PhUSE EU Connect 2018

Paper PP21

Program Validation – Care for a black sheep

Andrea Weber, PRA Health Science, Mannheim, Germany

ABSTRACT

Several methods are available to validate statistical programming outputs. All those have in common the steps of identifying and analyzing issues as well as communicating them to the primary programmer. While guidelines for good programming practice exist, the validation practice is a rather unattended topic. So daily work shows scenarios like:

"You are the quality control (QC) programmer of an analysis dataset, required to do double programming. The proc compare shows 'hundreds' of differences and you are at loss how to move on. "

"You are the main programmer and receive a mail stating 'There are discrepancies' or 'There are mismatches in obs 35277' and you do not know more than you did before. "

This paper provides some tips to prevent those situations using the example of double programming. It discusses ideas on structuring the QC process, as well as identifying, analyzing and reporting issues to the main side programmer.

INTRODUCTION

In a CRO environment it is indispensable to deliver each kind of programming output in time and in high quality. To ensure the second, PRA applies an independent QC programming for all kinds of output, e.g. datasets, tables, graphics and listings. In the following it is assumed that a main (or primary) programmer and an independent QC (or second) programmer have to get through this process.

The paper starts with a rough re-cap on double programming and proceeds with main ideas on optimizing the QC based on two daily scenarios. This includes explanations on the root cause of common problems with tips for their resolution. It deals with identification and analysis of discrepancies as well as the report of issues to the main programmer. The paper closes with a final conclusion.

DOUBLE PROGRAMMING – ROUGH RE-CAP

Double programming implies a second programmer creating the same output as done by the main side programmer. Both programmers work independently from each other without gaining any information from the other programming side. For a report such a programming output is mainly the dataset feeding the report procedure. For analysis datasets or converted datasets this is the dataset itself. In both cases, the datasets are compared using an automated comparison procedure as SAS® proc compare. The comparison is done for the initial version of the dataset as well as each time the dataset is refreshed during the course of the study.

This is the focus of this paper, the necessary layout review of tables, listings and figures is not elaborated.

The examples of the paper uses ADVS as the main and QC_ADVS as the QC dataset.

OPTIMIZING THE QC

The following discussion focuses on two scenarios. Both can give a bulk of frustration to a less experienced programmer. One is a large number of discrepancies after a first proc compare run, the other is the way to communicate and solve issues together with the main side programmer.

SCENARIO I: PROC COMPARE SHOWS HUNDREDS OF DIFFERENCES

Plain proc compare

A large number of discrepancies could be caused by unequal numbers of records (observations) or an inconsistent sorting order in the main and in the QC dataset. Thus the output of a plain proc compare procedure created by a SAS code as shown below, could exhibit a large number of discrepancies.

```
proc compare base = advs compare = qc_advs;
run;
```

With this code SAS compares both datasets record wise irrespective if the records correspond to each other or not. The below example shows the first record of the two datasets

Example 1: Plain proc compare as a starting point

USUBJID	PARAMCD	ADT	USUBJID	PARAMCD	ADT
32202-001	HEIGHT	2017-01-06	96953-002	PULSE	2017-09-15

Subject 32202-001 is compared with subject 096953-002. Thus the compared records do not correspond to each other and show differences in (almost) all variables. This happens if both datasets are sorted differently or contain a different number of observations with the mismatch beginning from the first additional record on.
To circumvent this issue it is necessary to find out if the number of observations is identical in both datasets and whether the datasets are sorted in the same way.

Number of observations
To check the number of observations several options exist. So a frequency count of the main and QC dataset could be performed and the results compared against each other.
This can be done as a first QC step ahead of proceeding the proc compare in particular if the dataset is QC'd the first time.
If the proc compare has already run, the first part of the comparison output can be used to determine the number of observations. It already provides this information in column Nobs (Number of Observations) as shown in the following output snippet.

Example 2:

```
Data Set Summary

Dataset                 Created           Modified  NVar    NObs

WORK.ADVS      30AUG18:07:06:57  30AUG18:07:06:57   273    6414
WORK.QC_ADVS   30AUG18:09:11:47  30AUG18:09:11:47   273    6416
```

In this example the column NObs shows 6414 observations for the base dataset ADVS and 6416 observations for the compare dataset QC_ADVS.

The log output provided by the proc compare procedure also gives the number of observations and can therefore serve as source for the check as well.

```
NOTE: There were 6414 observations read from the data set WORK.ADVS.
NOTE: There were 6416 observations read from the data set WORK.QC_ADVS.
```

To resolve the issue it is necessary to further isolate the subjects and records being obsolete in the one or missing in the other dataset. So it is advisable to count the number of subjects for a subsets of data. In the below example the frequency count was done per PARAMCD showing a difference for the parameter HEIGHT.

```
proc freq data = advs  noprint;
tables paramcd  / out = _advs_frq;
run;

proc freq data = qc_advs  noprint;
tables paramcd  / out = _qc_advs_frq;
run;
```

Frequency count of ADVS (dataset _ADVS_FRQ)

	PARAMCD	COUNT
1	DIABP	266
2	HEIGHT	1575
3	PULSE	147
4	SYSBP	1102
5	TEMP	1749
6	WEIGHT	1575

Frequency count of QC_ADVS: (dataset _QC_ADVS_FRQ)

	PARAMCD	COUNT
1	DIABP	266
2	HEIGHT	1577
3	PULSE	147
4	SYSBP	1102
5	TEMP	1749
6	WEIGHT	1575

To find out the subjects affected by the issue a second frequency count was done per USUBJID for parameter HEIGHT only.

```
proc freq data = advs (where =(paramcd = 'HEIGHT')) noprint;
tables usubjid  / out = _advs_frq2;
run;
```

```
proc freq data = qc_advs (where =(paramcd = 'HEIGHT')) noprint;
tables usubjid  / out = _qc_advs_frq2;
run;
```

The two result datasets _ADVS_FRQ2 and _QC_ADVS_FRQ2 were then compared against each other by using proc compare.

```
proc compare base = _advs_frq2 compare = _qc_advs_frq2;
id usubjid;
run;
```

The following comparison output was created

```
The COMPARE Procedure
Comparison of WORK. _advs_frq2 with WORK. _qc_advs_frq2
(Method=EXACT)

Value Comparison Results for Variables
```

	Frequency Count			
USUBJID	Base COUNT	Compare COUNT	Diff.	% Diff
34904-013	9.0000	10.0000	1.0000	11.1111
37828-004	6.0000	7.0000	1.0000	16.6667

It shows the two subjects 34904-013 and 37828-004 having the count of 9, 6 respectively in the _ADVS_FRQ2 dataset and 10, 7 respectively in _QC_ADVS_FRQ2.
This means that in the main dataset ADVS the subject 34904-013 has 9 records for the parameter HEIGHT while in the QC dataset QC_ADVS it has 10. The same for subject 37828-004 with 6 records versus 7 records.
Now the subjects are isolated and can be further examined by checking their individual data. So if it turns out that the records are required and thus are missing in the main dataset the issue must be reported to the primary programmer to get it resolved before the QC can continue. If it turns out that the records are obsolete then the QC program must be amended before the QC continues.

Dataset sort

If the number of observations are matching the correct sorting order should be checked. A quick method is the usage of proc compare with the ID statement and the review of the corresponding log message.
In the below example the expected dataset key is USUBJID, ADT, PARAMCD.

Example 3: Proc compare with ID statement

```
proc compare base = advs compare = QC_advs;
id usubjid adt paramcd;
run;
```

This enforces SAS to compare the records by the variable(s) defined in the ID statement and requires both datasets being sorted by them. If that is not the case a corresponding log message is generated.

```
WARNING: The data set WORK.ADVS is not sorted by the ID variables. Observations will be matched
using NOTSORTED logic with the assumption that the observations correspond one-to-one.
NOTE: At observation 2 the current and previous ID values are:
      USUBJID=30703-003 ADT=19JAN2017 PARAMCD=DIABP.
      USUBJID=30703-003 ADT=27JAN2017 PARAMCD=DIABP.
ERROR: The ID variable values do not match at observation 2 in the base data set WORK.ADVS and
observation 18 in the comparison data set WORK.QC_ADVS.
(When one or both data sets are not sorted by the ID variables, or when NOTSORTED is specified,
the observations must match one-to-one.)
NOTE: The current ID values in WORK.ADVS and WORK.QC_ADVS are:
       USUBJID=D0816C00010/E0703003 ADT=19JAN2017 PARAMCD=DIABP.
       USUBJID=D0816C00010/E0703003 ADT=27JAN2017 PARAMCD=PULSE.
```

Already the first two sentences are sufficient to identify the sorting issue. The “WARNING” message provides the name of the dataset not being sorted by the identifier variable. The “NOTE” message shows the record number at which the wrong sorting becomes evident. It also shows the ID variable(s) name as well as the value of the current and previous record. In this example the main dataset ADVS is mentioned as not being sorted as exepcted from observation number 2 on. The ID variables are USUBJID, ADT, PARAMCD with the values ‘30703-003’, ‘19JAN2017’, ‘DIABP’ in the current and ‘30703-003’, ‘27JAN2017’, and ‘DIABP’ in the previous record. Thus the

date in the previous record is later than in the current record while the values of PARAMCD are the same. This gives the impression that the variables PARAMCD and ADT are reversed in the sorting key of the main programmer or a sorting key was not applied at all.
The remaining part of the message provides the impact of the sorting issue on the compare procedure. It is not worth being checked further because it only provide the consequences but no more information on the root cause of the problem.
As next step the expected sorting key as per specification should be checked. In this example the specifications states a sorting key of USUBJID, ADT, PARAMCD in which case the issue should be corrected on the main side before the QC can continue.

Size and complexity
After there is agreement in the number of observations and the corresponding sort order, the QC can go into detail. Depending on the size of the dataset or the complexity of the data it is advisable to structure the QC in meaningful steps and thus have multiple circles of it. This reduces the amount of issues to be analyzed in one step to a lower number and thus enables the programmers to stronger focus on the individual issues rather than being overwhelmed by the amount of them. This also increases the motivation by seeing more and more parts of data getting QC completed.
So, one option could be to split up the datasets into meaningful subsets of observations or to restrict to a bunch of variables and do a proc compare for each of them separately. Meaningful subsets should comprises all records of a particular category, parameter or visit (e.g. all records of PARCAT =‘A’, PARAMCD= ‘B’ or AVISIT = ‘C’). This depends on the content of the dataset and should be done in a logical way to easily find the section of the program causing issues.
A bunch of variables could comprise name, start and end of study treatment (TRT01P, TRTSDT, TRTEDT) or analysis flag, analysis result and analysis date variables (ANL01FL, AVAL, ADT).
In case the subsets are independent from each other this method also allows parallel work of the main and the QC programmer as the one can already implement the issues from the first subset while the other is reviewing the second subset and so on. This procedure should be applied with care, unless both programmers feel confident that they do not interfere with other variables or records.

To check the datasets per subset the proc compare can be modified as follows

```
proc compare base = advs compare = qc_advs;
id usubjid adt avisitn;
where paramcd = 'SYSBP';
run;
```

To check the datasets for particular variables the proc compare can be modified as follows:

```
proc compare base = advs compare = qc_advs;
id usubjid paramcd adt avisitn ;
var adt anl01fl aval;
run;
```

A combination of both is also possible.

```
proc compare base = advs compare = qc_advs;
id usubjid adt avisin ;
where paramcd = 'SYSBP';
var adt anl01fl aval;
run;
```

Alternatively the datasets can be split before being fed into the proc compare procedure.

Additional checks
Another option to reduce the amount of issues fired by a proc compare are high-level plausibility checks done before a detailed QC starts, because they could give a first impression on the data correctness. Such checks include but are not limited to frequency counts of categorical variables, simple summary statistics such as minimum, maximum on continuous variables, or visual comparisons of values against a controlled terminology. So e.g. a frequency count on subgroups or a visual check of values could highlight an expected value being missing or misspelled. Unexpected outliers for vital sign or laboratory values could give a hint on the usage of a wrong conversion factor. The following examples illustrate this.

Example 4: Counts of categorical variables

The results for the parameters Height and Weight should be categorized as shown in the table below:

PARAMCD	AVALCAT
HEIGHT	<130 cm including missing
HEIGHT	>130-150 cm
HEIGHT	>150-170 cm
HEIGHT	>170 cm
WEIGHT	30-50 kg
WEIGHT	<50 - 80 kg
WEIGHT	>80 kg

A frequency count on this parameter shows the following result:

	PARAMCD	avalcat
1	HEIGHT	
2	HEIGHT	130-150 cm
3	HEIGHT	>170 cm
4	WEIGHT	30-50 kg

For the parameter HEIGHT a missing category exists although even missing results should be categorized as '<130 cm including missing'. This is a clear programming issue and can be resolved ahead of commencing a proc compare.
For parameter WEIGHT only one category exist although three are expected. This could be a programing issue or caused by the data not having any results below 30 kg or above 50 kg. In this case further investigation of the source data is required.

Example 5: Simple summary statistic:

The minimum and maximum value for AVAL of parameter HEIGHT in a study for adults shows the following results:

	PARAMCD	PARAM	min	max
1	HEIGHT	Height (cm)	59.05515	180

Per parameter label (Height (cm)) the values are expected to be in the unit cm. Therefore a value of 59 cm for an adult is implausible and needs further investigation. To find out the corresponding subject, ADVS was reduced on all records where non-missing AVAL was lower than 60 and subject 34904-013 was selected. His raw data shows a value of 59.05515 but a unit of 'lb'. In this case the programming code for converting 'lb' to 'cm' failed or was not implemented.

Example 6: Visual comparison against controlled terminology

The following unique values of variable PARAMCD were selected from ADVS and compared against the ADaM specification.

ADVS

	PARAMCD
1	DIABP
2	HEIGHT
3	PUL
4	SYSBP
5	TEMP
6	WEIGHT

ADaM specification

Codelist / Controlled Term	Codelist Name	Data Value
PARAMCD_VS	Vital Signs Parameter Code	DIABP
PARAMCD_VS	Vital Signs Parameter Code	HEIGHT
PARAMCD_VS	Vital Signs Parameter Code	PULSE
PARAMCD_VS	Vital Signs Parameter Code	SYSBP
PARAMCD_VS	Vital Signs Parameter Code	TEMP
PARAMCD_VS	Vital Signs Parameter Code	WEIGHT

It became visible that parameter PULSE is not spelled correctly what is the result of a programming issue.

All the above mentioned issues were detected without the need of proc compare. Therefore these checks could also be performed by the main side programmer to increase the quality of the output already before releasing it to QC.

The same holds true for the QC programmer who also should ensure a high quality of the QC output before starting the compare against the primary output.

Dependencies
Data dependencies could lead to an increased amount of issues in dependent variables caused by mismatches in the parent data. So, a mismatch in the change from baseline variable (CHG) could be caused by a mismatch in the result variable (AVAL) or a mismatch in the variable containing the base value (BASE). A mismatch in the variable BASE could be caused by a mismatch in variable AVAL or a different flagging of the baseline record (ABLFL). Therefore in this case it is advisable to check the variables AVAL and ABLFL first. If those are correct it is expected that BASE is also correct and if not the issue is isolated to the programming section where BASE is derived. The same holds true for CHG. If AVAL and BASE are correct, then a correctly derived variable CHG will automatically match as well.

Example 7: Solving mismatches with dependencies between variables

In the below example a comparison output shows differences for AVAL, ABLFL, BASE and CHG

```
Variables with Unequal Values

Variable  Type  Len   Label                Ndif   MaxDif  MissDif

AVAL      NUM     8   Analysis Value          9   10.000        0
BASE      NUM     8   Baseline Value         46   20.000        5
CHG       NUM     8   Change from Baseline   51   20.000        5
ABLFL     CHAR    2   Baseline Record Flag    6                 6
```

The column Ndif (Number of differences) shows the number of observations with different values in the corresponding variable. So for AVAL it is 9, for BASE 46, for CHG 51 and for ABLFL 6.

After variable AVAL were corrected the comparison output shows the following remaining issues

```
Variables with Unequal Values

Variable  Type  Len   Label                Ndif   MaxDif  MissDif

BASE      NUM     8   Baseline Value         39   20.000        5
CHG       NUM     8   Change from Baseline   39   20.000        5
ABLFL     CHAR    2   Baseline Record Flag    6                 6
```

The discrepancy for AVAL has disappeared and the number of observations with mismatches in BASE and CHG have automatically been reduced to 39. The number of records with discrepancies in variable ABLFL has not changed as ABLFL is independent from AVAL.

After correction of ABLFL all discrepancies disappeared and the compare provides a clean output;

```
Number of Observations with Some Compared Variables Unequal: 0.
Number of Observations with All Compared Variables Equal: 735.
NOTE: No unequal values were found. All values compared are exactly equal.
```

A record-wise dependency example are the parameters Weight, Height and BMI. While the first two are independent from each other, BMI is usually derived using the both. So a compare of BMI related records makes only sense once it is ensured that the records for the parameters Height and Weight are matching.

SCENARIO II: ISSUES REPORTED IN AN USELESS MANNER

Communication QC findings to the main programmer
If not otherwise determined or agreed it is the QC programmer's responsibility to identify issues and to prove that those are caused in the main program rather than in the QC program.
So a note like 'There are differences' without any further explanation is not helpful. It is missing the information what differs (record, variable) and an assessment whether the main program is indeed incorrect. The primary programmer would have to repeat the QC steps to find this out. This leads to inefficiencies and repeated work.

Furthermore only true main side issues should be communicated to the main side programmer. This means that at best of his or her knowledge the secondary programmer should be sure that the QC program is correct. It is further advisable to explain why the main side is considered being wrong by indicating the expected result as per specification, SAP, shells or any other client specific or CDISC specific guidelines. If the root cause of the problem can already be assumed it could be helpful to mention it as well, however it is still the main programmer's responsibility to verify this. All this prevents the primary side programmer to repeat the QC steps to find the

necessary information of his or her own.

How to report issue identifiers
To enable the main side programmer to see the issue and analyze the root cause quickly, it is advisable to provide at least one example case that allows a reduction of the dataset to the affected record(s), also enabling the programmer to easily track the success of the programming update. The sample case, however, is often provided by mentioning the observation number ("obs number xx") the records have in the dataset the QC programmer used for examination. As the observation number is dependent from the (subset of) data that is stored in the dataset as well as the sorting order, it is a rather ambiguous identifier.

Example 8: True record identifier

In the below example the QC programmer has created a temporary dataset as a subset of ADVS by selecting all post-baseline records. He finds an issue in observations number 6.

	USUBJID	PARAMCD	AVAL	AVISITN	ADT
1	30501-002	DIABP	85	18	30NOV2016
2	30501-002	PULSE	76	18	30NOV2016
3	30501-002	SYSBP	128	18	30NOV2016
4	30501-002	TEMP	36.2	18	30NOV2016
5	30501-002	WEIGHT	96	18	30NOV2016
6	30703-003	DIABP	80	8	13OCT2016
7	30703-003	PULSE	93	8	13OCT2016
8	30703-003	SYSBP	100	8	13OCT2016
9	30703-003	TEMP	36.1	8	13OCT2016

This observations points to USUBUJID= 30703-003 with PARAMCD = DIABP, AVISITN= 8 and ADT = 13OCT2018

In the full ADVS however observations number 6 shows USUBUJID= 30203-005 with PARAMCD = WEIGHT, AVISITN= 1.2 and ADT = 03APR2017.

	USUBJID	PARAMCD	AVAL	AVISITN	ADT
1	30203-005	DIABP	80	1.2	03APR2017
2	30203-005	HEIGHT	162	1.2	03APR2017
3	30203-005	PULSE	80	1.2	03APR2017
4	30203-005	SYSBP	130	1.2	03APR2017
5	30203-005	TEMP	36.7	1.2	03APR2017
6	30203-005	WEIGHT	81	1.2	03APR2017
7	30207-002	DIABP	70	1.2	06APR2016
8	30207-002	DIABP	70	1.9	06APR2016
9	30207-002	HEIGHT	150	1.2	06APR2016

So in this example the primary programmer would examine the wrong case and would either not find the issue or would make an unwanted update.

As long as it is not 100% ensured that the dataset the QC programmer looks into, contains exactly the same subset with exactly the same sorting order as it is for the dataset the main side programmer has access to, only true record identifiers such as subject id and any further key variables (e.g. categories, parameters, dates, visit, terms etc.) should be reported.
So instead of mentioning observation number 6 the QC programmer should have at least provided subject number 30703-003 with PARAMCD = DIABP and ADT = 13OCT2018.

Circles of Reporting
To reduce the QC circles to a minimum it is advisable to provide as much issues as possible in one step. This allows the main program being touched once for a bunch of issues, rather than dropwise for separately reported issues. As outlined for data dependencies above, this is only meaningful if the issues are independent from each other.
In case of dependent data, it is more efficient to work on the parent data first as then the number of issues will be reduced automatically with an increase in effort already on the QC side as shown in example 7 above. In that example the values for AVAL and ABLFL are independent from each other and thus should be reviewed in one step. This allows the primary programmer to implement corrections for the two parameters at one time.
The variables BASE and CHG are dependent from AVAL and ABLFL and thus can only be reviewed in a second step after any issues for AVAL and ABLFL are resolved.

CONCLUSION

Several steps are possible to reduce the burden and the time needed for the QC. So a meaningful usage of proc compare reduces the amount of false-positive issue.
A limited size of data increases the visibility of progress and allows parallel work of the main and the QC programmer for independent data. It leads to an increase in motivation by seeing more and more variables matching as well as a shorter timeframe in which the work is completed.
Considering data dependencies by focusing on issues in the parent data first, reduces the amount of mismatches in dependent data and thus minimize the workload already for the QC process.
Identifying and providing as much issues as possible in one step reduces the amount of cycles for updating the main program as well as the amount of QC cycles with an increase of efficiency and a reduction of the overall time needed to get an output QC complete.

RECOMMENDED READING

Paper RG04
Validation Strategies for Report Objects and Programs –
Challenges and Obstacles in a Risk Based Reporting World

Paper 149-2010
PROC COMPARE – Worth Another Look!

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:
Author Name: Andrea Weber
Company: PRA HealthScience
Email: weberandrea@prahs.com.