Paper CT11

Practical implementation of Good Programming Practice and other theories in macro development

Fred Ross, Parexel, Stockport, United Kingdom

ABSTRACT

When putting theory into practice during the development of a macro, concepts and ideas might not always hold up as intended. This could be due to various reasons such as but not limited to: the nature of the project the macro is intended for and/or limited time to spend on the development of the macro. In a previous PHUSE presentation I have outlined concepts for Good Programming Practice and other ideas to follow or consider when developing macros. In this presentation I will explore these previously established ideas and the practicality of what worked and what didn't in a real world example.

INTRODUCTION

In this paper I will be looking at the application and the following of good programming practice in the RDM Lab Grading macro project I undertook over several months in 2022. Using the PHUSE Good Programming Practice Guidance as a reference point, I will be comparing the ideals outlined in both with my experiences in the project.

RDM LAB GRADING MACRO PROJECT, STARTING POINT & GOOD PROGRAMMING PRACTICE

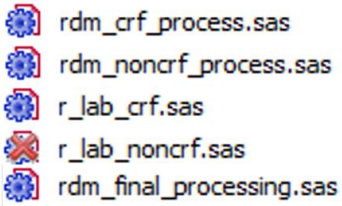
ORIGINAL PROGRAM'S PURPOSE & USE?

I was tasked with working with an existing superuser maintaining team (henceforth referred to as the maintainers or maintaining team) to refine a suite of macros. The macros and programs took a reasonably standardised raw data, turned it into SDTM data and the pushed it through a Labs ADaM macro to create lab test gradings. The original programs were maintained & managed by a handful of individuals, but the main user base was a larger number of other programmers. The main issue to solve was for the maintainers to reduce the large number of tickets and queries which these programs generated.

There was not a pool of SDTM or ADaM knowledge to utilise in refining these macros which is where I came in with a previous background in ADaM programming and macro development theoretical practice.

The most relevant points of the RDM Lab Grading macro project are as follows:

- Originally a suite of macros for taking raw data and turning this into SDTM equivalent data which is then pushed through a LAB ADaM macro to create lab gradings.
- 6 Differing versions of macros for Early Phase, Late Phase and PDMA. With CRF and NONCRF versions of each.
- Used across hundreds of different studies
- Being run by multiple programmers who aren't ADaM experienced programmers
- Maintained by a handful of individuals who have other responsibilities
- Original macros were generating many issues for the maintainers



Example of the programs used for Early Phase CRF & NONCRF version

MY STARTING POINT

The main challenges I faced starting out were:

- Familiarity – I had not used these macros myself
- Understanding – I had not worked on this area of programming before
- Time – Some study work to be starting up around the same time meant I couldn't work full time on this

The starting point for the RDM Lab Grading macro project involved lots of meeting and discussions on the requirements and issues the current team was facing. To be able to follow good programming practice it is important to comprehensively understand the task.

BOILING DOWN THE CODE, RESTRUCTURING, ADDING CODE

BOILING DOWN THE CODE AND RESTRUCTURING

Following the initial meetings and discussion the next step was to refine the code from multiple programs and areas into one program with several macros in a macro area. Initially the focus was to combine where it made logical sense to do so, removing or merging code where applicable. The result of this was 12 macros and 1 final program, this was a similar number to where we started however this was due to firstly reducing the original code down so the original code was contained within less programs, in the same area in logically named macro programs. Secondly additional macros were added for additional features to the final program.

Final program used by programmers in red and the list of macros contained within that below.

ADDING TO THE CODE

The most numerous additions to the overall code in the RDM Lab Grading macro project was user errors, warnings and notes. The scope of which included warning and aborting if incorrect formats or inputs were used in our new set of macros. Throughout the project there was space to add additional ideas into the code but instead of adding this all now in the first pass and due to time limitations, the focus was more on making the new programs very easy to put this new code in.

Program Header

```
%macro lbtgr_studystart(date=);
%***Warnings;
%if %qsubstr(%str(&date.),5,1) ne %str(-)
  or %qsubstr(%str(&date.),8,1) ne %str(-)
    or %length(&date.) ne 10
      %then %do;
        %put %str(ERR)OR: Study start date inputed is not of correct format YYYY-MM-DD. Aborting program.;
        %abort;
      %end;

%*** Set _studystart;
%global _studystart;
%if &date. ne %then %let  _studystart = &date.;

%else %do;

%* Automatically detect study start date code here;
%put %str(ERR)OR: Study start date is empty and requires it to be populated by user. Aborting program.;
%abort;

%end;

%mend lbtgr_studystart;
```

Example code from the lbtgr_studystart.sas program.

FINAL MACRO PROGRAM

The final macro program as summarised below we can see most of the code is hidden behind macros. This is due to the setup of these programs with a small group of maintainers and a larger group of programmers who would run this program on their studies. The macros called here have logical names so that an unfamiliar programmer can read and understand the program easily. Should the program have issues then it will hopefully mostly be straightforward to query and resolve. The minimal expectation is that a programmer will be able to run this with a higher success rate than the old macros, meaning less time spent by the maintainers in resolving issues.

Program Header

```
  %lbtgr_setup(studyphase=  /* EP LP or PDMA */ ,
               crf_or_noncrf=  /* CRF or NONCRF or BOTH*/,
               debug= /*Y or N*/);
%* When debug is set to Y diagnostic datasets will be created in outdata;


%*** Update study start in macro lbtgr_studystart to pick up correct LB metadata;
%*** Date must be in ISO format YYYY-MM-DD;
  %lbtgr_studystart(date=);


%*** Put lab grading criteria from the list below into lbtgr_grade_criteria;
%*** _EXT_NCICTCV4_ : for NCICTCAE version 4;
%*** _NCICTCV4_ : for strict NCICTCAE version 4 (Potential use of LAb Interface only);
%*** _NCICTCV5_ : for NCICTCAE version 5;
%*** _WHOV1_ : for WHO grading version 1;
  %lbtgr_grade_criteria(criteria=);
```

Additional Macro Calls and Pre-processing Options

```
%*** Call to macros lbtgr_(EP/LP/PDMA)_(CRF/NONCRF) depending on phase and crf option selected in lbtgr_setup;
  %lbtgr_crf_noncrf(crf_dsin_lab    = staging._lab /* Name of input lab dataset for CRF */,
                    noncrf_dsin_lab = staging._lab_nc /* Name of input lab dataset for NONCRF */,

                    crf_dsout    = outdata.r_lb_rdm_vad /* Name of output CRF dataset */,
                    noncrf_dsout = outdata.r_lb_ncrf_rdm_vad /* Name of output NONCRF dataset */);
```

Post-processing Options

Example code from the final program r_lab_crf_noncrf.sas.

PHUSE GPP RELATING TO THE COMPLETE LAB MACRO TASK

Below is the PHUSE GPP outlined on the PHUSE wiki with context on how this has been applied across the RDM Lab Grading macro project:

- Write code that is clear and easy to read and review
  - I had taken existing code and made it easier to read and review.
  - All code is contained within one main program with multiple macros called instead of being spread across multiple areas.
- Write code that is easy to maintain, modify and debug
  - During the project I had refined the existing modularity of the code, then stored it in relevant folder areas.
- Use robust programming to reduce the need for code maintenance
  - The code was already partly robust but there was lots of space to improve the code here which was achieved with the help of the maintaining team.
- Develop code that can be reused or easily adapted
  - The original code was already being used on hundreds of studies however, having followed other GPP outlined this has naturally made the new programs more reusable and easily adaptable.
- Write code that uses minimal computer processing resource (efficient code)
  - Not hugely applicable in the project but having the program abort should an invalid macro-option be inputted saves running time on an incorrect run.
- Write code in such a way to reduce logical and syntax errors
  - Part of the new programs was to have additional user generated errors and warnings to reduce user input and option selection mistakes. In addition, the code has been restructured so should anyone want to add extra ways to avoid logical and syntax errors this has been provisioned.

CONCLUSION

How well GPP is followed can depend on time, scope, and purpose of any given programming project. Being aware and consciously evaluating your work in relation to it when beginning, working on, and finalising a project can help to easily and efficiently follow GPP guidance.

REFERENCES

PHUSE WIKI

https://advance.phuse.global/display/WEL/Good+Programming+Practice+Guidance

BIG SAS® CODE NAVIGATION & MANAGEMENT FRED ROSS & JESSICA WHITTAKER-DIXON

https://phuse.s3.eu-central-1.amazonaws.com/Archive/2019/Connect/EU/Amsterdam/PAP_CT14.pdf

ACKNOWLEDGMENTS

Thanks to; Violet the cat; the PHUSE Coding Tips & Tricks chairs for their hard work and review comments on papers and presentations.

CONTACT INFORMATION

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

Author Name: Fred Ross

Company: Parexel

Address: Parexel International, Navigation House, 1 South Quay Drive, Sheffield, S2 5SU, United Kingdom

Email: fred.ross@parexel.com

Web: https://www.parexel.com/