Paper TT12

Why are old wines sometimes tastier than new ones?

Henning Pontoppidan Föh, Novo Nordisk A/S, Copenhagen, Denmark
Lars Lehmann Andersson, SAS Institute, Copenhagen, Denmark

ABSTRACT

For decades, SAS software has been a substantial part of the toolbox for professionals in life sciences involved in the data handling, analysis and results reporting of clinical trials. Today, at PHUSE conferences and elsewhere, we observe a trend towards transitioning to other technologies, often from the open source realm. As an example of this, Novo Nordisk has a programming language agnostic strategy and a Statistical Computing Environment (SCE) that currently makes R and Python available to its users alongside SAS.

Like most programmers, the authors of this paper tend to favour the programming language that we already know, in our case SAS. But we realize that some of the successes to which we contributed were not only due to our language of choice, but also to the structured approach to its use in the context of the clinical trials to which we were assigned. As such, we will in this paper present aspects of an initially small SAS setup for ADaM generation in a phase 2 trial that evolved into an efficient and adaptable framework supporting large phase 3 trials across several clinical development projects. We believe that many of the approaches can and to some extent should also be followed if using another tool than SAS.

With our walkthrough of programming principles which helped us meet quality and timeline requirements, and which we implemented using SAS, we hope to demonstrate that the best processes can be achieved in a set-up where SAS is allowed to co-exist with, rather than be replaced by, newer tools. Like the old wine stored at the rear of the wine cellar, waiting for a special occasion to be opened…

INTRODUCTION

We, the authors of this paper, collaborated some time back as statistical programmers in the same team at Novo Nordisk, focusing on ADaM generation (note, that the role-name of statistical programmer at Novo Nordisk recently has changed to Clinical Data Scientist; however, this paper refers to the previous role as statistical programmer that to a large degree is identical to the new Clinical Data Scientist role). In retrospect, we can see that the work done by the team had desirable results, leading us to wonder how we might achieve something similar another time. Our paper is based on experiences made back then as well as later, as some of the learnings have been propagated and adopted throughout the statistical programming function at Novo Nordisk. The aim is to share what we believe to be good practices, hopefully inspiring adoption elsewhere. The views presented are those of the authors, not necessarily those of their respective employers.

The paper has three descriptive sections followed by a discussion. In the first of these sections, we outline the background givens that we had no or limited ability to influence, including the constraints under which we had to operate. In the next section, we go through some of the decisions made or influenced by us, which we believe were contributing factors to the team repeatedly being able to deliver high quality within expected timelines. The focus of the last section is on some of the programming principles that we adopted, with emphasis on the rationales and the possible universality of those principles.

In the subsequent discussion, we summarize and elaborate on learnings from what was done.

THE GIVENS: TASK, EXPECTATIONS, AVAILABLE TOOLS AND TEAM

At the time of our original collaboration, Novo Nordisk's statistical programmers working on clinical trials were organized in project teams focused on each of their set of products. These teams also included statisticians. The programmers were to handle the programming tasks for all related trials in close cooperation with the statisticians. SDTM had been established outside the team and would act as the main data source, with the programmer tasks being mainly to build and document programs for creating ADaM datasets and the tables, figures and listings needed for the CTRs. Other tasks included programming of the integrated summaries of efficacy and safety. Part of the expectations for documentation was that ADaM programs should be "submission ready", i.e. sufficiently well-structured and legible to be shared with and potentially even re-run by regulatory authorities.

The work ahead would last for years, with planned phase 3 studies for an investigational medicinal product. However, our first joint experiences with programming ADaM were done on a smaller phase 2 trial on a concurrent project. After the ADaM side of things for this phase 2 trial was done, learnings had been made and the team was in a good position to make the right choices for the phase 3 ADaMs.

The core of the phase 3 study plan included five efficacy trials that were similar in terms of approximate duration, visit structure, assessments made, type of primary endpoint etc., with the main difference being the comparator drug. The first of these had FPFV before the four others, with the others starting concurrently before the first study was concluded. In addition to these five trials, the study plan also included a long-term safety trial that obviously differed from the others on several counts, and some country specific phase 3 trials as well.

The team of which we were part of consisted of in-house statisticians in the company HQ, as well as in-house and external programmers from both HQ and other parts of the world. Whereas the team did not remain the same for the entirety of the project, several key people on both the statistician and the programmer sides took part for years.

Back then, available ADaM standards were less mature than today, and more decisions were left to the sponsor. To prevent different project teams from making contradictory decisions about how to implement the various “free choice” aspects of ADaM, Novo Nordisk established a cross-team standards governance board which would regularly make decisions that would be “in-house additional standards” for all teams to adhere to.

Technology-wise, the organization relied mostly on a SAS 9.4 installation in a managed Unix server environment, with R also being available for exploratory tasks.

DECISIONS AND CONSIDERATIONS MADE

The goal was to be able to submit data, programs and documentation to authorities, and we had to decide on a programming strategy that would allow us to accomplish this. Everything we made should be transparent and enabling us to share both within and outside Biostatistics. We also knew that we had to be iterative in our development of the data processing, enabling scalability and adaptability of business rules as blinded clinical data arrived gradually until DBL.

Facing the phase 3 study plan, decisions had to be made about how to best organize the work around building ADaMs for the constituent trials, based on the information we knew in advance: tool (SAS), approximate timelines and available programmers. In the following sections, we will go through some of the choices made and the underlying rationale. Some of our decisions were the result of initial planning and discussion, others were made while development work was ongoing. We believe that we made some sensible decisions, many of which will also have relevance if the tool used had been something other than SAS.

We aimed for easily readable programs (submission friendly), adaptable and easy to execute. Knowing that most statistical programmers within our company and at the FDA were de facto mainly skilled in SAS (including the two authors of this paper), we decided to use SAS as the programming language.

PROGRAMMERS AND THEIR ASSIGNMENTS

One of the first things considered had to do with how to make best use of the available programmers. An initial decision that remained in effect for the entire project was that some programmers would focus on programming ADaM datasets, while others would focus on using the ADaM datasets to produce tables, figures and listings. This decision probably helped ensure that the quality of ADaMs remained "top of mind" for the responsible programmers, but it also introduced some constraints in terms of TFL programmers having to await the relevant ADaM datasets being ready, and ADaM programmers being expected to prioritize the datasets needed for high-priority outputs without necessarily having the most up-to-date needs as expressed from e.g. statisticians to TFL programmers. Such challenges were mostly overcome by communicating.

Initially, each trial had its own team of associated ADaM programmers, and the idea was that programmers from trials 2 and onwards would "copy/paste and adapt" from the first trial, consulting the original programmer in case of questions or doubts. Due to organizational changes, this concept did not hold for the entire duration of the project, and the original ADaM programmers from the first trial ended up taking part in adapting programs for the subsequent trials. This proved effective in terms of ensuring that any given ADaM dataset was made as consistently as possible across trials, with only trial-specific needs driving introduction of changes. It was found that a programmer who had already done a dataset X for trials 1, 2 and 3 was in a better position to ask relevant questions to a stakeholder suddenly requesting something different for the same dataset in trial 4, sometimes causing reversal of a request, thus contributing to the overall quality.

As such, a recommendation for programmer assignment is to largely let one programmer be responsible for the same dataset across multiple trials within the same program. Figure 1 contrasts this with a more "random assignment" model.

	ADaM dataset		
	ADSL	ADLB	ADVS
Trial 1	P1*	P2*	P3*
Trial 2	P3*	P3*	P1*
Trial 3	P2*	P1*	P3
Trial 4	P2	P1	P2*

Pros: Very agile – first available resource grabs and executes on task with highest priority.
Cons: Disregards that most ADaM datasets are complex matters where initial "get to know it" is time consuming. Also, non-programmer stakeholders may expect or require consistency across trials, which is difficult to ensure when it is not the same programmer responsible.

	ADaM dataset		
	ADSL	ADLB	ADVS
Trial 1	P1*	P2*	P3*
Trial 2	P1	P2	P3
Trial 3	P1	P2	P3
Trial 4	P1	P2	P3

Pros: Learnings from first time are carried forth. "Single point of contact" programmer for other stakeholders in the trials.
Cons: Programmers must be able to prioritize tasks to avoid waiting time, typically meaning a near full time allocation to ADaM programming for the trials in question.

Figure 1: Three programmers P1-P3 assigned to make three different ADaM datasets in each of four related and similar trials. The asterisks indicate each programmer's "first experience" with the dataset in question for these trials, assuming they are conducted sequentially. The random assignment model on the left has 9 first encounters, the other one has only 3.

COLLABORATIVE DEVELOPMENT

Some ADaM datasets are more complex to produce than others. Some require iterations between programmer, statistician and other stakeholders to build a dataset that is fit for purpose. Consequently, many detailed specifications were created iteratively and typically in dialogue between the statistician and the programmer. During this process, program logic was often looked upon as part of the dialogue, and it would gradually be refined as understandings were reached. It was typically also applied on available blinded data to exemplify the impact and potential consequences. Here a readable programming language understood by the involved persons – in our case SAS - proved its capability. The consistent availability in SAS of intermediate results in a navigable, rectangular format (work datasets) further facilitated the dialogue.

An example of such a complex dataset was the Time-to-Event dataset (ADTTE). It also became a good example of how various stakeholders used their individual skills to add value to the iterative process. First, the statistician specified how he wanted first draft of ADTTE to be (of course with special emphasis on the rules for determining what constituted an event of interest and when it took place), and the programmer made an early version of the dataset in accordance with these specifications. This was done quickly, as the programmer had plenty of experience in the programming tools used. The statistician then initially worked with the resulting data to familiarize himself with them ensuring that they met expectations.

He then proceeded to build an R-Shiny application that used interactive visualization of the dataset and used it to facilitate further dialogue with other stakeholders in the study group. With his own observations and further input from the study group, he was able to come back to the programmer with updated specifications, allowing an iterative but not overly lengthy way of developing an ADTTE that was "fit for purpose".

Such a visualization could have been made by any kind of available tool or application characterized by having an output-side which both technical and non-technical stakeholders would understand.

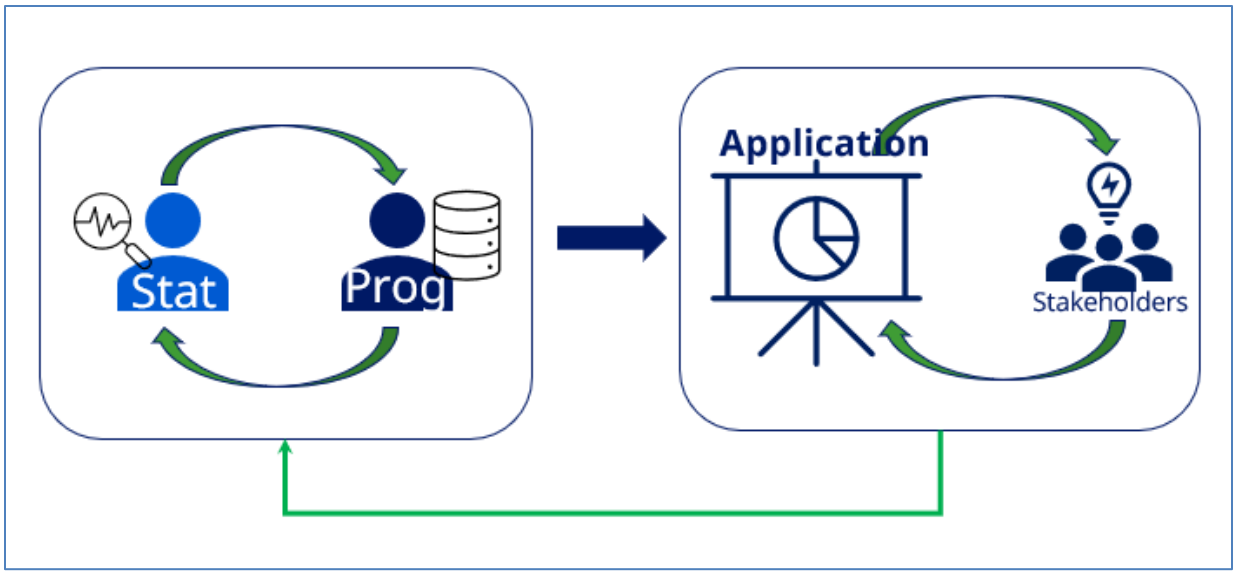

Figure 2: Collaborative development of more complex ADaM datasets. First step is an iterative specification and development of prototypes of the datasets, feeding it into a dynamic visualization application of the dataset for stakeholders to navigate. Insights and new understandings could then lead back to the specification and development step.

ADOPTING COMPANY STANDARDS AS THEY ARE MADE

As mentioned earlier, Novo Nordisk created a cross-team standards governance board to ensure that project teams across the organization would interpret ADaM standards in the same manner, whenever there was something open to interpretation. In the daily work with building ADaMs, questions would often arise that in principle would require a recommendation from this board. The team thus had to choose between awaiting a decision from the board (which met at most on a biweekly basis), or to assume a decision and move ahead. To preserve momentum, it was often decided to do the latter, flagging to the involved programmers that there was a risk that the board would overturn the immediate decision made in the team.

Quite often, programmers would be able to keep the programs sufficiently flexible to be able to quickly change them later to a different interpretation of the rules, if the decision of the standards governance board was “against us”. But by frequently being the first movers and having a representative on the standards governance board, the work done in the team would often influence and inspire the decision of the board.

On rare occasions, it did happen that the board would decide on an interpretation that was contrary to something deep-rooted in work already done within the team. But it was allowed for on-going trials to stray from the in-house standards if timelines did not permit otherwise.

PROGRAMMING PRINCIPLES THAT PROVED USEFUL

In the following sub-sections, we describe some of the general programming principles that we found to be useful in helping us ensure a timely high-quality delivery, and which we also believe are not necessarily “SAS only”. I.e., we focus on principles that we believe possess a certain universality, so that it would also be beneficial to adhere to them even if another language than SAS was used for building ADaM datasets.

INDIVIDUAL PROGRAM STRUCTURE

Despite being spread across several datasets, a collection of ADaM datasets for a given trial must constitute a coherent whole. One approach to achieving this could be to let as few programs as possible, ideally just one, create all the ADaM datasets. We opted for the opposite, going with a "one program creates one ADaM dataset only" principle. There were several reasons for this:

- Easier and faster QC
- Easier re-run of relevant ADaM programs only
- Easier to involve many programmers while preserving “sense of ownership”

To keep the ADaM programs as similar as possible to facilitate QC and handover, they were all written with a degree of adherence to the ETL (“Extract, Transform, Load”) principles [1] known from data warehousing: Initially, relevant data are extracted (in our case read from source SDTM and/or other ADaM datasets) and written to temporary WORK datasets. Then the transformation takes place (often the bulk of the program) during which the extracted data are combined and augmented with derived variables etc. until we have a WORK dataset identical to the desired

ADaM dataset as defined as target. Finally, the data are loaded, i.e. written to the permanent output location for ADaM datasets.

Writing programs in this manner helped everybody understand what was made and what was missing, and QC via code review and/or parallel programming was easier than if every new program had been made with entirely its own structure. The Base SAS programming language supported the effort by having readily available features for all parts of the ETL process.

CREATING PROGRAM LOGIC SHARED ACROSS PROGRAMS

When constructing programs intended to create ADaM datasets, it quickly becomes apparent that much of the logic needs repeating across multiple programs. Thus, it is relevant to consider whether code should be centralized in callable modules, and if so, how. The team adopted a set of principles for this which helped ensure an appropriate degree of modularization that was not overly cumbersome to work with:

- If operations A and B often or always occur one after the other in ADaM programs, but they are not part of the same logical thing (maybe A is visit reallocation and B is imputation of missing values), then they should not be part of the same centralized module. It is preferable to have more modules that are less complex over having fewer modules that are more complex.
- If an operation is so trivial that it can be robustly implemented via simple code (in SAS, a simple procedure call or data step), there is no reason to embed that simple code in a central module and replace the simple code with a custom module call. A central module should be a reasonable trade-off between program legibility and the time involved in developing and maintaining the central module.

The strategy of centralizing shared program logic into callable modules while avoiding overcomplication proved effective. It ensured that common operations were handled efficiently without burdening the programs with unnecessary complexity. This modular approach balanced program readability with development and maintenance efforts, ensuring a streamlined and manageable programming environment.

HANDLING SIMILAR DATASETS

Many of the ADaM datasets in a typical set of trial data belong to one of the two classes BDS or occurrence data. We found that typically, the similarities between ADaM datasets of the same class did not end with the structure. Rather, statisticians would often request the same kind of methods applied across datasets for e.g. visit reallocation, imputation etc. Consequently, we tried to keep the different programs within each of the two classes similar in terms of order of operations etc., so that central program logic modules for doing e.g. visit reallocation or imputation could be reliably called from each program. Keeping much of the complex logic central, contributed to a higher quality via the same logic being subjected to a larger and more varied set of input data, and faster resolution time by having only one place in which to fix a problem.

It was not a goal to have e.g. all BDS programs look "as alike as possible", but rather an approach to ensuring relatively high quality relatively rapidly. Conceptually only, there was a "master BDS" program containing every possible operation relevant to (any) BDS in the trial, and each actual BDS program in the trial would be an adapted copy of this master program with all the unnecessary steps removed.

About half of a typical BDS program would be "free code" (much adapted from the "master program", i.e. more or less identical across datasets), and the remaining would be calls to central modules.

By maintaining consistency in the order of operations and central logic across similar ADaM datasets, the team achieved higher quality and faster resolution times – especially in the iterative specification and development process. This approach allowed for complex logic to be centralized, ensuring robust and reliable methods across datasets.

REBUILDING PROGRAM FOR AN ADAM DATASET FOR THE NTH TRIAL

It takes time to write an ADaM program. It is tempting to try reducing the time needed by only making the program for a given ADaM dataset once, and then using the same program on multiple trials. A disadvantage to this approach is that detailed needs of the later trials must be known and analysed already while working on the program for the first trial.

Our approach to this dilemma was not to aim for a "one size fits all" program, but rather to aim for a sensible data processing structure that could easily be copy/pasted and subsequently adapted to new trials. A reasonable compromise between the two extremes on the scale of program generalness can be seen in Figure 3.

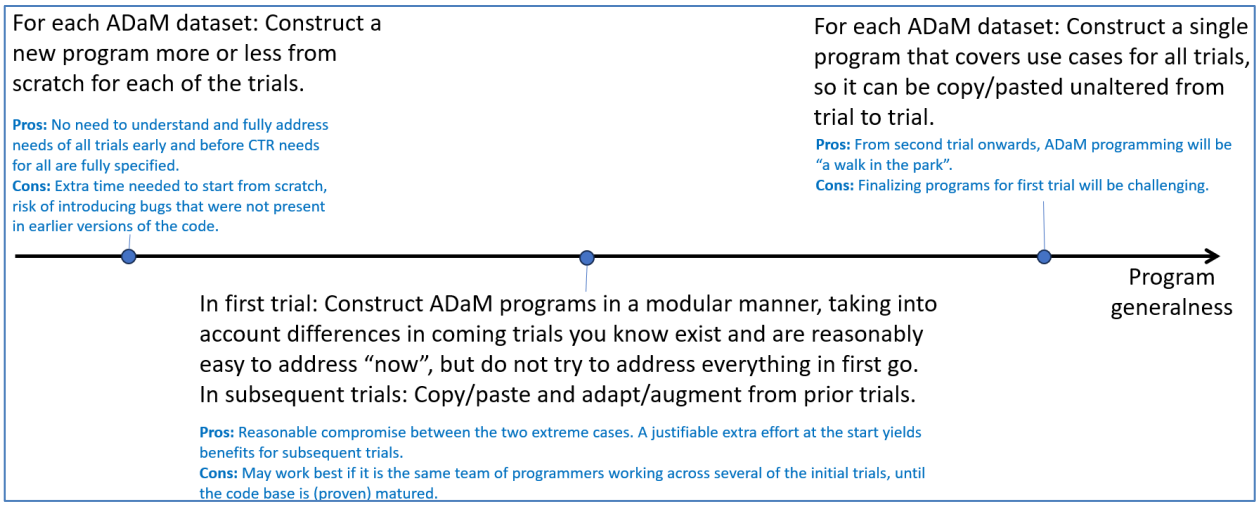

Figure 3: Pros and cons of the two extremities of achieving a very general program that works in all trials vs. creating a new program for each trial, and a reasonable compromise that proved to work well.

This flexible structure provided a practical compromise between generalness and specificity, enabling efficient reuse of code while accommodating the unique needs of each trial. This approach facilitated faster development and higher quality in creating ADaM datasets.

ORDER OF ADAM CREATION

In ADaM, the ADSL dataset exists specifically for containing subject level data, whereas the datasets of BDS or occurrence class may contain multiple entries per subject, linking the data of each record to e.g. a timepoint as well as a subject. However, for certain uses for reporting and analysis, it is sometimes necessary to include certain derived variables in the subject-level data, e.g. treatment responder flags. These may need to be derived based on other ADaMs rather than the corresponding data from SDTM, since the ADaM-level operations such as visit reallocation and imputation should have taken place on the data used to determine whether e.g., a subject is treatment responder.

This poses a challenge in terms of how to achieve this. Since most ADaM datasets contain certain subject-level core variables, the ADSL dataset is normally one of the first ones created, with the BDS and occurrence datasets being created afterwards. But then how can we have ADSL contain a treatment responder variable based on e.g. the contents of ADLB? And what if that variable is considered so essential that you want it to be included as a subject level variable in several other ADaM datasets?

One possibility is to have e.g. the ADLB logic (full, or parts) embedded in a callable module, so that both the ADLB program but also the ADSL program can invoke the ADLB logic. This way, the relevant derivations can be done when ADSL is produced, and repeated when ADLB is made. Alternatives include updating datasets after initial creation. Figure 4 shows our perception of pros and cons associated with the possible ways to address the challenge.

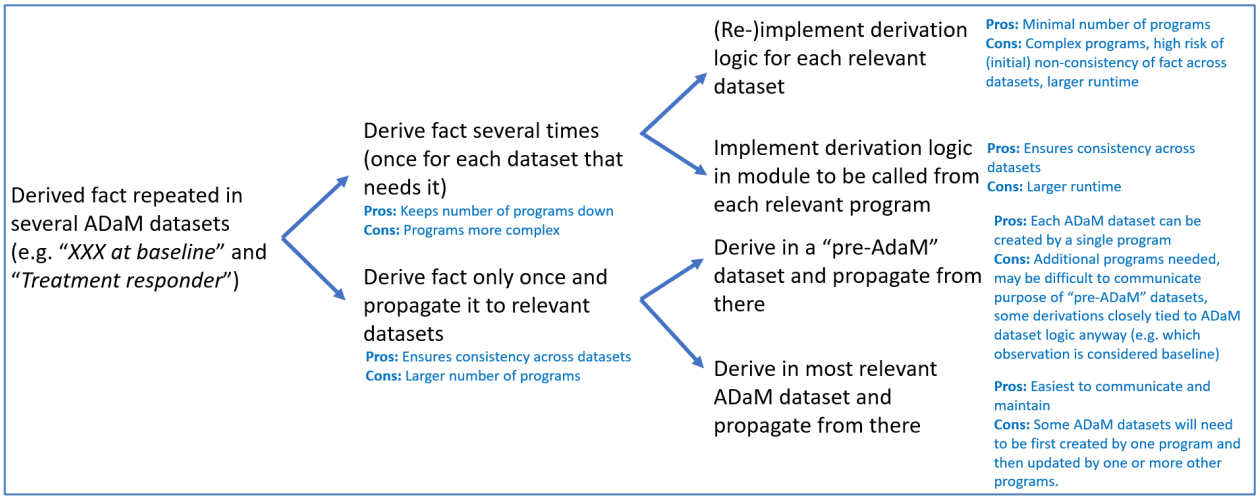

Figure 4: Different ways to address the challenge of derived facts repeated in several ADaM datasets.

We opted to not pursue the “derive facts several times” model, since it would mean increased processing time (and during the development phase, we would often re-create ADaMs several times a day), and it would also make e.g. the ADSL program more complex than it needed to be, with increased risk of introducing bugs and having to spend precious time addressing these. Instead, we strayed from the concept of "one ADaM dataset - one program" and accepted that an ADaM dataset may be initially created in one step, and then updated in one or more later steps.

E.g. we would
1) Create ADSL with all empty treatment responder flags
2) Create ADLB with all empty treatment responder flags
3) Update ADSL by deriving values for treatment responder flags from ADLB and updating them in ADSL
4) Update ADLB by propagating treatment responder flag values from ADSL to ADLB

This may seem overly cumbersome, but update programs were largely calls to central modules, keeping complexity down. A sensible naming standard for the programs helped ensure that they were executed in the right order, see Figure 5.

<fixed prefix><ADaM dataset name>.<program file extension>
E.g. **mk_adlb.sas**

Pros: If prefix omitted completely, there is an easily understood 1:1 relationship between program name and the ADaM dataset created by the program.
Cons: Required execution order of programs not clear – whenever you want to “run all”, a different source must be consulted (or time wasted with trial and error). Not really possible to have programs updating ADaM datasets with “late derivable facts” within this naming scheme.

<information-carrying prefix><ADaM dataset name><optional suffix>.<program file extension>
E.g. **mk450_adlb.sas**

Pros: The prefix can contain a number that communicates desired execution order: Smaller numbers have to be run before bigger numbers, and programs with the same number can be run concurrently.
Secondary update programs possible, e.g. **mk905_adlb_update.sas**.
Cons: Slightly more challenging to locate the relevant program(s) for a given dataset in a list of program files.

Figure 5: Simple vs. heavily information-carrying program names.

It is essential that ADaM programs are executed in the correct order, but it is open to discussion whether the information about “correct order” need to be part of the program names, or if this information can be maintained in a flat file or other extra entity. In favour of the extra entity is that execution order can then very easily be changed. However, only by having the information in the program name is a reminder to think about upstream dependencies served whenever a program is submitted for execution.

The decision to update ADaM datasets in multiple steps rather than adhering to a single program per dataset allowed for manageable complexity and reduced processing time. The use of a sensible naming standard and central modules facilitated the correct execution order, ensuring consistent dataset creation.

ENSURING ADHERANCE TO STANDARDS

As mentioned earlier, in addition to the official ADaM standards and guidance from CDISC, the team also had to adhere to company- or project-wide decisions limiting the sometimes quite open ADaM standards. Once datasets were created, the Open CDISC Validator tool was used for checking consistency with ADaM standards. However, running this tool was somewhat cumbersome in terms of both required number of steps and actual run time, so it was not natural for programmers to do this daily during the early stages of programming.

Since new and updated ADaM datasets were instantaneously available to the TFL-programmers, it was of particular importance to quickly rectify any (unintended) deviations from the standards, so that TFL-programmers would not waste valuable time programming against something that would perhaps be changed the next day. Using SAS programming, we made a solution in which a check program could quickly compare the actual structures of the currently available ADaM datasets with the desired one as described in manually maintained “metadata”. The check program would alert about deviations, so that the responsible programmer could take corrective action by modifying the program and/or the Excel file describing the desired structure. See top part of Figure 6.

Whenever the desired structure of an ADaM dataset at the given trial was updated and converted to SAS, there was also a program to be run that would compare the new structure with the available standards and highlight in case of any discrepancies. See bottom part of Figure 6.

Implementing a system to check and adapt dataset structures against manually maintained metadata ensured adherence to both official and internal standards. The integration of these checks into the daily work ensured consistent compliance with ever evolving standards.

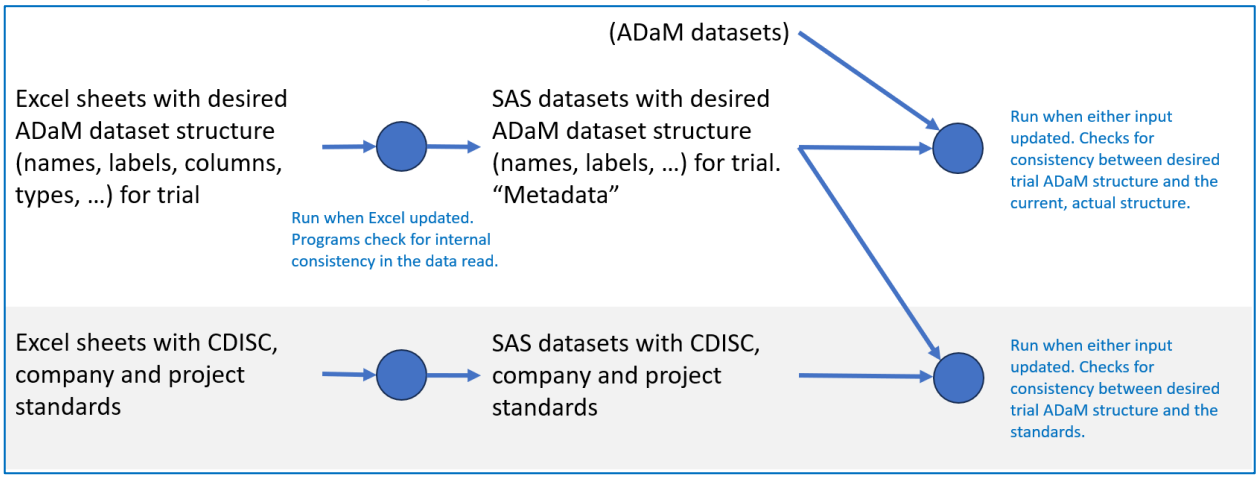

Figure 6: Use of manually maintained "metadata" describing standards at global, company, project and trial level.

DISCUSSION

Although the principles we found useful in our projects were established several years ago, we have on an ongoing basis since been able to observe that these principles have merit. We believe our projects have shown that, when tasks are approached in a thorough and well-considered manner, successful ADaM creation can be achieved, scaled, and adopted relatively easily when using SAS. While other programming languages may also be used for ADaM creation, they are still in the early stages of maturity and adoption within the pharmaceutical industry. This is an area that requires further investigation, as seen in initiatives like Pharmaverse [2].

As other tools become more commonplace, it is crucial to retain certain valuable aspects of our SAS programming practices, even if it is decided to switch to another language. The decision to switch from one language to another should not be driven by the desire to follow new technology trends or the fear of missing out. Instead, short-term and long-term benefits and side effects of a transition should be considered before deciding. Due to the highly regulated nature of the industry, adoption of new tools should not happen at a pace under which the quality is compromised.

Undoubtedly, many of the concepts we have described could be implemented using other programming languages or solutions, and some may even offer better results, such as the compliance check example where other commercially available tools show significant benefits. However, the question remains: why replace SAS entirely with a new language? We believe that SAS continues to hold merit for ADaM creation because it allows for the implementation of all the principles we have described. Furthermore, SAS offers a consistent syntax across various parts of the language. In contrast, some of the available alternatives appear to have a 'conventions are decided per package' approach to syntax, potentially making it harder for a programmer to master a sufficiently large part of it.

SAS should coexist with other languages that demonstrate strengths in their respective domains. This approach aligns with the "fit for purpose" philosophy rather than adopting a "one language fits all" mindset.

Additional more general discussion points which are not directly linked to our above experiences, are:

i. Regulatory Compliance and Validation
   One of the key advantages of SAS is its long history of use in the pharmaceutical industry, which has led to a robust framework for regulatory compliance and validation. Regulatory agencies are familiar with SAS outputs and processes, which can streamline the review process. Any new programming language adopted must undergo rigorous validation to ensure it meets the stringent requirements of regulatory bodies like the FDA and PMDA.
ii. Training and Skills Development
   A driver for an organization to transition to another programming language may be the availability of job candidates for open positions. If a certain language is favoured in university courses, it may seem logical to follow. But for existing staff, transition to a new programming language is likely to involve significant investment in training and skills development. A rapid cut-over may also mean inadvertently retiring

expertise and good practices accumulated over years. As such, integration of new languages should happen in a gradual manner, mitigating the risks associated with a transition.

iii. Interoperability and Integration
With the clinical datasets (not the TFL's only) being more and more central, the ability to integrate and interoperate with other systems and platforms is crucial. SAS has proven capabilities in this area, with extensive support for various data formats and integration with other software tools. Any new language adopted must demonstrate similar or superior capabilities to ensure seamless data flow and integration within the organization.

iv. Support
The SAS programming language is well-established, with extensive resources, documentation, and support available. This support can be invaluable, especially when troubleshooting complex issues. While emerging languages may have growing communities, the depth and breadth of support available should be a key consideration in the decision-making process, and investments or establishment in new communities must be considered.

v. Innovation and Futureproofing
While SAS has a proven track record, it is also essential to consider the pace of innovation and futureproofing. Emerging languages may offer features and capabilities that align better with future technological trends and business needs. Balancing the stability and reliability of SAS with the innovative potential of new languages can provide a strategic advantage.

In conclusion, the decision to adopt a new programming language should be made with careful consideration of various factors, including regulatory compliance, training, interoperability, community support, cost, and innovation. By adopting a thoughtful and balanced approach, organizations can leverage the strengths of both SAS and emerging languages to achieve their objectives effectively.

CONCLUSION

Our experience in statistical programming at Novo Nordisk, particularly in ADaM dataset generation, has led to several key insights and practices that have been instrumental in our project in delivering high-quality deliverables within expected timelines. These practices, rooted in the principles of structured programming, collaborative development, and adherence to standards, have shown their effectiveness in the context of SAS, although supposedly also hold potential applicability across other programming environments.

Key takeaways from our work include:

- *Clear Division of Peoples Work*
  Assigning specific roles to team members, such as differentiating between individual ADaM and TFL programmers, helped maintain focus and quality.

- *Iterative and Collaborative Development*
  Engaging statisticians and other stakeholders in iterative development processes, exemplified by the ADTTE dataset creation including a visualization application, fostered better specifications and outcomes.

- *Modular Programming Principles*
  Centralizing shared logic into callable modules and structuring individual ADaM programs according to ETL principles facilitated easier maintenance, QC, and scalability.

- *Adaptable Program Structures*
  Developing flexible program structures that could be easily adapted for new trials ensured efficiency and consistency across similar datasets.

- *Order of Dataset Creation/Dataflow*
  Implementing a multi-step approach to creating and updating ADaM datasets helped manage complexity and dependencies effectively

- *Adherence to Standards*
  Utilizing a cross-team standards governance board ensured consistency across projects, while internal mechanisms helped quickly rectify deviations from standards.

While these practices were developed within a SAS-dominated environment, they emphasize principles that are universally beneficial, *regardless of the programming language used*. During the shift towards new tools and languages, the pharmaceutical industry should continue to embrace these core principles to ensure quality and compliance.

As the industry evolves, the decision to transition to new programming languages should be made with a balanced consideration of regulatory compliance, training needs, interoperability, community support, and future innovation

potential. SAS, with its proven track record and robust framework, should coexist with emerging languages, leveraging the strengths of both to achieve optimal results.

In essence, these programming principles and learnings can be likened to valuable old wine from the far end of the wine cellar, while the choice of programming language can be seen as the choice of the surrounding bottle and label.

REFERENCES

[1] Base SAS vs. Data Integration Studio: Understanding ETL and the SAS tools used to support it (lexjansen.com)

[2] https://pharmaverse.org/

ACKNOWLEDGMENTS

While gaining the experience which is basis for this paper, we worked together with a number of great people within and exterior to the team of which we were part. We would like to acknowledge the following for having influenced the work with ADaM datasets: Lars Holm Damgaard, Thomas Gulholm-Hansen, Oluf Højbjerg Hansen, Jens Wej Modvig and John Roth.

CONTACT INFORMATION

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

Henning Pontoppidan Föh
Novo Nordisk A/S
Knud Højgaards Vej 1
2860 Søborg
Denmark
Work Phone: +45 3079 4981
Email: hpf@novonordisk.com

Lars Lehmann Andersson
SAS Institute A/S
Havneholmen 8
2450 Copenhagen SV
Denmark
Work Phone: +45 2721 2793
Email: Lars.L.Andersson@sas.com