PharmaSUG 2018 - Paper LD-12

What Makes a Good QC Programmer ?

Timothy J Harrington, DataCeutics, Inc.

ABSTRACT

This paper is a discussion of the methods employed by Quality Control (QC) Programmers, when performing QC by means of independent SAS® programming, for the most effective identification and reporting of discrepancies between a SAS Programmer/Analyst (SPA) and QC program set of results, with a given program specification. The discussion concerns both comparing the output of a program, that is SAS data sets, tables, listings, or graphic output, and on reviewing log files for coding issues and input data problems. Specific skills and traits are identified which are compatible with Good Clinical Programming Practices (GCPP) and the recently implemented 42 CFR Part 11 during the QC process. The content of this paper is based on many years of experience and past formal training of the author and of both current and prior project team members.

INTRODUCTION

The object of a QC review is to ensure that the contents of analysis files (created SAS data sets), tables, listings, and graphs is fully compliant with the specification, factually correct, numerically accurate, and is unambiguous. All applicable data must be included, but there must be no superfluous data present. A proper QC plan and effective application greatly increase the likelihood of identifying errors and potential problems early in the reporting process when corrective action is much less costly. Such a QC plan and process should be auditable and be seen to be in place and be well performed before any content is passed on to any FDA submission such as a New Drug Application (NDA) or Integrated Safety Study (ISS) review.

METHODS OF QC

There are three common methods of performing QC. Firstly, there is *Independent Programming*, where a QC programmer creates a SAS program from the same specification document, and using the same input data sources as used by the SPA, and then compares the results to reveal any differences and hence errors on the part of either programmer. There is an assumption that both programmers do not make the same programming error. Secondly, there is the *Structured Walk-though*, where a reviewer inspects the programmer's code and SASLOG file. This method is good for checking compliance with coding standards. Finally, there is the *Result Check*, where there is list, tabular, or graphic output. The content of the SPA and QC programmer outputs are manually checked against each other, expected results, and the table or chart mock-up (specification). Summary variables such as the total, mean, median, minimum, and maximum are visually checked. For example, the maximum for 'all patients' should be the same as the highest maximum for each of the patient groups. Results are also checked against other corresponding items in other tables, for example, the total number of patients in a given group should always be the same.

MANAGING QC WITHIN AN ORGANIZATION

Many organizations use a formal QC check sheet, even if this is not strictly followed in practice, such a document is a good basis for training new team members. Also recommended is a code management system to control the versioning of each program, maintain a chronological history of updates, protect programs from unwanted access, and provide an audit trail. Typically, there are three levels of hierarchy; Development, QA, and Production. As each program passes QC, it is promoted to the next hierarchy.

There is a need for a communication tool, which may be incorporated into the code management system, for indicating when a program requires QC, for the QC programmer to report actual or possible shortcomings, and for the SPA and QC programmer to communicate their findings and progress. This communication tool should maintain an audit trail of all typed correspondence. Such tools also provide a

means of accessing prior QC cases for finding solutions to past situations and are a standard for training, learning, and teaching within project teams.

The chronological sequence in which QC is performed on dependent programs is critical. Care must be taken to ensure dependent data sets are synchronized, that is a second program using the output of a first program must not be subject to QC until the QC has been successfully completed on the first program and its output.

QC OF ANALYSIS FILES (SAS DATA SETS)

A very useful SAS tool for validating a QC programmer produced data set against the corresponding SPA produced data set is SAS PROC COMPARE. This utility shows which variables and observations have differences between the corresponding values and lists any observations that are present in one data set but not the other. Column attributes are also compared and any differences are listed. Before using PROC COMPARE for this purpose, care must be taken to ensure appropriate options are set. METHOD should be ABSOLUTE and CRITERION should be set so insignificant rounding differences between corresponding numeric variables are not shown. A good enough accuracy of comparison is usually achieved by setting CRITERION to 0.00001. The SAS option VALIDVAR=UPPERCASE will prevent differences in upper and lower case letters of variable names that are spelled the same from being shown. PROC COMPARE does, however, indicate character variable contents case differences. Both the SPA and QC data sets should be sorted by a suitable sort key (BY variables, such as USUBJID VISIT LBTESTCD). If any observations in either data set have duplicate key values a message to that effect is written to the SASLOG. An important fact to note about using PROC COMPARE for QC is this procedure only shows the *differences* between two data sets. An error in the specification that is subsequently followed by both programmers will result in the same shortcoming in both data sets and will not show as a difference.

The following are the specific checklist of items the QC programmer should follow when evaluating an analysis file:

- All the required data set observations are included except any which match specification defined exclusion criteria, for example where PCSTATUS='NOT DONE'.
- There are no duplicated observations or observations with duplicate key values. If this happens the duplicate observations should have been dropped, perhaps observation selection criteria were not met, such as omitting a WHERE clause item. Another cause is a need for finer granularity of the sort-key, for example USUBJID VISIT LBTESTCD LBSEQ to handle multiple LBTESTCD values performed at the same VISIT.
- All required columns are present, including all columns from the source data set(s), and columns containing derived values. If a column is named incorrectly, this column will appear in one data set and the correctly named (or also incorrectly but differently named) column in the other.
- Columns are of the applicable data type, that is numeric or character. Sometimes flag variables are numeric with values 0, 1, 2, but flag variables can be character with values like '0', '1', '2'.
- Character variables must be of matching length and of sufficient length to prevent possible truncation, that is they must be long enough to contain the longest expected text string.
- Columns must be labeled and the labels match the contents, with correct spelling, case, punctuation, and suitably abbreviated terms where applicable. Labels should be 40 characters in length or less. When programming and assigning label texts care must be taken to use single quotation marks to enclose the text if an ampersand (&) or percent sign (%) is part of the text, so the SAS macro pre-compiler is not triggered. Similarly, when using single quotes a possessive apostrophe will cause problems such as an unclosed quoted string error.
- Where a column contains a derived value that value must be accurate, and if numeric and non-integer, of sufficient precision. The CRITERION option in PROC COMPARE should be set accordingly.

- Numeric data must be quantified in the correct and corresponding units of measurement. If the unit of measurement is specified in a column label, it must correspond. Note: Sometimes there is a separate character unit of measurement column, so the contents must correspond to the numeric unit of measurement. Sometimes, when a patient has no such values, the numeric column has all missing values for the patient and the character column is left blank. A useful SAS utility for checking grouped counts, including any missing values, is PROC FREQ.
- When there is a finite range of values, whether numeric or character, (which may or may not include missing) there should be no values present which are not in that range.
- Character strings contain text which is factually correct and meaningful, is spelled correctly, has good grammar and punctuation and is in the required case (upper, lower, or propcase). Abbreviations, if any, should be the standard abbreviations. Note: Some texts, such as investigator comment fields, are verbatim.
- When formats or informats are used the format labels must be checked as well as the ranges of values that map to each format. Format libraries must be in the proper hierarchy. When using formatted numeric values care must be taken with rounding and truncation errors. Decimal precision handling may be different on differing operating system platforms.
- When conditional selection is used, for example IF THEN ELSE, SELECT IF, or a PROC SQL CASE construct, a check should be made not only for assigned values but for failing values. An example is a character variable SEX, which should be 'M' or 'F' or possibly ‘Unknown’ but may, for any reason, be blank for one patient. There should always be a final unconditional ELSE, and an OTHERWISE clause with SELECT IF, and a final ELSE clause before the END statement when using the PROC SQL CASE construct.
- Some columns in a data set should not normally be missing, examples are date of birth, sex, race, and ethnicity. If such data was not, for some reason, collected there should be a defined 'unknown' value (which may be blank).

There are also data dependencies to consider, such as an end time must be later than a start time (Under some conditions the end date may equal the start date or be missing). If a flag such as DIEDYN is set to 'Y' there must also be a date of death, which should be after all other event dates for that patient in the entire study! An exception in this case a date of autopsy - which must be on or after the date of death. If the death date is missing the patient is assumed to be still living and DIEDYN should be 'N'. Another example is as a pregnancy test performed on a male patient. There may be ambiguities over some items, such as the number of times a patient has given birth. The specification document should indicate a missing value for situations which are not possible, as opposed to zero for an event which has not occurred but could have. For example, when considering the number of times a patient has given birth, for a male patient, this would be missing. For female patients, the number of times should be the number of times, which includes zero for a female who has never given birth. There is also an ambiguity, that is when a patient gives birth to twins the question arises as to whether this is one or two births. This birth data should also be checked against related data such as age and whether the patient has had an earlier sterilization procedure such as a hysterectomy. This check should not be performed against methods of birth control because there is no certainty, contraception can fail, but an apparently high failure rate should be questioned. A dependent data item is the method of birth, which can only be ‘Vaginal’ or ‘C-section’ for females who have given birth, and can only be missing (blank) for males and for females who have never given birth.

When working with data sets with relatively small numbers of observations there is a significant possibility of a particular situation which could happen not happening. An example is with AE data. When there are a large number of patients there is an increasing likelihood of a patient being there who had no AEs at all. This should be allowed for. A successful independent programming QC may occur initially, but when more patients are added to the study data, and one of them has no AEs a problem will occur if this situation was coded.

TABULAR OUTPUT

QC guidelines similar to those listed above for analysis files also apply to tables, but there are these following extra issues, specific to the QC of tabular output.

The table must have the correct table ID number and correct title.

The table layout must match the table mockup (specification).

Table, column, and row titles must match the table contents. Examples are patient population treatment group or a demographic subset. Text in all headings, labels, annotation, and footnotes must be factually accurate and free of ambiguities. A illustration of this type of text is "Half fresh grapefruit" could mean a whole grapefruit that is half-fresh, or one of the halves of a cut in half fresh whole grapefruit. Headings and labels should be appropriately centered or justified and be in upper case or mixed case as required. Spelling, punctuation, and grammar are most important too. A spell checking utility can help but care must be taken with a similar but incorrect word that is spelled correctly, an example is 'Manger' instead of 'Manager' or 'Altar' instead of 'Alter'. When checking spelling a medical dictionary spell check should be used, since some medical terminologies may not be recognized. Footnote references must be correct and clear using numerically or alphabetically ordered superscripts. All texts should be inspected for truncation or wrapping into adjacent cells, rows, or across page boundaries.

Table columns and rows must be appropriately ordered, alphabetically, by decreasing frequency, or otherwise as defined in the specification and table mockup. Summary columns and rows, such as 'All Patients' must be present if required.

Repagination is important. Page breaks should be in appropriate places, such as after summary rows or the end of a related group of rows. Footnotes must not straddle a page break or be truncated by the edge or end of a page. Pages must be numbered 'Page x of y' where x is the current page number and y the total number of pages for the whole table, this assures the reader that all of the table's pages are present, and should be followed even when the table covers only one page.

A table may be well laid out but be factually inaccurate. Obviously, the values being shown must be truthful and displayed with the required decimal precision, and, where applicable, using correct formats. Summary rows and columns should be manually checked. Figures in a 'Total' or 'All Patients' column must be the total of the corresponding figures in the relevant columns, for example, the maximum in the totals column or row should be equal to the highest figure in the individual columns or rows. Statistics must be the correct statistics, the mean must be the mean, and the median the median. Such statistics can be verified using a SAS utility such as PROC MEANS or PROC UNIVARIATE, or PROC FREQ. More complex statistics such as p-values should be evaluated using the applicable SAS procedure (or a previously verified applicable department macro) stated in the specification document.

What is not in a table is just as important as what is. Specific issues to check are results which are missing or zero, and absent data, a row or column with no data, or in some cases a table with no result at all. Absent data must be appropriately indicated and such indication must be consistent within a study.

Finally, tables must be checked against other tables, even tables that are unrelated. If all tables for a study are produced at the same time with the same input data the contents must be consistent across the study. An example is the patient totals for treatment groups should be the same for the same patient population in all of the tables, listings, and graphs where that total is displayed.

A problem with tabular QC can be a table's sheer size. Examples are AE summaries by treatment group and body system, vital signs, and laboratory findings. If there are a large number of patients these tables can easily exceed 100 pages and have more than 50,000 individual numbers. Checking many pages with many figures is not only tedious and time consuming but errors are more likely to be missed. In such situations spot checking is recommended. This is checking the first and last pages and a few randomly chosen other pages, or a randomly chosen sub section, such as a particular body system in an AE table. Any overall summary figures, such as the total AE count for each body system, and the summary values for all the body systems are good checkpoints. Another good data verification is the extremes such as the highest totals and any zero or small totals. An example is the body system in an AE table with the most

AEs reported. A large total being slightly too large or too small may be indicative of a problem with a MERGE, multiple occurrences of a particular AE, or data sub-setting criteria.

What has been discussed so far regarding tabular output QC is strictly manual QC methods, the best way to verify any table of any size is to use a standard output package which produces exactly the same formatting for the results of both the SPA and QC program. Typically, the program creates a SAS data set of the results, which is then used by the standard output package to produce the tables. After the QC programmer produces their output, this is compared exactly with the SPA output. An example of such a comparing tool is UNIX DIFF. Any difference is highlighted making one mismatching figure clearly visible amongst many matching figures. A timestamp on the table will show the difference in the time for each time that table is produced. Even if such a comparing tool is used the manual checks described above should still be performed, since conformity to the mock up and the possibility of both programmers making similar misinterpretations of the specification needs to be taken into account.

LISTINGS

Performing QC on a listing is very similar to performing QC on a table, but typically, there are fewer numeric items and only a small number of columns, but listings are sometimes many pages in length. Listings should be checked against related tables. Totals should match the number of corresponding entries in the listing.

GRAPHS, CHARTS, AND FIGURES

The guidelines for performing QC on graphic output are again verify similar to those of tables. The QC programmer my produce their own rough plot to compare with the figure, or a listing of the values of the points at the plotted points to check against the graph trace. Extra considerations regarding graphic output are:

The graph type (such as a line graph, bar chart, whisker plot, blobogram) must be the same as detailed in the specification.

The legend must match the graph color, shading, line type, and icon scheme. All data displayed must be present in the legend.

Any annotation required in the specification must be shown in the correct place.

The axes must be correctly labelled and show the applicable units of measurement. Axis should have markers and quantity labelling at appropriate intervals. Labels must not be truncated or overlap. Axes ranges should be such that all plotted points lie within the graph area but are not too crowded together. The plot must match the axes scales, for example logarithmic axes should show the plot of the logarithmic data and be labelled as such.

VERIFIYING THE SAS LOG FILE

Both the QC and SPA SASLOG file outputs must be checked for the specific messages and situations described below, either by using a log file scanning utility program or by opening the log file as read only in an editor and searching.

If the program stopped executing before reaching its end the cause is likely a syntax or lexical error, such as an unclosed IF THEN END or DO block. Setting the RUN statement to RUN CANCEL in the applicable DATA step(s) will cause the SAS compiler to check the SAS code statements but not attempt to execute the code. Another cause is an expected input data set or library is not found or an expected variable in a KEEP, WHERE, or RENAME statement is not on that data set. Quotation marks can be problematic, an unclosed quotation mark, just as with an incorrectly placed comment delimiter, will result in a large amount of code being skipped.

Specific items a QC programmer should look for in the SASLOG are:

- Any ERROR messages, including all types of error, whether syntax, lexical, or run time. The word 'Error' should be searched for in upper, lower, and mixed case. This would include the SAS system

variable _ERROR_ being set to 1. Amongst the causes of runtime errors are an input data set, or variable name is not found, attempted division by zero, or an invalid subscript in a SCAN or SUBSTR function. The SAS automatic variable _N_ indicates the observation number where the error occurred. The log file may also indicate the row and column number where the problem happened and the values of all the variables at that point. Using a PUT _ALL_ statement displays all the variables and their contents at that point. PUT _NUMERIC_ and PUT _CHARACTER_ display all the numeric or all the character variables respectively.

- Any WARNINGS, even if the program output is good, indicates a flaw in the program code. Examples are non-existent variables in a DROP or KEEP statement or the length of a variable is defined more than once, which could lead to text truncation.
- Any variables which are uninitialized. Uninitialized variables are undefined or unused variables. Such an indication may be the result of a mistyped variable name or a variable was defined in a LENGTH statement or given a FORMAT but was never used, or a variable no longer needed was not deleted.
- Repeats of BY variables in MERGEs. This is an attempt at a 'many to many' MERGE, which may produce a large Cartesian product. This is also an indication of one or more duplicate key values which should be unique. MERGEs performed without due care can, under some conditions produce reasonable but still incorrect results.
- Implicit type conversion. A message such as 'Numeric values were converted to character at ...' indicates an incorrect data type, or an improperly formatted PUT or INPUT. Values may be truncated, or be set to an incorrect numeric precision, or be set to missing. PUT and INPUT are strongly recommended over implicit conversion when converting between data types.
- Generated missing values. The message 'Missing values were generated at ...' may indicate a section of code or a SAS function has come across a missing value as input. The general rule is the SAS system propagates any missing values in arithmetic operations. This may also be a missing result of a function with improper arguments, such as looking up the square root of a negative number, or attempted division by zero. Searching for the expressions 'Invalid', 'Not valid', 'not a valid' may reveal the site of such a problem.
- Searching for and finding the text 'Not found' indicates an attempt at reading a variable which is not in a data set (This should produce an ERROR or a WARNING anyway), KEEP and DROP listings should be checked. Another cause is a reference to a data set which does not exist, in this case data set names and libraries, including any LIBNAME statements should be checked.
- Another useful source of information is observation counts, which should be checked for consistency and any unexpected occurrences of ' 0 observations were read from/written to', or unbelievably large observation counts, or created data sets which should be empty having observations present. These situations may be caused by a missing or incorrect WHERE clause, a problem with a MERGE or PROC SQL JOIN clause.

A good practice is to annotate the log output by printing, using a PUT or %PUT line of text indicating expected results at specific points. Examples are:

```
%put Now Executing Macro &sysmacroname;


proc sql noprint;
  select count(distinct trt) into :ngrps from ana.asl;
  select distinct trt into :trtgrps separated by ‘ ‘ from ana.asl;
quit;

%put Number of Patient Treatment Groups in the Report = &ngrps;

%put Treatment Groups are: &trtgrps;
```

```
data _null_;
  set sdtm.pc;
  if shcm ne ‘ ‘ then do;
    put ‘Sample Handling Comment Text for’ usubjid= visit= sample_id= shcm=;
  end;
run;
```

PROC FREQ is available to show summaries of data set contents and provides a quick check on a created data set. For, example, this code will show the demographic summary of a group of patients:

```
proc sort data=demo(keep=usubjid age sex race ethnic) out=demotest nodupkey;
  by usubjid;
run;

proc freq data=demotest;
  tables age;
  tables sex;
  tables race;
  tables ethnic;
run;
```

Another useful SAS tool is the MPRINT and MLOGIC options. These show how macro variables change during program execution. All %MEND statements should have the name of the macro they refer to. All macro variables within a macro should be declared as %LOCAL, this is something to check if a macro variable has an unexpected value.

QC PROGRAMMER TRAITS

How effectively all these items discussed above are followed depends on both written procedures such as SOPs and the skill and experience of the QC programmer (and of the original SPA). Troubleshooting requires a detective and investigative mentality, looking for possible, as well as actual shortcomings, that is a need to ask 'what if' questions. A problem found during a QC program run is a symptom, the QC programmer must find the cause to identify if the problem is in their program. Only when the problem is strongly believed to be with the SPA results should the QC programmer communicate this finding as a possible SPA error. Such communication should be courteous and diplomatic as well as brief and to the point showing an example of the erroneous output and an example of the data causing the problem, such as one of the patients whose result does not agree. The QC programmer should also ask the right questions of the SPA when in doubt or there is ambiguity in the specification. In many situations there are tight timelines, but the QC programmer should be result rather than process focused and must keep their SPA and team leader informed and up to date with progress. Missing a due date by a modest margin is generally much less serious than finding a problem with a program after it has already been used in the production environment.

The QC programmer must be resourceful when unsure with a given situation. A quick email to a coworker who is known to have prior experience in the same field is good way to learn and pass on knowledge. Many organizations have knowledge databases of accumulated companywide facts and experience, and there are also internet search engines. University and college websites often have solutions to quite complex analytical problems, as do CDISC, PHUSE, and other relevant SAS websites. Then there are past papers and presentations from prior relevant conferences such as the SAS Global Forum (formerly SUGI), regional SAS Users' groups, and relevant organizations including CDISC, PhUSE, and PharmaSUG.

Experience is rightly considered a strong asset, but ironically, lack of experience has an advantage, that is a new team member may spot and report an issue existing team members have become so used to they no longer feel it is serious. New facts often become known when someone new reviews a ‘cold case’.

CONCLUSIONS

To minimize the likelihood of inaccurate submissions to the FDA or any regulatory body QC must be effectively implemented and be fully auditable for all data processing steps prior to such a submission. Methods of QC should be consistent across all studies within an organization and be fully documented (e.g. training manuals, SOPs). Audits should be performed regularly. There must be open communication and in depth understanding of the issues involved. A company-wide system for monitoring QC submissions and communicating findings is strongly recommended. A detective mindset and an ability to learn from past problems is very much an asset for a QC programmer. Whilst there are often tight timelines to be met, in the long run quality is more important than exact timeliness.

ACKNOWLEDGMENTS

The author would like to express gratitude to the following for their assistance in writing and reviewing this paper:

Jeffrey Dickinson, Sharon Hall, and Kathy Greer at DataCeutics, Inc., 215 West Philadelphia Avenue, Boyertown, PA, 19512 | 610.970.2333 | info@dataceutics.com.

Paul Slagle, David D'Attilio and Mira Shapiro PharmaSUG 2018 Academic Chair, at AcademicChair@PharmaSUG.org

REFERENCES AND SUGGESTED FURTHER READING

Paul Gorrel, 2003, "Quality Control with SAS® Numeric Data" Annual Northeast SAS Users' Conference, Washington DC. https://www.albany.edu/~msz03/epi697/papers/at001.pdf

Prashanthi Selvakuma, 2014, "QC Made Easy Using Macros" Paper CC-33 Annual Pharmasug Conference San Diego, CA. https://www.pharmasug.org/proceedings/2014/CC/PharmaSUG-2014-CC33.pdf

Frederick P. Brooks, Jr. 1982, "The Mythical Man Month – Essays on Software Engineering", University of North Carolina, Chapel Hill. Addison-Wesley, Inc.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Timothy J. Harrington
DataCeutics, Inc.,
215 West Philadelphia Avenue,
Boyertown,
PA 19512
(610) 970 2333
harringt@dataceutics.com