PhUSE 2009

Paper AD15

# Main challenges for a SAS programmer stepping in SAS developer's shoes

Sebastien Jolivet, Novartis Pharma AG, Basel, Switzerland

## ABSTRACT

Whether you work for a large pharma or a local CRO, you are used to spend most of your time creating datasets, tables, figures and listings. Programs that generally have been developed (or adapted) for one particular task, one particular study.

Imagine you get the opportunity of developing some SAS™ code, that all programmers in your group would use for the derivation of the main efficacy datasets. They would be the users of this code... and you would become the developer. The challenge is highly attractive. Do you think you are ready for it?

This presentation intends to highlight the difficult steps for a statistical programmer to take on the role of a SAS developer, the challenges, the expectations, etc...

## INTRODUCTION

From a personal perspective this paper will look at the difficulty of developing standard programs for a programmer who has only ever worked on his own clinical studies.

The complexity of the task is to reach the very natural expectations that users have about the tool, with limited experience in macro development and no clear idea of what it is to develop a standard program. Naively, the initial impression was that it is nothing more that some validated code being shared with others.

The aim of this paper is neither to explain how to become a good SAS developer nor to explain how to write standard macros. This paper simply intends to share a personal experience and to express some important points one should be very aware of before going on this unsteady path.

## DESCRIPTION OF THE NEEDS

### HIGH LEVEL TERMINOLOGY

In this paper, the following terminologies are being used:
A SAS programmer/reporter is somebody whose daily work is to support the development of study drug, by producing derived datasets, tables, figures, listings and all related activities. They have a very good understanding of the data, the way they are being collected, cleaned and analyzed. They are also experienced users of the company reporting tools. They might also be involved in the development of standard programs, but those are limited to a specific project or client. Their main interest is to make sure timelines are met with all the needed information in a correct manner (usually in terms of contents and layout).

A SAS developer is somebody who is working on the implementation of company standards. They are involved in the settings of tools and programs that users need and use. They are responsible for the upgrade of newer versions, the appropriate trainings of users, and the technical support as needed.
Those definitions are only based on personal appreciation from the author of this paper.

### DESCRIPTION OF THE NEED

Within oncology, efficacy endpoints are usually based on tumor assessments. Those are often derived according to the RECIST criteria for tumor responses (Therasse and Arbuck et al., 2000, New Guidelines to Evaluate the Response to Treatment in Solid Tumors, Journal of National Cancer Institute, Vol. 92; 205-16).

RECIST (REsponse Criteria In Solid Tumors) is a set of tumor assessment criteria developed by major public institutions (EORTC, NCI), cooperative groups and industry to achieve a standardized way of assessing tumor response. RECIST is now widely accepted in the oncology community and became an industry standard in claiming efficacy results in solid tumor, adopted by all Health Authorities. RECIST is meant to be a guideline to standardize research results across the industry that sponsors can adopt.

After deriving responses according to the RECIST criteria, ultimately, the aim is to derive standard study endpoints most commonly used when working on these data (e.g. Progression free survival, Best Overall response, etc..).
When working in Oncology, the existence of a standard program handling RECIST data is a clear bonus to the efficiency of the group, guaranteeing consistency across projects and standardise the datasets provided to health authorities. The main core of the tool is therefore basically to create derived datasets which contain all the key variables for the analysis based on RECIST criteria (derived responses + efficacy endpoints).

Several programs already existed but referring to previous versions of the RECIST paper. Those were all reviewed and assessed whether they could be upgraded into a company standard. All those programs however had the same particularity: It was very difficult to imagine promoting them into a broader use than what they had been written for. They were all consisting of one big macro with loads of proc and data steps, and very large %if.. %then statements. This code was generally copied and adapted from one study to another.
It was therefore decided to start another program from scratch. One of the already existing programs would be used for validation purposes.
The end product was nothing more than a large set of SAS macros creating derived datasets used in the analysis of tumor assessments based on RECIST criteria.

#### EVERYBODY'S EXPECTATIONS AT THE BEGINNING

Management was willing to get these macros in place in a usual manor: minimise the cost and maximise the functionalities of the tool. In their dreams, not only should the tool derive responses and the conventional study endpoints derived from RECIST, but in a longer term, the whole package should also contain standard analysis (e.g Kaplan Meier graphs of progression free survival). A good way of reducing the cost was clearly to ask a programmer, working in a team with primary analyses on RECIST data, to take over the responsibility of implementing these activities. They ensured this individual had some time off from project activities and was completely dedicated to this new task. In order to ensure flexibilities in the tool, it was planed for it to be tested on studies with different settings (e.g. Paper and electronic CRFs, in-house and outsourced data management, central radiology review and local investigator, Phase I and III studies, etc..)

Study programmers, in general, would be happy having a program already available to derive those standard endpoints. They won't need to know in every details how the variables are coded. From the documentation of the tool, they should be able to get the algorithm/definitions corresponding to those, and therefore have a good understanding of what is reported in their datasets. The documentation should also help them setting up their calling programs accordingly.
Due to the sensitivity of the data, an independent program will still need to be created (most probably based on one of the original programs discussed above) to validate the execution of the standard tool.
In general, this should mean study programmers get more time to concentrate on other parts of their analysis, having this big section handled in a standardised way.

The persons identified for the task have a clear opportunity of setting their own standards, of extending their technical knowledge, of improving their knowledge on the matter (RECIST in this example), etc… This is a fairly rare opportunity for a SAS programmer. At this stage, their impression was that it would take between 3 and 6 months of hard work, intense programming, a bit of documentation, some slides that could be used for a training and that will be it. Once this is finished they will quickly be able to go back to their usual activities, following their trials again.

## IMPLEMENTATION OF THE TOOL MEANT SOME DEVELOPMENT… AND SO MUCH MORE

The development of the SAS code in itself, was more or less as it had been envisaged:

- a lot of reflexion on how best to develop the programs,
- a lot of discussion on the assumptions that could be taken,
- a lot of thoughts on the structure of the output datasets,
- a lot of testing,
- a lot of adding pieces in and/or of taking some other out, etc..

**HOW DIFFERENT IS IT TO PROGRAM SOME STANDARD CODE IN COMPARISON TO A SINGLE USE PROGRAM?**

One of the most obvious points is that in the development of programming standards, there is kind of no end to it. There is always places for some improvement, for some new features to be added... In the development of a single use derived dataset or output, the program development ends when the outcome is final. In macro development, there could always be a need for some improvement or modification to the code.

Here is a very brief history of the set of macros used for the implementation of the RECIST macros:

| **History of the macro** |
|---|
| The very first component of the macro finalised in July 2007 |
| Complete macro finalised in February 2008 |
| 1st update done in early May 2008 |
| 2nd update done in late May 2008 |
| 3rd update done in August 2008 |
| 4th update done in early February 2009 |
| 5th update done in late February 2009 |
| 6th update done in April 2009 |
| 7th update done in May 2009 |
| *And two years later.... here I am still talking about it!!!* |

The longer these macros are being used, the more likely there will be several developers reviewing and updating the code.

Standard codes need to run on every type of data that is possible, and not just on the ones available in one specific study. Standard code needs to run adequately on all the different possible branch given in the RECIST criteria. The developer needs to think of all those scenarios and to implement solutions. He needs to become an expert of the subject, expert in the way the data are collected, but also in the way they are being analysed. In this particular case, the database structure was different between a study using an electronic CRF and a study using paper CRF. The standard code needs to run no matter which type of CRF is being used. In this case, in order to avoid potential confusion, this should really be transparent to the users.

The development is all done in small modules. This eases the development, the review and the updating of the entire tool. This also gives the opportunity of singling out one particular module and to use it independently from the other modules. Modular programming also gives the advantage that different developers could be involved in the entire project, but working on their own little area of expertise, and that at the end it would eventually all fit in together. In such a setting, the role of a project leader would be essential. The different small modules should be clearly identified before the beginning of the development phase. Each modules should have its own documentation (the goal, the assumptions, description of input and outputs).

Everything that is input and/or outputs that is required or made by the macro has to be considered very carefully. Variable names and SAS macro parameter names needs to be self explanatory, but also needs to follow general company practices. Every input that is deemed to be requested by the overall tool has to be discussed within a team. Even the way this input will be done should be discussed. Will this input be done via a macro parameter? Or an input dataset with all the necessary information? Or a combination of both? The names needs to be clear and self explanatory, but the purposes of it needs also to be clear. For example, why does the RECIST macro need to know the tumor assessment schedule for the particular study? And how should this information be collected so that it is easy to get for the user and easy to use in the macro?

The release of any versions of standard macros needs to be done very carefully.
If, for example, in one version of the tool, you have added a variable which you later on realise might actually not be that necessary or interesting. To remove it in the subsequent versions of the tool is not such a straight forward action as one could think. You need to assume that this variable was potentially used. Users needs to be informed of it, even if the added variable was nonsense. The impact of removing this variable needs to be assessed carefully.
Every effort needs to be taken in order to have any new versions backward compatible. If a new version is not made backward compatible, the developer will need to spend a lot of time documenting the differences and informing users on the changes they need to implement.
Every finalised version of the macro should be kept, as users might not want to use the latest version. For example, it could happen that one user has used a validated version of the tool for his/her interim analysis. At the time of his final run, there is a new version in place. In order to keep consistency between interim and final analyses, the user might decide to keep using the old macro.

**AND ONCE THE TOOL IS FINISHED?**

The main developer of the tool has the advantage of having their name at the top of a set of macros. Those macros should ideally simply be referenced to and not modified. If a user has any difficulty with its use or any questions on how to set it up, it is then very easy for that user to contact the developer and to discuss directly with the developer of the application.
The developer is clearly identified as a key contact for every single issue/concern/question that programmers have on the related data (from CRF design, cleaning of the database to the reporting).
Programmers might even wish to contact the developer in order to get their point of view on other different endpoints that they need to derive, or on how to get the macro to work when none of the general assumptions are met (things like the structure of the raw data or the contents of the data collected).
The developer is a subject expert in the area... even when the area is not exactly the same one.

In addition to being the subject expert, the developer can also very quickly become the technical support. He is of course the ideal person to understand any errors/warnings that users might get. He knows the intermediate datasets and variables that users normally don't really need to know and look into. By looking at those, he might be able to point at some issues in the data, in the setting of the calling programs…. Or in the coding of the macro.

Having a vey detailed documentation in place will be a must when related data are been submitted to Health Authorities. They could required specifications on how those key variables were derived. Having a great documentation is also a great source of information for more advance users who would want to understand in detail the derivations of the different endpoitns created by the tool.
The minimum in terms of documentation should contain the following :

- some technical specifications (clear list of every bit of code that was developed, how, why, and how it was validated and tested)
- some user manual (clear list of all the assumptions, the options available, a description of the outputs, etc..).

The next step is to inform users of the existence of a new standard tool. Sending an email or waiting for the rumor to spread is one way, but having some training available at hands is so much more powerful. Users are generally eager for this tool to be ready and very keen on learning how they might be able to use it. The absence of training slows down the spread of the tool. The absence of user manuals increase the time spent in technical support. Training is also a good way of creating an interaction between users and developers, this interaction is important in order to get pertinent feedback on the tool. Getting feedback is essential for the tool to stay up-to-date. What is it they like? What is it they don't like? What is it they cannot do and wish they could? What is it that they find unclear? What could be improved? Etc…

## CONCLUSION

Although developing standard macros is a fantastic experience, challenging by all means, this work is very demanding and the technical aspect is probably not the hardest part.

The developer needs to understand the data even more than a user would. They need to imagine all possible cases that might occur in the data, to understand the impact of different study designs, to know the different analyses that might be done with these data.

The developer must take a good care at the documentation and, as for any other programs, the importance of this documentation is often only realised when this documentation is actually not there. Specifications should be written, discussed and agreed beforehand. Development should only start after. When the documentation is nicely in place, any update needed would greatly benefit from this.

The communication of the tool is also something very important. With a good supporting documentation, the spread of the tool will be much more facilitated and feedback would also come more naturally.

## REFERENCES

RECIST criteria for tumor responses (Therasse and Arbuck et al., 2000, New Guidelines to Evaluate the Response to Treatment in Solid Tumors, Journal of National Cancer Institute, Vol. 92; 205-16).

## ACKNOWLEDGMENTS

I would like to acknowledge Annabelle Millischer Foulon and Dave McCandless for reviewing this paper and Michel Gauthier for his valuable suggestions.

## CONTACT INFORMATION

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

Sebastien Jolivet
Novartis Pharma AG
WSJ-103.2.10.21
Forum 1
Novartis Campus
CH-4056 Basel
Switzerland
Phone: +41 61 32 40607
Fax: +41 61 32 40064
Email: sebastien.jolivet@novartis.com