Ein einfaches Versions-Kontroll-System für SAS Programme in der klinischen Entwicklung

Matthias Post
Accovion GmbH
Helfmann-Park 10
65760 Eschborn
matthias.post@accovion.com

Zusammenfassung

Die Notwendigkeit der Versions-Kontrolle ergibt sich für ein in der klinischen Entwicklung tätiges Unternehmen aus den regulatorischen Anforderungen, den Lebenszyklus aller verwendeten SAS-Programme lückenlos zu dokumentieren. Wurde ein Programm für eine Analyse verwendet, muss die entsprechende Programmversion zum Zweck der Reproduzierbarkeit dauerhaft archiviert werden. Darüber hinaus muss sichergestellt werden, dass nur validierte Programme in Analysen verwendet werden.

Um diese Anforderungen zu erfüllen wurde ein einfaches, aber effizientes Versions-Kontroll-System entwickelt, das aus folgenden Komponenten besteht:

1. Eine auf diese Vorgaben zugeschnittene Linux-Verzeichnisstruktur,
2. eine Sammlung von UNIX-Command-Line-Tools, die Validierungsstatus und Zugriffsberechtigungen der SAS-Programme verwaltet sowie Programmversionen archiviert und
3. eine Oracle-Datenbank, in der alle Versionsänderungen protokolliert werden.

Das Ziel dieses Beitrags ist es zu demonstrieren, wie bei Accovion ohne Anschaffung von Zusatzsoftware ein effektives System zur Versions-Kontrolle implementiert wurde.

Schlüsselwörter: Versionskontrolle

1 Einführung

Als Contract Research Organisation (CRO) ist die Accovion GmbH im Auftrag ihrer Kunden für die Planung, Durchführung und Auswertung klinischer Studien zuständig. Für die statistische Datenanalyse kommen dabei SAS-Programme zum Einsatz, die z. B. zur Erstellung von standardisierten Datensätzen (im CDISC-SDTM oder ADaM Format), Tabellen, Listen und Grafiken verwendet werden.

Bevor ein Programm einen Output erzeugt, der dann an den Auftraggeber weitergegeben und später eventuell bei der Zulassungsbehörde eingereicht wird, muss es einer formalen Validierung unterzogen werden. Diese beinhaltet üblicherweise einen Testlauf des Programmes, die Durchsicht des Programmcodes sowie die Verifikation von Logfile und Output. Darüber hinaus kann auch eine unabhängige Zweitprogrammierung mit anschließendem Vergleich der beiden Outputs erforderlich sein.

Für eventuelle Rückfragen durch den Auftraggeber oder die Zulassungsbehörden ist es erforderlich, dass der Lebenszyklus eines Programmes lückenlos dokumentiert werden kann.
Des Weiteren besteht die Anforderung, dass finale Analysen jederzeit aus den Source-Daten reproduziert werden können. Somit besteht die Pflicht, neben den Analyseergebnissen und den zugehörigen Daten auch die verwendeten Programme dauerhaft zu archivieren.

Darüber hinaus existieren weitere Anforderungen aus der Softwareentwicklung, die den Einsatz eines geeigneten Versionierungssystems sinnvoll erscheinen lassen. Programmierer müssen in der Lage sein, auf ältere Programmversionen zugreifen zu können, Vergleiche zwischen Programmversionen anstellen zu können oder auch einfach einen Zwischenstand so ablegen zu können, dass er wiederauffindbar ist.

Aus den genannten Gründen war es unerlässlich, ein auf diese Bedürfnisse zugeschnittenes Versions-Kontroll-System zu implementieren.

2 Die SAS-Architektur bei Accovion

Im Mittelpunkt unserer Programmier-Umgebung steht ein LINUX-Server mit einer validierten SAS-Installation (s. Abbildung 1). Auf diesem Server befindet sich auch unser zentrales Filesystem auf dem die SAS-Datenbestände, Outputs, Programme und zugehörige Bibliotheken abgelegt sind.
Auf einem Windows-Citrix-Server betreiben wir eine zweite SAS-Instanz, die über SAS/Connect mit der SAS-Installation auf dem LINUX-Server verbunden ist. So besteht die Möglichkeit, Programme auf der etwas komfortableren Windows-Umgebung zu entwickeln, um sie später per Remote Submit auf dem Linux Server produktiv laufen zu lassen. Viele Benutzer arbeiten aber auch während der Programmentwicklung direkt auf dem LINUX-Server.
Über einen SAMBA-Share wird das LINUX-Filesystem auf den Windows-Server gemappt, so dass die Benutzer auch von dort auf alle Programme und Daten zugreifen können.
Die SAS-Umgebung unter Windows dient nur als Entwicklungsumgebung, produktive Programmläufe werden grundsätzlich im Batch-Modus auf dem LINUX-Server ausgeführt.

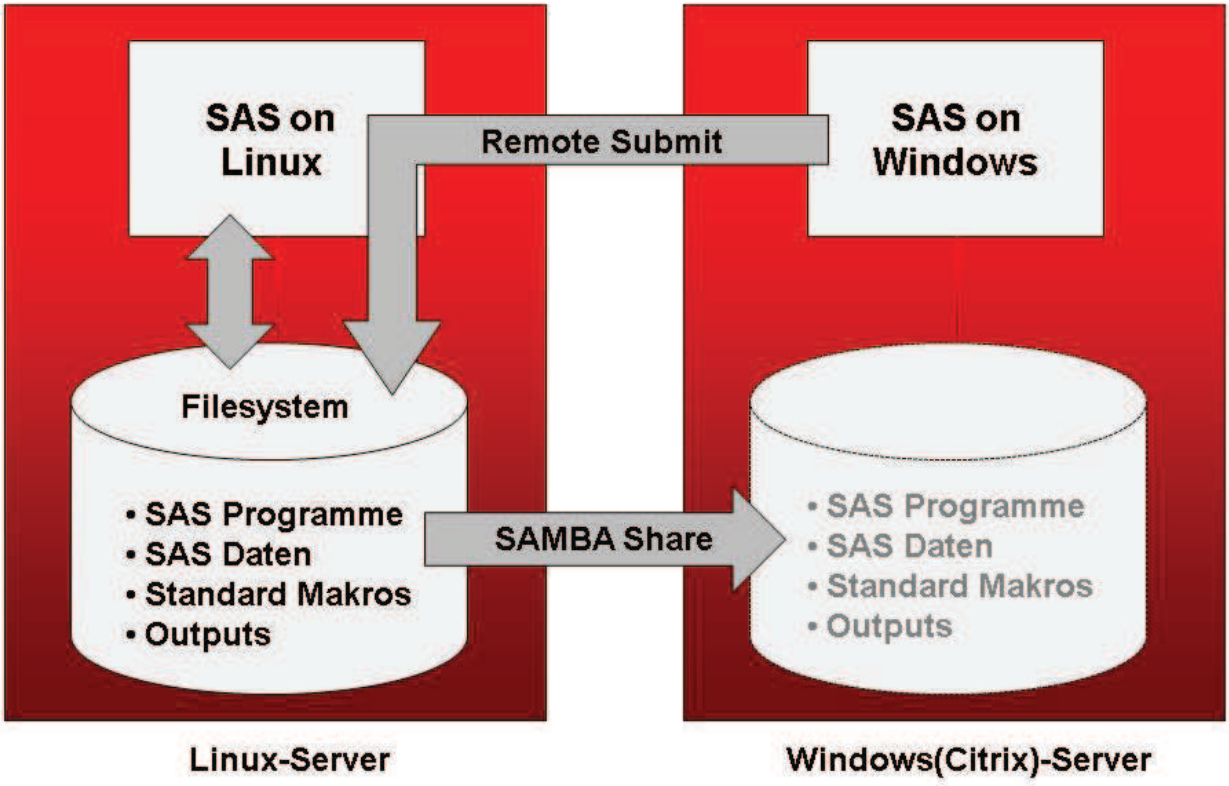

Abbildung 1: SAS-Architektur bei Accovion

3 Validierungs-Ebenen für SAS-Programme

Zur Umsetzung des angewandten Validierungskonzeptes nutzen wir eine Verzeichnisstruktur, die in drei Ebenen unterteilt ist. Programme und Makros werden zunächst auf der Entwicklungsebene (im Folgenden als DEV-Ebene bezeichnet) erstellt. Nach Fertigstellung wird das Programm auf die Validierungsebene (VAL-Ebene) verschoben. Dort erfolgt eine Validierung durch einen zweiten Programmierer oder Statistiker nach vorher festgelegten Regeln. Im Falle einer erfolgreichen Validierung wird das Programm dann auf die produktive Ebene (PROD-Ebene) verschoben, wo es zur Erstellung von validierten Outputs genutzt wird.

Falls bei der Validierung Fehler bemerkt worden sind, die eine Änderung des Programmcodes erfordern, muss das Programm dazu erst wieder auf die DEV-Ebene verschoben werden.
Auf den VAL- und PROD-Ebenen dürfen keine Änderungen an Programmen durchgeführt werden. Deshalb wird den Programmen beim Verschieben auf diese Ebenen auch das Schreibrecht entzogen. Diesen Vorgang bezeichnen wir deshalb auch als LOCK. Der umgekehrte Weg von der VAL- oder PROD-Ebene auf die DEV-Ebene bezeichnen wird als UNLOCK, hier werden die Schreibrechte (für den Programmbesitzer) wieder gesetzt.

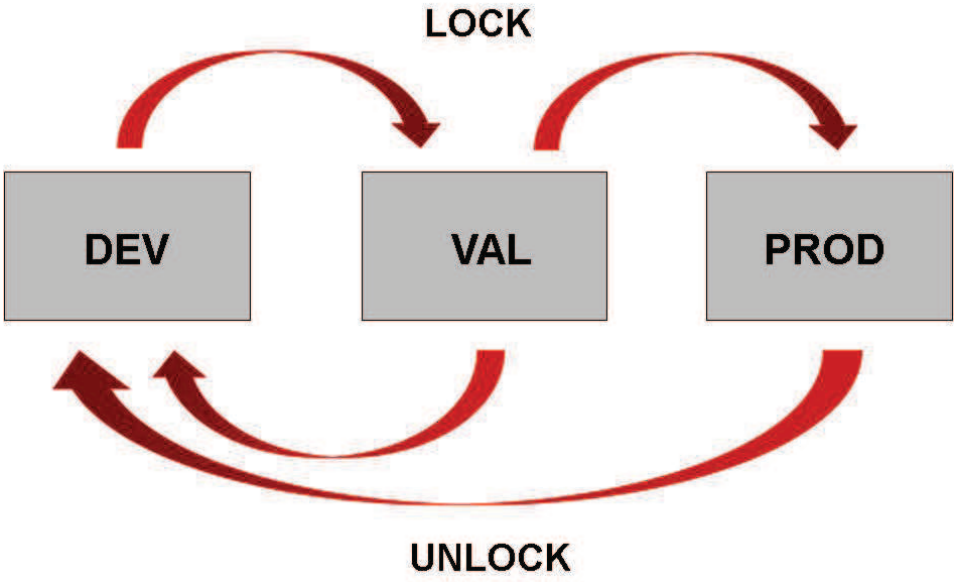

Abbildung 2: Die drei Validierungsebenen

Die Validierungsebenen sind innerhalb des Filesystems als Directories implementiert, die unter der jeweiligen klinischen Studie aufgehängt sind.
Unter jeder der drei Validierungsebenen befinden sich weitere Unterverzeichnisse, in denen die Programme nach Typ, Zweck und weiteren Kriterien organisiert sind.
Diese Unterverzeichnisstruktur ist auf allen Validierungsebenen identisch. So wird zum Beispiel ein Programm, was sich im Verzeichnis

```
/project/study/DEV/reports/safety
```

befindet, bei einem LOCK nach

```
/project/study/VAL/reports/safety
```

verschoben.

4 Das Versions-Kontroll-System

Für die Entwicklung unseres auf den beschriebenen Validierungsebenen basierenden Versions-Kontroll-Systems standen zunächst die folgenden Anforderungen im Vordergrund:

- Verschieben der Programme zwischen den verschiedenen Validierungsebenen
- Setzen der entsprechenden Zugriffsrechte
- Automatisches Erstellen eines Backups von einem Programm, sobald sich dessen Status ändert (z.B. durch einen LOCK)
- Vergabe von eindeutigen Versionsnummern für diese Backup-Kopien.
- Protokollierung aller Bewegungen zwischen den Validierungsebenen in einer Datenbank

Die genannten Funktionen wurden auf unserem Linux-Server in Form von Command-Line Tools umgesetzt. Diese Anforderung ergab sich daraus, dass unsere statistischen

Programmierer außerhalb der SAS-Session grundsätzlich direkt auf Command-Line Ebene arbeiten, wo auch weitere, ähnlich aufgebaute Funktionen unserer integrierten SAS-Umgebung zur Verfügung stehen.
Darüber hinaus ist natürlich auch ein Aufruf aus der SAS-Session durch das X-Kommando denkbar. Für die Entwicklung einer grafischen Benutzeroberfläche sahen wir bis jetzt keinen Bedarf.

Bei der Entwicklung des Versions-Kontroll-Systems kam ausschließlich vorhandene Software zum Einsatz.

Da ein Großteil der Funktionalität aus Operationen auf dem LINUX-Filesystem besteht, wurde die Programmlogik in Form von BASH-Shell-Scripts implementiert. Für das Schreiben der Records in die History-Datenbank wird in den Shell-Scripts ein Insert-Statement generiert und per SQLPLUS-Client an die ORACLE-Datenbank geschickt. Die Datenbank (die aus einer einzelnen Tabelle besteht) wurde in einer unserer vorhandenen ORACLE-Instanzen integriert, hätte aber theoretisch auch in jeder anderen relationalen Datenbank untergebracht werden können.

4.1 Backup-Kopien und Versionsnummern

Alle Versionierungs-Funktionen, außer UNLOCK, erstellen bei Ausführung automatisch eine Backup-Kopie des jeweiligen SAS-Programmes. Diese Backups werden in einer dedizierten History-Verzeichnisstruktur pro Studie abgelegt. Die Unterverzeichnisse des ursprünglichen Pfadnamens eines Programmes werden analog in diesem History-Verzeichnis abgebildet, wobei pro Programm nochmal ein Verzeichnis angelegt wird, das alle Backup-Kopien dieses Programmes enthält. Dieses Unterverzeichnis heißt dann so wie das Programm, jedoch ohne die Extension .sas.

Beispiel:
Alle Backups des Programmes

`/project/study/`**`DEV`**`/reports/safety/lab_listing.sas`

werden im History-Directory

`/project/study/`**`HISTORY`**`/reports/safety/`**`lab_listing`**

abgelegt.

Im History-Verzeichnis noch nicht vorhandene Unterverzeichnisse werden ggf. beim ersten Backup, das für ein Programm erstellt wird angelegt.

Die Backup-Kopien erhalten eine dreistufige Versionsnummer, die in den ursprünglichen Dateinamen integriert wird:

<filename>`.sas` → *<filename>*`_v`*PP.VV.DD*`.sas`

- Die unterste Teilnummer *DD* wird um 1 erhöht, wenn ein neues Backup erstellt wird, das Programm sich aber weiterhin auf der DEV-Ebene befindet. (z.B. durch die Funktionen SAVE, MODIFY, RENAME – siehe spätere Kapitel)
- Die mittlere Teilnummer *VV* wird beim LOCK eines Programmes von DEV nach VAL erhöht.
  Dabei wird die unterste Teilnummer *DD* gleichzeitig auf 00 zurückgesetzt.
- Die oberste Teilnummer *PP* wird beim LOCK eines Programmes von VAL nach PROD erhöht.
  Dabei werden die beiden Teilnummern *VV* und *DD* wieder auf 00 zurückgesetzt.

Die Dateiattribute aller Backup-Kopien werden außerdem auf „Read-only“ gesetzt.

5 Beschreibung der einzelnen Funktionen

5.1 Die LOCK Funktion

Verschiebt ein Programm auf die nächsthöhere Validierungsebene (DEV -> VAL oder VAL -> PROD)

Syntax

```
lock [-l] [-c "comment"] filename.ext [filename2.ext …]
```

Parameter

-l Falls ein zum SAS-Programm gehörendes .log oder .lst File gleichen Namens existiert, so wird von diesem ein Backup (mit der gleichen Versionsnummer wie das zugehörige Programm) im History-Verzeichnis erstellt.

-c "*comment*"
Hier kann ein kurzer Kommentar angegeben werden, der dann zusammen mit dem LOCK-Vorgang in der Datenbank gespeichert wird.

Funktionsweise

- Verschiebt ein oder mehrere Programme in die nächsthöhere Validierungsebene und passt die Zugriffsrechte an.
- Erstellt pro File ein Backup im History-Verzeichnis.
- Erstellt pro File einen Eintrag in der History-Datenbank.

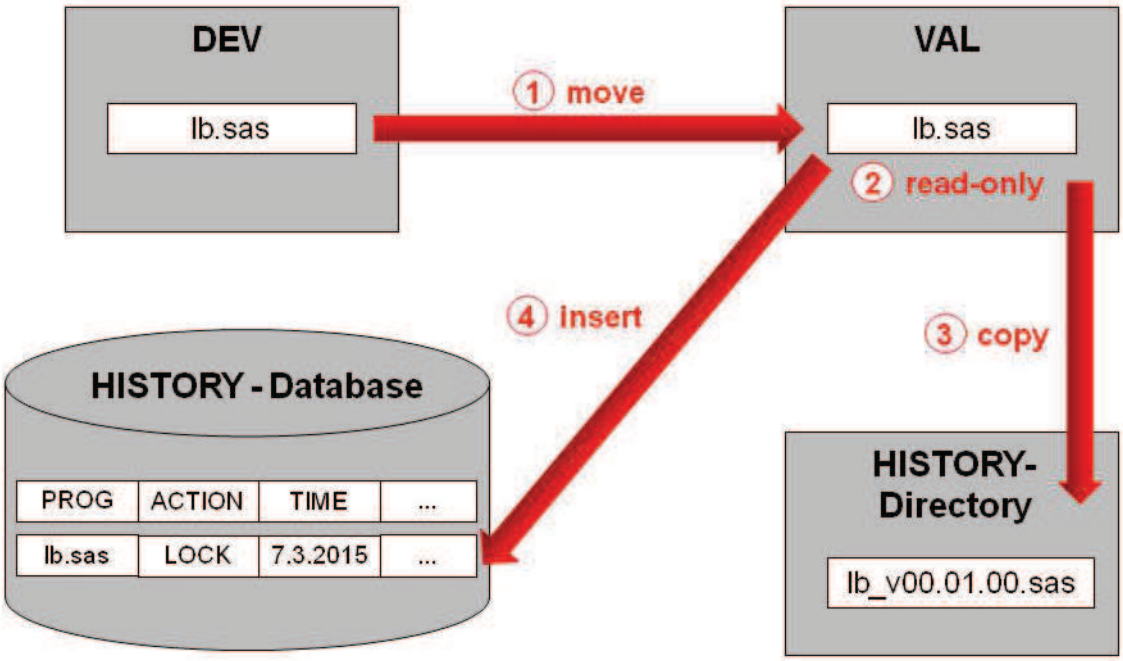

Abbildung 3: Arbeitsweise der Funktionen am Beispiel der LOCK-Funktion

5.2 Die UNLOCK Funktion

Verschiebt ein Programm zurück auf die DEV Ebene

Syntax

```
unlock [-c "comment"] filename.ext [filename2.ext …]
```

Funktionsweise

- Die Schreibrechte werden nach dem Verschieben wieder gesetzt.
- Beim UNLOCK wird kein Backup erstellt, da sich das Programm auf den höheren Validierungsebenen nicht geändert haben kann (Dateiattribut: read-only)
- Pro File wird ein Eintrag in der History-Datenbank erstellt.

5.3 Die RENAME Funktion

Die Identifizierung der Programme im History-Verzeichnis und in der History-Datenbank erfolgt über den Programmnamen und den Pfadnamen innerhalb des Studienverzeichnisses.

Verschiedene Gründe können dazu führen, dass ein Programm umbenannt oder in einem anderen Verzeichnis gespeichert werden muss.

Damit in diesen Fall die Verbindung zu existierenden Backups und Datenbank-Einträgen nicht verloren gehen, war es notwendig, eine Funktion zu entwickeln, die ein „kontrolliertes“ Umbenennen oder Verschieben von SAS-Programmen ermöglicht.

Das Ergebnis ist die RENAME Funktion, deren Syntax und Funktionsweise dem UNIX-Kommando „mv“ entspricht.

Syntax

```
  rename [-c "comment"] oldfilename.ext  [newdir/]newfilename.ext
oder
  rename [-c "comment"] filename.ext  [filename2.ext …] newdir
```

Funktionsweise

- Benennt ein Programm um und/oder verschiebt ein oder mehrere Programme in ein anderes Verzeichnis.
- Erstellt pro File ein Backup im History-Verzeichnis.
- Existierende Backups werden in das zum neuen Programmnamen gehörende History-Verzeichnis umgezogen.
- In diesem Fall wird im zum alten Programmnamen gehörenden History-Verzeichnis auch noch ein Shortcut („Symbolic Link“) erstellt, der auf das neue History-Verzeichnis zeigt.
- Die Umbenennung wird durch einen Eintrag in der History-Datenbank dokumentiert.
  Dabei wird unter anderem auch der alte Pfad- und Programmname gespeichert. Bestehende Datenbank-Einträge, die sich auf den alten Pfad-/Programmnamen beinhalten werden allerdings nicht abgeändert.

5.4 Die MODIFY Funktion

Die Berechtigungsstrukturen unserer SAS-Programme auf der Entwicklungsebene (DEV) sehen wie folgt aus:

```
permission   owner group     filename
rw- r-- ---  user1 projectX  addm.sas
rw- r-- ---  user2 projectX  adlb.sas
```

Nur der Besitzer hat Schreibrechte, andere Angehörige des Projektteams haben durch ihre Gruppenzugehörigkeit Leserechte.

Nun kann es vorkommen, dass ein Programm von einem anderen Programmierer modifiziert oder komplett übernommen werden muss. Dazu ist es notwendig, dass dieser dann auch der Besitzer („Owner“) des Programmes wird.

Da es in diesem Fall sinnvoll ist, ein Backup zu erzeugen und diesen Vorgang auch in irgendeiner Form zu dokumentieren, wurde diese Funktion in unser Versions-Kontroll-System integriert.

Syntax

```
modify [-c "comment"] filename.ext [filename2.ext …]
```

Funktionsweise

- Der Benutzer, der die Funktion ausführt wird Owner des Programmes. Er muss dazu allerdings Leseberechtigung auf das Programm durch seine Gruppenzugehörigkeit haben.
- Es wird ein Backup im History-Verzeichnis erstellt.
- Der Vorgang wird durch einen Eintrag in der History-Datenbank dokumentiert. Dabei wird unter anderem auch der Name des vorherigen Besitzers gespeichert.
- Der vorherige Besitzer wird über den Vorgang durch eine Email informiert. Dabei bekommt er auch Speicherort und Versionsnummer „seiner" letzten Programmversion mitgeteilt.

Im Übrigen führen auch die Funktionen UNLOCK und RENAME automatisch einen Benutzerwechsel („Ownerchange") durch, sobald ein Programm eines anderen Benutzers wieder nach DEV zurückgeholt bzw. umbenannt wird. Auch dabei wird der frühere Besitzer des Programmes per Email informiert und das ganze wird in der Datenbank dokumentiert.

5.5 Die SAVE Funktion

Neben den automatischen Programm-Backups, die beim Wechseln der Validierungsebenen (z.B. LOCK) erstellt werden, kann es auch sinnvoll sein, einen Zwischenstand während der Entwicklung in Form eines Backups abzulegen. Dazu wurde die SAVE Funktion entwickelt.

Syntax

```
save [-l] [-c "comment"] filename.ext [filename2.ext …]
```

Parameter

-l Falls ein zum SAS-Programm gehörendes .log oder .lst File gleichen Namens existiert, so wird von diesem ein Backup (mit der gleichen Versionsnummer wie das zugehörige Programm) im History-Verzeichnis erstellt.

Funktionsweise

- Erstellt pro File ein Backup im History-Verzeichnis.
- Erstellt pro File einen Eintrag in der History-Datenbank.

5.6 Weitere Funktionen

- Bei allen Funktionen des Versions-Kontroll-Systems können die unter LINUX üblichen Wildcards zur Generierung von Filenamen benutzt werden um mehrere bzw. bestimmte Programme auszuwählen (*, ? und []).
- Obwohl das System zur Versions-Kontrolle von SAS-Programmen konzipiert wurde, lässt sich damit aber auch mit allen anderen Datei-Typen arbeiten. So setzen wir es z.B. zur Versionierung von CSV- oder Excel-Files ein, die neben den

Programmen und SAS-Datasets Bestandteil einer statistischen Analyse sein können.

- Um eine höchstmögliche Verfügbarkeit zu garantieren, wurde ein Mechanismus implementiert, der es erlaubt, mit dem System auch dann zu arbeiten, wenn die zugehörige History-Datenbank nicht zur Verfügung steht (z.B. wegen eines Offline-Backups oder Wartungsarbeiten). Neue Records, die nicht erfolgreich in die Datenbank geschrieben werden können, werden in einem separaten Verzeichnis gesammelt und später nachgeladen.

6 Die History-Datenbank

Alle Aktionen des Versions-Kontroll-Systems werden in einer Datenbank protokolliert. Dabei wird pro Aktion und File genau ein Record angelegt.
Die Datenbank besteht aus einer einzelnen Tabelle, auf die noch einige Indexe zur Beschleunigung der Lese-Zugriffe gesetzt wurden. Schreibzugriffe erfolgen (ohne dass sich die Benutzer authentifizieren müssen) über einen eingeschränkten Account, der lediglich INSERT-Rechte hat und somit nur neue Records anlegen darf. Bestehende Einträge dürfen und können somit nicht mehr abgeändert oder gelöscht werden. In der Datenbank werden die folgenden Informationen protokolliert:

Tabelle 1: Inhalt der History-Datenbank

Spaltenname	Typ	Inhalt
PROJECT	Text	Projekt, zu der das Programm gehört
STUDY	Text	Studie, zu der das Programm gehört
FILENAME	Text	Programmname
PATH	Text	Pfadname relativ zum Studienverzeichnis
OLDPATH	Text	Absoluter Pfadname vor der Aktion
NEWPATH	Text	Absoluter Pfadname nach der Aktion
SOURCE	Text	Validierungsebene (DEV, VAL oder PROD) vor der Aktion
TARGET	Text	Validierungsebene (DEV, VAL oder PROD) nach der Aktion
ARCFNAME	Text	Name der Backup-Kopie (Programmname + Versionsnummer)
ARCPATH	Text	Pfadname der Backup-Kopie
VERSION_PROD	Integer	Vorder Teil der Versionsnummer (PP)
VERSION_VAL	Integer	Mittlerer Teil der Versionsnummer (VV)
VERSION_DEV	Integer	Hinterer Teil der Versionsnummer (DD)

ACTION	Text	Ausgeführte Aktion : "LOCK", "UNLOCK", "RE-NAME", "MODIFY" oder "SAVE"
USERID	Text	Benutzer, der die Aktion ausgeführt hat
PREVUSERID	Text	Früherer Besitzer des Programmes (im Falle eines Benutzerwechsels („Owner-Change"))
PREVNAME	Text	Früherer Pfad- und Programmname, falls ein RE-NAME stattgefunden hat.
TIMESTAMP	Date	Zeitpunkt der Aktion im Format DD.MM.YYYY HH:MM:SS
TIMEFILE	Date	Letztes Modifikationsdatum des Programmes
COMMENT	Text	Kommentar, der mit der Option -c mitgegeben wurde.
SAVELOGS	Text	Enthält ein "L", wenn Log- oder Listfiles durch die Option –l mitgesichert wurden
LOGFILENAME	Text	Name des durch die Option –l mitgesicherten Logfiles
LSTFILENAME	Text	Name des durch die Option –l mitgesicherten Listfiles

Der Lesezugriff auf die Datenbank erfolgt über das Modul SAS ACCESS TO ORACLE.
Nach Ablauf eines kleinen Stück Programmcodes, das über die autoexec.sas eingebunden werden kann, steht der Inhalt der Datenbank gefiltert auf die aktuelle Studie als SAS-View zur Verfügung. Über die standardmäßigen Such- und Filterfunktionen kann der Lifecycle der zur Studie gehörenden Programme nun komfortabel analysiert werden.
Auch ein abschließender Report für den Auftraggeber lässt sich problemlos erzeugen.
Regelmäßig gestellte Fragen wie „Sind denn alle Programme einer Studie validiert oder befinden sich noch welche auf der Entwicklungsebene?" lassen sich schnell beantworten.

7 Fazit

Es ist uns gelungen, in kurzer Entwicklungszeit mit Bordmitteln ein einfaches, auf unsere Bedürfnisse zugeschnittenes und intuitiv nutzbares Versions-Kontroll-System zu entwickeln, das allen regulatorischen Anforderungen entspricht.
Der Aufwand für die Programmierung hielt sich dabei in Grenzen, das System besteht aus nur etwa 1500 Zeilen Programmcode (ohne Kommentare). Als Eigenentwicklung kann es jederzeit an neue Bedürfnisse angepasst werden. Zudem ist es relativ unabhängig von Releasewechseln der verwendeten Komponenten (LINUX, ORACLE, SAS).
Ein Nachteil des Systems besteht allerdings darin, dass es theoretisch von den Benutzern „umgangen" werden könnte, z.B. indem Programme auf konventionellem Weg

zwischen den Validierungsebenen verschoben oder umbenannt werden, ohne die dafür vorgesehenen Funktionen zu benutzen. In diesem Fall würde auch der Inhalt der History-Datenbank nicht mehr den aktuellen Zustand des Dateisystems wiedergeben.
Insofern sind die Einfachheit des Systems und eine gute Schulung der Benutzer von essenzieller Bedeutung. Allerdings sind die Programmierer gewohnt, auf diese Weise zu arbeiten, denn das vorgestellte Versions-Kontroll-System hat ein Vorgänger-System mit ähnlicher, aber geringerer Funktionalität abgelöst.
Als schlankes und transparentes System genießt das neue Versions-Kontroll-System, das seit etwa einem Jahr im Einsatz ist, eine hohe Akzeptanz bei den Benutzern.