Versionsverwaltung

5.3.1 Erzeugung einer Sicht auf die Versionsverwaltung

ClearCase stellt für die Sicht auf das Repository einen View bereit. Dieser View stellt nichts weiter als einen Container dar, welcher die mittels einer Config-Spec ausgewählten Verzeichnisse und Files zur Verfügung stellt.

Ein Tool zur Erzeugung eines Views muss folgende Funktionalität bereitstellen:

- Abfrage des Namens, falls dieser nicht angegeben wurde.
- Der Name des Views muss mit dem Username des Benutzers erweitert werden, um eine Eindeutigkeit innerhalb des Unix-Systems zu gewährleisten.
- Die Lage des View-Storage-Verzeichnisses wird vor dem User verborgen.
- Der View wird sofort gestartet, und für ein Projekt konfiguriert. Stehen mehrere Projekte zur Auswahl, erfolgt eine interaktive Abfrage.
- Ist es nötig, in eine andere Gruppe zu wechseln, wird dieses veranlasst.
- Es wird in das für das ausgewählte Projekt konfigurierte Arbeitsverzeichnis gewechselt und -falls konfiguriert- Einstellungen des Projekts gesetzt (z.B. setzen von PATH, LD_LIBRARY_PATH etc.).
- Die eingestellte Sicht auf die Versionsverwaltung (Config-Spec) entspricht dem aktuellen Releasestand des Projekts.

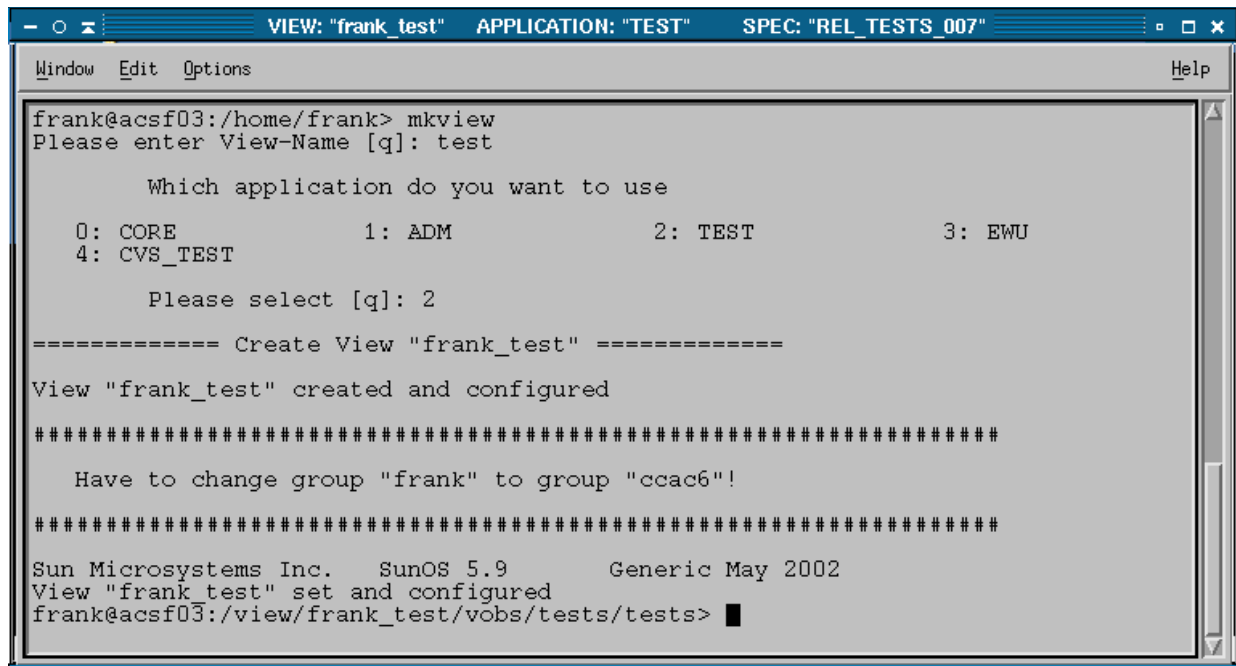
⁷ Samba ist ein Open-Source Projekt zur Integration von Unix und Windows im Netzwerk

Ein neuer View wird mit dem Kommando:

```
mkview [view_name]
```

angelegt.

Beispiel:



```
frank@acsf03:/home/frank> mkview
Please enter View-Name [q]: test

Which application do you want to use

0: CORE          1: ADM          2: TEST          3: EWU
4: CVS_TEST

Please select [q]: 2

===== Create View "frank_test" =====
View "frank_test" created and configured
#####
Have to change group "frank" to group "ccac6"!
#####
Sun Microsystems Inc. SunOS 5.9 Generic May 2002
View "frank_test" set and configured
frank@acsf03:/view/frank_test/vobs/tests/tests>
```

Bild 10: Erzeugen eines Views

Im obigen Beispiel wird der View für das Projekt 'TEST' konfiguriert.

Die Config-Spec wird auf die aktuelle Releasesicht eingestellt (diese Config-Spec lässt keinen Checkout von Files zu).

Anschließend wird ggf. die Umgebung auf die Belange des Projekts eingestellt und in das Arbeitsverzeichnis dieses Projekts gewechselt.

Da für das angegebene Projekt eine Gruppe definiert ist (siehe ewu.conf), welche nicht der aktuellen Gruppe des Users entspricht, wird die Gruppe gewechselt und der View erneut gesetzt.

5.3.2 View auf ein neues Projekt konfigurieren

Natürlich legitim für jedes Projekt einen eigenen View zu erzeugen. Es muss allerdings auch möglich sein, in einem vorhandenen View leicht von einem Projekt auf ein anderes wechseln zu können.

Ein Tool zur Konfiguration eines Views muss folgende Funktionalität bereitstellen:

- Auflisten aller eventuell im View ausgecheckten Elemente und ggf. Abbruch der Aktion.
- Es wird ggf. in die für das ausgewählte Projekt konfigurierte Gruppe gewechselt.

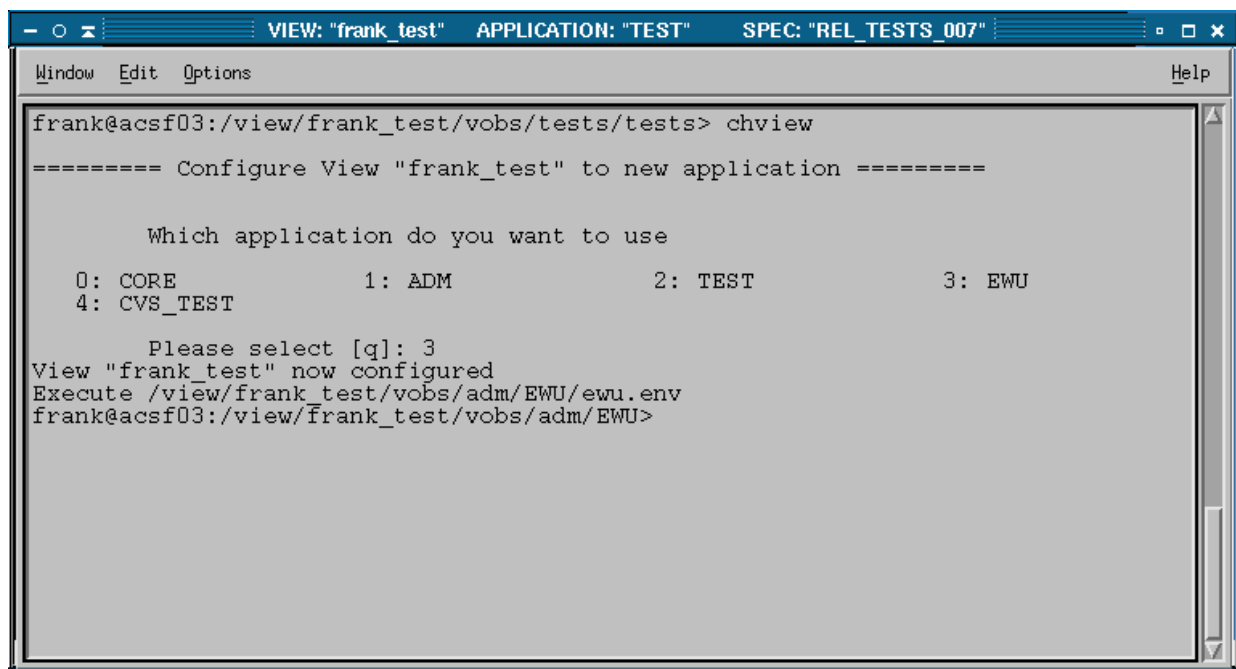
- Anschließend erfolgt ein Wechsel in das Arbeitsverzeichnis und sowie ggf. die Einstellungen des Projekts (z.B. PATH, LD_LIBRARY_PATH etc. setzen).
- Die eingestellte Sicht auf die Versionsverwaltung (Config-Spec) entspricht dem aktuellen Releasestand des Projekts.

Ein View wird mit Kommando:

chview

um konfiguriert.

Beispiel:



```
VIEW: "frank_test" APPLICATION: "TEST" SPEC: "REL_TESTS_007"
Window Edit Options Help
frank@acsf03:/view/frank_test/vobs/tests/tests> chview
===== Configure View "frank_test" to new application =====

Which application do you want to use

0: CORE          1: ADM          2: TEST          3: EWU
4: CVS_TEST

Please select [q]: 3
View "frank_test" now configured
Execute /view/frank_test/vobs/adm/EWU/ewu.env
frank@acsf03:/view/frank_test/vobs/adm/EWU>
```

Bild11: Um konfigurieren eines Views

Im obigen Beispiel wird der View vom Projekt 'TEST' auf das Projekt 'EWU' konfiguriert.

Die Config-Spec wird wieder auf die aktuelle Releasesicht eingestellt.

Die Umgebung wird auf die Belange des neuen Projekts eingestellt. Für diese Projekt ist in der Datei ewu.conf eine Konfigurationsdatei ewu.env angegeben. Diese wird in der aktuellen Shell ausgeführt.

Danach wird in das Arbeitsverzeichnis gewechselt.

Die Gruppe wird diesmal nicht gesetzt, da sie bereits der konfigurierten Gruppe entspricht.

Falls User die Config-Spec händisch editieren kann es vorkommen, dass der View nicht mehr korrekt reagiert. In diesem Falle kann der View mittels 'chview' wieder repariert wird. Dazu ist einfach auf ein anderes Projekt umzukonfigurieren und durch eine weiteren Aufruf von 'chview' wieder auf das ursprüngliche Projekt zurück zu schalten. Anschließend ist noch die gewünschte Config-Spec zu setzen.

5.3.3 Vorhandene Views zur Arbeit nutzen

Hat der User bereits Views erzeugt und konfiguriert, müssen diese auch ansprechbar sein. Die Konfiguration des Views und die Config-Spec entsprechen den zuletzt eingestellten Werten.

Ein Tool zum Starten eines Views muss folgende Funktionalität bereitstellen:

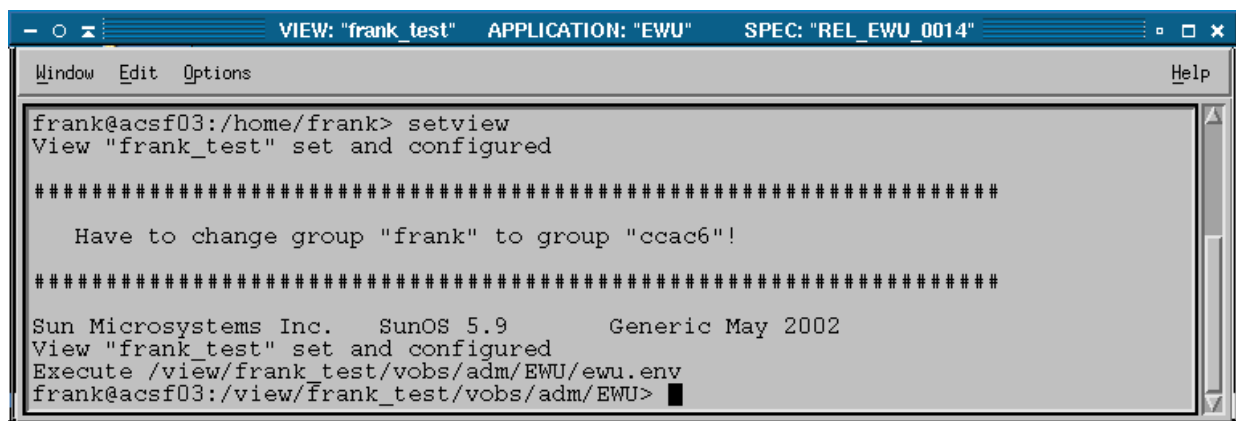
- Existiert nur ein View für den Benutzer, wird dieser sofort gestartet, die Umgebung konfiguriert und in das Arbeitsverzeichnis gewechselt.
- Bei mehreren Views des Benutzers wird der gewünschte View abgefragt, gestartet, die Umgebung konfiguriert und anschließend in das Arbeitsverzeichnis gewechselt.

Ein vorhandener View wird mit Kommando:

```
setview [view_name]
```

gestartet.

Beispiel:



```
VIEW: "frank_test" APPLICATION: "EWU" SPEC: "REL_EWU_0014"
Window Edit Options Help
frank@acsf03:/home/frank> setview
View "frank_test" set and configured

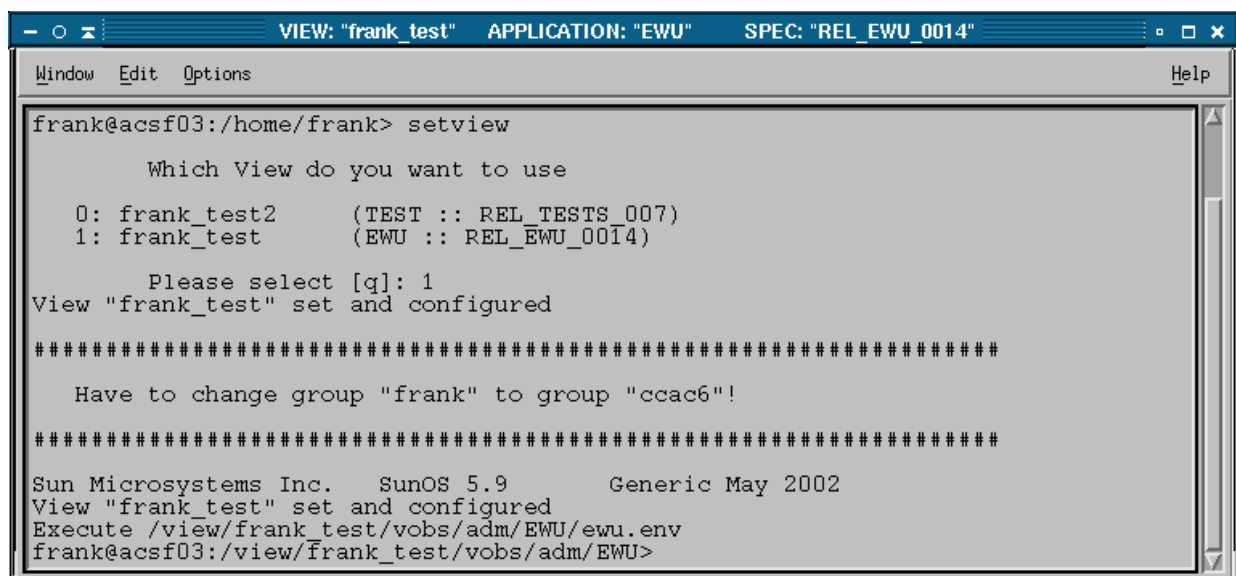
#####

Have to change group "frank" to group "ccac6"!

#####

Sun Microsystems Inc. SunOS 5.9 Generic May 2002
View "frank_test" set and configured
Execute /view/frank_test/vobs/adm/EWU/ewu.env
frank@acsf03:/view/frank_test/vobs/adm/EWU>
```

oder



```
VIEW: "frank_test" APPLICATION: "EWU" SPEC: "REL_EWU_0014"
Window Edit Options Help
frank@acsf03:/home/frank> setview

Which View do you want to use

0: frank_test2 (TEST :: REL_TESTS_007)
1: frank_test (EWU :: REL_EWU_0014)

Please select [q]: 1
View "frank_test" set and configured

#####

Have to change group "frank" to group "ccac6"!

#####

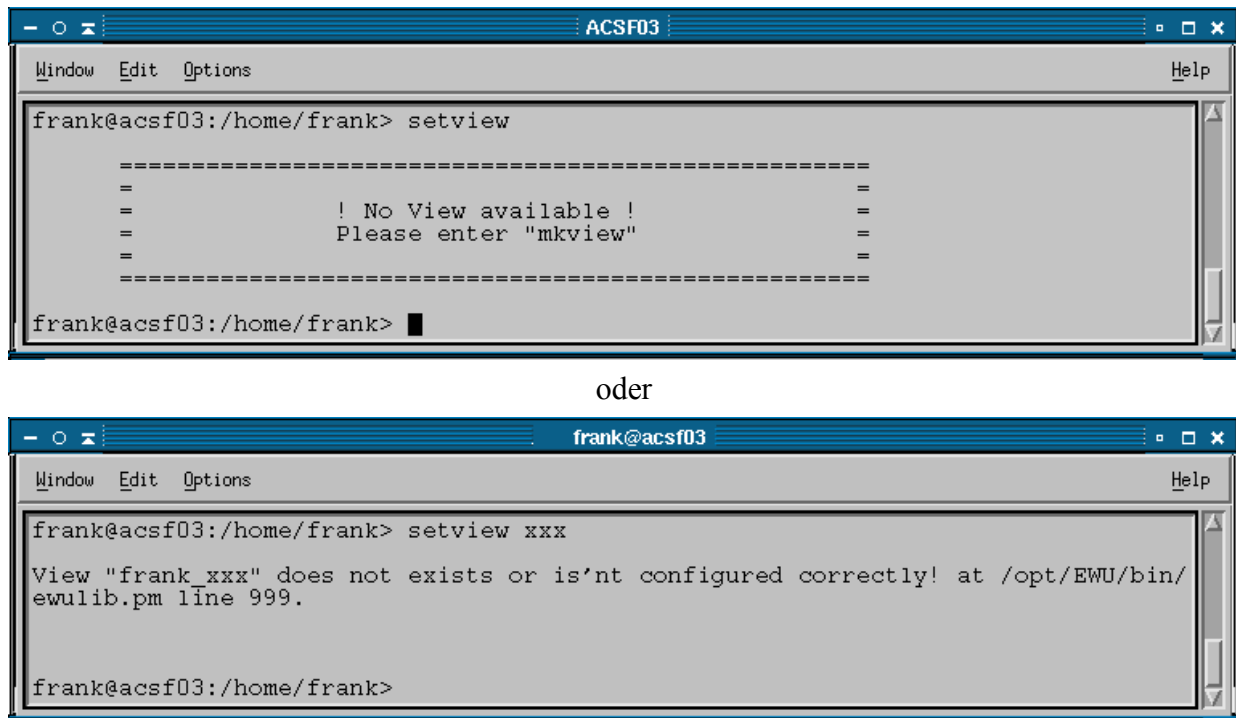
Sun Microsystems Inc. SunOS 5.9 Generic May 2002
View "frank_test" set and configured
Execute /view/frank_test/vobs/adm/EWU/ewu.env
frank@acsf03:/view/frank_test/vobs/adm/EWU>
```

Bild 12: Starten eines Views

Bei der Auflistung mehrerer Views werden zu jedem Viewnamen auch das konfigurierte Projekt sowie die eingestellte Config-Spec angezeigt.

Hat der User keinen View oder der Viewname wird unkorrekt angegeben, erfolgt eine entsprechende Fehlermeldung:

Beispiel:



The image shows two screenshots of a terminal window titled 'ACSF03'. The first screenshot shows the command 'setview' being executed, which results in a message: '----- ! No View available ! ----- Please enter "mkview" -----'. The second screenshot, labeled 'oder', shows the command 'setview xxx' being executed, which results in an error message: 'View "frank_xxx" does not exists or is'nt configured correctly! at /opt/EWU/bin/ewulib.pm line 999.'

Bild 13: mögliche Fehlermeldungen beim Starten eines Views

5.3.4 Einstellen der Sicht in einem View

Der View reicht für eine Sicht auf die Versionsverwaltung nicht aus. Welche Dateien und Verzeichnisse gesehen werden, wird durch die Config-Spec bestimmt.

Diese Config-Spec muss entsprechend ausgewählt werden können.

Ein Tool dazu muss folgende Funktionalität bereitstellen:

- Bei Aufruf ohne weiteren Parameter, wird immer der aktuelle Releasestand des Projektes ausgewählt. Ansonsten wird die durch den Namen identifizierte Config-Spec gesetzt.
- Tritt ein Fehler auf, wird wieder auf die ursprüngliche Config-Spec zurück gesetzt.
- Da ausgecheckte Elemente durch Umsetzen der Config-Spec unter Umständen unsichtbar werden, muss ein entsprechender Check durchgeführt werden und die Aktion ggf. abgebrochen werden.

Ein View wird mit dem Kommando:

```
setspec [-l] [spec_name]
```

auf eine gewünschte Config-Spec eingestellt.

Beispiel:

```

VIEW: "frank_test2"  APPLICATION: "TEST"  SPEC: "PR_TEST_0009"
Window Edit Options Help
frank@acsf03:/view/frank_test2/vobs/tests/tests> setspec -l

Which config-spec do you want to set

  0: REL_TESTS_001      1: PR_TEST_0001      2: REL_TESTS_002      3: PR_TEST_0003
  4: PR_TEST_0004      5: PR_TEST_0005      6: INTEGRATION001     7: REL_TESTS_003
  8: PR_TEST_0002      9: PR_TEST_0006     10: INTEGRATION002    11: REL_TESTS_004
 12: PR_TEST_0007     13: INTEGRATION003   14: REL_TESTS_005     15: PR_TEST_0008
 16: INTEGRATION004   17: REL_TESTS_006    18: PR_TEST_0009     19: INTEGRATION005
 20: REL_TESTS_007    21: PR_TEST_0010

Please select [q]: 18
Set Config-Spec "PR_TEST_0009"

!!!! Attantion: View-Configuration has changed !!!!

frank@acsf03:/view/frank_test2/vobs/tests/tests>

```

Bild 14: Setzen einer Config-Spec

Die Config-Spec kann je nach Aufruf aus einer Liste ausgewählt werden (siehe Beispiel) oder per Name angegeben werden. Wird kein Parameter beim Aufruf übergeben, erfolgt die Einstellung des aktuellen Releasestandes.

Sind im View Files ausgecheckt, wird darauf hingewiesen und eine Sicherheitsabfrage gestartet:

Beispiel:

```

VIEW: "frank_test2"  APPLICATION: "TEST"  SPEC: "PR_TEST_0010"
Window Edit Options Help
frank@acsf03:/view/frank_test2/vobs/tests/tests> setspec
setspec.pl: Warning: You still have checked-out files:
/view/frank_test2/vobs/tests/tests/bin

You should do "end_work" or "cleartool unco ..."!
Do you want to change the config-spec anywhere [y|n] [q]: q
Terminated .....
frank@acsf03:/view/frank_test2/vobs/tests/tests>

```

Bild 15: Setzen einer Config-Spec mit Sicherheitsabfrage bei ausgecheckten Files

Der User kann den Wechsel der Config-Spec erzwingen, sollte sich aber darüber im Klaren sein, dass unter Umständen die ausgecheckten Files nicht mehr sichtbar sind.

5.3.5 Ermitteln der aktuellen Release- und Produktionsstände

Für einen schnellen Überblick über die aktuellen Stände zu verschaffen, wird der Aufruf:

```

get_act_release -r|-p [-a] [-s]
-p           Get actual production.
-r           Get actual release.
-s           Silence.

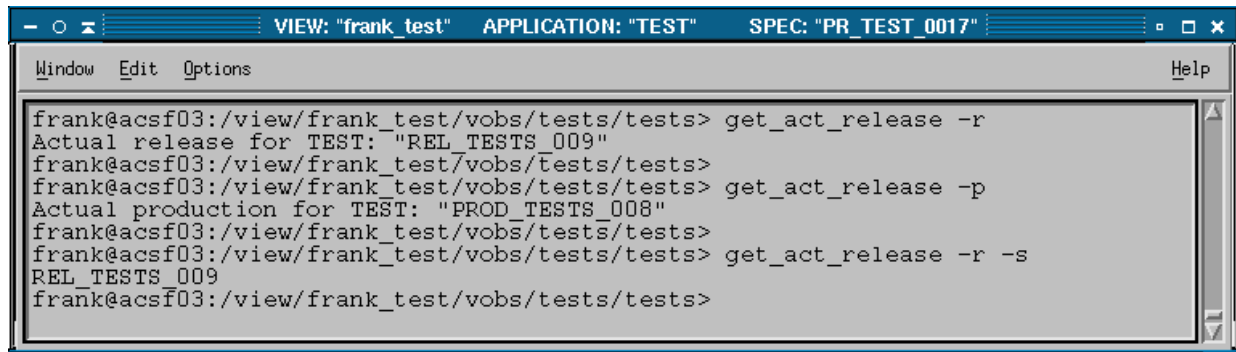
```


-a Production/release over all applications.

Genutzt.

Dieses Tool kann sowohl innerhalb eines gesetzten Views als auch ohne View genutzt werden.

Beispiel:



```

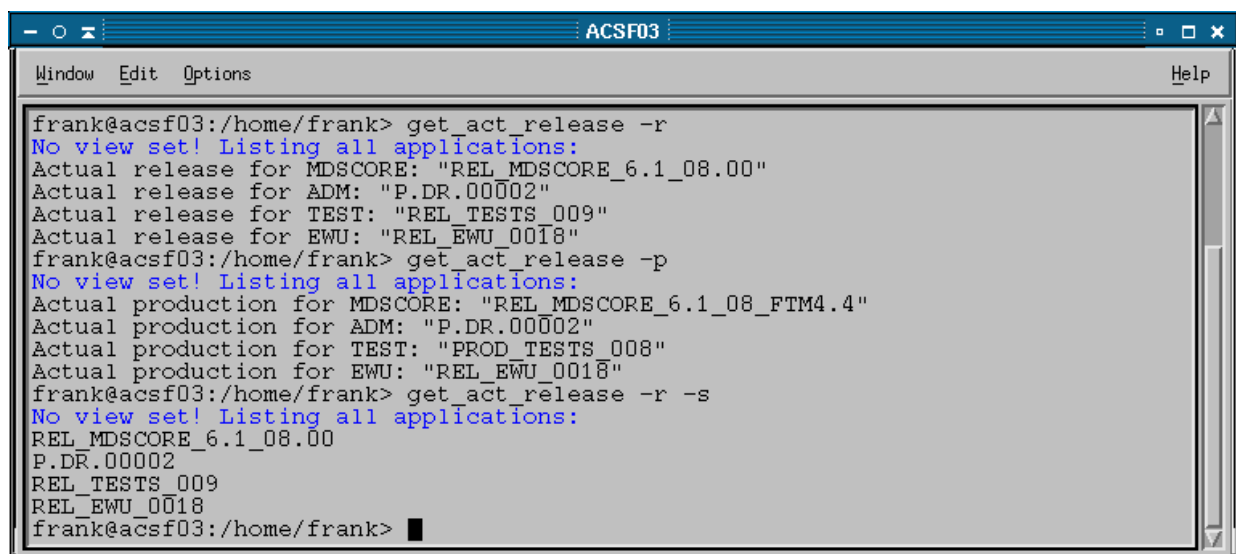
VIEW: "frank_test" APPLICATION: "TEST" SPEC: "PR_TEST_0017"
frank@acsf03:/view/frank_test/vobs/tests/tests> get_act_release -r
Actual release for TEST: "REL_TESTS_009"
frank@acsf03:/view/frank_test/vobs/tests/tests>
frank@acsf03:/view/frank_test/vobs/tests/tests> get_act_release -p
Actual production for TEST: "PROD_TESTS_008"
frank@acsf03:/view/frank_test/vobs/tests/tests>
frank@acsf03:/view/frank_test/vobs/tests/tests> get_act_release -r -s
REL_TESTS_009
frank@acsf03:/view/frank_test/vobs/tests/tests>

```

Bild 16: Beispiele für `get_act_prod` innerhalb eines gesetzten Views

Ausserhalb eines gesetzten Views werden die Informationen für alle Projekte aufgelistet

Beispiel:



```

ACSF03
frank@acsf03:/home/frank> get_act_release -r
No view set! Listing all applications:
Actual release for MDSCORE: "REL_MDSCORE_6.1_08.00"
Actual release for ADM: "P.DR.00002"
Actual release for TEST: "REL_TESTS_009"
Actual release for EWU: "REL_EWU_0018"
frank@acsf03:/home/frank> get_act_release -p
No view set! Listing all applications:
Actual production for MDSCORE: "REL_MDSCORE_6.1_08_FTM4.4"
Actual production for ADM: "P.DR.00002"
Actual production for TEST: "PROD_TESTS_008"
Actual production for EWU: "REL_EWU_0018"
frank@acsf03:/home/frank> get_act_release -r -s
No view set! Listing all applications:
REL_MDSCORE_6.1_08.00
P.DR.00002
REL_TESTS_009
REL_EWU_0018
frank@acsf03:/home/frank>

```

Bild 17: Beispiele für `get_act_prod` innerhalb ohne gesetzten View

Hier macht der Schalter `-s` keinen Sinn, da dann die Zuordnung zum Projekt verloren geht.

5.3.6 Löschen einer Sicht auf die Versionsverwaltung

Nicht mehr benötigte Views müssen natürlich auch komfortabel gelöscht werden können.

Dabei sind folgende Anforderungen zu erfüllen:

- Falls im View ausgecheckten Elemente existieren, Auflisten und Abbruch der Aktion.

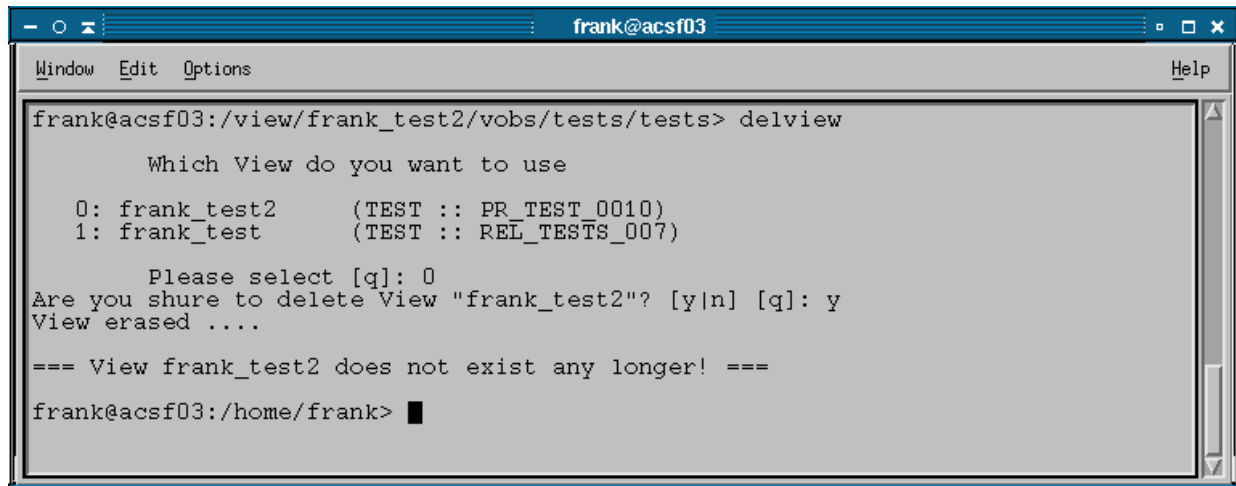
- Nachfrage, ob der View wirklich gelöscht werden soll.
- Löschen aller Konfigurationsdateien für diesen View.
- View selbst löschen.

Ein View wird mit dem Kommando:

```
delview [view_name]
```

gelöscht.

Beispiel:



```
frank@acsf03:~/view/frank_test2/vobs/tests/tests> delview
Which View do you want to use
0: frank_test2      (TEST :: PR_TEST_0010)
1: frank_test      (TEST :: REL_TESTS_007)

Please select [q]: 0
Are you shure to delete View "frank_test2"? [y|n] [q]: y
View erased ....

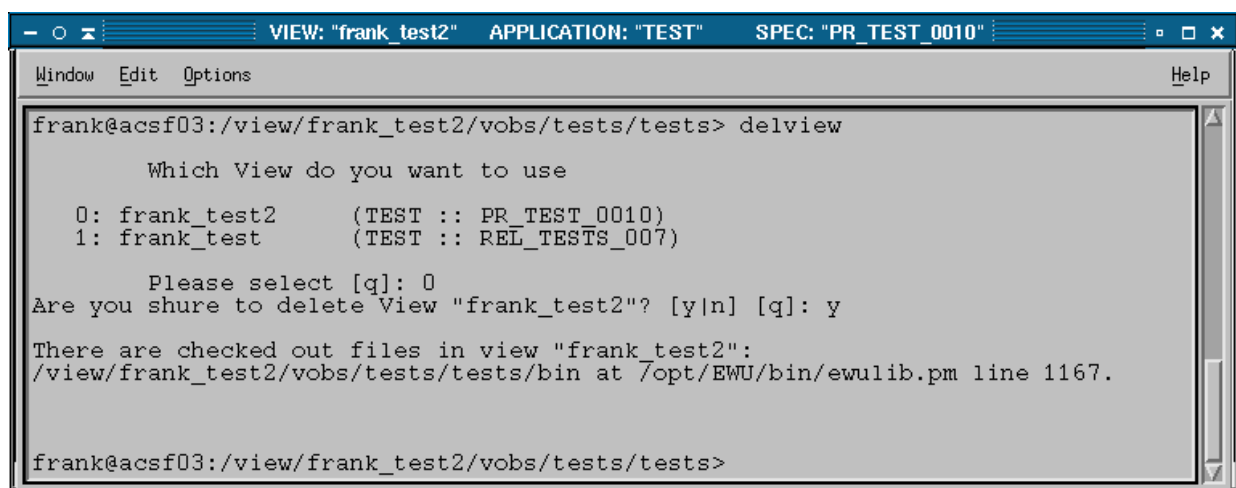
=== View frank_test2 does not exist any longer! ===
frank@acsf03:/home/frank>
```

Bild 18: Löschen eines Views

Der View wird erst nach einer Sicherheitsabfrage gelöscht.

Falls im View noch ausgecheckte Files vorhanden sind, wird eine Fehlermeldung ausgegeben und der View wird nicht gelöscht.

Beispiel:



```
frank@acsf03:~/view/frank_test2/vobs/tests/tests> delview
Which View do you want to use
0: frank_test2      (TEST :: PR_TEST_0010)
1: frank_test      (TEST :: REL_TESTS_007)

Please select [q]: 0
Are you shure to delete View "frank_test2"? [y|n] [q]: y
There are checked out files in view "frank_test2":
/view/frank_test2/vobs/tests/tests/bin at /opt/EWU/bin/ewulib.pm line 1167.

frank@acsf03:~/view/frank_test2/vobs/tests/tests>
```

Bild 19: Löschen eines Views mit ausgecheckten Elementen

5.3.7 Kommunikation der Views in verschiedenen Terminals

In einer normalen Entwicklungsumgebung in UNIX ist es üblich in mehreren Terminals zu entwickeln.

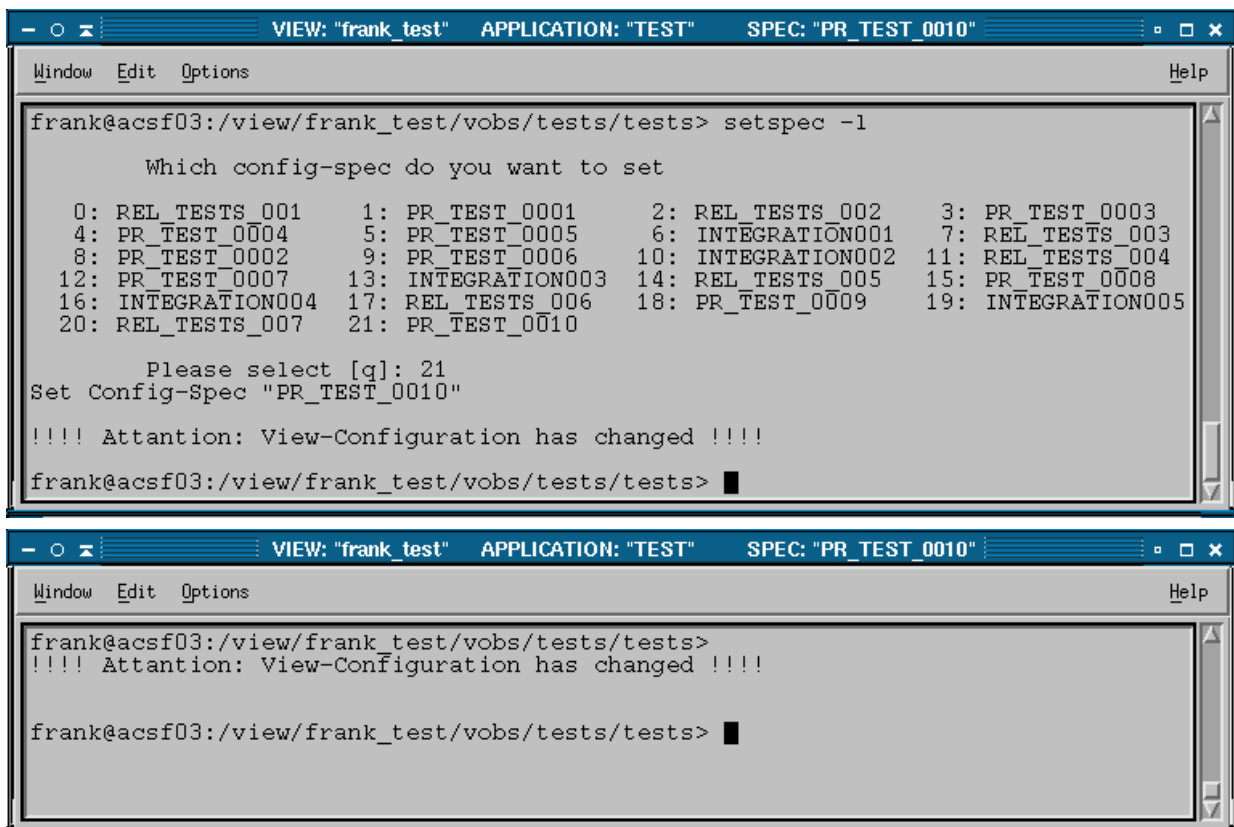
Z.B. ist es möglich in mehreren Terminals im gleichen View zu entwickeln und in einem weiteren Terminal Tests durchzuführen.

Falls nun die Konfiguration des Views geändert wird, müssen zumindest alle anderen Terminals, welche diesen View benutzen, über diese Änderung informiert werden.

Folgende Informationen müssen ausgetauscht werden:

- Wechsel der Config-Spec.
- Wechsel des Projektes.
- Löschen des Views.

Beispiel:



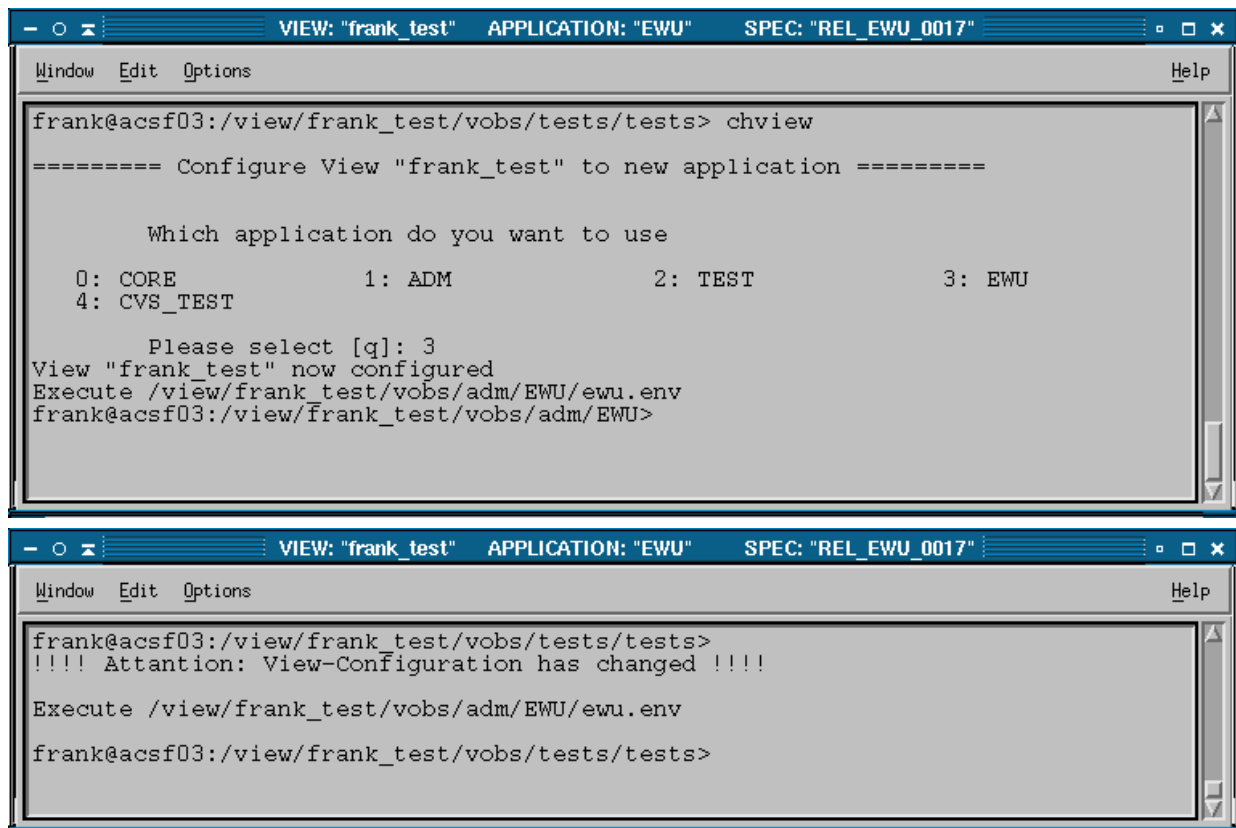
```
VIEW: "frank_test" APPLICATION: "TEST" SPEC: "PR_TEST_0010"
Window Edit Options Help
frank@acsf03:/view/frank_test/vobs/tests/tests> setspec -l
Which config-spec do you want to set
0: REL_TESTS_001      1: PR_TEST_0001      2: REL_TESTS_002      3: PR_TEST_0003
4: PR_TEST_0004      5: PR_TEST_0005      6: INTEGRATION001     7: REL_TESTS_003
8: PR_TEST_0002      9: PR_TEST_0006     10: INTEGRATION002    11: REL_TESTS_004
12: PR_TEST_0007     13: INTEGRATION003   14: REL_TESTS_005     15: PR_TEST_0008
16: INTEGRATION004   17: REL_TESTS_006    18: PR_TEST_0009     19: INTEGRATION005
20: REL_TESTS_007    21: PR_TEST_0010
Please select [q]: 21
Set Config-Spec "PR_TEST_0010"
!!!! Attantion: View-Configuration has changed !!!!
frank@acsf03:/view/frank_test/vobs/tests/tests> █

VIEW: "frank_test" APPLICATION: "TEST" SPEC: "PR_TEST_0010"
Window Edit Options Help
frank@acsf03:/view/frank_test/vobs/tests/tests>
!!!! Attantion: View-Configuration has changed !!!!

frank@acsf03:/view/frank_test/vobs/tests/tests> █
```

Bild 20: Kommunikation von Views beim Wechsel der Config-Spec

Im Beispiel oben wird beim Wechsel der Config-Spec eine Warnung ausgegeben und die Titelzeile des Terminals angepasst.



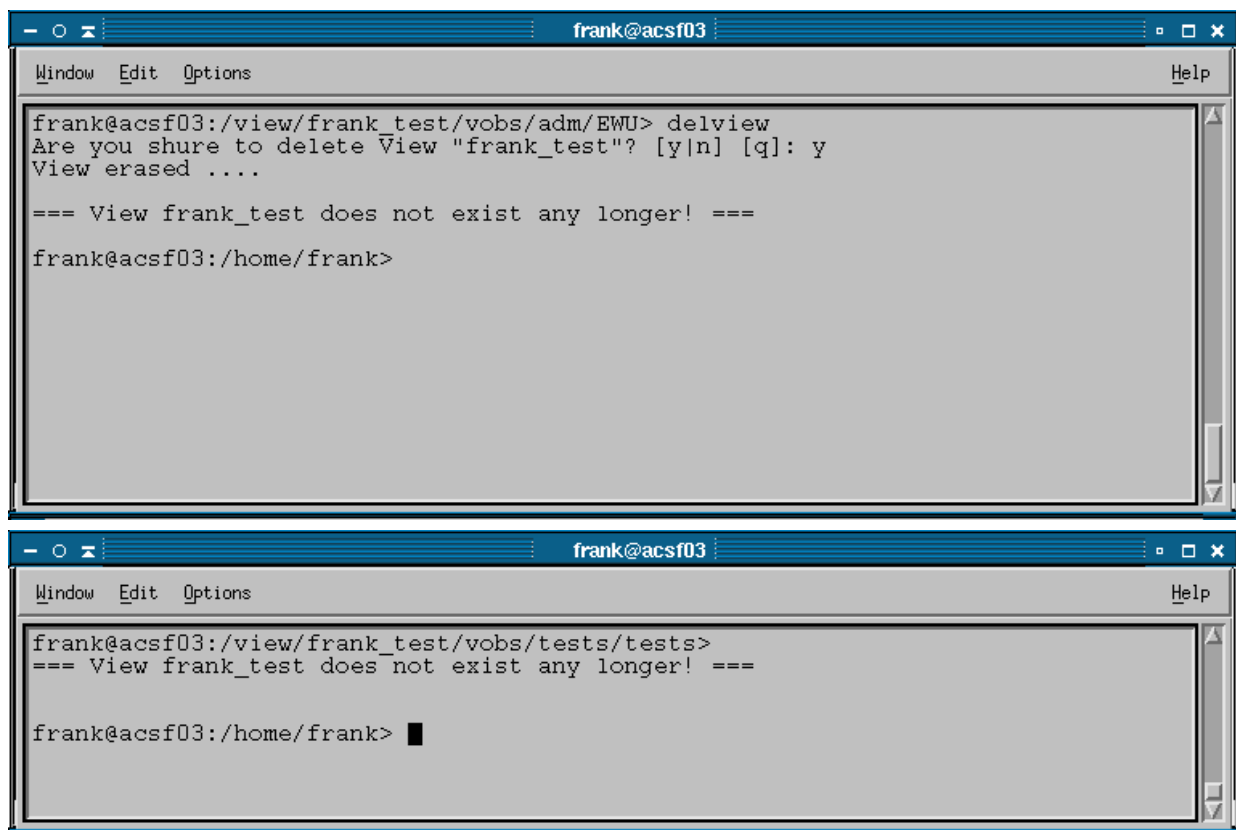
The image shows two screenshots of a terminal window in a ClearCase environment. The window title bar indicates the current view is "frank_test", the application is "EWU", and the specification is "REL_EWU_0017".

The first screenshot shows the user running the `chview` command. The terminal displays a prompt to configure the view to a new application. A list of available applications is shown: 0: CORE, 1: ADM, 2: TEST, 3: EWU, and 4: CVS_TEST. The user selects option 3. The terminal confirms the configuration and prompts the user to execute the environment file `/view/frank_test/vobs/adm/EWU/ewu.env`.

The second screenshot shows the user executing the environment file. The terminal displays a warning message: "!!!! Attantion: View-Configuration has changed !!!!". The user then runs the `chview` command again, and the terminal returns to the prompt.

Bild 21: Kommunikation von Views beim Wechsel des Projektes

Beim Wechsel des Projektes wird sowohl die Titelzeile des Terminals angepasst, als auch in das Arbeitsverzeichnis des Projektes gewechselt.



The image shows two terminal windows from the user frank@acsf03. The top window shows the execution of the 'delview' command. The user enters 'delview' and is prompted 'Are you shure to delete View "frank_test"? [y|n] [q]: y'. The terminal outputs 'View erased', followed by a confirmation message '=== View frank_test does not exist any longer! ===', and finally the prompt changes to the user's home directory: 'frank@acsf03:/home/frank>'. The bottom window shows the user in a different directory, '/view/frank_test/vobs/tests/tests>', where they receive the same confirmation message '=== View frank_test does not exist any longer! ===' and the prompt returns to the home directory: 'frank@acsf03:/home/frank>'. Both windows have a menu bar with 'Window', 'Edit', 'Options', and 'Help'.

Bild 22: Kommunikation beim Löschen von Views

Beim Löschen von Views wechseln alle Terminals, welche diesen View gesetzt haben, ins Home-Verzeichnis des Users. Die Titelzeile wird wiederum angepasst.

5.4 Development-Tools

5.4.1 Start einer Entwicklung

Ausgehend vom allgemeinen Workflow bekommen die Development-Teams einen Entwicklungsauftrag über das Releasemanagement gestellt und melden die Fertigstellung wieder an das Releasemanagement.

Sie benötigen dazu nur die Information, um welches Projekt es sich handelt, die Identifikation des CR/PR und ob es sich dabei um eine normale Entwicklung oder um einen Hot-Fix handelt.

Ein Tool muss dazu folgende Aufgaben erfüllen:

- Ermitteln des gültigen Release- oder Produktionsstandes in Abhängigkeit von Standartentwicklung oder Hotfix.
- Erzeugen und sichern der Entwickler-Config-Spec.
- Erzeugen von Branch- und Labeltypen in allen zum Projekt gehörenden Vobs sowie Wechsel der Eigentümerschaft dieser Typen auf den Integration-User.

Mit dem Kommando:

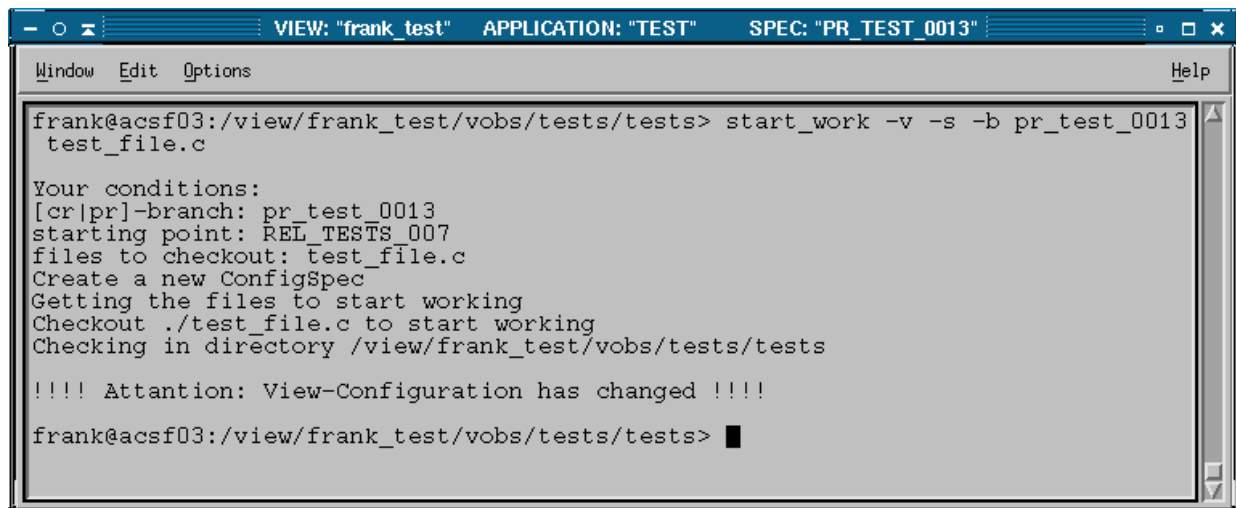
```
start_work.pl -s|-p [-l label] [-b branch] [-i] [-v] filelist
  -p                production hotfix
  -s                standard development
  -l <label>       Label is the starting point of your development.
                   For std-dev the default is the actual Release.
                   (Label may be omitted)
  -b <branch>      The branch, you want to work on ( f.e. Cr### )
  -i                interactive mode
  -v                verbose
```

wird die Entwicklung gestartet.

5.4.1.1 Standardentwicklung

Die Standardentwicklung wird den Großteil des täglichen Entwicklungsgeschäftes ausmachen. Diese Entwicklungen setzen immer auf dem zum Zeitpunkt des Entwicklungsstarts gültigen Releasestand auf.

Beispiel:



```
frank@acsf03: /view/frank_test/vobs/tests/tests > start_work -v -s -b pr_test_0013
test_file.c

Your conditions:
[cr|pr]-branch: pr_test_0013
starting point: REL_TESTS_007
files to checkout: test_file.c
Create a new ConfigSpec
Getting the files to start working
Checkout ./test_file.c to start working
Checking in directory /view/frank_test/vobs/tests/tests

!!!! Attantion: View-Configuration has changed !!!!

frank@acsf03: /view/frank_test/vobs/tests/tests > █
```

Bild 23: Start einer Entwicklung mit Anlegen einer neuen Datei

Im obigen Beispiel wird im Rahmen einer Standard-Entwicklung ein neues File in die Versionsverwaltung übernommen. Entwickelt wird im Branch '**pr_test_0013**'.

Das Tool ermittelt den letzten Release-Stand, erzeugt auf dieser Basis eine Config-Spec (falls diese noch nicht existiert) und erzeugt anschließend das File 'test_file.c'.

Wird beim Aufruf des Tools keine File-Liste angegeben, wird nur die Config-Spec erzeugt und gesetzt. Jeder weitere Aufruf des Tools wird diese dann neu setzen. Natürlich kann die Config-Spec jetzt auch mittels '**setspec pr_test_0013**' gesetzt werden.

Wenn eine File-Liste beim Aufruf übergeben wurde, erfolgt die Bearbeitung nach folgenden Regeln:

- Wenn das File existiert und ein ClearCase-Element ist, wird es Abgebrant und ausgecheckt.
- Handelt es sich um ein view-private File sind folgende Unterscheidungen zu beachten:
 1. Es ist ein reguläres File: Es wird in ein ClearCase-Element überführt, gebrant

und ausgecheckt.

2. Es handelt sich um ein Verzeichnis: Es wird nicht bearbeitet. Im Logfile wird eine Warnung geschrieben.
 3. Es handelt sich um eine Kombination Verzeichnis/File: Wenn alle Verzeichnisse existieren, wird das File gebrannt und ausgecheckt. Hier darf es sich auch bei den Verzeichnissen um view-private Objekte handeln.
- Das File existiert noch nicht:
 1. Es handelt sich nicht um eine Kombination Verzeichnis/File: Es wird als ClearCase-Objekt erzeugt, gebrannt und ausgecheckt
 2. Es ist eine Kombination Verzeichnis/File: Wenn das Verzeichnis nicht existiert oder ein view-private Objekt ist, wird wiederum nichts gemacht und eine Warnung im Logfile hinterlegt.

Bei neuen Files und Verzeichnissen wird natürlich auch das Basis-Verzeichnis gebrannt, ausgecheckt, gleich gelabelt und wieder eingchecked.

Beispiel:

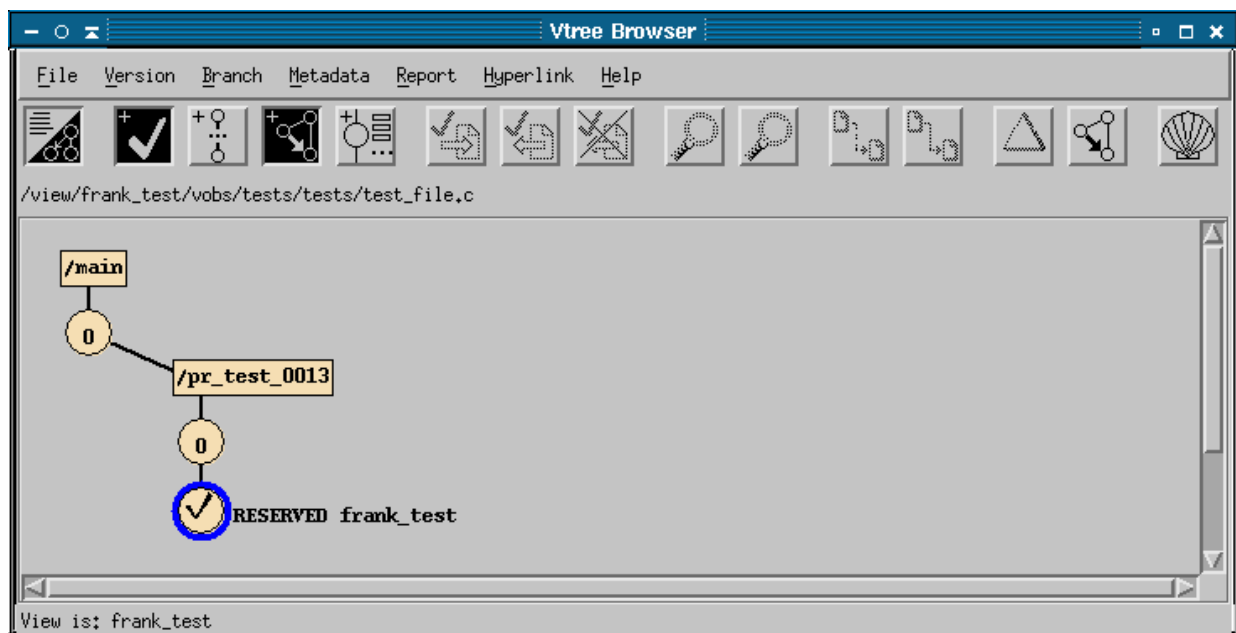
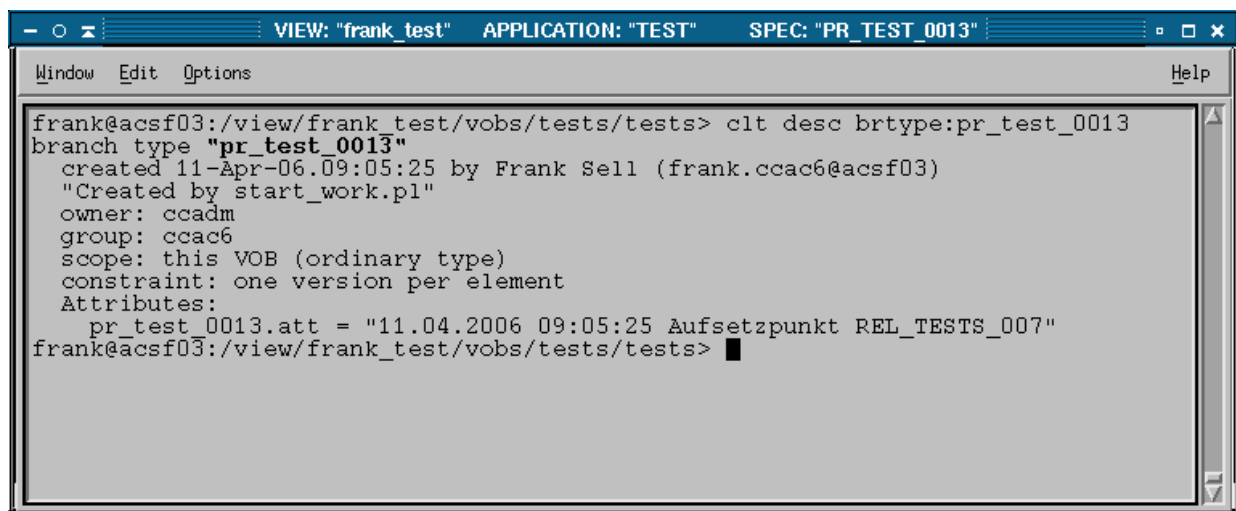


Bild 24: Versionsbaum der neuen Datei

Da es sich in diesem Fall um eine neue Datei handelt, wird grundsätzlich von main/0 abgebrannt. Für dieses File wird erst ein Release-Branch (definiert in ewu.conf) erzeugt, wenn es tatsächlich zur Auslieferung kommt. Damit vermeidet man bei Auslieferungen während der Entwicklungsphase, dass ein 0 Byte großes File ausgeliefert wird.

Beim Erzeugen der Entwickler-Config-Spec werden auch gleichzeitig die erforderlichen Label- und Branch-Typen in den entsprechenden Vobs erzeugt.

Der Branchtyp der Entwicklung bekommt ein Attribut mit den Informationen zur Entstehung:



```
VIEW: "frank_test" APPLICATION: "TEST" SPEC: "PR_TEST_0013"
Window Edit Options Help
frank@acsf03:/view/frank_test/vobs/tests/tests> clt desc brtype:pr_test_0013
branch type "pr_test_0013"
  created 11-Apr-06.09:05:25 by Frank Sell (frank.ccac6@acsf03)
  "Created by start_work.pl"
  owner: ccadm
  group: ccac6
  scope: this VOB (ordinary type)
  constraint: one version per element
  Attributes:
    pr_test_0013.att = "11.04.2006 09:05:25 Aufsetzpunkt REL_TESTS_007"
frank@acsf03:/view/frank_test/vobs/tests/tests> █
```

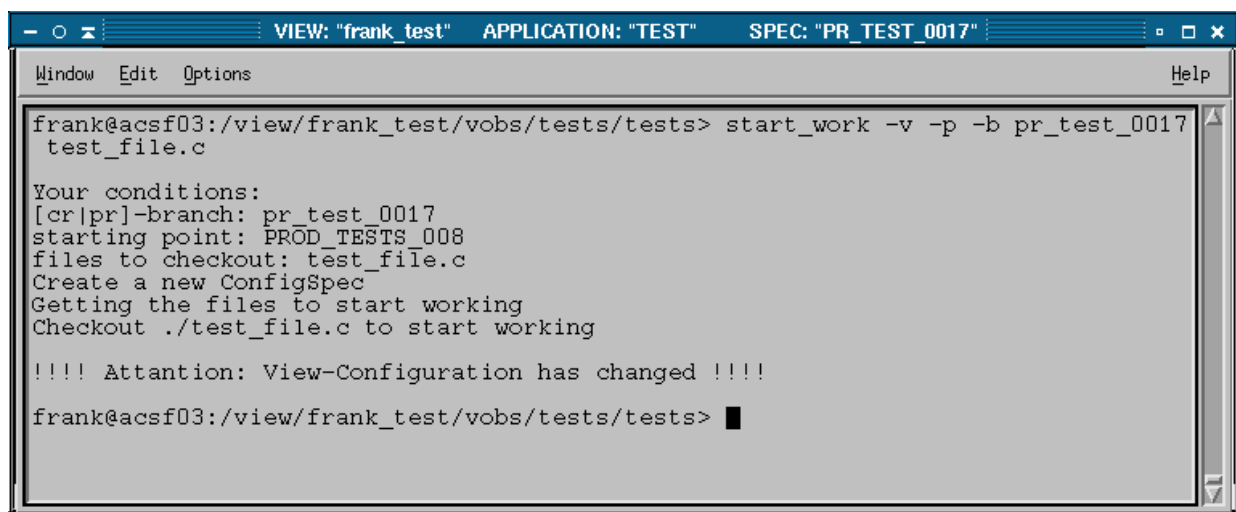
Bild 25: Informationen zum Entwicklungsbranch

5.4.1.2 Hotfixentwicklung

Die Bearbeitung von Hotfixen ist eine Sonderart der Entwicklung. Sie trägt der oben beschriebenen Diskrepanz zwischen Releasestand und produktiven System Rechnung.

Im folgenden Beispiel ist der Releasestand REL_TESTS_008 als produktiver Stand gekennzeichnet. Aktueller Releasestand ist REL_TESTS_009.

Folgendes Bild ergibt sich bei der Hotfix-Bearbeitung



```
VIEW: "frank_test" APPLICATION: "TEST" SPEC: "PR_TEST_0017"
Window Edit Options Help
frank@acsf03:/view/frank_test/vobs/tests/tests> start_work -v -p -b pr_test_0017
test_file.c

Your conditions:
[cr|pr]-branch: pr_test_0017
starting point: PROD_TESTS_008
files to checkout: test_file.c
Create a new ConfigSpec
Getting the files to start working
Checkout ./test_file.c to start working

!!!! Attantion: View-Configuration has changed !!!!
frank@acsf03:/view/frank_test/vobs/tests/tests> █
```

Bild 26: Aufruf von start_work für einen Hotfix

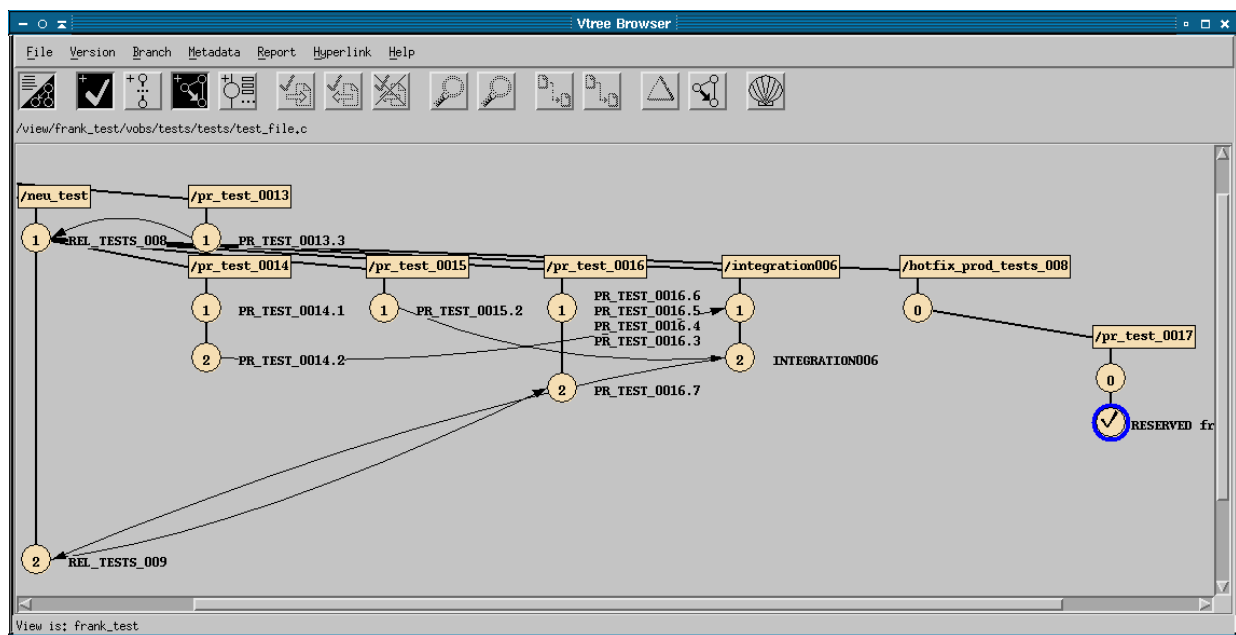


Bild 27: Versionsbaum der Datei

Durch die Bearbeitung des Hotfixes wird vom Produktionsstand unter Zwischenschaltung des Hotfixbranches 'hotfix_prod_tests_008' die betroffene Datei dann im PR-Branche ausgecheckt.

Hier endet die Toolunterstützung vorerst.

Nach den Modultests wird der Hotfix vom PR-Branche ausgeliefert. Nach der Auslieferung muss der PR sowohl auf den Hotfixbranch gemerged werden als auch in das nächste Release einfließen.

5.4.2 Abschluss einer Entwicklung

Der Abschluss eines Entwicklungsauftrags ist dann gegeben, wenn alle Files und Verzeichnisse eingecheckt sind und mit einem Label versehen wurden.

Diese Aufgabe übernimmt der Aufruf

```
end_work.pl [-i] [-v] filelist
-i          interactive mode
-v          verbose
```

Dieses Tool muss innerhalb der Entwicklungs-Config-Spec aufgerufen werden.

Es ermittelt das höchste Update-Label zum Entwicklungsauftrag und erzeugt ein neues Update-Label.

Das Tool wird alle Files und Verzeichnisse, welches das bisherige maximale Label haben, mit dem neuen Label versehen.

Anschließend werden ausgecheckte Files und Verzeichnisse im aktuellen View ermittelt und nachgefragt, ob diese eingecheckt werden sollen. Wird dieses positiv beantwortet, wird alles eingecheckt und gelabelt.

Die einfachste Vorgehensweise für den Entwickler ist also:

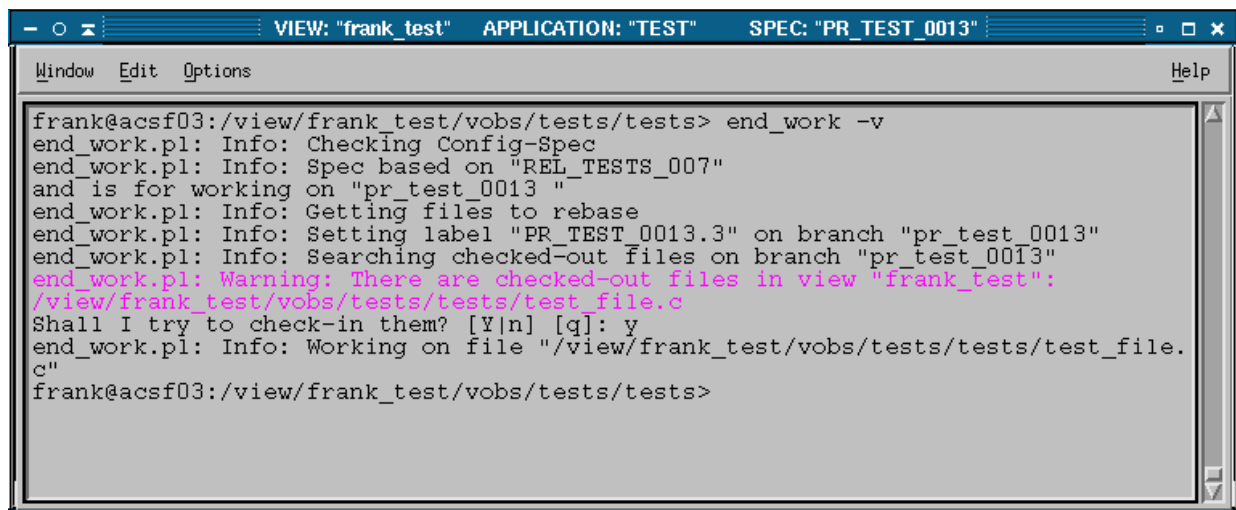
Alles bis zum Ende der Entwicklungstätigkeit ausgecheckt lassen und den Rest dem Tool **end_work** überlassen.

Hat der Entwickler seine Files mittels ClearCase-Kommandos eingechekkt, müssen diese zum Labeln explizit mit angegeben werden!

Hier bietet sich auch die Möglichkeit versehentlich bearbeitete Files nicht in die Entwicklung zu übernehmen, indem einfach ein **cleartool unco ...** durchgeführt wird, und dieses File beim **end_work** nicht anzugeben.

Bei paralleler Bearbeitung von Entwicklungsaufträgen durch mehrere Entwickler muss jeder Entwickler **end_work** in den betreffenden Views aufrufen.

Beispiel:



```
frank@acsf03:/view/frank_test/vobs/tests/tests> end_work -v
end_work.pl: Info: Checking Config-Spec
end_work.pl: Info: Spec based on "REL_TESTS_007"
and is for working on "pr_test_0013 "
end_work.pl: Info: Getting files to rebase
end_work.pl: Info: Setting label "PR_TEST_0013.3" on branch "pr_test_0013"
end_work.pl: Info: Searching checked-out files on branch "pr_test_0013"
end_work.pl: Warning: There are checked-out files in view "frank_test":
/view/frank_test/vobs/tests/tests/test_file.c
Shall I try to check-in them? [Y|n] [q]: y
end_work.pl: Info: Working on file "/view/frank_test/vobs/tests/tests/test_file.
c"
frank@acsf03:/view/frank_test/vobs/tests/tests>
```

Bild 28: Ende einer Entwicklung kennzeichnen

Das File hat anschließend folgenden Versionsbaum:

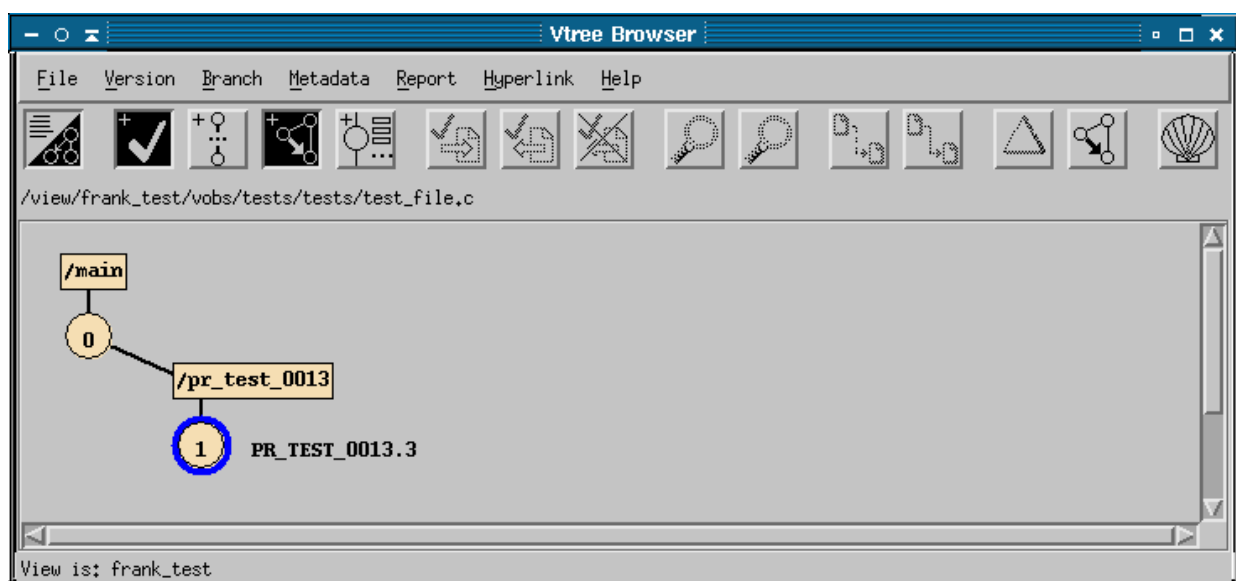
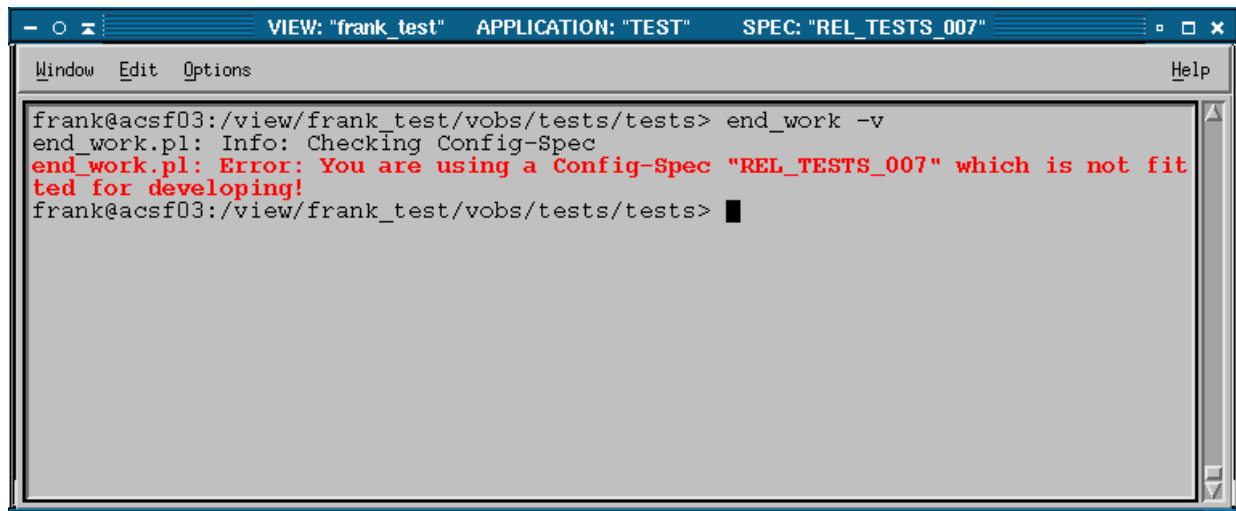


Bild 29: Versionsbaum nach Entwicklungsende

Über den Zählerstand des Update-Labels braucht sich der Entwickler keine Gedanken zu machen.

Werden obige Hinweise beachtet, wird das Tool auch alles korrekt durchlabeln.

Wird **end_work** aus einer falschen Config-Spec heraus aufgerufen, erfolgt die Ausgabe einer Fehlermeldung:



```
VIEW: "frank_test"  APPLICATION: "TEST"  SPEC: "REL_TESTS_007"
Window Edit Options Help
frank@acsf03:/view/frank_test/vobs/tests/tests> end_work -v
end_work.pl: Info: Checking Config-Spec
end_work.pl: Error: You are using a Config-Spec "REL_TESTS_007" which is not fit
ted for developing!
frank@acsf03:/view/frank_test/vobs/tests/tests> █
```

Bild 30: Fehlermeldung von `end_work` bei falscher Config-Spec

5.4.3 Rebase eines neuen Releasestandes auf eine laufende Entwicklung

Bei schnellen Entwicklungszyklen ist es an der Tagesordnung, dass Entwicklungsaufträge vom Produktionsstand überholt werden. Damit die Entwicklung immer nahe am aktuellen Releasestand bleibt und damit später keine Merge-Konflikte auftreten, muss das Development-Team regelmäßig Releasestandsänderungen in die Entwicklung einfließen lassen.

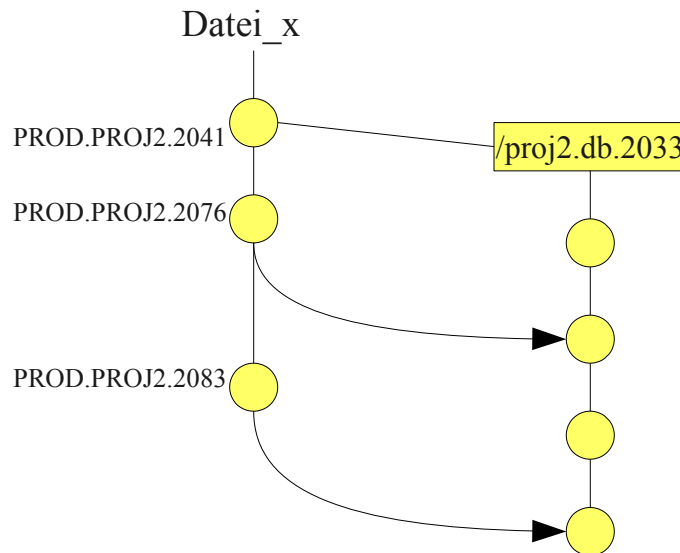


Bild 31: Beispiel des Rebase einer Entwicklung

Nach jedem Rebase wird die Config-Spec des Entwicklungsauftrags angepasst. Das ist notwendig, damit immer der releasenahe Stand plus der Entwicklung gesehen wird.

Würde die Config-Spec nicht umgesetzt werden, könnten beim Rebase Files gemerged werden, welche nicht Bestandteil der Entwicklung sind.

Im folgenden Bild wird dieses verdeutlicht.

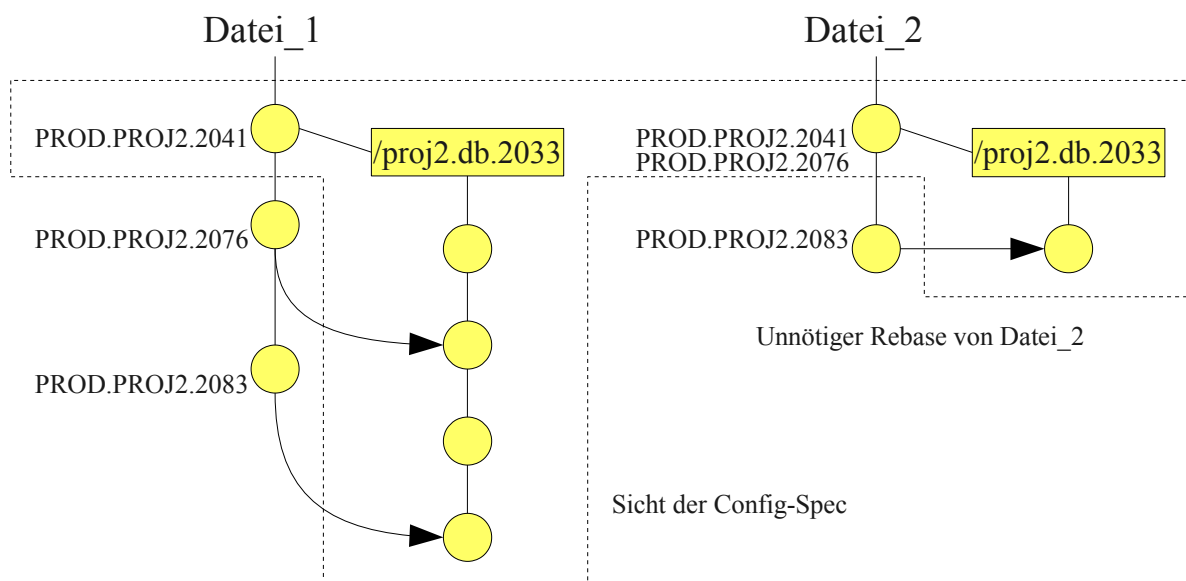


Bild 32: Beispiel eines fehlerhaften Rebase

Ein korrekter Rebase wird die Sicht wie folgt einstellen:

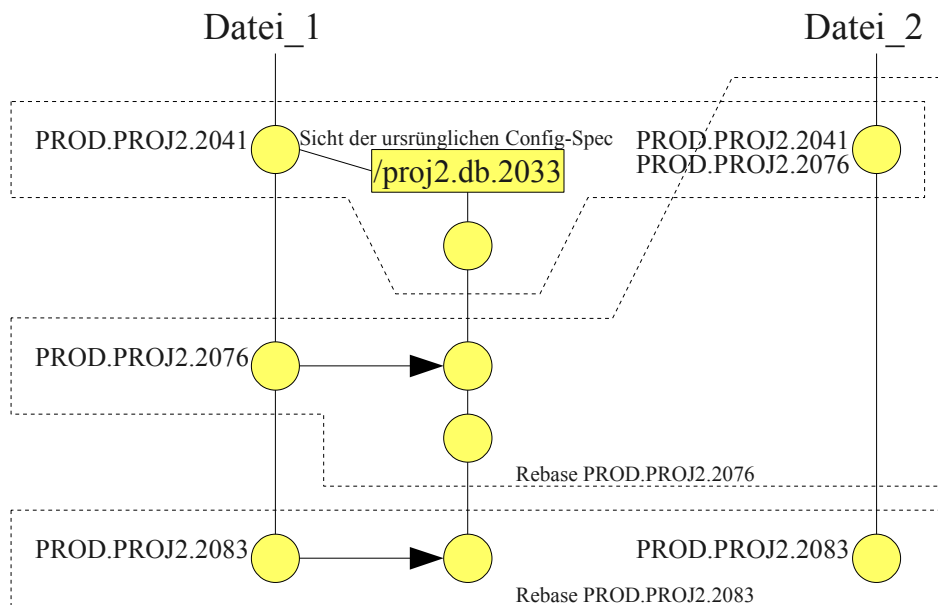


Bild 33: Beispiel eines korrekten Rebase

Da bei jedem Rebase eine neue Config-Spec aufgebaut und gesichert wird, ist es unbedingt nötig, dass alle am Entwicklungsauftrag beteiligten Entwickler über den Rebase informiert werden. Sie müssen dann jeweils die neue Config-Spec in ihrem Entwicklungsview setzen.

Spätestens vor der Fertigmeldung des Entwicklungsauftrags muss dieser, sofern notwendig, zwingend durchgeführt werden.

Das dazu notwendige Tool muss folgendes sicherstellen:

- Check, ob innerhalb des Entwicklungsauftrags noch ungesicherte Files vorliegen (vor dem Rebase muss alles eingecheckt werden)
- Integration des aktuellen Produktionsstandes
- Das Attribut des Branchtypes wird erweitert
- Aufbau einer neuen Entwickler-Config-Spec auf Basis des neuen Produktionsstandes

Das nachfolgende Bild verdeutlicht den Rebase eines Files während einer Entwicklung:

```

VIEW: "frank_test_2"  APPLICATION: "TEST"  SPEC: "PR_TEST_0016"
Window Edit Options Help
frank@acsf03:/view/frank_test_2/vobs/tests/tests> start_work -s -v -b pr_test_00
16 test_file2.c
start_work.pl: Info: Getting files to rebase
start_work.pl: Warning: A rebase is required for files:
/vobs/tests/tests/test_file.c

Your conditions:
[cr|pr]-branch: pr_test_0016
files to checkout: test_file2.c
Getting the files to start working
Checkout ./test_file2.c to start working
frank@acsf03:/view/frank_test_2/vobs/tests/tests>

```

oder

```

VIEW: "frank_test"  APPLICATION: "TEST"  SPEC: "PR_TEST_0016"
Window Edit Options Help
frank@acsf03:/view/frank_test/vobs/tests/tests> end_work -v
end_work.pl: Info: Checking Config-Spec
end_work.pl: Info: Spec based on "REL_TESTS_008"
and is for working on "pr_test_0016 "
end_work.pl: Info: Getting files to rebase
end_work.pl: Warning: A rebase is required for files:
/vobs/tests/tests/test_file.c
end_work.pl: Info: Setting label "PR_TEST_0016.3" on branch "pr_test_0016"
end_work.pl: Info: Searching checked-out files on branch "pr_test_0016"
end_work.pl: Warning: There are checked-out files in view "frank_test":
/view/frank_test/vobs/tests/tests/test_file.c
Shall I try to check-in them? [Y|n] [q]: y
end_work.pl: Info: Working on file "/view/frank_test/vobs/tests/tests/test_file.
c"
frank@acsf03:/view/frank_test/vobs/tests/tests> █

```

Bild 34: Hinweis auf einen Rebase bei Aufruf von start_work und end_work

Es ist also notwendig einen Rebase anzustoßen. Dieses erfolgt mit dem Aufruf:

```

rebase [-b branch]
  -b <branch>      The branch, you want to work on ( f.e. Cr### )
  -v                verbose.

```

Beispiel:

```

VIEW: "frank_test"  APPLICATION: "TEST"  SPEC: "PR_TEST_0016"
Window Edit Options Help
frank@acsf03:/view/frank_test/vobs/tests/tests> rebase -v
Please, enter the [cr|pr]-identifier for your work:
(it has to match regular expr.: ^(cr-test-\d{4}-\d{3}$)|(pr_test_\d{4,5}$) ): [q
]: pr_test_0016
Checking checked-out files
rebase.pl: Warning: There are checked-out files with branch pr_test_0016 in othe
r views than your view:
/view/frank_test/vobs/tests/./tests/test_file2.c
Please check! Exiting ....
frank@acsf03:/view/frank_test/vobs/tests/tests> █

```

Bild 35: Durchführung des Rebases mit ausgecheckten Files in anderen Views

In einer parallelen Entwicklung kann es durchaus sein, dass andere Entwickler noch Files ausgecheckt haben. In diesem Fall darf kein Rebase erfolgen, solange noch Checkouts bestehen.

Alle Entwickler sollten dann ein **end_work** starten und alles einchecken lassen.

Danach kann der Rebase durchgeführt werden:

```

VIEW: "frank_test"  APPLICATION: "TEST"  SPEC: "PR_TEST_0016"
Window Edit Options Help
frank@acsf03:/view/frank_test/vobs/tests/tests> rebase -v -b pr_test_0016
Checking checked-out files
Process rebase for "pr_test_0016" ..... Please wait!
Rebased files:
tests/tests/test_file.c
Files are still checked-out. - Call "end_work" again!
rebase.pl: Warning: Config-Spec for pr_test_0016 has been changed!
Please inform all developers!

frank@acsf03:/view/frank_test/vobs/tests/tests> end_work -v
end_work.pl: Info: Checking Config-Spec
end_work.pl: Info: Spec based on "REL_TESTS_009"
and is for working on "pr_test_0016 "
end_work.pl: Info: Getting files to rebase
end_work.pl: Info: Setting label "PR_TEST_0016.7" on branch "pr_test_0016"
end_work.pl: Info: Searching checked-out files on branch "pr_test_0016"
end_work.pl: Warning: There are checked-out files in view "frank_test":
/view/frank_test/vobs/tests/tests/test_file.c
Shall I try to check-in them? [Y|n] [q]: y
end_work.pl: Info: Working on file "/view/frank_test/vobs/tests/tests/test_file.
c"
frank@acsf03:/view/frank_test/vobs/tests/tests>

```

Bild 36: Korrekt durchgeführter Rebase

Der Rebase lässt alle gemergeden Files ausgecheckt. Damit hat der Entwickler die Möglichkeit, die Merge-Ergebnisse zu überprüfen.

Treten beim Rebase Mergekonflikte auf, wird automatisch das grafische Mergetool von

ClearCase gestartet (eine funktionierende X-Umgebung wird vorausgesetzt).

Sind alle Merges ok, kann per **end_work** wieder alles eing检eicht werden.

Nach dem Rebase stellt sich folgendes Bild dar:

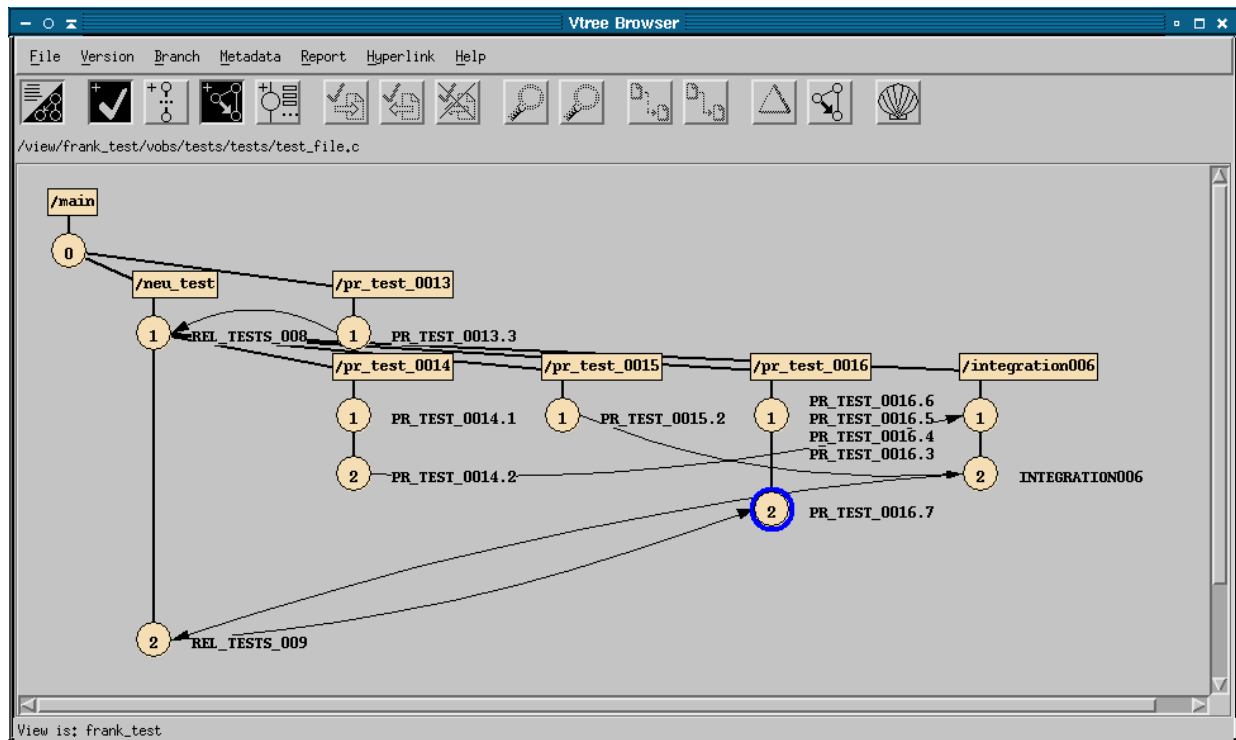
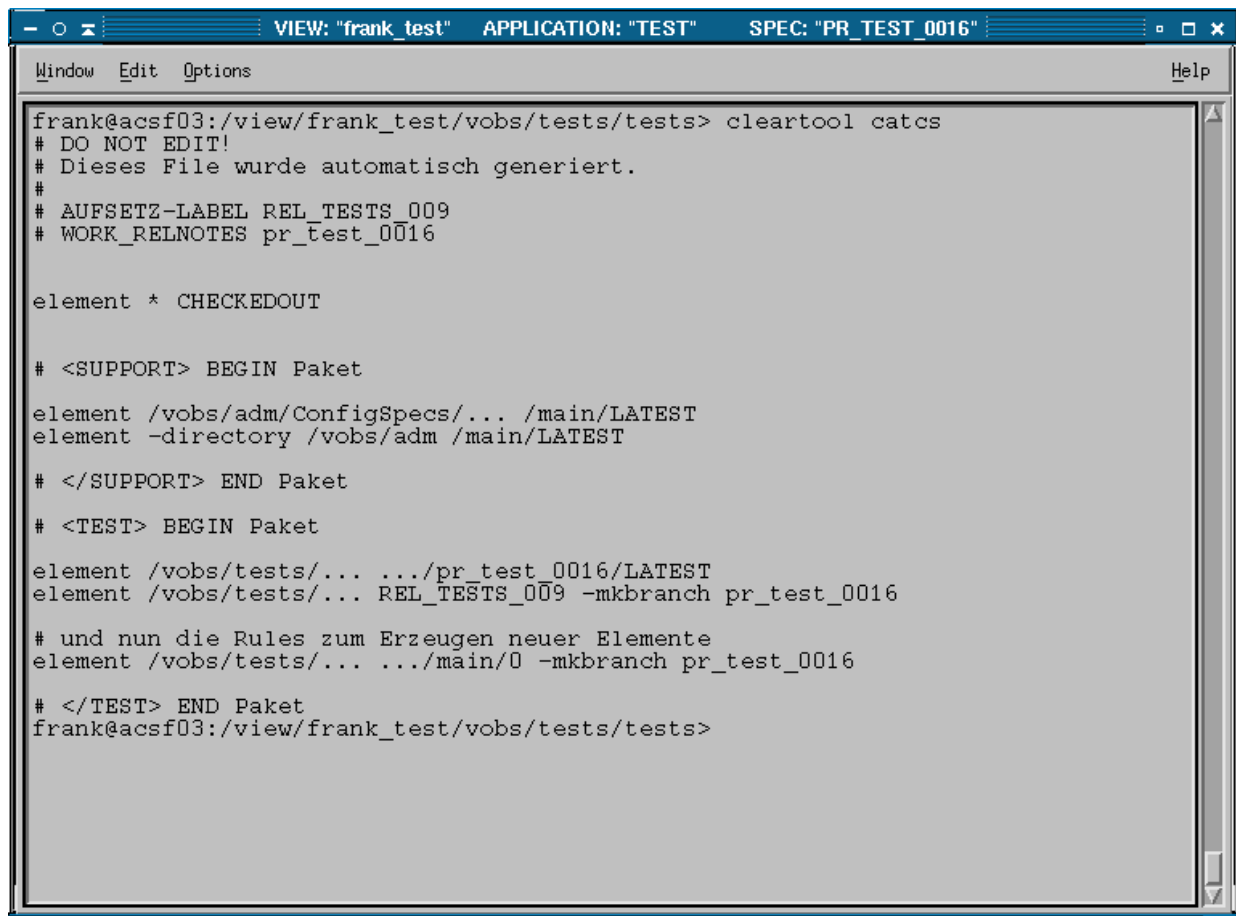


Bild 37: Versionsbaum eines Files nach dem Rebase

Alle Entwickler müssen über den Rebase informiert werden. Sie müssen ihre Entwickler-Config-Spec entweder mit **start_work** oder **setspec** neu setzen.

Die Config-Spec hat jetzt einen neuen Aufsetzpunkt und sieht folgendermaßen aus:



```
frank@acsf03:/view/frank_test/vobs/tests/tests> cleartool catcs
# DO NOT EDIT!
# Dieses File wurde automatisch generiert.
#
# AUFSETZ-LABEL REL TESTS_009
# WORK_RELNOTES pr_test_0016

element * CHECKEDOUT

# <SUPPORT> BEGIN Paket
element /vobs/adm/ConfigSpecs/... /main/LATEST
element -directory /vobs/adm /main/LATEST
# </SUPPORT> END Paket

# <TEST> BEGIN Paket
element /vobs/tests/... ../pr_test_0016/LATEST
element /vobs/tests/... REL_TESTS_009 -mkbranch pr_test_0016

# und nun die Rules zum Erzeugen neuer Elemente
element /vobs/tests/... ../main/0 -mkbranch pr_test_0016
# </TEST> END Paket
frank@acsf03:/view/frank_test/vobs/tests/tests>
```

Bild 38: Config-Spec nach dem Rebase

5.5 Deployment-Tools

Das Deployment-Team hat als Organ zwischen Entwicklung, Test-Team und Produktion qualitätssichernde Aufgaben wahrzunehmen.

Die Tools für das Deployment-Team müssen dieses widerspiegeln.

Da diese Tools eine gewisse Sonderstellung haben, werden auch sie auch unter den Rechten eines Integration-Users ausgeführt.

Dieses ist nicht nur aus Gründen der Sicherheit und der Abgrenzung zu den Entwicklern notwendig. Ein Hauptgrund ist der, dass die Entwicklungstätigkeit zwar als realer User im ClearCase durchgeführt wird, aber für alle von den Entwicklungstools angelegten Branch- und Labeltypen wird der Integration-User der Eigentümer. Damit ist es ihm möglich diese Objekte im Zuge des Releasebaues im ClearCase zu sperren (locken).

Mitglieder des Deployment-Teams loggen sich mittels

login2merge

als Integration-User ein.

5.5.1 Integration eines Entwicklungsauftrages

Ausgehend vom Entwicklungs-Workflow stehen nach den Modultests der Entwickler Integrationstest auf den unterschiedlichsten Testumgebungen an.

Dies machen natürlich nur Sinn, wenn mehrere Entwicklungsaufträge gemeinsam getestet werden sollen.

Es könne für gezielte Tests auf unterschiedlichen Testumgebungen die verschiedensten Kombinationen aus aktuellem Releasestand und den einzelnen Entwicklungsaufträgen zusammengestellt werden.

Es sind folgende Anforderungen zu erfüllen:

- Erzeugen des Integrations-Banches, falls dieser noch nicht existiert.
- Merge aller Dateien des zu mergenden Branches.
- Mitnahme möglichst vieler Informationen aus den zu mergenden Branches.

Zur Integration mehrerer Entwicklungsaufträge wird der Aufruf:

```
integrate.pl -t test_branch|hot -b branch [-ci] [-v]  
-t test_branch           The branch, you want to merge to ( f.e. Test### )  
                        or "hot" if you want to merge to a hotfix-branch.  
-b branch              The branch, you want to merge from ( f.e. Cr### ).  
-ci                   check in merged files  
-v                   verbose
```

genutzt.

Beispiel:

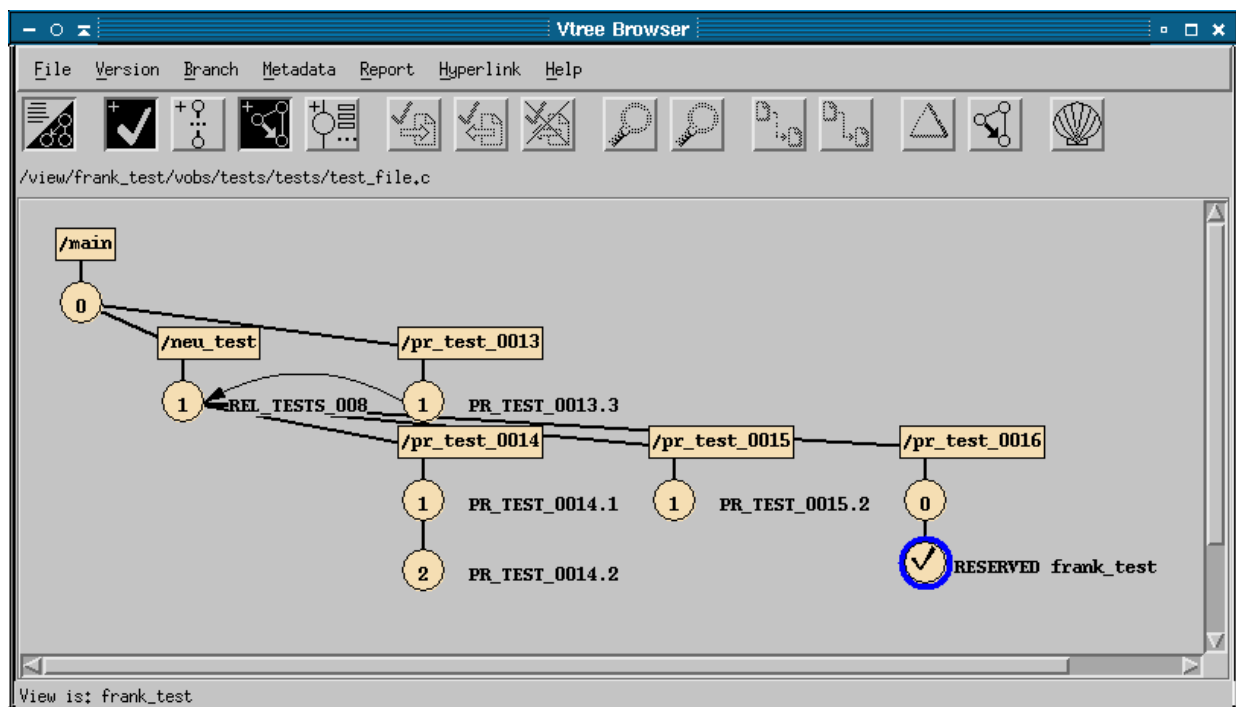


Bild 39: Ausgangsbeispiel zur Integration mehrerer Entwicklungsaufträge

Im obigen Beispiel sollen die Entwicklungsaufträge pr_test0014 und pr_test_0015 zusammen getestet werden. Dazu sind sie auf einen gemeinsamen Branch zu mergen:

```

VIEW: "ccadm_test"  APPLICATION: "TEST"  SPEC: "INTEGRATION006"
Window Edit Options Help
ccadm@acsf03:/view/ccadm_test/vobs/tests/tests> integrate -v -ci -b pr_test_0014
-t int_1
integrate.pl: Info: Checking Config-Spec
integrate.pl: Info: Source-Spec based on "REL_TESTS_008"
and is for working on "pr_test_0014 "
Please, enter the test-branch-identifier for your work
(it has to match regular expr.: (^(\cr-test-\d{4}-\d{3}$)|(pr_test_\d{4,5}$))|(^i
ntegration\d{3}$) ): [q]: integration006
integrate.pl: Info: Config-Spec set
integrate.pl: Info: Merging branch "pr_test_0014" labeled with "PR_TEST_0014.2"
Merged files from branch pr_test_0014:
/view/ccadm_test/vobs/tests/tests/test_file.c

!!!! Attention: View-Configuration has changed !!!!

ccadm@acsf03:/view/ccadm_test/vobs/tests/tests> integrate -v -ci -b pr_test_0015
-t integration006
integrate.pl: Info: Checking Config-Spec
integrate.pl: Info: Source-Spec based on "REL_TESTS_008"
and is for working on "pr_test_0015 "
integrate.pl: Info: Config-Spec set
integrate.pl: Info: Merging branch "pr_test_0015" labeled with "PR_TEST_0015.2"
Merged files from branch pr_test_0015:
/view/ccadm_test/vobs/tests/tests/test_file.c
ccadm@acsf03:/view/ccadm_test/vobs/tests/tests>

```

Bild 40: Integration von zwei Entwicklungsaufträgen

Im ersten Aufruf wurde der Branch zur Integration falsch angegeben. Demzufolge wird das Tool

zur korrekten Eingabe auffordern. Weiterhin erstellt das Tool eine Config-Spec und speichert diese. Sie kann dann jederzeit wieder gesetzt werden

Nach dem Merge stellt sich folgende Situation dar:

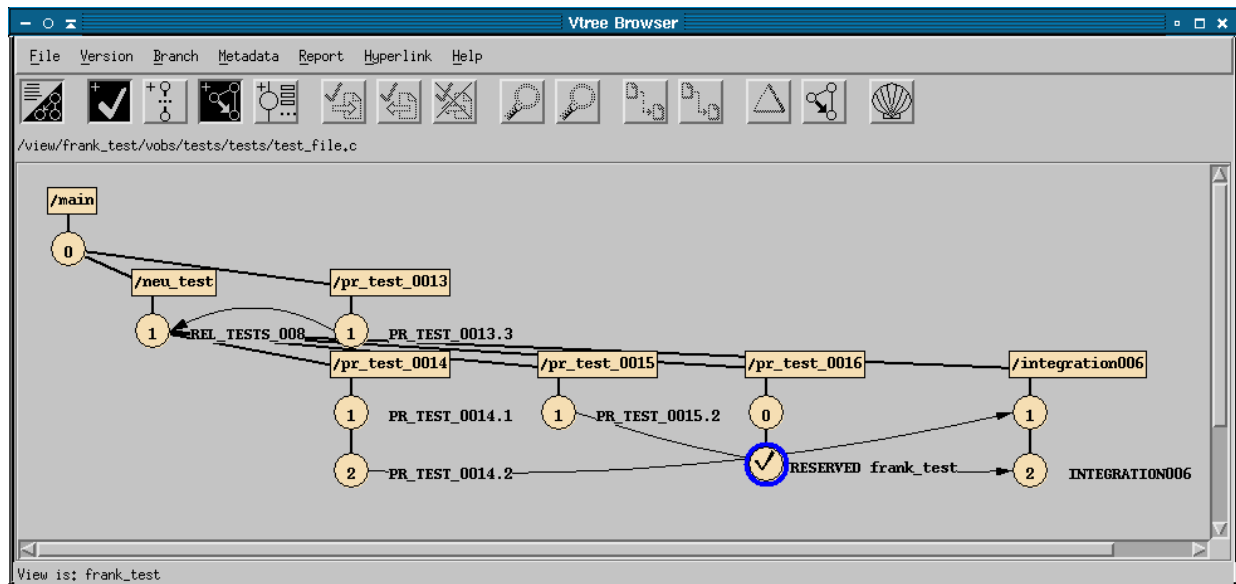


Bild 41: Ergebnis der Integration

Mit dieser Integration können jetzt Testinstallationen gemacht und die notwendigen Integrationstests durchgeführt werden.

Es ist weiterhin möglich, Entwicklungen untereinander zu integrieren.

Die Mergeergebnisse müssen allerdings korrekt gelabelt werden.

Dazu hat man entweder die Möglichkeit, die Integration ohne automatischen checkin durchzuführen und **end_work** das Labeln zu überlassen oder aber **end_work** für alle gemergeden Files explizit aufzurufen.

Zur Integration von Hotfixen ist statt des Integrationsbranches das Schlüsselwort „hot“ anzugeben. Damit wird eine Sonderbehandlung eingeschaltet:

```

VIEW: "ccadm_test"  APPLICATION: "TEST"  SPEC: "HOTFIX_PROD_TESTS_008"
Window Edit Options Help
ccadm@acsf03:/view/ccadm_test/vobs/tests/tests> integrate -v -t hot -b pr_test_0
020 -ci
integrate.pl: Info: Checking Config-Spec
integrate.pl: Info: Source-Spec based on "PROD_TESTS_008"
and is for working on "pr_test_0020 "
To make your config-spec for hotfix-delivery do:
call ccredit -l PROD_TESTS_008
edit config-spec using label "HOTFIX_PROD_TESTS_008.3"
call cssave and give the correct name to that config-spec
integrate.pl: Info: Config-Spec set
integrate.pl: Info: Merging branch "pr_test_0020" labeled with "PR_TEST_0020.1"
Merged files from branch pr_test_0020:
/view/ccadm_test/vobs/tests/tests/test_file.c
ccadm@acsf03:/view/ccadm_test/vobs/tests/tests>

```

Bild 42: Integration eines Hotfixes

Alle gemergeden Files bekommen ein eindeutiges Label:

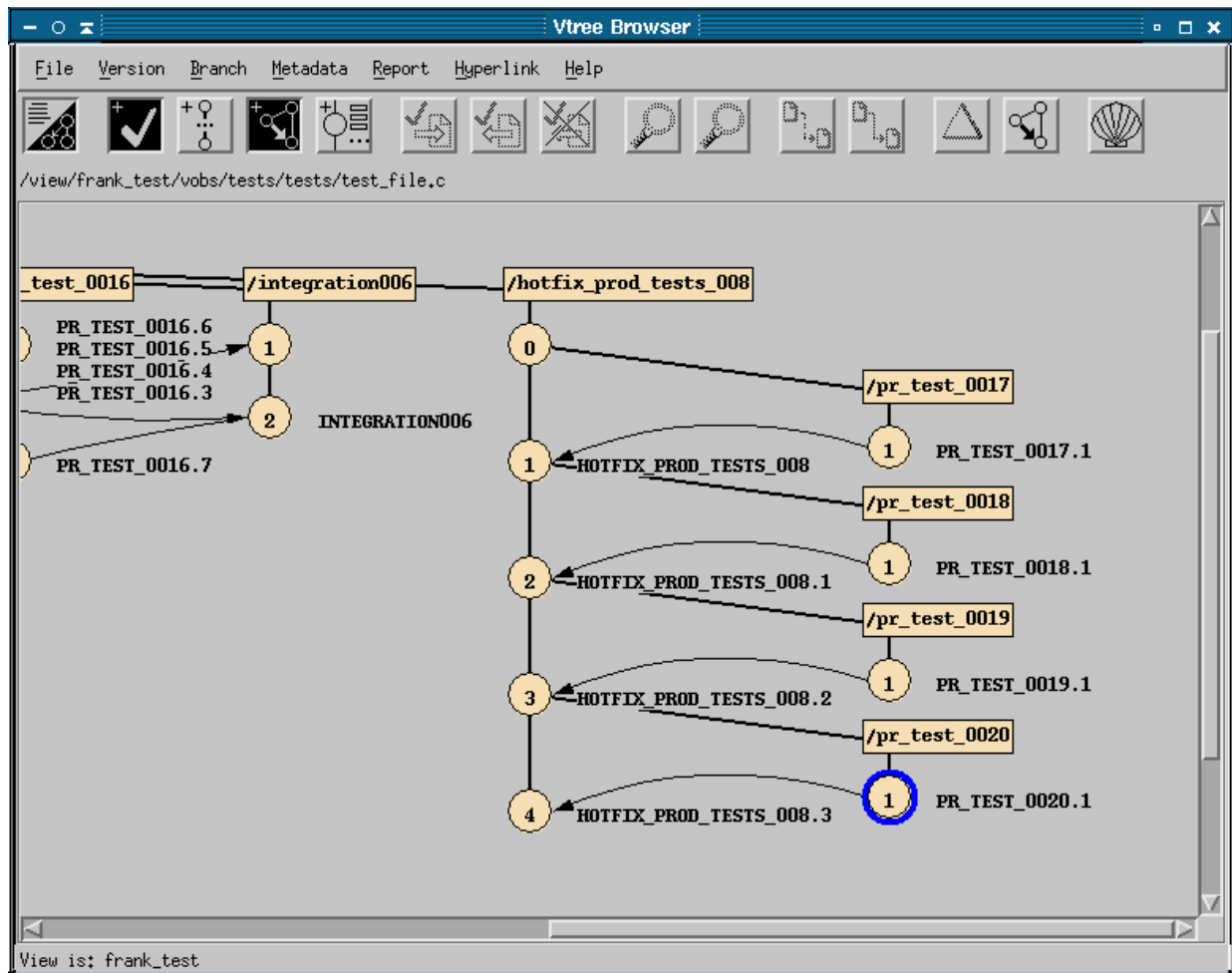


Bild 43: Ergebnis der Integration eines Hotfixes

Für eine Auslieferung des gemergeden Hotfixes wird nun noch eine gesicherte Config-Spec benötigt.

Dazu werden die Tools **csedit** / **cssave** benutzt.

Beispielsweise wird die Config-Spec PROD_TESTS_008 in folgender Weise erweitert:

```
# SPEC-LABEL:      PROD_TESTS_008
# APP_ANW:        TEST
```

```
# allgemeines Paket zur Sicht auf die Config-Specs (do not edit!)
# <support> BEGIN INCLUDE
element    /vobs/adm/ConfigSpecs/...    /main/LATEST
element    -directory /vobs/adm        /main/LATEST
# </support> END INCLUDE
```

```
# Paket TEST
# <test>
element    /vobs/tests/...              HOTFIX_PROD_TESTS_008.3 -nocheckout
element    /vobs/tests/...              REL_TESTS_008 -nocheckout
# </test>
```

Damit wird der Produktionsstand zuzüglich der Hotfixe gesehen.
Die Config-Spec ist dann unter einem geeigneten Namen abzuspeichern.
Mit dieser Hotfix-Config-Spec kann dann die Lieferung vorgenommen werden.

Natürlich darf auch nicht vergessen werden, die Hotfixes in das normale Release zu integrieren.

5.5.2 Erzeugen eines neuen Release-Standes

Wenn alle Tests eines oder mehrerer Entwicklungsaufträge erfolgreich gelaufen sind und vom Releasemanagement der Auftrag zum Erstellen eines neuen Releases erteilt wird, werden entweder der geforderte Entwicklungsbranch oder der Integrationsbranch auf den Releasebranch gemerged.

Es sind folgende Anforderungen zu erfüllen:

- Merge aller Dateien des zu mergenden Branches.
- Erzeugen des Release-Branches, falls dieser noch nicht existiert.
- Mitnahme möglichst vieler Informationen aus den zu mergenden Branches.
- Locken aller Quellbranches, um eine weitere Bearbeitung durch den Entwickler zu unterbinden.

Dazu dient der Aufruf:

```
release -r label -b branch [-ci] [-v]
-r label           The label, you want to give to the release.
-b branch         The branch, you want to merge from ( f.e. Cr### ).
-ci              check in merged files
-v              verbose
```

Beispiel:

```
VIEW: "ccadm_test" APPLICATION: "TEST" SPEC: "MERGE_REL_TESTS_009"
Window Edit Options Help
ccadm@acsf03:/view/ccadm_test/vobs/tests/tests> release -v -ci -r REL_TESTS_009
-b integration006
release.pl: Info: Checking Config-Spec
release.pl: Info: Source-Spec based on "REL_TESTS_008"
and is for working on "INTEGRATION006 "
release.pl: Info: Config-Spec set
release.pl: Info: Merging branch "integration006" labeled with "INTEGRATION006"
Merged files from branch integration006:
/view/ccadm_test/vobs/tests/tests/test_file.c

!!!! Attantion: View-Configuration has changed !!!!
ccadm@acsf03:/view/ccadm_test/vobs/tests/tests> █
```

Bild 44: Mergen eines Entwicklungsbranches auf den Release-Branch

Im obigen Beispiel handelt es sich um den direkten Merge eines Entwicklungsbranches auf den Releasebranch.

Wenn das Tool zum ersten Mal für dieses neue Release aufgerufen wird, werden alle auf dem derzeitigen Releasestand sichtbaren Files und Verzeichnisse mit dem neuen Release-Label

versehen.

Anschließend wird eine Merge-Config-Spec erzeugt und diese gesetzt.

Danach erfolgt der Merge des geforderten Branches. Es werden nur Files und Verzeichnisse berücksichtigt, welche den Entwicklungsbranch haben und das letzte Update-Label tragen.

Alle auf den Releasebranch integrierten Branches werden gesperrt.

Der Versionsbaum eines Files sieht anschließend z.B. wie folgt aus:

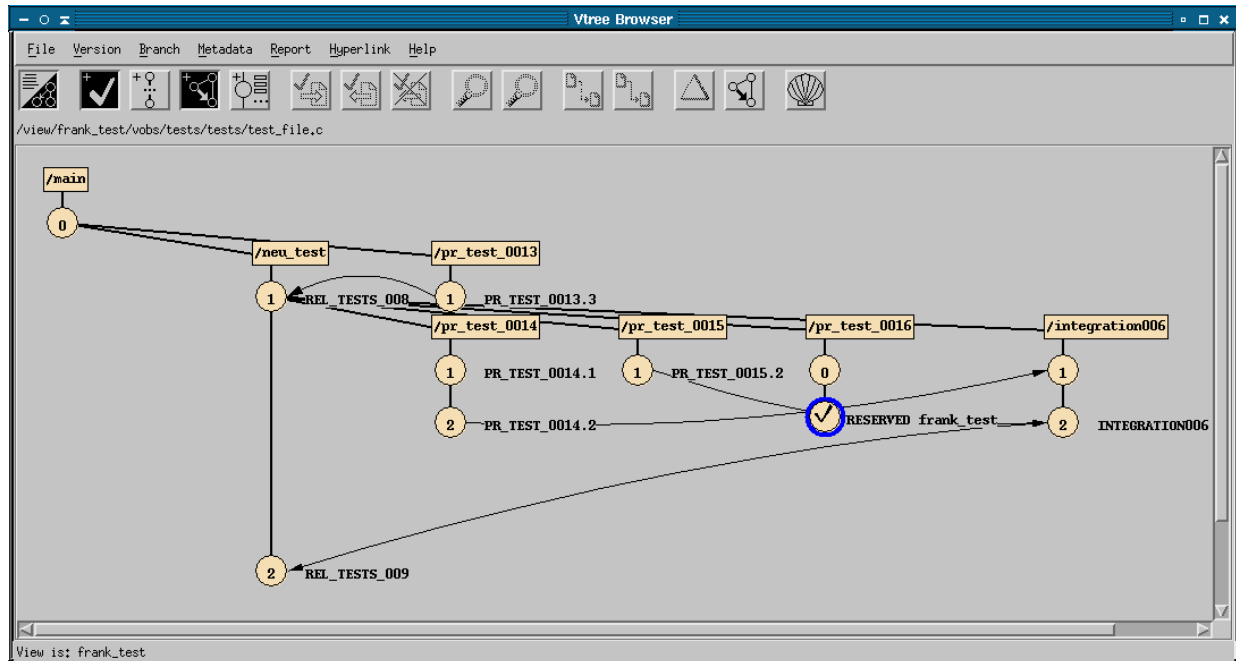
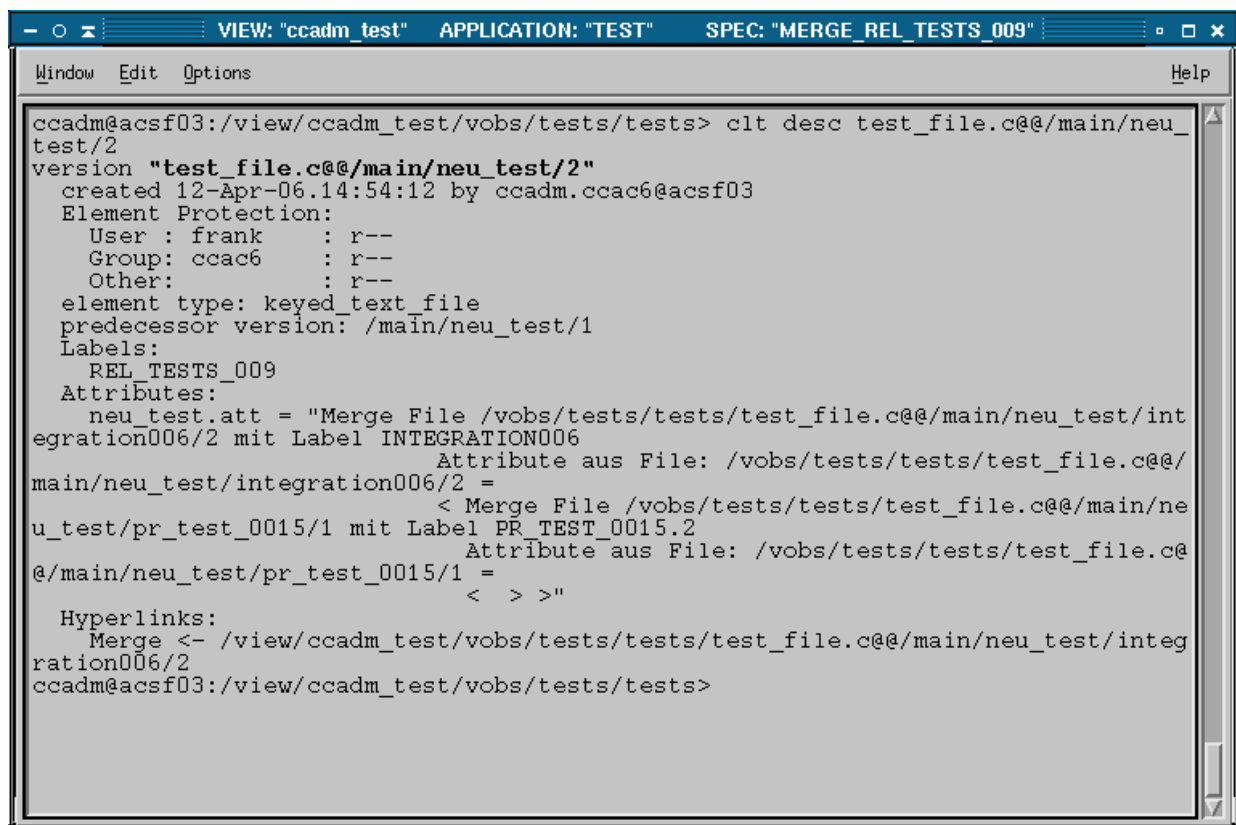


Bild 45: Versionsbaum einer Datei nach dem Release-Merge

Die auf den Releasesstand gebrachte Version der Datei wird mit einem Attribut versehen:



```
VIEW: "ccadm_test" APPLICATION: "TEST" SPEC: "MERGE_REL_TESTS_009"
Window Edit Options Help
ccadm@acsf03:/view/ccadm_test/vobs/tests/tests> clt desc test_file.c@@/main/neu_
test/2
version "test_file.c@@/main/neu_test/2"
  created 12-Apr-06.14:54:12 by ccadm.ccac6@acsf03
  Element Protection:
    User : frank      : r--
    Group: ccac6      : r--
    Other:              : r--
  element type: keyed_text_file
  predecessor version: /main/neu_test/1
  Labels:
    REL_TESTS_009
  Attributes:
    neu_test.att = "Merge File /vobs/tests/tests/test_file.c@@/main/neu_test/int
egration006/2 mit Label INTEGRATION006
                    Attribute aus File: /vobs/tests/tests/test_file.c@@/
main/neu_test/integration006/2 =
                    < Merge File /vobs/tests/tests/test_file.c@@/main/ne
u_test/pr_test_0015/1 mit Label PR_TEST_0015.2
                    Attribute aus File: /vobs/tests/tests/test_file.c@
@/main/neu_test/pr_test_0015/1 =
                    < > >"
  Hyperlinks:
    Merge <- /view/ccadm_test/vobs/tests/tests/test_file.c@@/main/neu_test/integ
ration006/2
ccadm@acsf03:/view/ccadm_test/vobs/tests/tests>
```

Bild 46: Attribute auf der Zieldatei

Der Releasebranch wird auch mit einem Attribut versehen und die entsprechenden Informationen werden ein gepflegt:


```

VIEW: "ccadm_test"  APPLICATION: "TEST"  SPEC: "MERGE_REL_TESTS_009"
Window  Edit  Options  Help

23.03.2006 12:48:34 Merge Branch integration004 mit Label IN
TEGRATION004 auf Label REL_TESTS_006
  Attribute aus Branchtype: integration004 =
  < 23.03.2006 12:43:24 Merge Branch pr_test_0008 mit Label
PR_TEST_0008.3 auf Label INTEGRATION004
  Attribute aus Branchtype: pr_test_0008 =
  < 23.03.2006 12:33:43 Aufsetzpunkt REL_TESTS_005 > >
23.03.2006 13:18:46 Merge Branch integration005 mit Label IN
TEGRATION005 auf Label REL_TESTS_007
  Attribute aus Branchtype: integration005 =
  < 23.03.2006 13:12:56 Merge Branch pr_test_0009 mit Label
PR_TEST_0009.1 auf Label INTEGRATION005
  Attribute aus Branchtype: pr_test_0009 =
  < 23.03.2006 13:04:30 Aufsetzpunkt REL_TESTS_006 > >
11.04.2006 13:19:07 Merge Branch pr_test_0013 mit Label PR_T
EST_0013.3 auf Label REL_TESTS_008
  Attribute aus Branchtype: pr_test_0013 =
  < 11.04.2006 09:05:25 Aufsetzpunkt REL_TESTS_007 >
12.04.2006 14:54:14 Merge Branch integration006 mit Label IN
TEGRATION006 auf Label REL_TESTS_009
  Attribute aus Branchtype: integration006 =
  < 12.04.2006 14:40:02 Merge Branch pr_test_0014 mit Label
PR_TEST_0014.2 auf Label INTEGRATION006
  Attribute aus Branchtype: pr_test_0014 =
  < 12.04.2006 13:54:56 Aufsetzpunkt REL_TESTS_008 >
12.04.2006 14:44:31 Merge Branch pr_test_0015 mit Label
PR_TEST_0015.2 auf Label INTEGRATION006
  Attribute aus Branchtype: pr_test_0015 =
  < 12.04.2006 14:17:37 Aufsetzpunkt REL_TESTS_008 > >
ccadm@acsf03:/view/ccadm_test/vobs/tests/tests>

```

Bild 47: Attribute auf dem Releasebranch

Es ist möglich, **release** mehrfach aufzurufen.

Diese sollte aber nur sehr gezielt angewendet werden, da in diesem Fall aus Sicht der Qualitätssicherung das neu entstandene Release komplett neu getestet werden muss.

Dazu wird in den seltensten Fällen genügend Zeit vorhanden sein.

In der Regel sollte ein Merge reichen. Dieser kann dann zur Sicherheit nochmal einem Regression-Test unterzogen werden.

Falls es doch nötig ist, mehrfach auf Release zu mergen, wird das Release-Label immer auf den aktuellen Stand verschoben.

Wird das neu entstandene Release für in Ordnung befunden, muss dafür noch eine gültige Release-Config-Spec erzeugt werden. Dazu werden die Tools **csedit** und **cssave** genutzt.

Ab diesem Zeitpunkt ist das neue Release Basis jeder weiteren Standard-Entwicklung.

5.5.3 Auswertungen über den Releasestand

Selbstverständlich muss es auch möglich sein, dass sich das Releasemanagement bzw. das Deployment-Team schnell einen Überblick über den Stand des Projektes zu verschaffen.

Dazu wird das Tool

```

show_integration.pl [-b branch] [-v]
  -b branch      The branch, you want to you want to have information on
                 (default: latest Release)

```

-v**detailed list**

bereitgestellt.

Beispiel:

Bild 48: Kurzinformation über den Releasebranch

Die Informationen werden aus dem Attribut des optional zu übergebenen Branchtype gewonnen.

Die detaillierte Auflistung liefert auch rekursive Informationen sowie eine Liste von Entwicklungen, welche noch nicht auf den Releasebranch gebracht wurden. Falls die integrierten Entwicklungsbranches eine höhere Updatenummer haben, als die beim Merge verwendete, wird eine Warnung zu diesem Branch erzeugt.

Beispiel:

Bild 49: Ausführliche Information über den Releasebranch

6 Zusammenfassung

Die oben vorgestellten Verfahren und Tools sind ein Vorschlag, wie man den Workflow einer Entwicklung innerhalb einer Versionsverwaltung abbilden kann.

Die Möglichkeiten der gegenseitigen Integration und des Rebases gewährleisten zum einen eine Entwicklungstätigkeit, die nahe am aktuellen Geschehen ist, zwingt die Development-Teams allerdings auch nicht, eine stabile Entwicklungsumgebung durch vorzeitige Merges zu gefährden.

Durch die konsequente Mitnahme und Pflege der Attribute an Files, Label- und Branchtypen ist es möglich sehr komfortabel den Entwicklungsfluss zu verfolgen.

Wie bereits erwähnt ist dieser Text auf Grundlage eines realisierten Projektes in einer ClearCase-Umgebung entstanden.

Die Einbindung anderer Versionsverwaltungssysteme ist vorbereitet und sollte schnell zu implementieren sein.

7 Abkürzungsverzeichnis

CR	Change Request (in der Regel eine Weiterentwicklung auf Grundlage neuer Anforderungen)
PR	Problem Report (generiert durch einen Incident)