

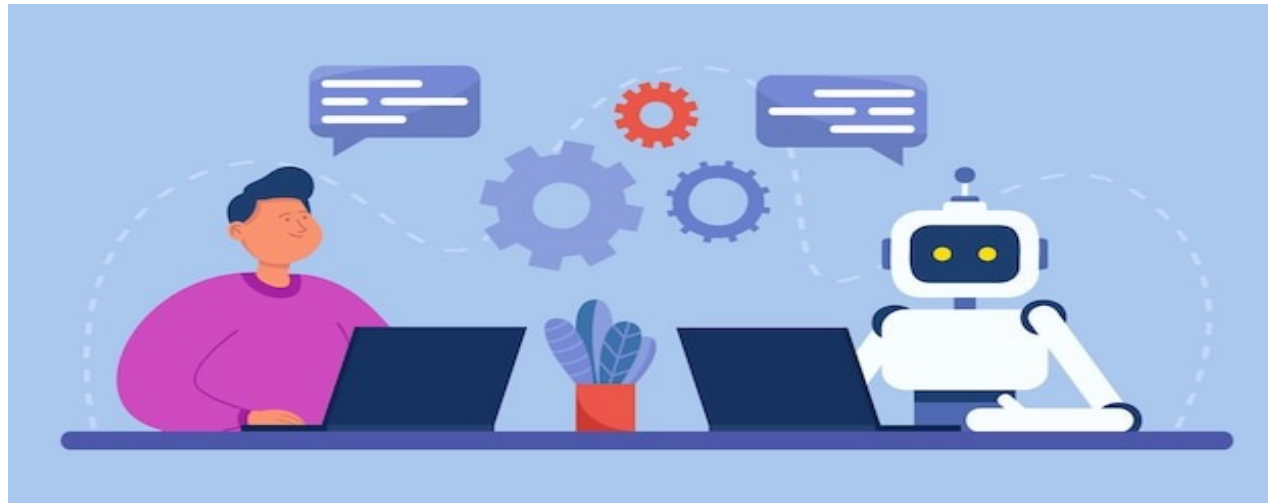
SM03: SDTM/ADaM Automation – Challenges and Issues from a Programmer's Perspective While Exploring an Automation Attempt to Address Them.

Name: Alistair Dsouza

Title: Associate Director

Company: Takeda Pharmaceutical

Alistair has 18 years of experience in clinical SAS programming. He has been leading studies, compounds, automation efforts and of course still hands on programming.

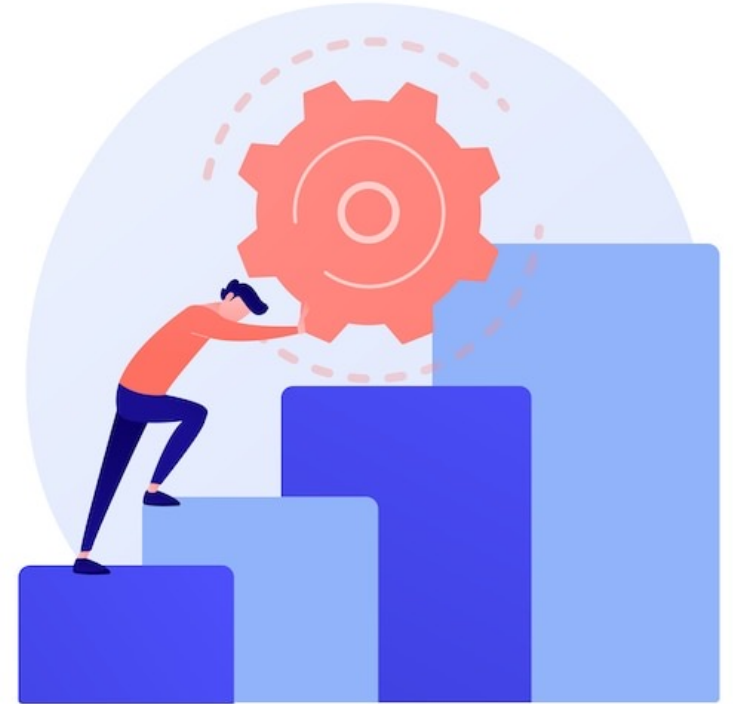


- In this presentation I would touch on some of the challenges and points to consider in SDTM/ADaM automation development from a programmer's perspective.
- I would also present high-level schematics of an innovative automation tool to address some of these challenges.
- We would look at how it can handle unexpected scenarios, help the programmer reduce overall effort and time needed for programming.
- Possible additions and efficiencies that were built or can be we added later..

CHALLENGES TO SDTM/ADaM AUTOMATION



- Fear of losing programming skill and trying to stay relevant in the industry.
- My way or the highway design. Processes becoming shackles.
- Fear of being stuck and helpless with the automation, less flexibility.
- Complex design with hundreds of parameters making it confusing and difficult to use.
- Dependency on the automation support team to provide solutions



LEARNINGS AND THINGS TO REMEMBER

- Deal with programmer's fear instill confidence that automation still needs programming.
- Remember end users are capable SAS programmers and not naïve users.
- Reduce the complexity that programmers get exposed, when things fail allow a fix that programmer can handle easily.
- When developing processes and SOPs for the automation system envision from programmers' perspective.
- Remember the show must go on.
- Involve some experienced clinical SAS programmers with rich programming experience



AUTOMATION EFFORT GROUNDWORK



BOTTOM-UP APPROACH

- Tools are trying to solve a bigger problem from a top-down approach.
- Instead break down the programming tasks and identify the repetitive basic building blocks of codes.

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

Deriving duration or change from baseline or AGE, BMI etc. → Arithmetic operations.

Baseline Flag → Selection based on criteria (pre first dose), logical comparison(non-missing), last record selected



AUTOMATION EFFORT GROUNDWORK



REDUCE REPETITIVE INSTRUCTIONS

There are definitions or derivations that we have been writing multiple times.

TEAE in Adverse events,

- Statistician defines treatment emergence in SAP
- Lead programmer will convert that definition into an algorithm for ADaM specs.
- Programmers will write the same thing in programming language such as SAS.

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?

AUTOMATION EFFORT GROUNDWORK



KEEP THE SHOW RUNNING

The biggest hindrance from programmers' perspective is getting stuck because the standard code fails for various reasons.

- The CRF variable changed,
- A data issue that was not handled in standard programming or some unforeseen situation,
- Rather complex derivation for a new variable.

Raise a ticket or demote the code to study level.



STRATEGY



SDTM or ADaM mapping categories.

- **Straightforward mapping:** Directly carried forward.
- **Conversions:** Conversions like Numeric to Character/ Character to Numeric/ Date conversions – ISO etc./ formats.
- **Simple derivations:** Duration, additions, subtraction, concatenation, split, logical comparisons for flags creations (Crit flags) ETC
- **Complex derivations:** Including multiple merging, selections and combinations of all other.



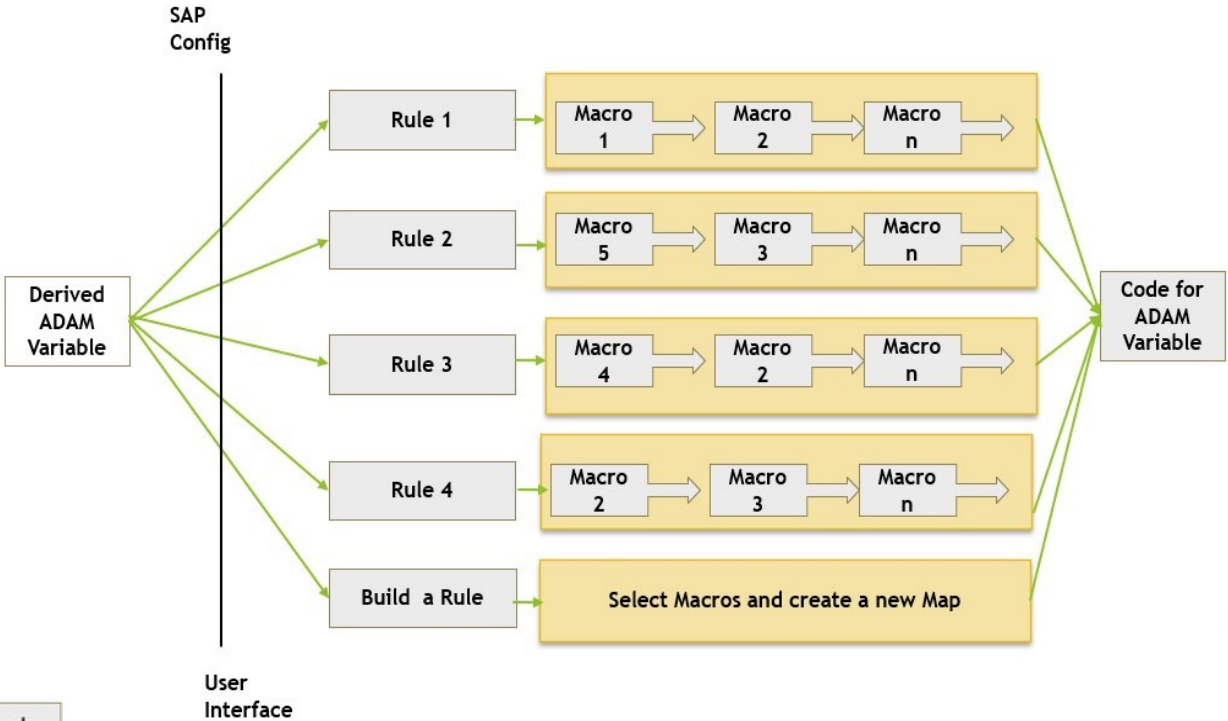
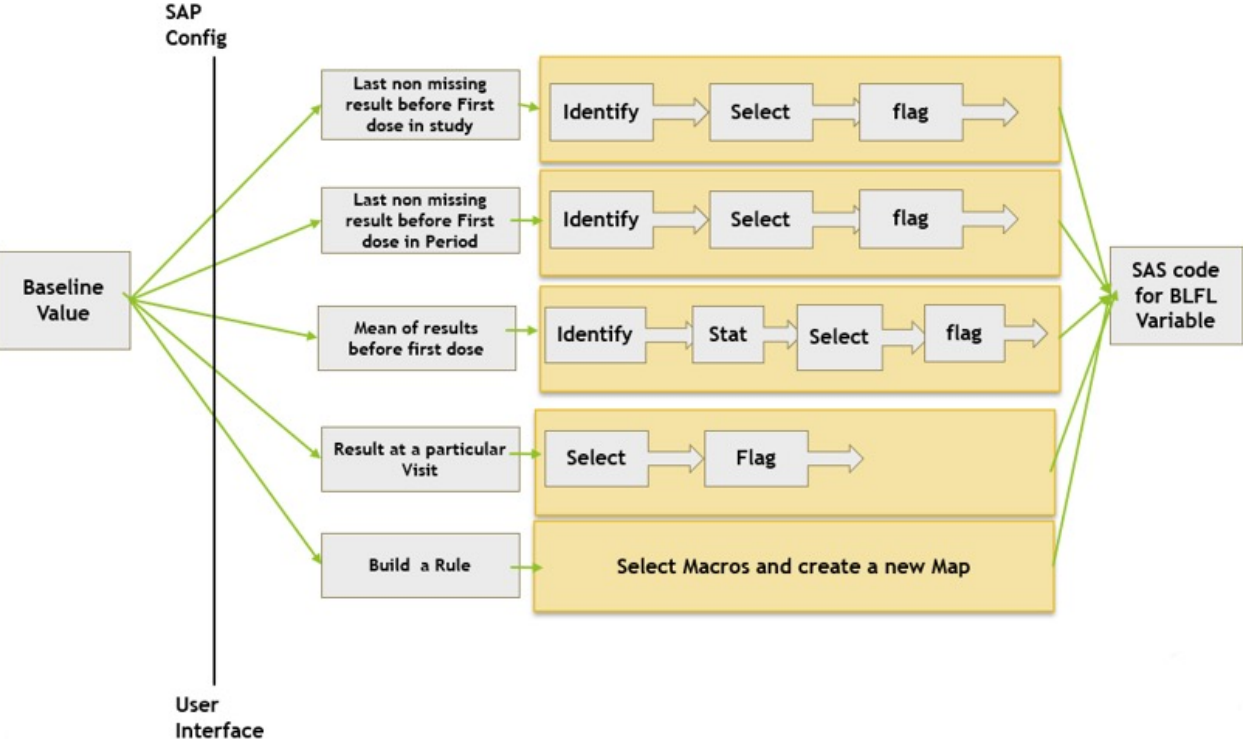
Few points we agreed on :

- The tool will not create SDTMs or ADaMs, rather it will be an SAS code.
- It will not be tied down to SDTMs and ADaMs structure, it should be flexible, less complex, and not a showstopper.
- The code generator engines input would be a machine-readable map... aka the specs.

SCHEMATICS

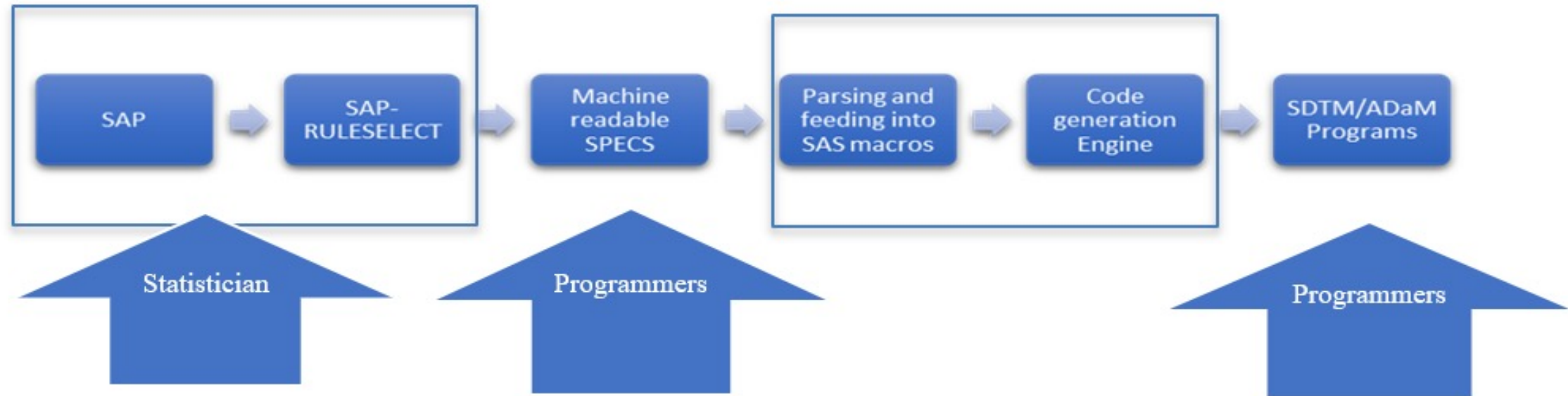


The basic blocks were converted into independent macro units such as conversion, concatenation, split, direct map, arithmetic macro, logical macro, merge macro, selection macro, flag macro and so on.



Try to imagine the macros as individual nodes following a map to reach the final output which is a variable. Each completed map outputs a SAS code which when run together is creating a variable from the SDTM/ADaM

HIGH LEVEL DIAGRAM OF THE AUTOMATION

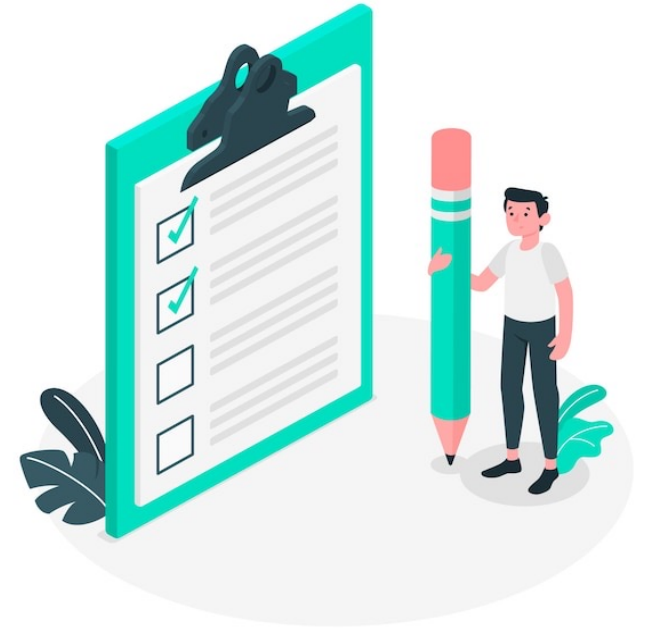


COMPONENTS



RULE LIBRARY

- An exhaustive collection of analysis that we normally use for ADaMs.
- These rules were then converted to maps for the user to select from.
- Maps get converted to machine readable specs.
- The statistician/programmer can select rules from drop down list that would trigger auto entry into the machine-readable specs.



MACHINE READABLE SPECS:

- This was like any other SDTM specs in excel format. The entries in derivation column was in pre-specified pseudo code.
- Most of the entries will be prefilled. SDTM/ADaM variables can be added, and derivations entered/modified in the derivation's column.
- The selections of the SAP rules also pre-populate the specs.

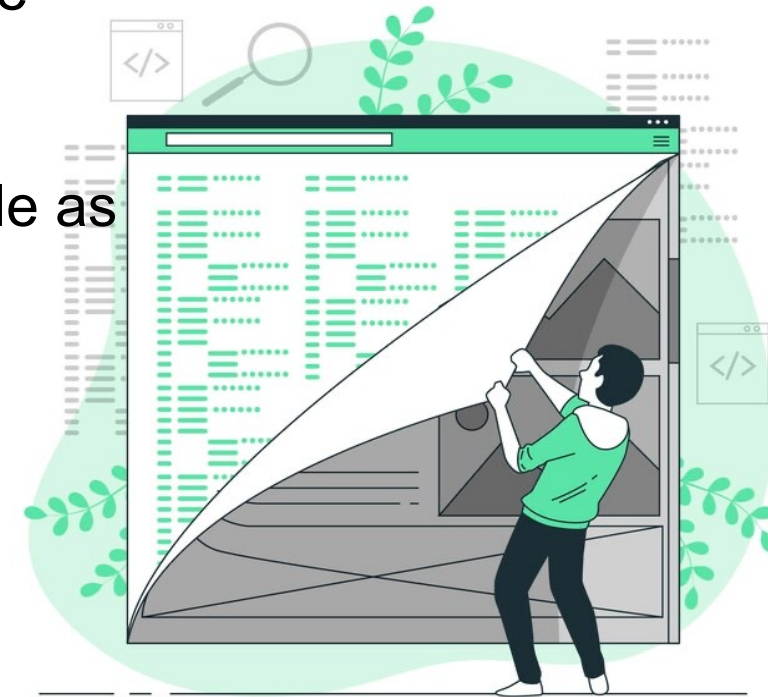


PARSER and CODE GENERATOR:

- The parser uses language processing concepts to tokenize the derivations column and create macro calls which generate the SAS program.

Simple example : Derivation of VSDY we write pseudo code as
`sdm.vs.vsdtc – sdm.dm.rfstdtc +1.`

Tokens → SDTM.VS.VSDTC, SDTM.DM.RFSTDC and VSDY.
→merge macro if DM not already merged
→arithmetic macro for duration.



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.
- It was a marriage between automation and manual programming.
- The sas-codes were deliberately kept simple with no complicated macros called.
- Initial program creation was one time effort per study.

EFFICIENCIES BUILT and PLANNED



Parser did multiple passes through the specs to build knowledge and use it to build efficiency

- The derivations can forward reference a variable that yet has not been derived.
- Variables with common derivation requirements were grouped together to avoid multiple data steps.
- Derivations that could not be entered or failed due to complexity appeared in the final code with comments at the exact location where the programmer needs to add code.
- Failure stopped the current map execution and went to the next map instead of stopping the process.
- A VLM parser could be built to handle PARAMCDs that have their own different derivations.

EFFICIENCIES BUILT and PLANNED



- Map or Macro failures to create SAS code were documented and handled later.
- The programmer always got a SAS code with few variables not programmed.
- The library of rules for SAP and previous entry in the specs can be maintained and made more efficient over time.
- In the worst scenario we would still get a skeleton SAS code.
- 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.

LIMITATIONS

- Developing the parser and using language processing concepts for complex derivations.
- Use of NLP was limited as the style of writing into the SAP/specs varied widely and was subjective.
- The derivations had to be written in pseudo code which had a learning curve for the programmers.
- Input data was needed structurally SDTM like i.e. vertical than horizontal.
- The output will be 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.



CLOSING COMMENTS



Its time to think radically different and evolve to address the automation needs in clinical SAS programming.

- Imagine breaking down of programming → Integration approach to derive an area of an uneven surface.
- The Pseudo-code and parser → concept of abstract programming language catered to the SDTM/ADaM needs.
- The SAP-Rules library → Eliminate repetitive efforts.
- The simplicity of end SAS programs was to reduce overhead for the SAS programmer.

NLP in SAP development
linked to the specs and
programs creation?

Food for thought



AI in selecting maps needed
for each variable. Each
macro is node storing & processing
information and passing it along?

Questions?



Thank you