Paper SM03

SDTM/ADaM Automation – Challenges and Issues from a Programmer's Perspective While Exploring an Automation Attempt to Address Them

Alistair Dsouza, Takeda Pharmaceuticals, New Jersey, United States

SDTM/ADAM AUTOMATION

This paper will touch on the challenges and issues encountered in implementing automation for SDTMs and ADaM from a programmers' perspective. It would also explore high level schematics of an automation attempt to address some of the issues.

ABSTRACT

Programmers' resistance to automation, lack of scalability, flexibility, and additional overhead are some of the reasons for tools failing or being scaled down after launching. We would investigate some of the challenges and points to remember in tool development from a programmer's perspective. This paper will talk about how such factors affect the overall success, effectiveness, and acceptance of the tool. It would also present high-level schematics of an innovative automation tool to address some of these challenges. With machine readable maps it is scalable and flexible due to a unique bottom-up and unit-based integration approach. A tool with which the programmer feels in control, without feeling stuck or stalled due to variations in the input data or unexpected scenarios, and helping the programmer reduce overall effort and time needed for programming. It also talks about the possible additions and efficiencies that can be built later.

CHALLENGES TO SDTM/ADaM AUTOMATION

Since the introduction of CDISC standards to bring consistency in clinical trial data, there have been multiple automation efforts to develop tools for programming in clinical trials. Most of us have experience of working on some successful tools or at least we would like to believe that they were successful. Some are being used to justify the investment and finally scaled down to eliminate the overhead in terms of maintenance and time, while others have opted for a middle path with some standard macros and tools being used to aid overall programming effort.

One classic example of which I was part of is a study which was being done on an existing system at sponsor side and was outsourced to the CRO as well. Time, quality, and cost were compared.

For every study the SDTMs and ADaMs are one big effort at the beginning and then with some updates for initial reruns as we keep on getting more data. Once the codes stabilize, we are pretty much good for subsequent reruns for different analysis with adding variables or flags here and there as per need.

PROGRAMMERS RESISTANCE

Some of the reasons a programmer has resistance towards using a system are as below:

- If a programmer is early in his career, with automation there is a fear that he will lose touch with SAS programming and hence may lose this skill. Fortunately, and unfortunately, programmers at all levels are tested for this skill when being hired.
- The system having my way or the highway design, where everything must be done as per the needs of the system. This leads to comparison in complexity and time if it was done with usual manual programming.
  Example: Companies standardize CRFs and hence at times develop standard SDTM codes which fail with the slightest of changes in CRF.
- Fear of being stuck and helpless with the automation due to variation which is very common in clinical studies. Less flexibility on the input parameters leading to being stuck until input or standard code is changed.
- Too much automation and flexibility leading to very complex design with hundreds of macro parameters making it confusing and difficult to use and remember all options.
- Dependency on the Automation support team to provide solutions and additional overhead of raising a ticket and similar approaches to address a code failure.

LEARNINGS AND THINGS TO REMEMBER

- We need to deal with programmer's fear of losing programming and instill confidence. An automation which still uses and needs programming ability. Remember the end users are capable SAS programmers and not naïve users.
- Reduce the complexity that programmers get exposed to and when things fail allow a fix that programmer can handle easily.

- Develop processes and SOPs for the automation system that don't become shackles but rather make working easy. Envision from programmers' perspective when developing them.
- The show must go on. Don't allow the system to make programming stop. If things fail, then allow programmers to take control easily and progress.
- When developing an automation, involve at least some experienced clinical SAS programmers with rich programming experience for clinical studies. I have seen highly skilled programmers from Data Science or other automation departments getting involved but missing the experience that gives the perspective and the variations involved in clinical SAS programming.

AUTOMATION EFFORT GROUNDWORK

The idea was to develop automation which will not try to solve the entire problem, but at least it aids the programmer as much as possible without breaking the programmer's spirit. As a parlor game experiences with different automations were discussed and debated. Few of the observations we had and were discussed during brainstorming are as following:

BOTTOM-UP APPROACH

Most tools are trying to solve a bigger problem from a top-down approach. Like entire SAS code to create SDTMs from source or to create ADaMs from SDTMs. You often see individual standard codes for each such SDTM/ADaM domains. Sponsors have tried to standardize CRFs and then written standard STDM codes to be used and adapted based on those CRFs. But if we break down the programming tasks and identify the granular repetitive stuff, we get basic building blocks of codes that we keep on doing repeatedly and can be standardized and grouped together.

This is based on the evolution of computer programming languages where we had assembly language and kept on developing higher level of programming languages catering to specific needs. Can we do something similar using SAS?

Examples: Selection of records, Arithmetic operations, logical comparison, conversions, merging etc

Imagine deriving any duration or change from baseline or AGE, BMI etc.→ Arithmetic operations.

For baseline Flag→ Selection based on criteria (pre first dose), logical comparison(non-missing),

last.recordselected→ merged back on one to many.

Basically, most of the derivations could be broken down into blocks of codes which appear to be repetitive with just the dataset names, variables and subset conditions changing.

REDUCE REPETITIVE INSTRUCTIONS

The underlying thought process behind this is that there are definitions or derivations that we have been writing multiple times for performing analysis. Say for example in SAP section for Adverse events, Statistician writes definition of treatment emergence. The statistician or lead programmer will convert that definition into an algorithm to be included in the ADaM specs. (Sometimes you have an intermediate document between SAP and Specs to aid the programmer which is then additional one more time) Finally, programmers will write the same thing in programming language such as SAS. The same thing was written three times but in different ways. Other examples are baseline flags, criteria flags, populations flags, visit windows, analysis flags etc.

How can we remove these multiple efforts trying to explain the same thing and merge them into one effort.

KEEP THE SHOW RUNNING

The biggest hindrance from programmers' perspective is getting stuck because the standard code fails. The CRF variable changed, a data issue that was not handled in standard programming or some unforeseen situation, or a rather complex derivation for a new variable.

Once the code fails, he must raise a ticket to the standards team or demote the code to study level and then update the standard code which can be a nightmare.

Let's now move towards the way the above three points were addressed.

STRATEGY AND SCHEMATICS

The first decision that was made was that the tool will not create SDTMs or ADaMs, rather it will be an SAS code generator that will generate programs to create STDMs and ADaMs.

It will not be tied down to SDTMs and ADaMs structure, it should be flexible, less complex, and not a showstopper when things change.

A code generator engine was planned whose input was a machine-readable map… aka the specs and efficiencies can be added gradually without disturbing the core.

SDTM or ADaM mapping were divided into the following broad categories.

- Straightforward mapping variables without changes. Directly carried forward from source to SDTM or SDTM to ADaM or variables whose names change but values stay the same.
- Conversions: All kinds of conversions like Numeric to Character/ Character to Numeric/ Date conversions – ISO etc./ formats to change variable values.
- Simple derivations: Duration, additions, subtraction, concatenation, split, logical comparisons for flags creations (Crit flags)

- Complex derivations including multiple merging, selections and combinations of all other: Select group of records based on conditions (within a specific range, non-missing result etc), flag one of the record from the group (max, min, last., first.)

Each of the basic blocks that we saw earlier were converted into independent macro units such as conversion, concatenation, split, direct map, arithmetic macro, logical macro, merge macro, selection macro, flag macro and so on.

Try to imagine the macros as individual nodes following a map to reach the final output which is a variable (Figure 1 and 2). The difference is each completed map outputs a SAS code which when run together is creating a variable from the SDTM/ADaM

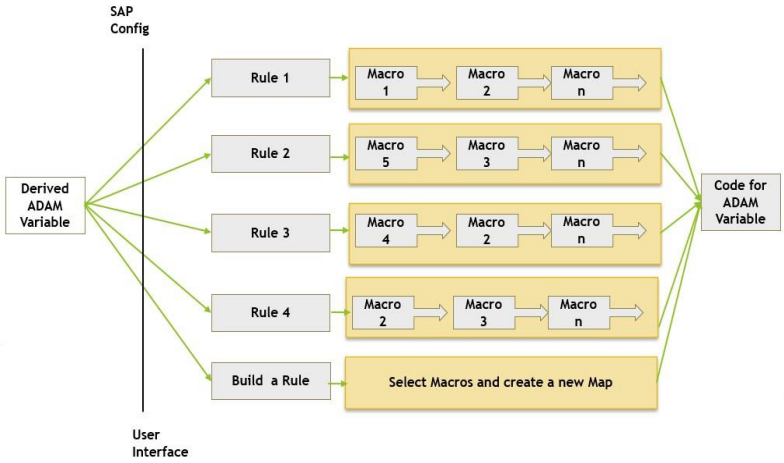


Figure 1

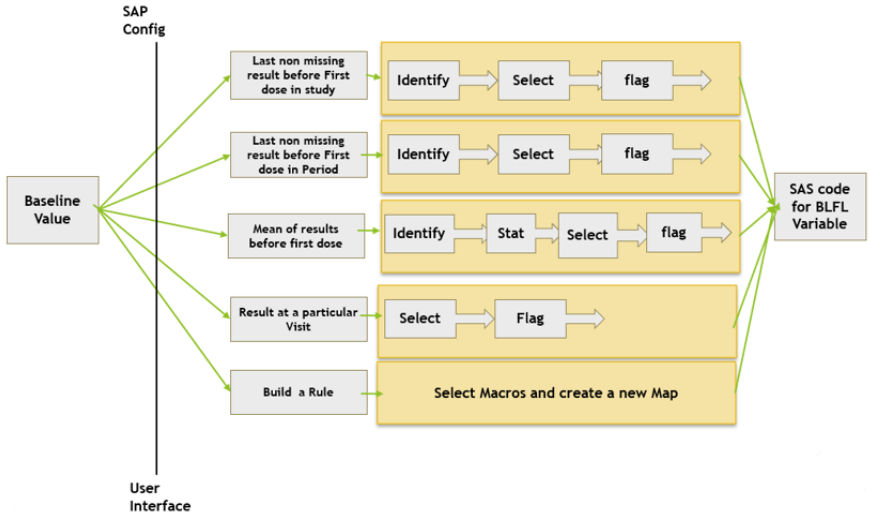


Figure 2

Example Baseline flag: 1) Select macro → Selects records before first dose and non-missing result+ flag macro which flags the last record when sorted by date + Merge one to many to get the base value → Arithmetic macros to get CHG and PCHG if these are part of the ADaM.

The way this was written in the derivation column for ABLFL was as below:
Last.LB.AVAL where LB.LBDTC <= DM.RFSTDTC and LB.AVAL ne ‘’

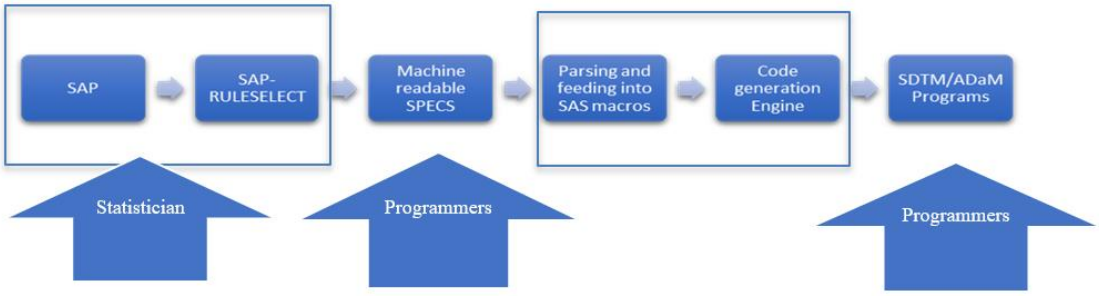
2) Select records within 7 days before first dose+ arithmetic macro to derive average+ flag and merge one to many
AVERAGE of LB.AVAL where 0 < DM.RFSTDTC - LB.LBDTC <=7

The fact that these are derivations of flags also auto-invokes the merge macro at the end which merges these flags

with the original dataset. Such features were already programmed for.

Each new map gets saved in the map library→ example population flags, baseline flags, discontinued flags,
The creation of maps or sequence of macro calls was handled by the parsing engine. The entry into the derivation column was controlled with an English like algorithmic language with its own simple syntax. An edit-controlled entry text box was provided to aid the programmer if he needs to modify anything in the derivation column.

HIGH LEVEL DIAGRAM OFTHE AUTOMATION



COMPONENTS:

- **Rule library**: The concept was to have an exhaustive collection of analysis rules that we normally use for ADaMs. Since most of my teams' experience was working for a CRO, it helped us gather a huge amount of data on different analysis done for the same needs across different therapeutic areas and sponsors.
  Each of these analyses can be considered as a rule. These rules were then converted to maps that were made available for the user to select from.
  Imagine a SAFFL flag definition which says: At least one dose and one safety assessment on treatment.
  While other derivation says at least one dose. Both the rules will be available for user to select from.
  Similarly, an exhaustive list of rules was made available for every ADaM domain. The maps to create the program for these rules were already created and available to select from.

  The plan was that while the SAP was being finalized, the statistician would be asked to complete a form where he would be able to select rules from drop down list.
  For example, baseline definitions, analysis flags derivations, visit windowing details, populations flag etc. The form can be one for entire study or you can add for individual domains like for ADLB or ADEG if they have a different definition for a particular flag. Once this form is prefilled by the statistician it triggers auto entry into the machine-readable specs for the rule selected. Hence the need to enter the same analysis multiple times in SAP, SPECS and SAS programs was eliminated. Any new rule can be added to the rule library as needed.

- **Machine Readable specs:** This is what would provide flexibility. This could be like any other SDTM specs template in excel format with entry controlled in the derivation's column. The programmer can change the entries into the derivation column. Most of the entries will be prefilled. New SDTM/ADaM variables can be added, and derivations entered in the derivation's column.
  The entries needed to be in a pre-specified English like pseudo code. Training was available with sufficient examples of how to enter details for any analysis variable. An edit check was provided to highlight wrong entries.
  The selections of the SAP rules also pre-populate the specs with the correct derivations.
  Common changes like CRF having a different variable name, additional variables collected or a variable need to be converted to numeric before using can easily be changed in the specs and the code generated will change automatically. Also, the specs are portable and can easily be copied over to the new study which shares common analysis needs.

- **PARSER and CODE Generator:** The parser would use language processing concepts to tokenize the input of the derivations column and create macro calls from it which then when executed generate the SAS program.

  Simple example of how the parser works: For derivation of VSDY we write pseudo code as
  sdtm.vs.vsdtc – sdtm.dm.rfstdtc +1.
  The parser would extract the tokens as SDTM.VS.VSDTC, SDTM.DM.RFSTDC and VSDY.

Then it would interpret that there are two different SDTM datasets referenced and would invoke the merge macro followed by the arithmetic macro for duration. Since we are using xxSTDTC variables it will also convert them into numeric.

- **SDTM/ADaM Programs:** The final outputs were simple SAS programs which are either complete or need more programming to be added by the programmer to complete the program. The variables that the Code generator was not able to create SAS code for are clearly indicated and documented for the programmer to step in and complete them. Below is a simple example of the output from the merger and arithmetic macro. The last output dataset of the earlier map is used as one of the inputs in the next map.

```
data v5;
 merge .......;
 .......;
run;

data vs6;
    merge vs5 sdtm.dm;
    by usubjid;
run;
data vs6;
    set vs6;
    ady=input(vsdtc,yymmdd10.)-input(rfstdtc,yymmdd10.);
    if ady>=0 then ady=ady+1;
run;

data v7;
  set v6;
  ......
run;

/*Variables CRIT6FL and CRIT6 not programmed**/
/**Please add code below***/
```

Efficiencies built and can be built.

The Parser did multiple passes through the entire specs and derivations for every variable to build knowledge and use it for structuring the code in efficient way. Some of the efficiencies built or planned were as below:

- The derivations can forward reference a variable that yet has not been derived. → So, a variable may be referenced in another variables derivation before itself being derived. Parser assigned a weightage to each variable to know which needs to be derived first. Some rules were inbuilt, for example flags like baseline flag should be derived after all parameters are derived and hence were assigned weightage accordingly.
- Variables having common derivation requirements were grouped together to avoid multiple data steps. Example: xSTDY and xENDY, BLFL and CHG and PCHG etc. So repeated data steps to merge or read the same datasets were avoided. Parser identified such scenarios and combined them into one data step.
- Any specification that could not be entered due to complexity will appear in the final code with comments clearly stating that the programmer will need to add his code here for that variable's derivation. The comments would be present at the exact location where the programmer needs to add code and it will have a dummy empty variable created.
- Any entry in specs that failed to generate a code due to some macro call failing will follow a similar approach to create an empty structure for that variable in the final SAS code and programmer can then add his own code. Any kind of error stopped the current variable map execution and went to the next map instead of stopping the process.
- A VLM parser was planned to be built in to read and code the details example for each PARAMCDs that have their own different derivations.
- After every rerun a log file will be created to document the macro failures to create SAS code and sent to the maintenance team. Also, the programmer can raise a ticket to notify that they were not able to enter a particular derivation using the allowed syntax.
  But note that none of these failures will stop the process of programming. The programmer will always get a SAS code with few variables not programmed which he can always add.
- The library of rules for SAP Config and previous entry in the specs were maintained and available to select from to the programmer. The rules were simple English language rules like we find in SAP and were linked to maps entered the Machine-readable specs.
- The worst scenario would be a complex ADaM dataset like ADEFF with each PARAMETER having textual description of complex derivations unable for us to enter it as per available syntax of derivations column. But the system would still give you a program that would create a SAS code with header, merge with ADSL, keep the required variables from ADSL, include the direct mapped variables and simple conversions and variables expected as per BDS or Occurrence datasets.

- The best scenario would create a complete SAS code that will generate a required dataset as per the specs. As the system is used more and more it will move closer to more than 90% code availability.
- The Machine-readable specs were easily portable from one study to other. A separate edit check code can be built in the parser to help with the portability. For example, to point out variables referenced in specs that are not part of the source dataset.

Limitations:

- One of the difficult tasks would be developing the parser and using language processing concepts. We face some difficulties when we come to very complex derivations, for example the kind of we see in ADEFF.
- Use of NLP was limited as the style of writing derivations and entries into the SAP and specs varied widely and was subjective. Hence the derivations had to be written in specific format in the specs documents which had a learning curve for the programmers.
- The output was a SAS code and not a dataset and hence even if the code generator was validated it didn't help in SDTM or ADaM validation. Validation was still parallel programming.

CONCLUSION

We have been doing things the traditional way for a long time now as far as programming is concerned.
Of course, we cannot automate the programming for SDTMs and ADaMs at 100%, but we need to think radically different and evolve to address the automation needed in clinical SAS programming. A thought-provoking question we had while thinking about the above-mentioned automation was how we can include AI concepts in selecting or creating/suggesting the maps needed for each variable.

The above method was simple yet different than usual automation efforts we have seen or encountered. This was like an integration method to derive an area of an uneven surface. We divide it into smaller identifiable parts and then add up the area. Similarly, we divide the bigger problem into identifiable smaller constructs of code and use them together to solve the problem. We based it on the concept of developing a higher abstract programming language more catered to the SDTM/ADaM requirements.

The above approach created SAS codes as complete as possible and provided opportunity to the end user to play around the SAS code. Also adding more efficiency to the automation could be done without stopping the programming and without adding burden to the programmer to remember more options and utilities.
In the end, I hope that this paper provides ideas and perspectives through our SDTM/ADaM automation story.

ACKNOWLEDGMENTS

Manali Pendse

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:
Author Name: Alistair Dsouza
Company: Takeda Pharmaceuticals
Email: alistair.dsouza82@gmail.com; alistair.dsouza@takeda.com