Paper SP11

Documentation, Applicability and Automation: How Best Practices Can Help You Achieve Best Results.

Aadesh Shah, GSK, Collegeville, USA
Susruth Ganoothula, GSK, Collegeville, USA

ABSTRACT

How best practices can help you achieve best results, this paper will walk you through briefly on documentation, what to document and how much to document as well as the factors of traceability, validation and accountability. Applicability talks about dependency around clinical programming and how we can secure it manually if no automated checks are available using a Quality Gate process. The last part of this paper talks about Automation, Automated checks can provide the rigidity of double programming with more efficiency, but only if time is invested upfront to develop the tools and processes. It can lower the potential of human error and be applied and executed faster than manual processes.

DOCUMENTATION

What to document: In pharmaceutical industry, it is implied that 'if it is never documented, it didn't happen'! While considering need of documentation in programming process, it is important that all programming functional specifications as derived from SAP needs to be documented with necessary and enough details. Any changes in programming specifications as requested by biostatistician and other stakeholders need to be documented as a part of functional specifications. As per 21 CFR part-11, all major revisions in SAS code as well as changes in functional specifications are required to be documented. Similarly, it is important to document QC findings, dates, and resolutions. All approvals must be signed off and should be appropriately dated.

How much to document: If the documentation is so much important, then obviously next immediate question that comes in mind is how much to document? From QA perspective there is some 'necessary' set of documentation which includes documentation of programming functional specifications, and decision logs. Besides that, any important deviations and changes need to be documented as a part of 'note to file'. All approval forms should be documented in a timely manner with necessary details. Besides this necessary documentation, some other documentation which provides more detailed information about specifications, audit trail, and issue logs can be considered as 'sufficient' documentation. Before deciding the necessity or sufficiency of the documentation, it is important that the programmer consults with appropriate management representative and if needed with the quality assurance representative regarding documentation needs.

TRACEABILITY

The design of ADaM datasets and associated metadata should facilitate explicit communication of the content of, input to, and purpose of submitted ADaM datasets. The Analysis Data Model should support efficient generation, replication, and review of analysis results. To satisfy these requirements, the quality of traceability plays an inevitable role.

Metadata Traceability: provides the information about data i.e. origin of variable, algorithm used to derive the variable etc. It establishes traceability between ADaM data and SDTM data by describing the algorithm used to derive or populate an analysis-ready variable, a parameter or a record.

Data Point Traceability: enables users (agency reviewers, QC programmers, Biostatisticians etc.) to go directly to the specific SDTM data record(s)used to derive an analysis value. This level of traceability is straightforward when a user is trying to trace a data manipulation path. It can be established by providing clear links in the data to the specific data values used as an input from predecessor to derive an analysis value.

Report Traceability: a virtual traceability that connects the analytical results to the ADaM data. This traceability allows users (agency reviewers, QC programmers, Biostatisticians etc.) to trace clearly a complex analysis report value to the underlying data, to PROC AWAY the analysis result easily, and to be able to reproduce the outputs with less hassle.

VALIDATION

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 production programmer, 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 production programmer 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.

Many organizations use a formal QC tracking sheet, even if this is not strictly followed in practice 😊, such a document is a good in practice. 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.

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

1. Tracking ADaM data back to SDTM; Here SAP is the Master and specification is support; Fitness of computational algorithms with required analyses is result.
2. Checking compliance with the ADaM Implementation Guide
3. Checking compliance of analysis variables and its derivation with required analysis results.
4. Pinnacle21 Validator report.
5. Create a study specific QC checklist which can check basic ADaM principle.

APPLICABILITY

Most organizations use a hierarchical folder structure to organize datasets, tables, listings, graphs, programs and associated validation work. Some systems have a built-in framework that automatically tracks analysis to raw/analysis data dependency. If such a framework is not available, programmers can keep track of dependencies by manually documenting them. The dependency list would then determine the order in which analysis data-set programs will be run.

An example below of Quality gate (QG) refers to check point in a clinical study process including programming at pre-defined milestones to assess if all existing processes as defined in SOP (Standard Operation Procedure) are followed until that point. A QG involves formal review of pre-selected relevant documentations for sufficiency to gauge satisfactory execution of previous project milestone before team starts work on next milestone. In a clinical programming processes QG's are placed at three pre-defined project milestones. First programming QG (Gate1) review occurs before start of programming, to make sure proper programming related initial set of documents (list of tables, table shells and SDTM logical data maps, analysis dataset specifications etc.) are created appropriately. Second QG (Gate 2) document review occurs about 10-15 days post LSLV (Last Subject Last Visit) assessing completion of programming code, validation and blinded data test run reviews. QG reviewers expect updated QG1 documents along with test run comments and resolution logs. Final programming QG (Gate 3) review occurs after completion of final study reports post database lock. At QG3 programming documents are reviewed for satisfactory execution of programming processes before commencement of medical writing Quality gate ensures efficient and orderly execution of programming tasks by reducing risk of programmers getting ahead of themselves and working out of sequence (e.g. completion of analysis datasets before specifications are complete) potentially increasing risk of human errors and hence impacting programming efficiency.

Other uses of QG would be tracking dependencies of date time stamps within the datasets process flow. Raw datasets -> SDTM -> ADaM -> TFL's, the documentation can consist of start and end dates of each dependencies (e.g. SDTM Feb 6th, 2019 – 11:45 AM – 2:45 PM) should be sequential to the ADaM datasets and TFL's.

TEMPLATE

Validation Checks	Date/Time Stamps
QC Spreadsheet Check	
Confirm appropriate raw/external data have been used. *Record type and date(s) of raw/external data.*	ECG – 08/29/2019 RAW – 08/29/2019 LAB – 08/29/2019
SDTM datasets/SDTM define.xml have been updated. *Confirm date/time stamps occur after date/time stamps of raw data.* *P21 Report run/commented*	Prod – 08/30/2019 – 2:48 – 2:56 PM QC - 08/30/20169 – 3:15 – 3:18 PM
Analysis datasets/ADaM define.xml have been updated. *Confirm date/time stamps occur after date/time stamps of SDTM* *P21 Report run /commented*	Prod: 7Sep2019 1:01 – 3:06 PM QC: 7Sep2019 4:08 - 4:10 PM
Table outputs have been updated. *Confirm date/time stamps occur after date/time stamps of analysis datasets.*	Prod: 8Sep2019: 2:41 -3:04 PM QC: 8Sep2019: 3:30 - 3:32 PM
Listing outputs have been updated. *Confirm date/time stamps occur after date/time stamps of analysis datasets.*	Prod: 8Sep2019: 3:04-3:07 PM QC: 8Sep2019: 3:37-4:34 PM
Figure outputs have been updated. *Confirm date/time stamps occur after date/time stamps of analysis datasets.*	Prod: 8Sep2019: 2:41-4:10 PM QC: 8Sep2019: 4:32 PM
Logs are clear of warnings and/or notes have been explained. *Confirm logs clear or all warnings/notes have been adequately explained.*	
Archive Performed	*15-Sep-19*
Comments:	

AUTOMATION

Automated scanning of PROC COMPARE outputs:

The general rule for truly efficient programming is to use SAS macros only when they add significantly to the process. Simple code that is used repeatedly throughout the many programs makes it more appropriate for macro usage.

Automated checks can provide the rigidity of double programming with more efficiency, but only if time is invested upfront to develop the tools and processes. It can lower the potential of human error and be applied and executed faster than manual processes. The development of validated macros is one way to approach automated testing. Another real-life example is Pinnacle 21, which has been developed to help programmers validate datasets for CDISC compliance. In order for these tools to be effective, they need to be developed with a stringent set of parameters and guidelines and they are only useful with the validation programmers understand the application and limitations.

Scanning PROC COMPARE output using an SAS macro instead of following the manual review process. This SAS macro reads all the .LST files provided a path and creates a summary of the list files and indicating if it has issue or not and also the type of issue.

Using SAS macro, the approach would become more efficient. The macro performs the following:

Reads all .LST files in path provided

Looks for discrepancies, if any

Checks if the number of variables or observations read from datasets are equal or not.

Looks for warnings related to duplicated observations.

Looks for variable attribute differences

Here, the macro loops through all the .LST files present in particular path, performs all the required checks and at last, the macro produces a summary output which consists of the name of the list file, if it contains issues or not, what type of issues does it have.

Automated scanning of Log files:

One of the most important and simplest steps to making validation easier is to start with a clean log. This means that your log should not only be free of errors, but it should be free of warnings and “helpful” notes.

Most programmers will check the log interactively whilst building programs, aiming for clean, error free, robust code. However, when data changes or programs are run in a batch environment there can be unexpected problems, therefore it is especially important to check all log files after a production run of data sets and / or tables, listings and graphs. Often there are some warnings and notes that are expected and can be confirmed as okay. Checking many log files can be timely and tedious; using an automated SAS macro tool to produce a quick report is an efficient solution. The advantage to this is that it is a SAS macro, making it user friendly and easy for anyone in the team to amend.

While the warning and error messages are clear problems, the notes listed above are often more subtle indications that the data or the code is not behaving as expected. While SAS can continue processing the data, it may not be doing what you really intended. It is critical to understand what each of these notes means and ensure that the results are what you intended. We Can use the below example macro code after each program to automate the log check process.

```
%macro m_logchk(showout=YES, showlog=YES, chkall=N);
  ods _all_ close;
  ods listing;
  proc printto;
  run;

  %if %upcase(&chkall)=Y %then %do;
    filename lfile "&ex_execall_lfile";
    filename pfile "&ex_execall_pfile";
    proc printto log=lfile print=pfile new;
    run;

    %let linsiz1=%sysfunc(getoption(linesize));
    %if &linsiz1 >= 120 %then %let linsiz2=120;
    %else %let linsiz2=%eval(&linsiz1-18);

    %put Execall: linsiz1=& linsiz2=&linsiz2;

    ods _all_ close;
    ods listing;
    proc report data=fmt.&ex_logchk_ds missing nowd headskip split='|';
      columns (grp ord linenum linenumc logcheck);
      define grp      / order order=internal noprint;
      define ord      / order order=internal noprint;
      define linenum  / order order=internal noprint;
      define linenumc / display ' ' width=9 right;
      define logcheck / display ' ' width=&linsiz2 left flow;
      compute after ord;
        line '';
      endcomp;
```

```
 run;

 %symdel ex_nochk ex_log_counter / nowarn;
 proc datasets lib=fmt;
  delete &ex_logchk_ds;
 quit;
 %goto exit;
%end;

%if %symexist(_SASPROGRAMFILE)=0 %then %goto exit;
%if %symexist(routelog)=0 %then %goto exit;
%if %upcase(&routelog) ne YES %then %goto exit;
%if %symexist(ex_nochk)=1 %then %do;
 %let showout=NO;
 %let showlog=NO;
%end;

%let linsiz1=%sysfunc(getoption(linesize));
%if &linsiz1 >= 120 %then %let linsiz2=120;
%else %let linsiz2=%eval(&linsiz1-12);

data _logchk _showlog;
 length logrow $2000;
 infile lfile;
 input;

 linenum=_n_;
 logrow = strip(_infile_);
 output _showlog;

 if find(_infile_,'ERROR')>0 or find(_infile_,'USER WARNING')>0 then output _logchk;
 else if find(_infile_,'WARNING:')>0 and
       not (index(upcase(_infile_),'WILL BE EXPIRING SOON')>0 or
           index(upcase(_infile_),'THE BASE PRODUCT PRODUCT WITH WHICH')> 0 or
           index(upcase(_infile_),'COMPRESSION WAS DISABLED FOR DATA SET')> 0 or
           index(upcase(_infile_),'SCHEDULED TO EXPIRE')>0) then output _logchk;
 else if find(_infile_,'NOTE:')>0 or find(_infile_,'INFO:')>0 then do;
  if indexw(upcase(_infile_),'WARNING:')>0 or
     index(upcase(_infile_), 'ERROR')>0 or
     indexw(upcase(_infile_),'MERGE')>0 or
     index(upcase(_infile_), 'UNINITIALIZED')>0 or
     index(upcase(_infile_), 'CATALOG IS OPENED FOR READ ONLY')>0 or
     index(upcase(_infile_), 'FORMAT WAS TOO SMALL')>0 or
     index(upcase(_infile_), 'VALUES HAVE BEEN CONVERTED TO')>0 or
     index(upcase(_infile_), 'MISSING VALUES WERE')>0 or
     index(upcase(_infile_), 'IS ALREADY USED OR NOT A VALID SAS NAME')>0 or
     index(upcase(_infile_), 'IS ALREADY ON THE LIBRARY')>0 or
     index(upcase(_infile_), 'DIVISION BY ZERO')>0 or
     index(upcase(_infile_), 'MATHEMATICAL OPERATIONS COULD NOT')>0 or
     indexw(upcase(_infile_),'INVALID')>0 or
     indexw(upcase(_infile_),'OVERWRITTEN')>0 or
     index(upcase(_infile_), 'AT LEAST ONE W.D FORMAT')>0 or
     indexw(upcase(_infile_),'TRUNCATED')>0 or
     index(upcase(_infile_), 'THE QUERY REQUIRES REMERGING SUMMARY STATISTICS BACK')>0 or
     index(upcase(_infile_), 'OUTSIDE THE AXIS')>0 or
     index(upcase(_infile_), 'OBSERVATIONS WERE DELETED')>0 or
     index(upcase(_infile_), 'REPEATS OF BY VALUES')>0 then output _logchk;
 end;
run;

proc sql noprint;
 select count(*) into :noobs
 from _logchk;
quit;

%if &noobs=0 %then %do;
```

```
  data _logchk;
    logrow='Log is clean.';
    linenum=' ';
  run;
%end;

title3 "Con File for %upcase(&TFL_part &pgm_type &pgm_name)";
proc report data=_logchk missing nowd headskip split='';
  columns (linenum logrow);
  define linenum  / order '' width=6 right;
  define logrow   / display '' width=&linsiz2 left flow;
run;
title;

%if %upcase(&showout)=YES %then %do;
  %if %sysfunc(fexist(pfile))=0 %then %do;
    data _noexist;
      NOTE='Output .lst file does not exist.';
    run;
    proc print data=_noexist noobs;
    run;
    %goto nextstep;
  %end;

  data _showout;
    length out $2000;
    infile pfile;
    input;
    out=_infile_;
  run;

  proc sql noprint;
    select count(*) into :noobs
    from _showout;
  quit;

  %if &noobs=0 %then %do;
    data _showout;
      out='         No output for this program.';
    run;
  %end;

  title3 "Output for %upcase(&TFL_part &pgm_type &pgm_name)";
  proc report data=_showout missing nowd headskip split='';
    columns (out);
    define out / display '' width=&linsiz1 left flow;
  run;
  title;

  proc datasets lib=work;
    delete _showout;
  run;
%end;

%nextstep:
%if %upcase(&showlog)=YES %then %do;
  title3 "Log for %upcase(&TFL_part &pgm_type &pgm_name)";
  proc report data=_showlog missing nowd headskip split='';
    columns (linenum logrow);
    define linenum  / order '' width=6 right;
    define logrow   / display '' width=&linsiz2 left flow;
  run;
  title;
%end;

proc datasets lib=work;
```

```
   delete _showlog _logchk;
  run;

  %exit:

  %symdel routelog / nowarn;
  proc printto;
  run;
%mend m_logchk;
```

CHECKING FOR MAJOR CHANGES IN NUMBER OF OBSERVATIONS:

In addition to checking for notes and warnings, it is important to follow the number of observations from one step to another. It is possible for code to execute with no notes or warnings, but due to issues in the data or unexpected problems with the code logic, the final result is unexpected. In many cases this will be evinced in the number of observations being different than expected. Prior to reviewing the final output, it is critical to review the log to make sure the number of observations flowing into and out of each data step or procedure makes logical sense. Chances are that if the number of observations doesn't make sense, the final output won't make sense either.

CONCLUSION

Maintaining quality while doing independent programming on dynamic data with lots of variables is always tricky and it is imperative that we identify and come up with crafty solutions where we are assured about the quality of the work done as a programming team. We are confident that this paper is a step in the right direction and help the programming community to implement them and achieve better quality outputs. Having said that to achieve ideal programming practices we need to constantly evolve and adapt to the new challenges in the industry and come up with effective methods and revisit the current practices and analyze their efficiency and the organizations must make it a priority to train the personnel and make adherence to qualitative programming a top metric.

REFERENCES

[1] https://www.lexjansen.com/nesug/nesug10/ph/ph09.pdf

[2] https://www.lexjansen.com/pharmasug/2018/LD/PharmaSUG-2018-LD14.pdf

[3] https://support.sas.com/resources/papers/proceedings17/0867-2017.pdf

[4] https://www.lexjansen.com/pharmasug/2008/cc/CC02.pdf

[5] https://www.pharmasug.org/proceedings/2015/IB/PharmaSUG-2015-IB01.pdf

CONTACT INFORMATION

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

Aadesh Shah
GlaxoSmithKline
1250 S. Collegeville Road,
Collegeville, Pennsylvania, 19426-0989, United States
Email: aadesh.x.shah@gsk.com

Susruth Ganoothula
GlaxoSmithKline
1250 S. Collegeville Road,
Collegeville, Pennsylvania, 19426-0989, United States
Email: susruth.x.ganoothula@gsk.com