Automatisches Testen mit SASUnit: mehr Qualität und Transparenz bei der Erstellung von SAS-Programmen

Andreas Mangold
HMS Analytical Software GmbH
Rohrbacherstraße 26
69115 Heidelberg
andreas.mangold@analytical-software.de

Zusammenfassung

Testen von Software ist notwendig und zeitraubend. Der Unittest-Ansatz, der insbesondere im Java-Umfeld mit JUNIT bekannt geworden ist, setzt darauf, dass das Testen Teil des Entwicklungsprozesses ist und dass Tests soweit wie möglich automatisiert ablaufen. Dies hat den Vorteil, dass die Tests jederzeit schnell wieder ausgeführt und somit negative Seiteneffekte von Änderungen schnell identifiziert werden können.
SASUnit ist ein von HMS Analytical Software entwickeltes frei verfügbares SAS-Makropaket. Es steuert die Ausführung von Testszenarien und erstellt automatisch übersichtliche Testberichte im HTML-Format. SASUnit in der aktuellen Version 0.9 ist verfügbar unter www.redscope.org/sasunit für SAS 9.1.3 Service Pack 4 unter Microsoft Windows.
Um ein Makro mit SASUnit zu testen, benötigt man unveränderliche Testdaten und muss mindestens ein Testszenario erstellen, das meist aus mehreren Testfällen besteht. Jeder Testfall ruft das zu testende Programm, oft ein Makro, mit bestimmten Daten und Parameterwerten auf und prüft anschließend, ob die gewünschten Ergebnisse zurückgeliefert wurden. Es können dabei beispielsweise Werte von Makrovariablen, Inhalte von SAS-Datasets und das Vorhandensein oder die Abwesenheit bestimmter Meldungen im SAS-Log geprüft werden.
SASUnit steuert die Ausführung der Testszenarien und erstellt aus den Prüfergebnissen Testberichte im HTML-Format, in die die Testdaten, die Testszenarien und erweiterte Prüfergebnisse (z.B. Output von PROC COMPARE) integriert sind. Für nicht-automatisch testbare Eigenschaften (z.B. korrektes Layout von Grafiken und Berichten) können Checklisten für die manuelle Prüfung in die Testberichte integriert werden.

Schlüsselwörter: Test, Unittest, Qualitätssicherung, Softwareengineering.

1 Unittests im Softwareengineering

Unittests, auch Modul- oder Komponententests genannt, dienen der Prüfung von einzelnen Komponenten eines Softwaresystems auf Korrektheit. Sie spielen in unterschiedlichen Vorgehensmodellen für die Softwareentwicklung eine wichtige Rolle.

1.1 V-Modell

Im Vorgehensmodell „V-Modell“ bilden die Unittests die Voraussetzung für Integrations- und Akzeptanztests. Integrationstests prüfen das Zusammenwirken der Kompo-

nenten. Akzeptanztests prüfen, ob ein IT-System im Geschäftsprozess erfolgreich anwendbar ist.

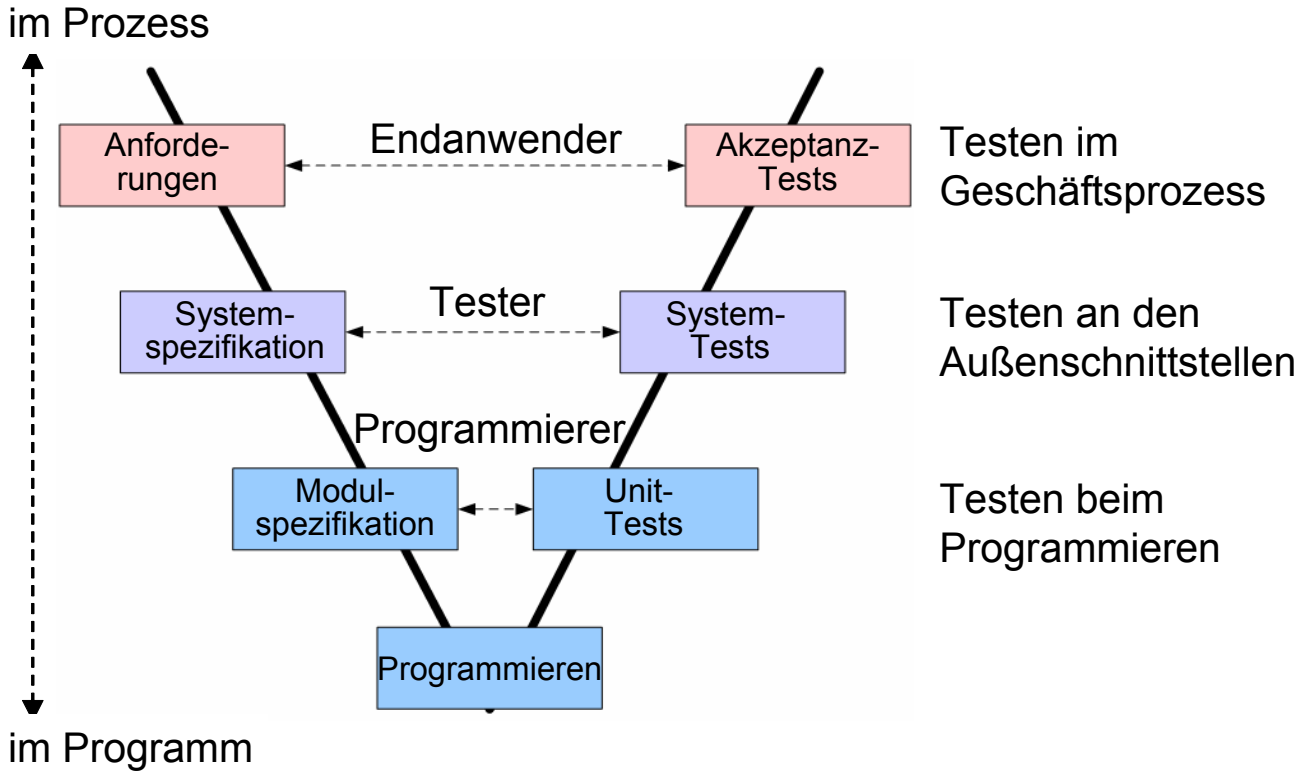

Abbildung 1: V-Modell

Das V-Modell eignet sich besonders für große Systeme, kann jedoch auch für kleine Systeme, sogar einzelne SAS-Programme, angepasst werden. Wichtig daran ist, dass eine Durchgängigkeit hergestellt wird von der Anwendung von Programmen im Geschäftsprozess bis hin zu Programmierung. Das Verfahren soll also nicht nur zu funktionsfähiger, sondern auch zu anwendbarer Software führen.

1.2 Testgetriebene Programmierung

Testgetriebene Programmierung ist eine Methode im Rahmen der Agilen Softwareentwicklung. Es wird Wert darauf gelegt, dass Qualität von vornherein in das zu entwickelnde Softwareprodukt eingebaut wird und dass Qualität nicht „nachrüstbar“ ist. Testen ist hier integraler Bestandteil der Softwareentwicklung. Tests ersetzen nahezu die Softwarespezifikation. Die Programmierung erfolgt in so genannten Mikroiterationen:

1. Schreibe einen kleinen Test für den nächsten zu implementierenden Funktionalitätshappen. Dieser Test sollte **nicht** funktionieren.
2. Erfülle den Test mit möglichst wenig Code, um schnell wieder zum "grünen Balken" (alle Tests laufen) zurückzukehren.
3. Räume den Code auf, dazu gehört die Entfernung von Duplikation, Einführung von notwendigen Abstraktionen und Umsetzen der Codekonventionen. Ziel dieses Aufräumens ist die *einfache Form* des Codes.

Diese Schritte werden solange wiederholt, bis dem Entwickler keine sinnvollen Tests mehr einfallen.
Die Testgetriebene Programmierung erleichtert die Einführung systematischer Tests im Arbeitsalltag des Programmierers. Ein Ersatz von dokumentierten Anforderungen durch Tests ist aber unter dem Gesichtspunkt der Anwendbarkeit von Software im Geschäftsprozess abzulehnen, siehe V-Modell.

2 Besonderheiten von Unittests und SASUnit

Unittests allgemein	Unittests mit SASUnit
Unittesting-Frameworks sind Systeme, die den Ablauf von automatischen Tests steuern und Testprotokolle erzeugen.	SASUnit ist ein Unittesting-Framework für SAS-Programme und SAS-Makros und ist selbst als **Makropaket** implementiert.
Unittests sind **Programme in der jeweiligen Programmiersprache**, die das zu testende Programm (den Prüfling) aufrufen und für den Vergleich der tatsächlichen mit den erwarteten Ergebnissen spezielle Funktionsaufrufe, **asserts** (Zusicherungen) genannt, verwenden.	**Testszenarien sind SAS-Programme**, in denen die Testfälle durch Makroaufrufe dokumentiert werden. Nach dem Aufruf des Prüflings werden **%assert**-Makros für die Ergebnisprüfung verwendet.
Eine **Testsuite** ist der Gesamtablauf aller Unittests. Die Testsuite kann **jederzeit erneut ausgeführt** werden, da unveränderliche Testdaten verwendet werden.	Alle Testszenarien einer Testsuite werden mit dem Makro **%runSASUnit** aufgerufen. Dabei läuft jedes Testszenario in einer **eigenen SAS-Sitzung**, um Seiteneffekte zu vermeiden.

Bei der Neuentwicklung eines Programms werden Unittests gemäß der **Spezifikation** entwickelt, um **Standardfälle** und **Ausnahmekonstellationen** zu testen. Das zu testende Programm (der Prüfling) und die Tests werden so lange modifiziert, **bis kein Test mehr fehlschlägt**.	SASUnit kann verwendet werden, um **SAS-Makros, SAS-Programme, Stored Processes** und **ETL-Jobs** zu testen. Es können **Makrovariablen**, **SAS-Dateien**, **SAS-Bibliotheken**, **SAS-Logs** und **ODS-Ausgaben** geprüft werden.
Regressionstest: Wenn sich **Anforderungen ändern** oder wenn man **nachträglich Fehler** findet oder wenn sich das **Systemumfeld ändert**, wird das zu testende Programm (der Prüfling) und die Tests so lange modifiziert, bis kein Test mehr fehlschlägt.	Regressionstests sind jederzeit möglich. Es werden nur solche Testszenarien ausgeführt, die **seit dem letzten Lauf der Testsuite geändert** wurden oder deren Prüfling sich geändert hat.
Testprotokolle werden automatisch im Anschluss an die Testausführung erzeugt. Die Prüfergebnisse werden farblich markiert: **Grün** bedeutet, dass die tatsächlichen Ergebnisse den erwarteten entsprechen, der andere Fall wird **rot** dargestellt.	Die Testprotokolle werden **im HTML-Format mit Navigationsframe** bereitgestellt, gegliedert nach Prüfling, Testszenario und Testfall. In das Testprotokoll integriert werden **Links** auf Quellverzeichnisse, SAS-Programme, Testszenarien, SAS-Logs, SAS-Dateien und ODS-Ausgaben.
Unittests werden wie jedes Programm **im Quellcode dokumentiert**.	Für SAS-Programme bietet sich **Doxygen** als Dokumentationsgenerator an. Siehe www.redscope.de/artikel/doxygen.

3 Aufbau von Unittests

3.1 Genereller Aufbau von Unittests

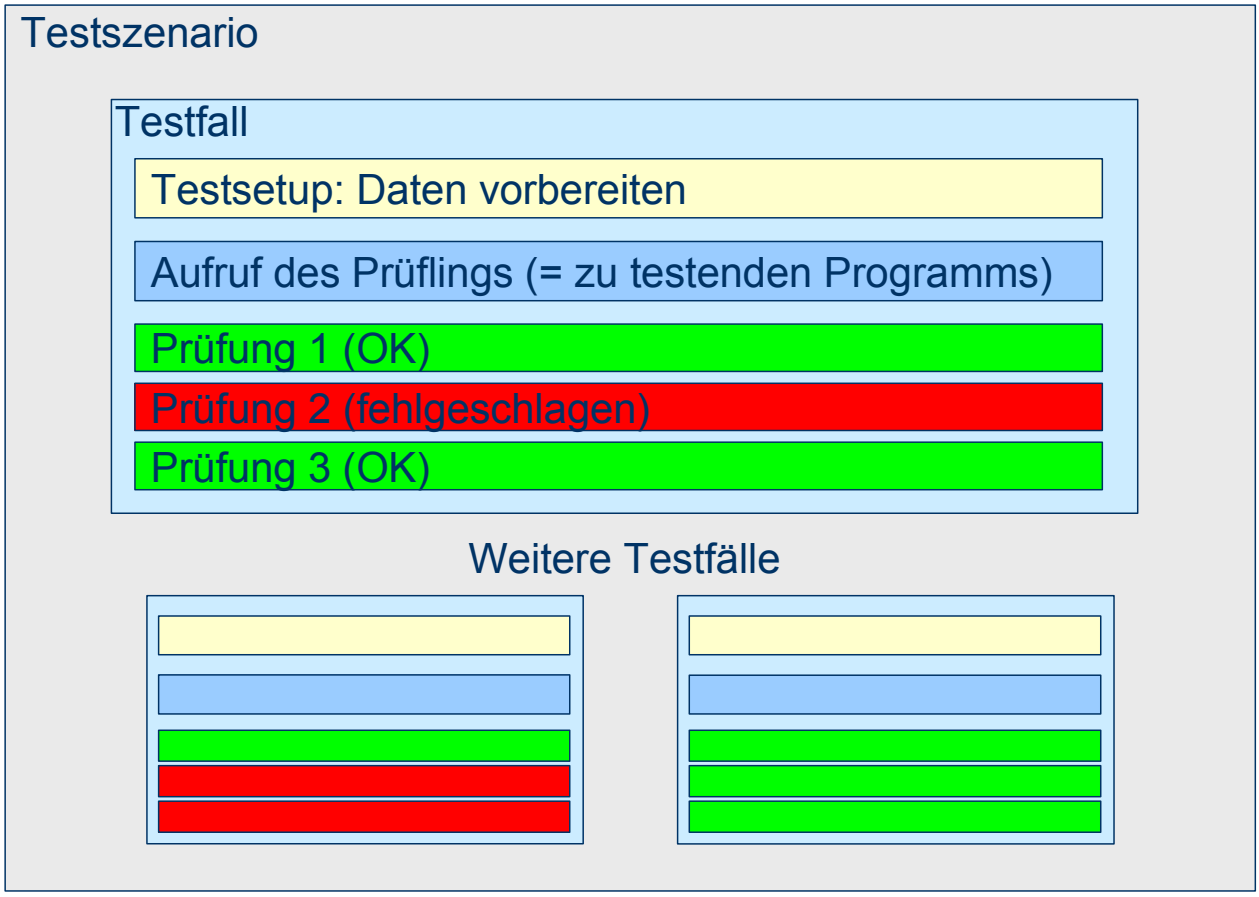

Abbildung 2: Aufbau von Unittests

Eine Testsuite besteht aus einem oder mehreren Testszenarien. Häufig wird pro Prüfling (also pro zu testendem SAS-Programm bzw. –Makro) ein Testszenario aufgesetzt, bei komplexen Prüflingen wird man jedoch auch mehrere Testszenarien pro Prüfling erstellen, bei einfachen Prüflingen kann man auch alle Tests aller Prüflinge in ein Testszenario packen. Jedes Testszenario enthält mindestens einen Testfall. Jeder Testfall ist für die Überprüfung einer bestimmten Eigenschaft eines bestimmten Prüflings zuständig.

Ein Testfall benötigt unveränderliche Daten, damit er jederzeit wiederholt werden kann. Diese werden entweder an entsprechender Stelle hinterlegt oder direkt im Testfall generiert (Testsetup). Nun wird der Prüfling aufgerufen.
Nach dem Aufruf des Prüflings werden die Ergebnisse des Aufrufs geprüft. Das Ergebnis kann sein „OK“ (grün) oder „fehlgeschlagen“ (rot).

3.2 Aufbau von Unittests mit SASUnit

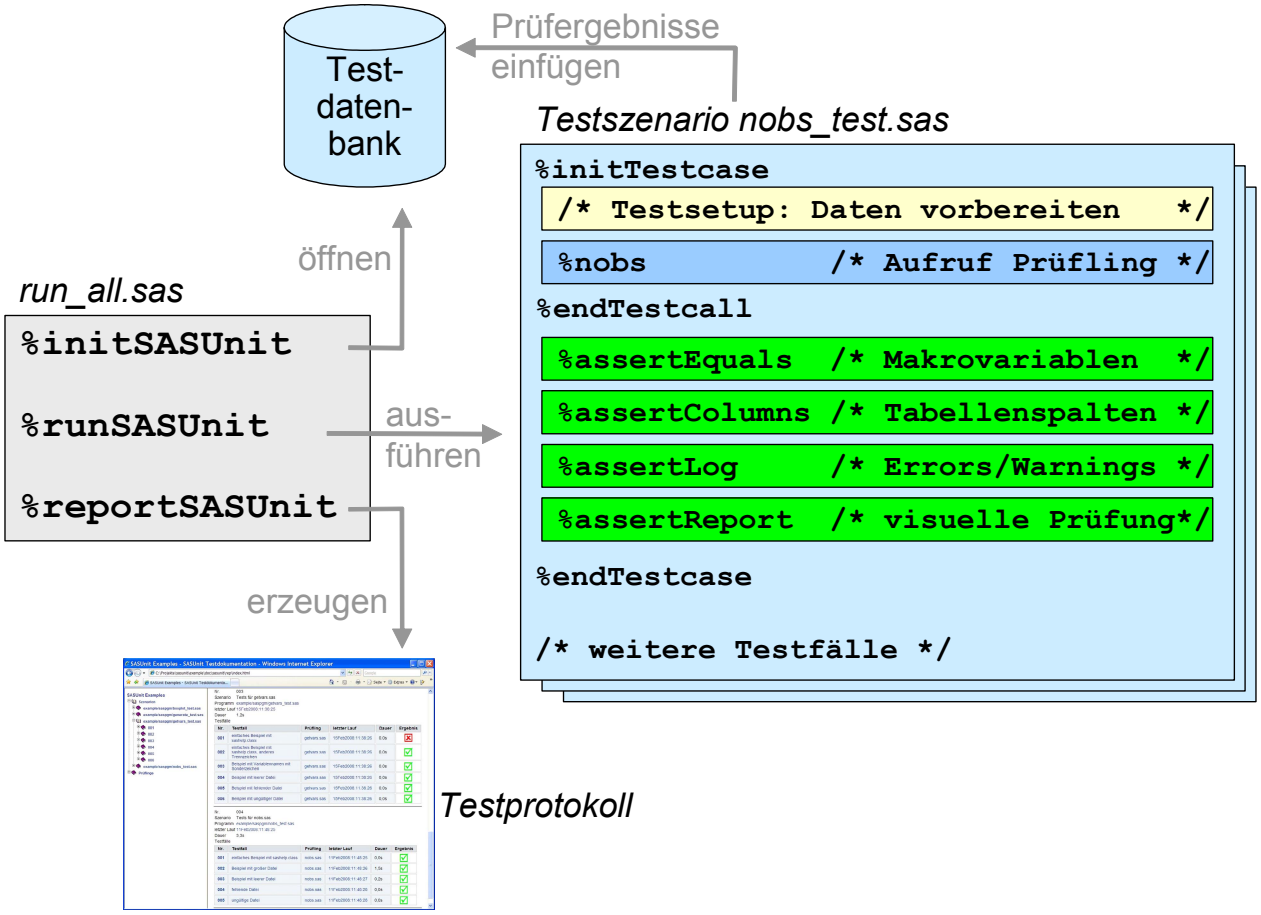

Abbildung 3: Aufbau von Unittests mit SASUnit

SASUnit basiert vollständig auf SAS und Betriebssystembefehlen.

Die Steuerung der Testausführung erfolgt in einer SAS-Sitzung im Programm run_all.sas.

- %initSASUnit öffnet die Testdatenbank, bestehend aus mehreren SAS-Dateien, SAS-Logs und weiteren Ergebnissen, die von den Testfällen dort eingefügt werden. Wenn die Testdatenbank noch nicht existiert, wird sie leer angelegt. Außerdem werden Makrovariablen, Librefs und Filerefs für die weitere Verarbeitung definiert.
- %runSASUnit erhält als Parameter eine Dateispezifikation, z.B. in der Form saspgm*_test.sas (alle Testszenarien im Verzeichnis saspgm ausführen, deren Name mit _test.sas aufhört). Jedes gefundene Testszenario wird in einer eigenen SAS-Sitzung ausgeführt, allerdings nur dann, wenn entweder das Testszenario selbst oder einer der darin referenzierten Prüflinge seit der letzten Ausführung modifiziert wurden.
- %reportSASUnit erstellt aus der Testdatenbank ein Testprotokoll mit Navigationsframe im HTML-Format. Das Testprotokoll ist gegliedert nach Prüflingen, Testszenarien, Testfällen und Prüfungen.

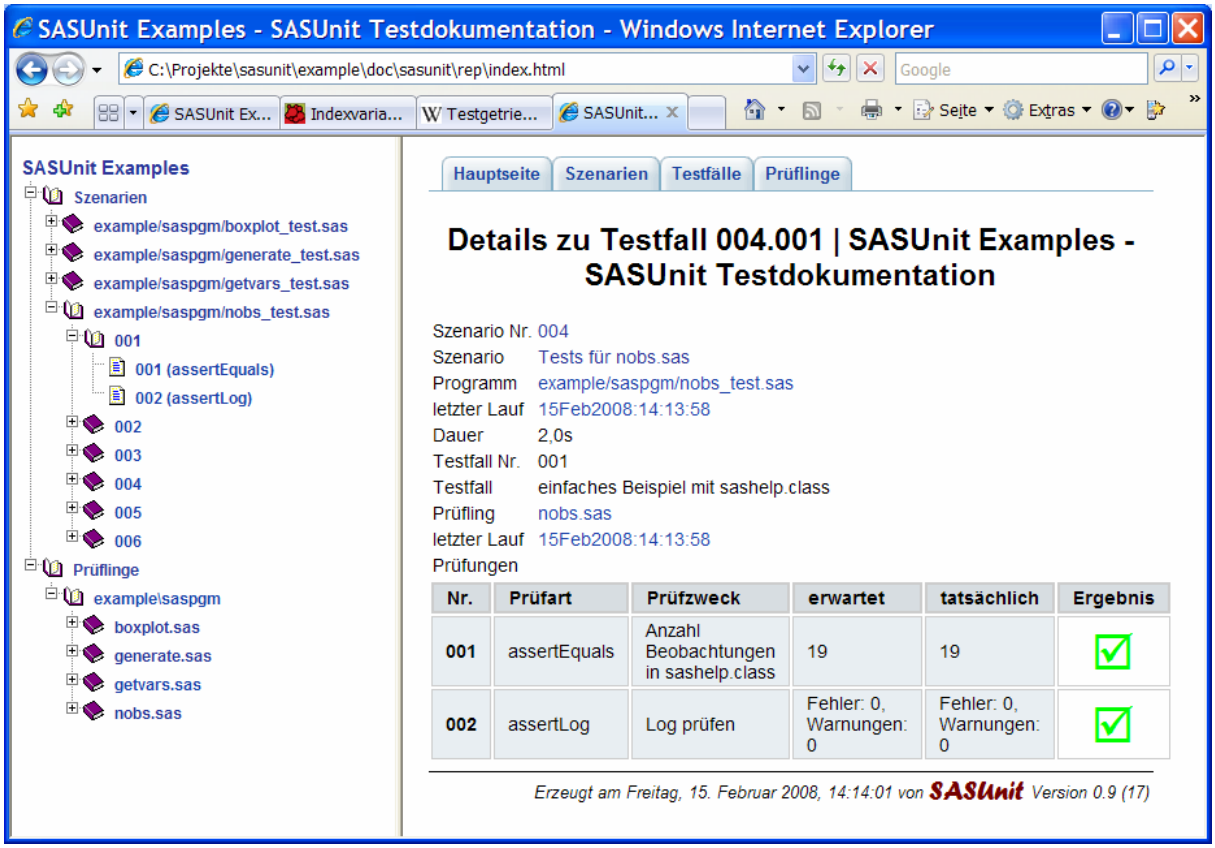

Abbildung 4: Testprotokoll mit Navigationsframe

Jedes Testszenario enthält einen oder mehrere Testfälle, die wie folgt aufgebaut sind:

- Aufruf von %initTestcase. Testbeschreibung und Name des Prüfling (hier: nobs.sas) werden in die Testdatenbank eingefügt. Der SAS-Log wird umgeleitet in eine eigene Log-Datei pro Testfall.
- Testsetup: Vorbereitung der Testdaten oder Zugriff auf statisch bereitgestellte Testdaten.
- Aufruf des eigentlichen Prüflings.
- Aufruf von %endTestcall: die Umleitung des SAS-Logs wird rückgängig gemacht.

- Prüfung der Ergebnisse, die der Aufruf des Prüflings zurückgegeben hat:
  - %assertEquals prüft den Wert einer Makrovariablen. Im Testprotokoll stehen erwarteter und tatsächlicher Wert der Makrovariablen sowie eine Markierung für Erfolg oder Misserfolg der Prüfung.

Nr.	Prüfart	Prüfzweck	erwartet	tatsächlich	Ergebnis
001	assertEquals	Anzahl Beobachtungen in sashelp.class	19	19	☑
001	assertEquals	Anzahl Beobachtungen in sashelp.class	20	19	☒

  - %assertColumns prüft Spalten in SAS-Tabellen. Im Testprotokoll erscheinen Hyperlinks auf ein Listing der tatsächlichen und der erwarteten Tabelle sowie auf einen mit PROC COMPARE erzeugten Vergleichsbericht.

001	assertColumns	die Beschreibung	Tabelle	Tabelle Bericht	☑

  - %assertLog scannt den SAS-Log und überprüft die Anzahl Fehler und Warnungen. Wurde nicht die erwartete Anzahl Fehler oder Warnungen gefunden (normalerweise 0), so wird das Symbol rot.

002	assertLog	Log prüfen	Fehler: 0, Warnungen: 0	Fehler: 0, Warnungen: 0	☑

  - %assertLogMsg scannt den SAS-Log nach einer ganz bestimmten Meldung. Das Symbol wird grün, wenn die Meldung gefunden wurde.

001	assertLogMsg	Logmeldung prüfen	Meldung "ERROR: boxplot: Datei XXXXX existiert nicht" vorhanden	Meldung gefunden	☑

  - %assertLibrary vergleicht komplette SAS-Bibliotheken und integriert Bibliothekslistings und einen Vergleichsbericht in den SAS-Log.

001	assertLibrary	Libraries prüfen	Bibliothek	Bibliothek Bericht	☑

  - %assertReport ist dafür da, mit ODS erzeugte Reports zu prüfen. Da eine inhaltliche Prüfung bei Reports nicht automatisierbar ist, wird nur geprüft, ob die Datei tatsächlich in dem Testszenario neu erstellt wurde und wenn das der Fall ist ein leeres Kästchen in das Testprotokoll integriert, so dass man eine Checkliste für die manuelle Prüfung erhält. Ein Link auf den Report wird direkt in das Testprotokoll integriert.

001	assertReport	bitte vergleichen Sie die Grafik mit der Spezifikation im Quellcode		.rtf	☐

4 Fallbeispiel

4.1 Der Prüfling: nobs.sas

```
%MACRO nobs(
   data
);
%local dsid nobs;
%let nobs=;
%let dsid=%sysfunc(open(&data));
%if &dsid>0 %then %do;
   %let nobs=%sysfunc(attrn(&dsid,nlobs));
   %let dsid=%sysfunc(close(&dsid));
%end;
&nobs
%MEND nobs;
```

Dieses Makro gibt die Anzahl Datensätze eine SAS-Datei zurück. Zum Beispiel:

```
%put %nobs(sashelp.class);
```

Gibt 19 in den SAS-Log aus.

4.2 Das Testszenario: nobs_test.sas

```
/*-- einfaches Beispiel mit sashelp.class ---------*/
%initTestcase(i_object=nobs.sas,
   i_desc=einfaches Beispiel mit sashelp.class)
%let nobs=%nobs(sashelp.class);
%endTestcall()
%assertEquals(i_actual=&nobs, i_expected=19,
   i_desc=Anzahl Beobachtungen in sashelp.class)
%endTestcase()

/*-- Beispiel mit leerer Datei --------------------*/
%initTestcase(i_object=nobs.sas,
   i_desc=%str(Beispiel mit leerer Datei))
data leer;
   stop;
run;
%let nobs=%nobs(leer);
%endTestcall()
%assertEquals(i_actual=&nobs, i_expected=0,
   i_desc=Anzahl Beobachtungen in Datei work.leer)
%endTestcase()
```

```
/*-- fehlende Datei -------------------------------*/
%initTestcase(i_object=nobs.sas,
   i_desc=%str(fehlende Datei))
%let nobs=%nobs(xxx);
%endTestcall()
%assertEquals(i_actual=&nobs, i_expected=,
   i_desc=Anzahl Beobachtungen bei fehlender Datei)
%endTestcase()

/*-- ungültige Datei ------------------------------*/
%initTestcase(i_object=nobs.sas,
   i_desc=%str(ungültige Datei))
%let nobs=%nobs(xxx);
%endTestcall()
%assertEquals(i_actual=&nobs, i_expected=,
   i_desc=Anzahl Beobachtungen bei ungültiger Datei)
%endTestcase()
```

4.3 Testprotokoll Hauptseite mit Einstellungen

Eigenschaften dieser Testsuite

Projektname	&g_project	SASUnit Examples
Wurzelverzeichnis	&g_root	**c:\projekte\sasunit**
Pfad zur Testdatenbank	&g_target	**example\doc\sasunit**
Programmbibliotheken (Makro-Autocall-Pfade)	&g_sasautos	**example\saspgm**
Verzeichnis mit Testdaten	&g_testdata	**example\dat**
Verzeichnis mit Referenzdaten	&g_refdata	**example\dat**
Verzeichnis mit Spezifikationsdokumenten	&g_doc	**example\dat**
Pfad zu den SASUnit-Makros	&g_sasunit	**saspgm\sasunit**
SAS-Log des Reporting-Jobs		**example/doc/sasunit/run_all.log**
Symbole	&g_error &g_warning	ERROR WARNING
Anzahl Testszenarien		4
Anzahl Testfälle		40
Anzahl Prüfungen		78

Erzeugt am Freitag, 15. Februar 2008, 14:14:00 von **SASUnit** *Version 0.9 (17)*

4.4 Testprotokoll Übersicht über alle Szenarien

Nr.	Szenario	Programm	letzter Lauf	Dauer	Ergebnis
001	Tests für boxplot.sas	example/saspgm/boxplot_test.sas	15Feb2008:11:38:15	10,4s	☐
002	Tests für generate.sas	example/saspgm/generate_test.sas	11Feb2008:11:48:21	2,7s	☑
003	Tests für getvars.sas	example/saspgm/getvars_test.sas	15Feb2008:14:13:57	1,1s	☑
004	Tests für nobs.sas	example/saspgm/nobs_test.sas	15Feb2008:14:13:58	2,0s	☒

4.5 Testprotokoll für Szenario nobs_test.sas

Nr. 004
Szenario Tests für nobs.sas
Programm example/saspgm/nobs_test.sas
letzter Lauf 15Feb2008:14:13:58
Dauer 2,0s
Testfälle

Nr.	Testfall	Prüfling	letzter Lauf	Dauer	Ergebnis
001	einfaches Beispiel mit sashelp.class	nobs.sas	15Feb2008:14:13:58	0,0s	☑
002	fehlgeschlagener Test	nobs.sas	15Feb2008:14:13:59	0,0s	☒
003	Beispiel mit großer Datei	nobs.sas	15Feb2008:14:13:59	0,9s	☑
004	Beispiel mit leerer Datei	nobs.sas	15Feb2008:14:14:00	0,0s	☑
005	fehlende Datei	nobs.sas	15Feb2008:14:14:00	0,0s	☑
006	ungültige Datei	nobs.sas	15Feb2008:14:14:00	0,0s	☑

4.6 Testprotokoll für den ersten Testfall

Szenario Nr. 004
Szenario Tests für nobs.sas
Programm example/saspgm/nobs_test.sas
letzter Lauf 15Feb2008:14:13:58
Dauer 2,0s
Testfall Nr. 001
Testfall einfaches Beispiel mit sashelp.class
Prüfling nobs.sas
letzter Lauf 15Feb2008:14:13:58
Prüfungen

Nr.	Prüfart	Prüfzweck	erwartet	tatsächlich	Ergebnis
001	assertEquals	Anzahl Beobachtungen in sashelp.class	19	19	☑
002	assertLog	Log prüfen	Fehler: 0, Warnungen: 0	Fehler: 0, Warnungen: 0	☑