Darf's ein bisschen robuster sein?

Sebastian Reimann
viadee Unternehmensberatung GmbH
Anton-Bruchausen-Straße 8
48147 Münster
sebastian.reimann@viadee.de

Zusammenfassung

Wer kennt das nicht. Das SAS Programm läuft und läuft und dann bricht es plötzlich und unerwartet ab. Eine kurze Analyse der Fehlersituation zeigt, dass eigentlich alles gut aussieht und mit einer kleinen Korrektur das Programm an der abgebrochenen Stelle weiterlaufen kann. Doch wie macht man das am besten? Kann ein SAS Programm an genau der Stelle weiterlaufen, an der es abgebrochen ist?
Genau diese Fragen stellen sich beim Betrieb einer SAS Umgebung mit größeren Batchverarbeitungen immer wieder. Es gibt Abhängigkeiten zwischen Programmen, die bei der Ausführung beachtet werden müssen. Weiterhin muss sichergestellt werden, dass nur korrekt beendete Programme die Fortführung der Batchverarbeitung zulassen.
Im Folgenden soll ein kleiner Einblick in den Alltag einer Batch-Produktionsbetreuung gegeben werden. Dabei werden Punkte aufgezeigt, die den Batchbetrieb einfacher und robuster gegen Fehlersituationen machen.

Schlüsselwörter: SAS Makro Programmrahmen, Restart-Framework, DOS Batch, Abhängigkeiten, Programmsteuerung, SAS im Batchbetrieb

1 Motivation

Nachdem im letzten Jahr mit dem KSFE Beitrag „Darf's ein bisschen schneller sein" [1] auf die verschiedenen Möglichkeiten eingegangen wurde, SAS Programme zu beschleunigen, soll in diesem Jahr der Fokus auf dem Betrieb einer SAS Installation liegen.
Fazit im letzten Jahr war, dass klassische SAS Programme, die auf dem DataStep aufbauen, häufig single threaded ausgeführt werden. Zur Beschleunigung wurde vorgeschlagen, die Abläufe aufzuteilen und zu parallelisieren.
Hinter dem Ergebnis
der Zerteilung und Parallelisierung steht nun der Betrieb der SAS Umgebung. Im konkreten Projekt wurden die Abläufe in kleine Häppchen zerschnitten. Jedes kleine Programm kann für sich alleine und eigenständig ausgeführt werden; unabhängige Programme dabei gerne auch parallel. In Summe führte dies zu einer stolzen Ansammlung von ca. 500 SAS Programmen, die vom Betrieb in der richtigen Reihenfolge, unter Beachtung aller fachlichen Abhängigkeiten und unter gleichzeitiger Optimierung der Abfolge der Programme ausgeführt werden müssen.

In den folgenden Kapiteln werden einzelne Aspekte vorgestellt, die genutzt wurden, um die Abläufe zu standardisieren und so die Betriebsprozesse überschaubar zu halten.

2 Standardisierung der Programmstruktur

Werden SAS Programme in einem Team von Entwicklern erzeugt, so hat jeder Entwickler seine persönliche Note, mit der er sein Programm erstellt. Dies liegt in der Natur der Sache und ist im Allgemeinen auch nicht hinderlich. Je größer die Menge der ausgeführten Programme jedoch wird, umso schwieriger wird es für den Betriebsbereich, Fehlersituationen im Ablauf zu erkennen und daraufhin die richtigen Maßnahmen einzuleiten.
Ein klassisches Beispiel ist das Fortschreiben von Zeitreihen in einem Datawarehouse (DWH). Dieses Fortschreiben wird mittels der Prozedur PROC APPEND durchgeführt. Kommt es in einem Programm des Bestandsaufbaus nun zu einem Fehler und damit zu einem Abbruch, so ist zu prüfen, was mit dem Programm nach der Fehlerkorrektur gemacht werden kann. Die Gründe für den Abbruch können dabei ganz unterschiedlicher Natur sein. Sie können in fehlerhaften Eingabedaten, in einem Programmfehler oder aber auch rein betriebliches Aspekte wie z.B. eine vollgelaufene Dateisystempartition sein. In jedem Fall ist zu prüfen, ob der Fehler vor oder nach dem Fortschreiben der Daten aufgetreten ist. Wird das Programm einfach erneut gestartet besteht sonst die Gefahr, dass die Daten doppelt in den Bestand eingefügt werden, was dann auf Dauer zu fehlerhaften Auswertungen im DWH führen wird.
Als erste Maßnahme wurde die einheitliche Nutzung eines SAS Programmrahmens vereinbart. Mit diesem Programmrahmen wurde gleichzeitig ein Nutzungskonzept erarbeitet, welches vorgibt, welche Programmierstandards von jedem Programmierer einzuhalten sind.

2.1 Der SAS Programmrahmen

Um einen einheitlichen Rahmen für alle Programm zu erhalten, müssen zunächst die unterschiedlichen Teile eines SAS Programms identifiziert werden. Jedes Programm besteht zwangsläufig aus einem Initialisierungs-Member und aus einem Steuerungsmember. Zusätzlich gibt es eine freie Anzahl an Einzelschritten je Programm, die in fester Reihenfolge durchlaufen werden müssen. Jedes auf diese Weise strukturierte Programm ergibt eine fachliche Einheit, die nicht weiter unterteilt werden sollte.

Initialisierungs-Member:
Im Initialisierungs-Member werden die notwendigen Rahmenbedingungen für die Programmausführung geschaffen. Es werden die notwendigen Bibliotheken zugewiesen und Ein-/Ausgabedateien definiert. Weiterhin werden spezifische Variablen zur Konfiguration des Programmablaufs je Programmvariante an dieser Stelle definiert.

Die Verantwortung für dieses Initialisierungs-Member liegt beim Betrieb der Umgebung. Die Entwicklung macht lediglich Vorgaben, welche Bibliotheken unter welchem

logischen Namen verfügbar sein müssen. Die wirkliche Zuweisung der Bibliotheken hat der Betrieb zu verantworten. Nur so ist sichergestellt, dass die Programme im Rahmen eines geprüften Programmeinsatzes durch die verschiedenen Stages Entwicklung / Funktionstest / Integrationstest / Abnahmetest / Produktion gebracht werden können und in jeder Umgebung korrekt ausgeführt werden.

Beispielhaft könnte ein Initialisierungs-Member wie folgt aussehen:

```
/*Aufbau der Vertragstabelle
  Dokumentation siehe http://saswiki.intern/...
  Im Abbruchfall: Entwicklungsbereich informieren*/

%M_GENLIBNAME(LIB=EINGABE, ACCESS=READ);
%M_GENLIBNAME(LIB=AUSGABE, ACCESS=WRITE);

%M_GENFILENAME(REF=VTR_IN, PFAD=DATENRAMPE\...\VTR.csv);

%M_GET_SYSVAR(SYSVAR=PROGRAMM);
%M_GET_SYSVAR(SYSVAR=VARIANTE);

%M_START_PROGRAMM;
```

Durch die Nutzung von eigenen Makros wird sichergestellt, dass die Bibliotheken immer gleich zugewiesen werden. Die Pfadangaben und weitere Allokationsparameter werden dabei aus den Metadaten der Umgebung gelesen. So muss keine weitere Anpassung an dem Member vorgenommen werden, wenn dieses in unterschiedlichen Umgebungen ausgeführt wird.

Weiterhin werden Systemvariablen genutzt, um von außen Steuerungsinformationen wie Programmname oder Programmvariante an die Initialisierung zu übergeben. Über das Makro M_START_PROGRAMM wird die Programmausführung dann gestartet. Hierbei wird zunächst anhand von Protokolltabellen geprüft, wie das Programm bei der letzten Ausführung beendet wurde. Dies ist wichtig, damit im Restart-Fall an der Stelle weiterverarbeitet wird, an der die Verarbeitung zuvor unterbrochen wurde. Damit entsprechende Restart-Stufen ermittelt werden können, wird das Steuerungs-Member benötigt.

Steuerungs-Member:
Im Steuerungs-Member gibt der Entwickler vor, in welcher Abfolge die einzelnen Programmschritte durchlaufen werden sollen. Dabei ist es wichtig, dass Metainformationen für den Restartfall mit angegeben werden.

Eine beispielhafte Steuerung könnte wie folgt aussehen:

```
/* Spezifische Parameter ermitteln */
%M_STEP(RESTART=1, MEMBER=VTR_S01, EXECUTE=A);

/* Einlesen der Dateischnittstelle */
%M_STEP(RESTART=2, MEMBER=VTR_S02);

/* Einlesen und aufbereiten der weiteren Eingabetabelle */
%M_STEP(RESTART=3, MEMBER=VTR_S03);

/* Daten zusammenführen */
%M_STEP(RESTART=3, MEMBER=VTR_S04);

/* Daten fortschreiben */
%M_STEP(RESTART=5, BASIS=AUSGABE.VTR_TABELLE,
        EINGABE=TMPLIB.VTR_TABELLE_TAG,
        VAR=DATUM, WERT=&DATUM., TYP=APPEND);
```

Auch in dieser Steuerung werden standardisierte Makros verwendet, um den Ablauf zu steuern. Die Parameter des Makros haben dabei folgende Bedeutung:

- Restart: Für jeden Schritt wird definiert, bei welchem Schritt die Verarbeitung wieder aufgesetzt wird, wenn es in dem jeweiligen Schritt zu einem Abbruch der Verarbeitung gekommen ist
- Member: gibt das einzelne Member an, welches zur Ausführung herangezogen werden soll
- Execute: Über den Parameter EXECUTE=A wird gesteuert, dass dieser Step immer ausgeführt wird, auch wenn im Restartfall eigentlich erst in einem späteren Step wiederaufgesetzt wird. Dies ist vor allem dann wichtig, wenn in einem ersten Step benötigte Makrovariablen für die Folgeschritte belegt werden.
- Typ: Ist standardmäßig mit INCLUDE belegt, so dass das unter MEMBER angegebene Member inkludiert wird. Es sind aber auch andere Ausführungsvarianten wie bspw. das Fortschreiben von Tabellen möglich.

Durch die Programmsteuerung mit passenden Kommentaren erhält man schnell einen Überblick über die einzelnen Schritte des Programms. Weiterhin ist leicht nachvollziehbar, in welchem Schritt ein Abbruch stattgefunden hat, da mit jedem Step-Aufruf ein entsprechender Hinweis in das SASLOG ausgegeben werden kann.

Restart-Verhalten:
Zu Beginn der Programmausführung (%M_START_PROGRAMM) wird aus der Protokolltabelle die letzte Restart-Stufe ausgelesen. I.d.R. ist dies 0, da nach einer erfolgreichen Beendigung des Programms die Restart-Stufe wieder auf 0 gesetzt wird. Nach einem Abbruch kann hier jedoch auch eine andere Stufe (z.B. 3) ermittelt werden.

Zu jedem Start eines Programmschritts wird geprüft, ob die global gespeicherte Restart-Stufe größer als die für den Step definierte Restart-Stufe ist. Ist dies der Fall und es ist gleichzeitig nicht die Option EXECUTE=A definiert, wird der Step übersprungen. Andernfalls wird die global definierte Restart-Stufe (Makrovariable und Protokolltabelle) direkt zu Beginn auf die Restart-Stufe des Steps geändert. (Greift die Option EXECUTE=A, so wird die globale Restart-Stufe nicht angepasst, jedoch der Step ausgeführt.)

Nach Abschluss des Programmsteps wird immer eine Fehlerprüfung durchgeführt. Hierbei wird geprüft, ob im Programmablauf globale Fehlerschalter eingeschaltet wurden, um so die Verarbeitung abzubrechen. Weiterhin wird geprüft, ob SAS Fehler erkannt hat und entsprechende Meldungen ins Log geschrieben hat. Hierzu wird die Sytemvariable SYSERRORTEXT ausgewertet. Immer wenn diese Variable gefüllt ist, hat es einen Fehler in einer SAS Prozedur gegeben.

Da die Restart-Stufe erst mit Beginn des Folgeschrittes erhöht wird, ist auf diese Weise gewährleistet, dass der abgebrochene Schritt erneut ausgeführt wird. Über die Belegung der Restart-Stufe in den einzelnen Schritten kann vom Entwickler vorgegeben werden, dass mehrere Einzelschritte auf derselben Restart-Stufe stehen. Somit wird auch beim Abbruch des zweiten Steps derselben Stufe, auch der erste Step wiederholt. Dadurch, dass die globale Restart-Steuerung zu Beginn des Programmablaufs eingelesen wird, kann durch den Betrieb im Abbruchfall ein Restart auf einer beliebigen Stufe aufgesetzt werden, wenn dies nach der Fehleranalyse vom Entwickler so vorgegeben wird.

Wurden alle Steps des Steuerungsmembers ausgeführt endet das Anfangsmakro M_START_PROGRAMM mit einigen abschließenden Prüfungen und Aufräumarbeiten. Damit ein Restart an jeder Stelle des Programms möglich ist, muss sichergestellt sein, dass temporäre Ergebnisse im Abbruchfall nicht verloren gehen. Daher muss auf die Nutzung der WORK-Bibliothek verzichtet werden. Beim Start des Programms wird automatisch eine TMPLIB-Bibliothek eingerichtet, die so wie WORK auf einer schnellen lokalen SSD-Platte liegt. Temporäre Ergebnisse werden nur in dieser Bibliothek gespeichert. Beim Beenden des Programms wird diese Bibliothek automatisch gelöscht sofern kein Fehler im Programmablauf festgestellt wurde. Nach dem Löschen wird die Restart-Stufe auf 0 zurückgesetzt damit das Programm bei der nächsten Verarbeitung von vorne startet.

2.2 Nutzungskonzept und Programmierkonventionen

Der Einsatz eines Programmrahmens, wie in Kapitel 2.1 vorgestellt, wird nur gelingen, wenn alle Entwickler diesen auch in ihren Programmen Nutzen. Daher wurde zusammen mit dem Programmrahmen ein Nutzungskonzept erarbeitet, welches die grundlegende Programmarchitektur vorgibt. So ist hier bspw. definiert, dass jedes Programm aus den oben beschriebenen Teilen Initialisierungsmember, Steuerungsmember und Ausführungsmember besteht.

Neben den Vorgaben für die Architektur von SAS Programmen werden im Nutzungskonzept auch grundlegende Programmierrichtlinien festgelegt, die von allen Entwicklern einzuhalten sind. Nur so ist gewährleistet, dass die erstellten Programme für alle Teammitglieder verständlich und nachvollziehbar sind. Auch wird im Nutzungskonzept beschrieben, welche Standard-Makros und –Funktionen für die Umsetzung von Anforderungen genutzt werden können.

Neuen Entwicklern dient das Nutzungskonzept als Leitfaden für den Einstieg in die Entwicklung während erfahrene Entwickler es als Nachschlagewerk für Standard Implementierungen von regelmäßig wiederkehrenden Sachverhalten nutzen.

Damit das Nutzungskonzept in der täglichen Praxis eingehalten wird, werden in regelmäßigen Abständen Code Reviews durchgeführt. Hierbei wird ein neues, in der Entwicklung befindliches Programm vom verantwortlichen Entwickler vorgestellt und von allen anderen Teammitgliedern analysiert.

Durch die vielen Augen, die in diesem Review auf das Programm schauen, fallen schnell diverse Verbesserungsaspekte auf, die dann direkt in das Programm mit einfließen können. Diese Aspekte können fachlicher Natur sein, aber auch technischer oder betrieblicher Natur. Hierbei gilt es, die im Nutzungskonzept vorgegebenen Konventionen auf das neue Programm zu spiegeln und ggf. vorliegende Abweichungen aufzuzeigen.

Auch können in diesem Zusammenhang die Vorgaben des Nutzungskonzeptes immer wieder hinterfragt und angepasst werden. Das Nutzungskonzept muss ein lebendes Dokument bleiben, in welches immer die aktuellen Erkenntnisse mit einfließen.

Mit einem überschaubaren Aufwand von ca. 2 Stunden je Entwickler gelingt es, das Nutzungskonzept stets aktuell und für alle Entwickler präsent zu halten. Weiterhin wird auf diese Weise ein Wissenstransfer zwischen den Entwicklern erreicht, so dass jeder zumindest überblicksartigen Einblick in die Programme der anderen Teammitglieder hat.

3 Steuerung der Programmausführung

Nachdem der interne Programmablauf standardisiert wurde besteht die nächste Aufgabe darin, die Programmausführung zu standardisieren. Bei der Ausführung der Programme sind folgende Nebenbedingungen zu beachten:

Jedes Programm hat einen genau definierten Ausführungsrhythmus. Der einfachste Rhythmus ist die tägliche Ausführung. Da es sich bei der Umgebung jedoch um ETL Prozesse eines Datawarehouses handelt, werden die Programme i.d.R. buchungstäglich ausgeführt. Dies bedeutet, dass ein individuell definierter Kalender vorgibt, an welchen Geschäftstagen Buchungsläufe stattfinden und an welchen Tagen nicht. Einzelne Pro-

gramme werden dagegen nur wöchentlich an einem bestimmten Tag, monatlich zum Monatsersten oder Monatsletzten, quartalsweise, halbjährlich oder jährlich ausgeführt. Es ist somit im Vorfeld der täglichen Batchausführung zu planen, welche Programme an diesem Tag ausgeführt werden müssen. Abhängig von den eingeplanten Programmen können sich unterschiedliche Abhängigkeiten zwischen den Programmen ergeben.

Die Planung für jede Batchausführung erfolgt jeweils täglich vor der Batchausführung. Voraussetzung für eine neue Planung ist, dass die Batchnacht des vorherigen Tages fehlerfrei abgeschlossen wurde bzw. dass alle Fehler behoben wurden oder die zugehörigen Programme ausgeplant wurden. Die Abhängigkeiten der Programme werden dabei in einer zentralen Datenbank gepflegt. Für jedes Programm werden die Laufrhythmen und die Abhängigkeiten, d.h. die Vorläufer gepflegt. Für jede Abhängigkeit kann dabei definiert werden, welche Bedingungen an eine Abhängigkeit gestellt werden. So kann bspw. festgelegt werden, dass ein Return Code 3 eines Vorläufers für ein Programm als fehlerfrei angesehen wird, für einen anderen Vorläufer jedoch nicht. Weiterhin kann für Nachfolger definiert werden, dass diese sich anhand des Return Codes eines Vorläufers ausplanen und diese somit nicht ausgeführt werden. Dies kann bspw. dann sinnvoll sein, wenn ein Programm einen Bericht erstellt und abhängig davon, ob der Bericht Datensätze enthält oder nicht einen unterschiedlichen Return Code liefert. So kann das Folgeprogramm – z.B. die E-Mail-Benachrichtigung der Fachanwender – ausgeplant werden, wenn der Bericht ohnehin keine Daten enthält. Alternativ kann auf diese Weise auch ein Archivierungsprogramm ausgeplant werden, so dass ein leerer Bericht nicht ins Langzeitarchiv überstellt wird.

Weiterhin können Vorläufer eines Programms als optional markiert werden, so dass während der Planung kein Fehler ermittelt wird, wenn ein Vorläufer zu einem Ausführungstag gar nicht eingeplant wird. Ein Beispiel hierfür wären Programme, die in der Monatsverarbeitung mehr Vorläufer haben, als in der Tagesverarbeitung. Wurden die Vorläufer für die Tagesverarbeitung nicht eingeplant, so führt dies bei einem optionalen Vorläufer nicht zu einem Planungsabbruch.

Die Planung für eine Batchausführung wird als Sammlung von Textdokumenten im Dateisystem bereitgestellt. Dies hat den Vorteil, dass ohne weitere technische Hilfsmittel (Web Browser, Datenbanktools, etc.) ermittelt werden kann, welche Programme ausgeführt wurden und welche noch nicht. Ein Planungsverzeichnis hat dabei folgenden Aufbau:

PLANDATUM JJJJ-MM-TT
- NACHLÄUFER
- SASLOGS
- STATUS
  - 01-EINGEPLANT
  - 02-BEREIT
  - 03-AUFGESCHOBEN

04-AKTIV
05-FEHLERHAFT
06-KORREKT
07-AUSGEPLANT
VORLÄUFER

Im Rahmen der Batchplanung werden alle Programme ermittelt, die an dem Plandatum ausgeführt werden sollen. Für jedes Programm wird eine Statusdatei im Ordner 01-Eingeplant erstellt, in der die grundlegenden Informationen zum Programm hinterlegt werden. Zu diesen Informationen zählen

- Programmname
- Max. Return Code, der als korrekt gewertet wird
- Startzeit
- Endzeit

Weiterhin wird für jedes Programm einmal ermittelt, welche Programm Nachläufer des jeweiligen Programms sind und einmal die Programme, die Vorläufer des Programms sind. Diese Informationen werden später benötigt, um zu bewerten, ob ein Programm bereit für die Ausführung ist oder ob noch gewartet werden muss.

Ablauf der Batchverarbeitung
Wird die Batchverarbeitung gestartet, werden zunächst alle Programme ermittelt, die keine Vorläufer haben. Diese dürfen mit Start der Batchverarbeitung gestartet werden. Um eine Parallelisierung zu erreichen, wird die Programmausführung in mehreren parallelen Threads gestartet. Jeder Thread prüft vor dem Start des Programms, ob das Programm in diesem Ausführungsplan bereits ausgeführt wurde. Wenn ja, so wird das Programm nicht erneut gestartet. Vielmehr wird der bei der letzten Ausführung festgehaltene Return Code erneut zurückgeliefert, so dass die Verarbeitung weiterläuft als ob das Programm erneut gestartet wurde.

Nach Abschluss eines Programms, welches Auswirkungen auf die Programmausführung hatte, wird geprüft, ob der Return Code des Programms kleiner oder gleich dem maximalen als korrekt zu wertenden Return Code ist. In diesem Fall wird die Status-Datei zu dem Programm in den Ordner 06-Korrekt verschoben. Andernfalls in den Ordner 05-Fehlerhaft. In der Statusdatei wird der Return Code als Variable vermerkt, so dass im Falle eines Restarts auf diesen Return Code erneut zurückgegriffen werden kann. Anschließend wird für alle Nachläufer des gerade abgeschlossenen Programms geprüft, ob für dieses Programm nun alle Vorläufer ausgeführt wurden. Wenn ja, dann ist dieses Programm als nächstes Programm bereit für die Ausführung und wird somit in den Ordner 02-Bereit verschoben.

Wurde ein Programm mit einem Fehler beendet, so werden alle nachfolgenden Programme in den Ordner 03-Aufgeschoben verschoben, damit diese Programme in der

weiteren Analyse nicht weiter betrachtet werden müssen. Nach einer Fehlerkorrektur werden diese Programme dann wieder eingeplant und dann ausgeführt.

Wurden alle Programme ohne Vorläufer bearbeitet, so werden die Programme, die im Ordner 02-Bereit stehen zur Ausführung gebracht. Die Vorläufer dieser Programme wurden bereits geprüft, so dass ein direkter Start erfolgen kann. Nach Abschluss des Programmes erfolgt die oben bereits beschriebene Prüfung der Nachfolger, so dass dann die nächsten Programme im Ordner 02-Bereit eingestellt werden.

Auf diese Weise werden der Reihe nach alle geplanten Programme ausgeführt. Kommt es im Laufe der Verarbeitung zu einem Abbruch, so kann der Fehler korrigiert werden Im Anschluss wird die gesamte Verarbeitung von vorne neu gestartet. Dadurch, dass jeder Job sich seinen ursprünglichen Return Code merkt und das Framework sicherstellt, dass ein Job nicht mehrfach ausgeführt wird, kann auf diese Weise die gesamte Batchkette fortgesetzt werden, ohne dass ein Punkt zum Wiederaufsetzen ermittelt wird.

Über die programminterne Restart-Steuerung ist weiterhin sichergestellt, dass ein einzelnes Programm im Restart-Fall auch nur die Schritte erneut ausführt, die noch nicht gelaufen sind. Für den Betrieb hat diese Herangehensweise den Vorteil, dass zu keiner Zeit analysiert werden muss, welche Teile im Abbruchfall erneut ausgeführt werden müssen, da durch das Framework sichergestellt ist, dass einzelne Programme, die im Status 06-Korrekt stehen, nicht erneut ausgeführt werden.

Weitergehende Fehlerermittlung

Die gesamte Ablaufsteuerung setzt darauf auf, dass das SAS Programm den korrekten Return Code seiner Verarbeitung liefert. Die Praxis hat jedoch gezeigt, dass manche Meldungen im SASLog den Return Code des Programms nicht weiter beeinflussen, aber trotzdem ein Indizes für die nicht korrekte Verarbeitung der Daten sind.

Ein Beispiel hierfür ist die Meldung

```
MERGE statement has more than one data set with repeats of BY values
```

Solange die Daten für den Merge korrekt sortiert sind, lässt sich der Merge fehlerfrei ausführen. Haben jedoch beide Tabellen des Merges Duplikate in der Schlüsselvariablen, so erfolgt je nach Datenkonstellation eine unterschiedliche Verarbeitung. Im Rahmen eines DWH Bestandaufbaus sind derartige Meldungen nicht wünschenswert bzw. wenn sie dann auftreten, muss i.d.R. geeignet darauf reagiert werden. Aus diesem Grund wird nach jeder Programmausführung das SASLog auf bestimmte Wortkombinationen gescannt. Werden relevante Treffer gefunden, so wird die Anzahl der Treffer auf den Return Code hinzuaddiert, um so die Folgeverarbeitung zu stoppen bis dass die Situation analysiert wurde. Auf diese Weise ist sichergestellt, dass nur einwandfreie Daten im DWH verarbeitet werden und die Datenqualität schon bei der Verarbeitung überwacht wird.

4 Fazit

Mit steigender Anzahl an Batchprogrammen und Entwicklern im Team wird es immer relevanter, dass gewissen Programmierrichtlinien und Programmstandards eingehalten werden. Nur so ist gewährleistet, dass sich jeder Entwickler in angemessener Zeit im Programm seiner Kollegen zu Recht findet.

Die Nutzung eines SAS Programmrahmens bietet den Vorteil, dass einzelne Programmteile im Restartfall nicht erneut ausgeführt werden, wenn sie zuvor fehlerfrei ausgeführt wurden. Auf diese Weise wird die Laufzeit der Programme im Restartfall deutlich verkürzt.

Durch regelmäßige Review-Prozesse kann sichergestellt werden, dass einmal festgelegte Programmierkonventionen von den Entwicklern auch eingehalten werden. Weiterhin erhalten alle Teammitglieder durch die Vorstellung von Programmen einen Überblick über den Tätigkeitsbereich der Kollegen. Die gemeinschaftliche Analyse der Programme deckt im Rahmen der entstehenden Diskussion Optimierungsmöglichkeiten schnell auf, so dass die Programme i.d.R. noch vor der Produktivsetzung optimiert werden können. Sind beim Review-Prozess die für den Batchbetrieb verantwortlichen Personen mit anwenden, so können auch betriebliche Aspekte schnell in die Entwicklung zurückgespielt werden. Durch die regelmäßige Diskussion der Programmierrichtlinien bleiben diese bei den Entwicklern präsent und können bei Bedarf in der Gruppe an aktuelle Situationen angepasst werden.

Durch die Nutzung eines Planungsprozesses für die Batchverarbeitung und eine lückenlose Ausführungsdokumentation lässt sich der Batchbetrieb soweit vereinfachen, dass gesamte Jobketten einfach neu gestartet werden können. Im Restart Fall werden dennoch nur die Programme ausgeführt, die zuvor noch nicht korrekt gelaufen sind. Durch die integrierte Prüfung der Vorläufer bzw. Nachläufer der Programme ist zudem noch sichergestellt, dass ein Programm erst dann ausgeführt wird, wenn alle Vorbedingungen erfüllt sind.

Literatur

[1] S. Reimann. "Darf´s ein bisschen schneller sein". In: T. Friede, R. Hilgers, R. Minkenberg: KSFE 2014. Shaker Verlag, Aachen 2014, 275-284