WUSS2015 – Paper 84

Building and Customizing a CDISC Compliance and Data Quality Application

Wayne Zhong, Accretion Softworks, Chester Springs, PA

ABSTRACT

Creating your own SAS® application to perform CDISC and data edit checks might seem to be a daunting challenge. After all, there are hundreds of CDISC rules for SDTM and ADaM alone! This paper aims to reduce the difficulty of such a proposition to an interesting project for 1 to 2 programmers looking for a challenge over three months. It does so by presenting coding techniques and a backbone that significantly reduce the amount of code needed for this data-driven application, and discussing the do's and don'ts of each phase of the development process, namely planning, design, development, testing, and rollout. Beyond the benefit of a private tool allowing unlimited customization to suit the unique needs of a company, programmers working on such a project also get a chance to build in-depth CDISC and SAS software skills.

INTRODUCTION

Data from a clinical trial is organized and presented using data structure standards developed by the Clinical Data Interchange Standards Consortium (CDISC). The Study Data Tabulation Model (SDTM) standard governs raw data, and the Analysis Data Model (ADaM) standard governs analysis data. Raw data includes all data collected during the trial, and analysis data supports safety and efficacy analysis conducted on the trial. Clinical data reviewers and analysts benefit from standards because data is presented the same way across different companies, which is why government agencies overseeing clinical trials recommend or require CDISC standards adoption.

To ensure the quality of CDISC datasets, free and paid tools are available which check data for standards compliance. Beyond compliance, these tools may also offer edit checks to check data for potential issues. The adoption of such tools benefits a company trying to comply with data standards. However, some companies may have experienced that off-the-shelf tools written for the industry are not necessarily a good fit for them. Some of the symptoms include false positives (i.e. reports containing many issues which are not really issues and result in needless investigations) and the continued need for separate quality assurance tools (e.g. company specific edit checks that cannot be incorporated into the off-the-shelf tool).

This paper aims to break down some of the hurdles associated with building a company-specific tool, hopefully making the idea more appealing. First, the volume of compliance checks published by CDISC can be compressed to around 10 programmed checks. Second, incorporating company specific edit checks results in better data issue detection and improved efficiency. Third, this project helps build internal CDISC experts, a must have in today's regulatory environment. The paper is organized as a draft of an actionable tool development plan, with sections on initial planning, design, development, testing, and rollout. A suggested timeline is 3 months for 1 to 2 experienced programmers.

PLANNING

This section discusses scope, resourcing, and timelines. (2 weeks total)

The scope of this project is defined by the sets of checks to incorporate into the tool. Suggested sets include standards i.e. SDTM, ADaM, and SEND (current versions of each should be identified), additional regulatory checks (FDA technical specifications, JANUS requirements), SDTM edit checks, and ADaM edit checks. While it is possible to focus on data edit checks and leave CDISC checks to free tools, it is the opinion of the author that CDISC checks are easier to program and can serve as an excellent segue into the development of data edit checks.

In terms of resourcing, this project is challenging due to the use of SAS logic not normally encountered in the everyday role of clinical programmer role. The ideal candidate would be experienced with Perl expressions, SAS macro language, dynamic code execution via CALL EXECUTE, and PROC SQL match merging. Beyond setting aside time for one to two programmers full time (this is not a 20% project), cross-functional liaisons are needed for the design phase of the project, and testers/trainers are needed for the testing/rollout phases, respectively. In the case of data edit checks, the compiling of possible data edit checks can be delegated to representatives of related functional groups and may begin immediately.

For timeline considerations, it is worth noting this project does not have to be completed in one piece. The suggested sets of checks may be tackled separately, at the cost of increasing the overall timeline. Additional resources will be tapped for the design, testing, and rollout phases of the project. Edit checks require the most time to design, therefore they are usually tackled at the end.

DESIGN

This section discusses tool input, check library, and output. (2 weeks total)

A list of all necessary inputs to the tool, along with the format of each input, should be compiled. The tool requires CDISC datasets as input, which can be in SAS7BDAT or XPT format. Using XPT may require an additional conversion step. Other inputs include controlled terminology (CT) and CDISC metadata. CT can be defined by CDISC, NCI (National Cancer Institute), coding dictionaries such as MedDRA and WHODRUG, and companies may define internal CTs for edit checks. CDISC metadata includes CDISC defined classes, dataset names, variable names, labels, types, and core. These pieces of information can be stored in the format of a dataset or spreadsheet. A dataset is simpler to code for but a spreadsheet will be easier to update and maintain by future users.

The check library for standards compliance are published and easily accessible. The main challenge is with edit checks, where ideas may come from data management, statistical programmers, statisticians, and perhaps other groups. There is no need for all groups to agree on a common list of checks, thus holding meetings with groups separately may simplify discussion. A list of requested edit checks should be prepared in advance, and the purpose of the meetings would be to clarify definitions and logic. It may be necessary to triage checks into practical, difficult, and impossible to implement categories so the first iteration of the tool will not be overly difficult.

The tool's output, a report of all check findings, must also be designed from the very beginning. While this includes the format of the output, such as a PDF, Word, EXCEL, or HTML file, it is critical to plan for the content of failed check messages. The content in the report must allow for unambiguous identification of the failure reason and also the datasets/records that failed without needing to search in manuals or data. It would be an extremely poor design if, for example, the failure report only refers to a check number and gives a generic message, leaving the reviewer to have to search for the failure reason manually. Understanding the content of finding messages allows the necessary information to be retained during the tool coding process.

DEVELOPMENT

This section discusses check condensing, metadata driven checks, and edit checks. (6 weeks total)

The number of checks published for SDTM and ADaM checks at the time of writing total over 600. To organize the code needed to support these checks, similar checks should first be grouped together. For example, consider the following checks:

(1) Description of Arm (ARM) must equal 'Screen Failure', when Arm Code (ARMCD) is 'SCRNFAIL', and vice versa
(2) ASTDTF is populated and ASTDT is not populated
(3) Study Day of Start of Event, Exposure or Observation (--STDY) must be less or equal to Study Day of End of Event, Exposure or Observation (--ENDY)
(4) A variable with a suffix of FL and a variable with the same root and a suffix of FN do not have the paired values (Y,1), (N,0), (NULL,NULL) respectively

These checks are similar in that they describe the relationship between variables in the same record and dataset. By condensing these checks to their common points, the same code may support them, demonstrated below:

```
%macro check1(data=, var1=, var2=, var3=, var4=, condition= );
    data finding;
        set &data(keep=&var1 &var2 &var3 &var4
                  rename=(
                    %do i=1 %to 4;
                        %if %length(&&var&i) %then &&var&i=v&i;
                    %end;
                  ));
        where &condition;
    run;
%mend check1;
```

At its core, macro *%check1* uses a WHERE statement to evaluate a condition. If the condition ever evaluates to be true, there will be records present in the dataset FINDING, indicating the check failed. In writing the condition, all input variables are renamed to V1, V2, etc. The reasoning behind this design is to allow wildcard variables such as --STDY or *FL to be processed with the same condition. Please note an additional step should be added to collect findings for the final report. Using macro %*check1*, the 4 rules above can be evaluated in the following ways:

```
(1) %check1(data=DM, var1=ARM, var2=ARMCD, condition= (v1='Screen Failure' and v2 ne
    'SCRNFAIL') or (v2='SCRNFAIL' and v1 ne 'Screen Failure') )

(2) %check1(data=ADCM, var1=ASTDTF, var2=ASTDT, condition= v1 ne '' and v2=. )

(3) %check1(data=EX, var1=EXSTDY, var2=EXENDY, condition= v1>v2>.z )

(4) %check1(data=ADSL, var1=SAFFL, var2=SAFFN, condition= not((v1='Y' and v2=1) or (v1='N'
    and v2=0) or (v1='' and v2=.)) )
```

Altogether, approximately 130 of the 600 aforementioned SDTM and ADaM CDISC checks can be supported with the logic provided in macro *%check1*, an enormous reduction in code compared with programming each check separately. When each check should be called will depend on the data the tool is operating on, hence the next topic is how to use metadata from CDISC datasets to dynamically choose appropriate checks.

Collecting the metadata of input datasets may be accomplished with PROC CONTENTS or SASHELP.VCOLUMN, and reorganized into a format useful for the tool. Table 1 shows a few records from the reorganized metadata dataset METADATA. There is one record per variable. The variable type has been added to variable names using symbol $ or #, and the variable ALLVARS provides a concatenated list of all variables in each CDISC dataset.

Table 1: METADATA

MEMNAME	NAME	ALLVARS
AE	AESTDTC$	STUDYID$ DOMAIN$ USUBJID$ AESEQ# AEGRPID$ AEREFID$ AETERM$ AEMODIFY$ AEDECOD$ AEBODSYS$ AESER$ AEACN$ AEREL$ AEPATT$ AEOUT$ AESDISAB$ AESDTH$ AESHOSP$ AESLIFE$ AESMIE$ AETOXGR$ AESTDTC$ AEENDTC$ AESTDY# AEENDY#
AE	AEENDTC$	STUDYID$ DOMAIN$ USUBJID$ AESEQ# AEGRPID$ AEREFID$ AETERM$ AEMODIFY$ AEDECOD$ AEBODSYS$ AESER$ AEACN$ AEREL$ AEPATT$ AEOUT$ AESDISAB$ AESDTH$ AESHOSP$ AESLIFE$ AESMIE$ AETOXGR$ AESTDTC$ AEENDTC$ AESTDY# AEENDY#
AE	AESTDY#	STUDYID$ DOMAIN$ USUBJID$ AESEQ# AEGRPID$ AEREFID$ AETERM$ AEMODIFY$ AEDECOD$ AEBODSYS$ AESER$ AEACN$ AEREL$ AEPATT$ AEOUT$ AESDISAB$ AESDTH$ AESHOSP$ AESLIFE$ AESMIE$ AETOXGR$ AESTDTC$ AEENDTC$ AESTDY# AEENDY#
AE	AEENDY#	STUDYID$ DOMAIN$ USUBJID$ AESEQ# AEGRPID$ AEREFID$ AETERM$ AEMODIFY$ AEDECOD$ AEBODSYS$ AESER$ AEACN$ AEREL$ AEPATT$ AEOUT$ AESDISAB$ AESDTH$ AESHOSP$ AESLIFE$ AESMIE$ AETOXGR$ AESTDTC$ AEENDTC$ AESTDY# AEENDY#
CM	STUDYID$	STUDYID$ DOMAIN$ USUBJID$ CMSEQ# CMGRPID$ CMTRT$ CMMODIFY$ CMDECOD$ CMCAT$ CMSCAT$ CMINDC$ CMDOSE# CMDOSTXT$ CMDOSU$ CMDOSFRQ$ CMROUTE$ CMSTDTC$ CMENDTC$ CMSTDY# CMENDY# CMSTRF$ CMENRF$
CM	DOMAIN$	STUDYID$ DOMAIN$ USUBJID$ CMSEQ# CMGRPID$ CMTRT$ CMMODIFY$ CMDECOD$ CMCAT$ CMSCAT$ CMINDC$ CMDOSE# CMDOSTXT$ CMDOSU$ CMDOSFRQ$ CMROUTE$ CMSTDTC$ CMENDTC$ CMSTDY# CMENDY# CMSTRF$ CMENRF$
CM	USUBJID$	STUDYID$ DOMAIN$ USUBJID$ CMSEQ# CMGRPID$ CMTRT$ CMMODIFY$ CMDECOD$ CMCAT$ CMSCAT$ CMINDC$ CMDOSE# CMDOSTXT$ CMDOSU$ CMDOSFRQ$ CMROUTE$ CMSTDTC$ CMENDTC$ CMSTDY# CMENDY# CMSTRF$ CMENRF$
CM	CMSEQ#	STUDYID$ DOMAIN$ USUBJID$ CMSEQ# CMGRPID$ CMTRT$ CMMODIFY$ CMDECOD$ CMCAT$ CMSCAT$ CMINDC$ CMDOSE# CMDOSTXT$ CMDOSU$ CMDOSFRQ$ CMROUTE$ CMSTDTC$ CMENDTC$ CMSTDY# CMENDY# CMSTRF$ CMENRF$
CM	CMGRPID$	STUDYID$ DOMAIN$ USUBJID$ CMSEQ# CMGRPID$ CMTRT$ CMMODIFY$ CMDECOD$ CMCAT$ CMSCAT$ CMINDC$ CMDOSE# CMDOSTXT$ CMDOSU$ CMDOSFRQ$ CMROUTE$ CMSTDTC$ CMENDTC$ CMSTDY# CMENDY# CMSTRF$ CMENRF$
CM	CMTRT$	STUDYID$ DOMAIN$ USUBJID$ CMSEQ# CMGRPID$ CMTRT$ CMMODIFY$ CMDECOD$ CMCAT$ CMSCAT$ CMINDC$ CMDOSE# CMDOSTXT$ CMDOSU$ CMDOSFRQ$ CMROUTE$ CMSTDTC$ CMENDTC$ CMSTDY# CMENDY# CMSTRF$ CMENRF$

This METADATA dataset can be used to drive checks based on variables present in the input data. By carefully selecting IF statements, wildcards can be handled with ease. For example:

```
data _null_;
    set metadata;
    if index(name,'STDY#') and index(allvars,' '||tranwrd(strip(name),'STDY#','ENDY#'))
         then call execute(cats(
            '%check1(data=',memname,
                  ', var1=',compress(name,'#'),
                  ', var2=',tranwrd(name,'STDY#','ENDY'),
                  ', condition= v1>v2>.z )'
                     ));
run;
```

Every occurrence of --STDY and --ENDY variable will invoke *%check1* with the appropriate variables and condition, resulting in a full implementation of check (3). The inclusion of symbols $ and # prevents variables with the wrong data type from executing, which would lead to SAS errors. A check can execute zero or multiple times, since it is solely driven by metadata. Any number of checks may be managed within one DATA _NULL_ step by adding more IF and CALL EXECUTE statements.

In addition to *%check1* which governs 130 WHERE statement checks, 3 condensed checks are suggested below:

1. A One-to-one/many-to-one relationship check, which compare two variables across multiple records within a dataset: Programmed using either PROC SORT NODUPKEY with DATA STEP and FIRSTdot/LASTdot, or PROC SQL with DISTINCT and COUNT

2. A CDISC metadata check, comparing input dataset metadata against CDISC metadata for correct variable names, labels, variable types, presence of variables: Programmed using PROC SQL match merge, a simple example can be found in paper *Automate Validation of CDISC ADaM Variable Label Compliance*, PharmaSUG 2012

3. A Variables governed by CT check, where CT may come from CDISC, NCI, coding dictionaries, or company standards: Easy to program, however upfront organization of all the input CT metadata into an easy to read and easy to update format will take time

Together, more than 550 of the 600 aforementioned CDISC checks can be supported with four macros. The remaining CDISC checks are more specialized, but can be done with 6 condensed checks or less. Overall, a duration of 4 weeks is recommended for the programming of all SDTM and ADaM checks based on the latest versions of each.

The last topic in the development section is edit checks. Even though edit checks will be presented in a separate section in the final report, new macros are not necessarily required. For example:

a) *DTC variables follow the ISO 8601 standard per CDISC, however stricter conditions will detect data issues: condition `%str(length(v1)>=10 and input(substr(v1,1,10),??yymmdd10.)>date())` used with *%check1* will flag dates in the future. Dates too far in the past can be detected the same way.

b) A set of checks using variables LBTESTCD, LBSTRESU, and LBSTRESN with conditions such as `v1 in ('AST' 'ALT' 'ALP') and v2='U/L' and (.z<v3<0 or v3>10000)` with *%check1* forms a filter to catch possible lab data errors.

c) CDISC checks on variables governed by extensible NCI CTs can be improved upon by switching to a non-extensible company specific CT. A process to maintain this internal CT would be needed, however any vagueness associated with extensible CTs would be eliminated.

In the first version of the tool, it is recommended to avoid complex edit checks hence a duration of 2 weeks is suggested, limiting edit checks to those that can be accommodated with existing macros. More specialized edit checks requiring new macros tend to require more testing and tweaking than standard CDISC checks. A few examples of advanced edit checks are:

i) Outlier detection using values from data, through the calculation of means, medians, quartiles, and standard deviations. Unlike b) above, it is not necessary to pre-specify limits.

ii) Site reporting error detection, to identify sites that report values significantly less than or greater than other sites, perhaps due to reporting in incorrect units.

iii) Coding consistency checking, to detect very similar (but not equal) verbatim terms being coded differently.

A discussion of algorithm based edit checks, with full code provided, can be found in paper *Let SAS Improve Your CDISC Data Quality*, PharmaSUG 2014.

TESTING

This section discusses development testing and independent testing. (2 weeks total)

Development testing is performed by the developers during the development process described in the previous section. The developers construct test data that is designed to trigger a PASS, FAIL, or NOT APPLICABLE for each individual check. As the code that governs a particular check is written, the test data is applied to verify that the given check is working properly. The test data may not cover all scenarios within every check, but will result in a better first draft than writing the code in a vacuum.

Independent testing is usually covered by UAT (User Acceptance Testing) requirements at most companies. At a minimum, a variety of study data should be reviewed with the tool by users who were not part of the development process. In addition to reporting bugs, feedback and questions may be incorporated into the user guide and training.

ROLLOUT

Launch documentation typically include a user guide, training (PowerPoint), and version notes. These items can be prepared during UAT testing, so that the tool may be released when UAT is complete.

Integration into the production processes is needed to maximize the benefit of a customized tool. If reports are created while draft SDTM is produced, before QC starts, the number of issues progressing to the QCer can be reduced. Likewise, edit check findings can either flow upstream to be queried and fixed, or downstream to help ADaM programmers plan for data oddities. Reports can be run when draft ADaM is available for the same reasons.

Once the tool is released, far from being the end of the work, comments from live use, updates to standards, and experience with new data issues can be incorporated into future versions of this application. This project will ensure continued improvements in understanding CDISC standards, regulatory requirements, and data quality automation.

ACKNOWLEDGMENTS

I would like to offer my special thanks to Tom Santopoli, Kim Minkalis, Tim Young, and Teressa Chen for their assistance in writing in this paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact Wayne at:

Name: Wayne Zhong
Company: Accretion Softworks
Address: Chester Springs, PA
Cell: (484) 354-2735
Email: wayne@asoftworks.com