PharmaSUG 2017 - Paper IB02

Good Programming Practices at Every Level

Maria Dalton, GlaxoSmithKline

ABSTRACT

Programming in the pharmaceutical industry focuses on transforming and analyzing clinical data. It is critical for pharmaceutical programs to be accurate since decisions about the safety and efficacy of drugs are made based on their results. In addition, programs must be well-documented, efficient, and reusable – this is necessary to meet tight timelines and resource constraints and to ensure that programs can be understood by other programmers and regulators. Thus, it is critical that pharmaceutical programmers follow good programming practices (GPP).

This paper explores the topic of GPP in the pharmaceutical industry at different levels – at the level of individual programs, at the study level, and at the compound or therapeutic area level. At each level, there are different yet complementary practices that a programmer should follow to produce accurate and robust programs.

The SAS® programming language is commonly used in the pharmaceutical industry, so some recommendations in this paper are SAS-specific, but the general principles can be applied to any programming language. This paper is most useful for beginner programmers, but it will be helpful to experienced programmers as well, especially for lead programmers and managers who are trying to embed good programming practices within their teams.

INTRODUCTION

The PhUSE Good Programming Practice Steering Board has developed a very useful guidance document for good programming practices, available on the PhUSE website[1]. This paper is based on the recommendations in the PhUSE GPP guidance document, with additional suggestions and examples. This paper examines good programming practices using a framework that first discusses the structure of individual programs, then considers study-wide GPP, and finally discusses compound or TA-level recommendations. I use this framework because it mirrors the way that many pharmaceutical companies organize their programs and programming staff.

GOOD PROGRAMMING PRACTICES AT THE LEVEL OF INDIVIDUAL PROGRAMS

The individual program is the foundation of a robust analysis of clinical data. Certain practices should be followed for every program to enable easier maintenance and reuse of programs. Here are some of the PhUSE GPP recommendations for individual programs[1]:

- Always include a program header in every program to explain the purpose and the revision history of the program.
- Write comments within programs that clearly explain the rationale of the code.

  Programs have to be maintained and revised over time, often by different programmers. Clear comments will help a new programmer quickly understand a program.
- Follow consistent coding conventions.

  As explained in the GPP guidance document, a standard code structure makes programs more

readable, thus making it easier to maintain and share with other programmers. The GPP guidance document lists many suggestions for coding conventions. I've listed a few here to give a flavor of the recommendations. See the GPP guidance for a full list of coding conventions.

- "Do not overwrite existing datasets"
- "Use 'data=dataset' option in procedure statements so that the dataset being used is explicitly stated to ensure that the statement will work if it is moved to another location"
- "End data steps and procedures with run or quit to provide a boundary and allow for independent execution"
- "Insert parentheses in meaningful places in order to clarify the sequence in which mathematical or logical operations are performed"
- "When converting character variables to numeric or vice versa, use the put and input functions to explicitly convert the variable to ensure that it is done in the way intended"

- Write programs to be reusable and avoid hard-coding.

  Most programmers are aware of the fact that they should not change data values within their programs in order to comply with the FDA's 21 CFR Part 11 Rule[2] which requires a clear audit trail of all data changes. However, sometimes programmers think it is okay to do another type of hard-coding – they sometimes include study-specific values within their programs. For example, they might hardcode the name of a specific drug or visit number. This is not a good practice since it makes the program less reusable. Programmers should instead strive to write programs that will work in general situations (e.g. programs that will work for any study treatment or any study design). If writing a macro, take advantage of macro parameters to allow flexibility in programs. If it is absolutely necessary to include study-specific values in a program, clearly document and encapsulate the study-specific portions of code, so that another programmer can easily see what portions of the code will need to be changed for a different study.

GOOD PROGRAMMING PRACTICES AT THE STUDY LEVEL

In the pharmaceutical industry, programs are organized around clinical trials. A set of programs are written to work together to analyze the data from the clinical trial. Usually, there is a lead study programmer who is responsible for overseeing all programs for the study and ensuring they work together accurately and efficiently.

To meet this responsibility, the study programmer must first understand the study and data, so that all programs for the study correctly process and analyze the study data. The programmer must review and understand all study documents, such as the Clinical Study Protocol and the Statistical Analysis Plan. It is also essential that the programmer understand the data – how it is collected and how it is represented in the datasets. I highly recommend that the study programmer be very familiar with the Case Report Forms and the site CRF guidelines.

The PhUSE GPP Steering Board recommends a practice called "defensive programming"[1]. The PhUSE GPP Guidance document describes defensive programming as "an approach to programming intended to anticipate future changes of the data that might influence the coding algorithms."[1] This is one of the more difficult practices to follow since it can be challenging to envision unexpected data values. But, being familiar with the Case Report Forms, the site CRF guidelines, and other study documents can help increase understanding of the nature and meaning of the data, thus improving the programmer's ability to anticipate values that may appear in the data in the future.

After the lead programmer understands the study, data, and analysis plan, he or she can start programming. But the lead programmer cannot only focus on individual programs. He or she must consider the entire set of programs for the study. For example, the study programmer must decide how to organize the programs that create the analysis datasets and the Tables, Listings, and Figures (TLFs). A common practice in the industry is to organize these programs as follows:

- For analysis datasets, write one main program per analysis dataset. Program all derivations into the analysis datasets, so the datasets are analysis-ready. This is one of the key principles of the CDISC ADaM model - "analysis datasets that have a structure and content that allows statistical analysis to be performed with minimal programming"[3]. This means that the programs that produce the Tables, Listings, and Figures (TLFs) should be able to summarize endpoints simply, without having to do complex manipulations or derivations of data. So, if a programmer is following CDISC standards, then this practice is required. But even if CDISC standards are not being used, organizing analysis datasets in this way is a good practice.
- For TLF programs, also write one main program per TLF. This makes it easy to identify which program produces which TLF, and it makes it easy to index and document these programs when providing them to regulatory agencies, such as the FDA.

Another key responsibility of the study programmer is compliance with industry data standards, namely CDISC. For CDISC compliance, a good programming practice is to check datasets for CDISC compliance early within the programming lifecycle, using the OpenCDISC tool to flag compliance issues. It is important to catch and correct structural issues with datasets early, prior to finishing the SAP-specified analysis for a study, to avoid changing the datasets after the analysis is produced.

The study programmer must also ensure that the set of programs for the study is well-documented. This may require writing documentation in addition to the program comments. The additional documentation and the program comments will work together to fully document the programming for the study. Here are the types of additional documentation that may be needed for a set of study programs:

- **Documentation of all the data cuts/reportings for the study**

  It is common to write programs for multiple data cuts/reportings during a study. For example, it may be necessary to provide programs to an IDMC, or to produce a dose escalation review, or produce a formal interim analysis prior to the full reporting for the Clinical Study Report (CSR). The programs for each data cut may be stored in different directories or libraries. It is very important to document the purpose of each set of programs, along with other useful info such as the date of the data cut (if this is not obvious in the programming system).

- **Documentation of all data sources for a study**

  Study analyses may sometimes use input data that does not come from the Case Report Form (CRF) data. For example, analysis programs may use data from an adjudication committee or from a clinically-reviewed spreadsheet. Any non-CRF sources of data need to be documented, so other programmers understand the data sources and so these data sources can be described in the Reviewer's Guides provided to regulatory agencies.

- **Specifications of analysis datasets**

  Analysis dataset specifications explain all the derivations programmed into the analysis datasets. These specifications are often in-scope for good programming practices because it is often (but not

always) the responsibility of the programming team to write them. The FDA has emphasized that clear metadata for analysis datasets is critical - "Sponsors should make certain that every data variables code list, origin, and derivation is clearly and easily accessible from the define file. An insufficiently documented define file is a common deficiency that reviewers have noted."[4] If programmers write the dataset specifications, it is good practice to have the study statistician review them to ensure consistency with the RAP. It is also good practice to draft the analysis dataset specifications prior to the start of production programming and finalize them as soon as production programming is complete. Then, the programmer that quality checks the analysis datasets can refer to the specifications during the QC process. The analysis dataset specifications can later form the basis for the Define.xml file in an ADaM CRT package, if one is created.

Documentation is sometimes omitted or abbreviated because of timeline pressures. But in my experience, the documentation of programs saves time and resource in the long run since it facilitates reuse of programs, and it allows new programmers to seamlessly step in and assist when needed. There are often situations (e.g. staff turnover, unplanned work, tight timelines) when we need to add additional programmers to a team. These new programmers will be able to fully contribute much sooner if they have documentation to bring them up to speed. Also, good programming documentation is the foundation for required submission documents such as ADaM CRT packages and ADaM Reviewers Guides.

Another important responsibility of the study programmer is to ensure programming consistency across programs and across different data reportings for a study. The PhUSE GPP Steering Board recommends following standard naming conventions for program names, datasets, and variables[1]. Also, the CDISC CDISC Analysis Data Model Version 2.1 states that analysis datasets must "apply naming conventions for datasets and variables consistently across studies within a given submission and across multiple submissions for a product."[3]

In addition to naming conventions, it is good practice to ensure that algorithms and programming are consistent across a study and across different data reportings, unless there is a scientific or analysis reason for changing definitions. For example, the study programmer should derive baseline flags in the same way in an interim and in a final analysis, or follow consistent date imputation rules in all of efficacy analyses. The study programmer should also ensure that variables stored in multiple datasets, such as age or planned treatment, have consistent values across datasets. Many programmers develop study-level macros to use across different data reportings to help ensure that programming is consistent across the study.

GOOD PROGRAMMING PRACTICES AT THE COMPOUND OR THERAPEUTIC AREA (TA) LEVEL

The considerations that I listed under the study-level also apply to compound or TA-level programming. But there are additional complexities to understanding the data, documenting programming work, and ensuring programming consistency at a compound or TA-level, since a compound may encompass many studies and multiple indications or drug formulations. Also, a programmer working at the compound or TA-level must pay special attention to rapidly developing therapeutic area standards. Through the Coalition for Accelerating Standards and Therapies (CFAST) initiative, CDISC has recently released "22 TA standards projects" and many more are planned in 2016[5].

Due to these additional responsibilities, some companies have a role of Project Programmer who manages and oversees programming at the compound-level. Also, many companies develop compound-level or Therapeutic Area – level macros for efficiency and to facilitate consistency of algorithms across the compound or TA.

CONCLUSION

There are many benefits to following good programming practices in the pharmaceutical industry – these practices produce accurate programs that can be efficiently maintained and reused. My experience in the pharmaceutical industry has shown me that if programmers invest the time and effort to write well-structured and reusable programs, properly document their programs and data sources, adher to standards, and implement consistency across studies and compounds, they will "future-proof" their work – ensuring it can be reused and understood in the future - as well as improve their own efficiency.

REFERENCES

[1] PhUSE Good Programming Practice Steering Board. "PhUSE Guidance on Good Programming Practice by the Steering Board for Good Programming Practice in Health and Life Sciences, Version 1.1 March 2014." Accessed March 21, 2017. http://www.phusewiki.org/wiki/index.php?title=Good_Programming_Practice_Guidance

[2] US Food and Drug Administration. "Guidance for Industry Part 11, Electronic Records; Electronic Signatures — Scope and Application." Accessed March 21, 2017. http://www.fda.gov/downloads/RegulatoryInformation/Guidances/ucm125125.pdf.

[3] Clinical Data Interchange Standards Consortium (CDISC). "CDISC Analysis Data Model Version 2.1." Accessed March 21, 2017. http://www.cdisc.org/system/files/all/standard/application/pdf/analysis_data_model_v2.1.pdf.

[4] US Food and Drug Administration. "CDER Common Data Standards Issues Document (Version 1.1/December 2011)." Accessed March 21, 2017. http://www.fda.gov/downloads/Drugs/DevelopmentApprovalProcess/FormsSubmissionRequirements/ElectronicSubmissions/UCM254113.pdf.

[5] Clinical Data Interchange Standards Consortium (CDISC). "The Coalition for Accelerating Standards and Therapies (CFAST)." Accessed March 21, 2017. http://www.cdisc.org/cfast.

ACKNOWLEDGMENTS

I am very grateful to the PhUSE GPP Steering Board for their review and input and for their "Guidance on Good Programming Practices" document.

RECOMMENDED READING

- *PhUSE Guidance on Good Programming Practice by the Steering Board for Good Programming Practice in Health and Life Sciences, Version 1.1 March 2014:* http://www.phusewiki.org/wiki/index.php?title=Good_Programming_Practice_Guidance

CONTACT INFORMATION

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

Maria Dalton
GlaxoSmithKline
maria.y.dalton@gsk.com