Good Programming Practice Guidance

Good Programming Practice Project

The PHUSE Good Programming Practice guidance and associated documents are maintained by the PHUSE Good Programming Practice project team.

Published Guidance Document

Version 1 of the guidance has been published and is available as an MS Word document for adoption.
GPP Guidance Document v1.1
We also have a one page poster summary of GPP principles for download

The guidance is open for public comment and comments will be incorporated in periodic revisions so please continue to review and provide your feedback.

Introduction

This document provides guidance for Good Programming Practices (GPP) for data manipulation, analysis and reporting of clinical data in health and life sciences organizations.

GPP addresses the way in which you write code and comments. The following key principles of Good Programming Practice and are followed in this document.

- Write code that is clear and easy to read and review
- Write code that is easy to maintain, modify and debug
- Use robust programming to reduce the need for code maintenance
- Develop code that can be reused or easily adapted
- Write code that uses minimal computer processing resource (efficient code)
- Write code in such a way to reduce logical and syntax errors

This guidance is primarily aimed at SAS programmers, however, the principles of GPP also apply to other languages such as R and Stata. In addition, although this is not produced with SAS macros in mind, the same principles apply to macros too.

We often have to update existing programs to add new rules, copy programs from one study to another, and take over programs written by others. This guideline aims to show how to produce well structured and well documented programs so that they are easy to read and maintain over time. It is meant to be applicable to all programs, and hence all programmers regardless of experience. Specific rules may be of more use to novice programmers, but applying the principles should be in mind for experienced programmers and mentors.

Why Good Programming Practice?

Good Programming Practice (GPP) is important within the life sciences and healthcare industries as an increased need for efficiency means that code that is clear, easy to maintain, and efficient is more important than ever. Efficient code and best practices should not conflict with one another. It is essential to have various guidelines to govern and regulate code on clarity, efficiency, re-usability, adaptability and robustness.

Good Programming Practices :

- Ensure the clarity of the code and facilitate code review;
- Save time in case of maintenance, and ease the transfer of code among programmers or companies;
- Minimize the need for code maintenance by robust programming;
- Minimize the development effort by development and re-use of standard code and by use of dynamic (easily adaptable) code;
- Minimize the resources needed at execution time (improve the efficiency of the code);
- Reduce the risk of logical errors;
- Meet regulatory requirements regarding validation and FDA 21CFR11 compliance;

What is Good Programming Practice?

Some examples of GPP from the above definition:

- Clarity and ease of maintenance might be promoted by

Use of headersCommentsSelf documenting codeStyle conventions (such as indenting, clear definition of datasteps and procedures)

- Robust programming can be met by

Dynamic programming- writing programs to accommodate potential changes to data or specifications. Test first design.

Good Programming Practice (GPP Project Team)

- Good Programming Practice project was established in late 2008, for the purpose of promoting and publicizing Good Programming Practices within the life sciences and the healthcare industries.

- Currently there are eight active members worldwide in the GPP team with the common aim of driving the development and promotion of a uniform approach of GPP within the industry. The practice encourages contributions from across companies, non-profit organizations and regulators through conference presentations at PharmaSUG, PHUSE, PSI, CDISC and collaborative contributions through the PHUSE projects.

- Most companies with programming personnel have a document or guideline where they describe their good practices for deriving syntax. This document usually contains aspect of programming techniques such as design appropriate use of macros and level of macros hard coding policy naming convention conservative strategies etc….

- A key aim of the GPP project is to achieve a consensus recommendations for a consolidated guideline which could potentially replace these documents.

Articles on Programming Practices

- Headers
- Commenting
- Coding Conventions
- Coding Efficiency
- Robustness
- Code Review

Quick guide to GPP courtesy of Shafi Consultancy

Useful Principles and Practices for Professional SAS Programmers

Good Programming Practice Conference and Discussion Clubs

GPP discussion Club at PHUSE Conference London 2014

GPP discussion Club at PHUSE Conference Brussels 2013

GPP discussion Club at PHUSE Conference Budapest 2012

Study on Good Programming Practices in Health and Life Sciences

Through this first study on Good Programming Practice in health and life sciences, we are looking to understand

- Do you as Programmers or organisations use Good Programming Practices?

If so-

- **Is this use mandatory?**
- **Which guidelines do you use?**
- **Which do you think are important?**

- This study has two components. The first is to register and provide information on the Good Programming Practices followed by your organization, and the second is to participate in the SAS code review sub-study. The registration will establish a platform from which we can launch additional sub-studies to collect data on areas of interest based on input from study participants. The results of the study will be kept strictly confidential but will be presented in summary form at conferences, on discussion boards, and the Good Programming Practice Advance Hub pages. We look forward to opening a robust dialogue on Good Programming Practices in health and life sciences, so please join us on this exciting initiative.

Click on this link for more information about the study.

How can you contribute?

1. Provide your comments to workinggroups@phuse.global for existing pages
2. Get involved with the team. Contact workinggroups@phuse.global

---

Bibliography

Some useful further reading on Good Programming Practice

Kirk Lafler Shares 25 Coding Techniques! 25 Best Practice Coding Techniques for SAS Users
Programming Guidelines (Lewin, L)

Good Programming Practices in SAS (Ford, J 2009)

Good Programming Practice in R (Maechler, 2004)

What Makes a Good Program? (Grundy, D 2010)

Macro Programming Best Practices:
Programming for Job Security Revisited

Include Debugging Code in Your Programs by Don Henderson

Writing Test Aware Programs

The Joel Test: 12 Steps to Better Code

Getting Started With a New Project

When starting work on any new study, it is important to familiarize yourself with the study. Review the study documents and try to understand the following:

- The objectives of the study.
- How many patients will be enrolled, randomized and treated.
- Schedule of events, i.e. screening, run-in, treatment periods, washouts, how many treatments and when they are taken.
- What is the primary endpoint and how, when and where is this data collected.
- Timelines for the trial, when is the database lock, when should the top line results be ready, and when should all the reporting be finalized.
- The current status of the project.

Study documents include:

- Protocol (CSP) - study outline and statistical sections are usually of relevance.
- Case Report Form (CRF, annotated CRF) - aCRFs are annotated with the data set name and variable name, - to understand where the data comes from and how it was collected and where it is stored.
- Statistical Analysis Plan (SAP) – to see what data is reported and how
- Analysis Datasets (ADS) specifications – describes which derived datasets should be created and what will be stored within them, including detailed definitions of endpoints. Used for ADS programming and validation.
- Table shells – used for tables, listings and graphs (TLG) programming or validation.
- Publications, if available (to check against already available results).
- Previous Clinical Study Reports (CSR), if available (to check against already available results).

Before you start programming, it is important that you familiarize yourself with all the relevant company IT systems, standards, SOPs and Guidelines on both programming and program validation. All these should be adhered to.

- Familiarize yourself with the system you are working on.
- Check for company specific programming standards.
- Check for study– and project specific standards.
- Check for industry standards like CDISC which are to be applied or can be applied.
- Check if a similar project/study has been worked on, i.e. check if available SAS code can be reused.
- Check for project-independent macros that can be applied.

Now that you are ready to start programming, keep in mind some basic standards:

- Flow of data from raw data tabulated data sets analysis data sets outputs.
- Do not derive anything in more than one place.
- All derivations should be implemented on ADS-level not during output programming.
- Structure your program to read in all external data at the top, do the processing, then produce any outputs or permanent analysis datasets.
- Keep in mind that you or somebody else might need to change your program in the future.
- Keep in mind that somebody will validate your program.
- Add comments as you are programming / do not plan to do that afterwards.
- Avoid data driven programming.
- Try to simplify your code to make it more readable and easy to perform source code review (e.g. do not make too many derivations in one data step, consider creating multiple data steps).

Language

The language used in programming code and within headers and comments is English.

Program Header

A standard header should be used for every program. The purpose of the header is to identify the program and provide documentation including revision history. It provides the necessary information to a code reviewer to identify and understand the program and its development life cycle. Standardizing the header will allow the information contained in the header to be leveraged programmatically for things such as auditing, project documentation, macro and dataset use tracking, consistency checking, and revision history reporting. The elements included in a header will vary from organization to organization but below is a discussion of some of the most common elements.

Required elements The following should be included in all program headers:

- Identification of the project of which the program is a part.
- Program name.
- Author identification which should be human readable and unique.
- Short description of program purpose.
- List of macros used in the program.
- Date program was first put into production, was finalized, or first passed validation.
- This date will be chosen based on the operational procedures used within the company /organization creating the program. The date should indicate the first date when the program was released for final use.
- Revision history.
- This is discussed further below.

Recommended elements The following are not required but are highly recommended in all program headers:

- The date on which program development started.
- All outputs generated by the program, including both file creation and modification.
- External files used such as datasets or databases that are used as data inputs to the program or macros used.
- Platform and operating system which the program was developed to run in.
- Software/programming language and version which the program was programmed in.

Revision History

The revision history section is critical to document the revisions made to the program once it is put into production. A well designed revision history section should include the author of the change, date of release of the change, a short description of the change. Revision history may also include a version number for changes which can be used as a reference in the code.

Comments

Comments are important to help anyone reviewing, modifying or using a program to be able to quickly understand the code. All major data or proc steps should be commented, especially data specific and complex code. Ideally comments should be comprehensive, and should describe the rationale and not simply the action. For example, instead of simply typing "Access demography data", describe which data elements you are accessing and why they are needed, for example, "Bringing in DM to get gender and age and subset to include only the intent to treat population". Comments can also include links to external documentation (requirement specifications, design documents. The programs can also be split up into sections by creating a different type of comments, e.g. many rows with asterisks. This helps to structure the program and make it easier for others to see an overview of the program.

Naming Conventions

All organizations should have standard naming conventions. Program naming conventions should make it possible to identify groups of related programs such as adverse events tables. Dataset and variable names should describe as best as possible their content. Temporary datasets and variables should be named consistently in a way that makes their purpose and role in the program clear. Where possible organisations should use industry level standards such as Clinical Data Interchange Standards Consortium (CDISC) standards for permanaent datasets and variables. This aids sharing of programs across companies and facilitates development of standard code. Space characters should be avoided in variable, dataset and output file names.

Coding Conventions

In order to be efficient and streamline the sharing of program code between programmers, with regulatory agencies, and with external partners or vendors, it is vital for code structure to follow standard conventions. SAS code which follows these conventions is much easier to read, modify, maintain, and correct. These conventions are divided into those which should be considered as required, and those which are merely recommendations to be followed as applicable.

Required conventions

- Do not overwrite existing datasets
- Each organization may have its own standards for using case within programming code but use of all uppercase should be avoided
- Separate data steps and procedures with at least one blank line
- 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
- Split data steps into logical parts
- Put each statement on a separate line
- Left justify global statements and data and procedure statements and their corresponding run and quit statements
- Indent statements belonging to a level by 2 to 5 columns (use the same number of spaces throughout the program), i.e. every nesting level should be visibly indented from the previous level.
- Do not use tabs for indentation because they will display differently depending on the platform and text editor being used, use blanks instead
- For do loops place the end statement in the same position as the do statement so that they can be easily matched
- 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.

Recommended conventions

- Perform only one task per module or macro
- Use logical groupings to separate code into blocks
- Double space between sections
- Group similar statements together
- Define new variables with the attrib statement in order to ensure that the variable properties such as length, format, and label are correct instead of allowing them to be implicitly determined by the circumstances in which they are initialized in the code

Log File Checking

As part of development and validation practices, it is often mandated that the log file generated is checked to ensure that the program has executed in the correct intention. Many companies may have their own automatic log file checking utilities to aid in this, and there are many examples of such tools in widely available papers. "ERROR" and "WARNING" in logs should normally be avoided. There are sometime exceptions to this, such as warnings that are output from statistical models that do not have enough data. Ordinarily, any warnings that are deemed acceptable are to be documented. There are also some specific "NOTE"s that can indicate a problem. The common "NOTE"s that should normally be avoided include those relating to "repeats", "more than one", "uninitialized" and "referenced".

Also, any user defined checks that have been added, such as from defensive programming, should be checked for in the log and followed up on. A company-specific naming convention for user defined checks can aid in this, so as the specific string can be searched for within the log. Examples of such conventions include "ISSUE:", "USER:", and "ALERT:". Avoid the use of user-generated errors and warnings labeled "NOTE:", "WARNING:" or "ERROR:", as these may make it difficult to find genuine problems when searching the log.

Portability

Most organizations are now working across multiple platforms, commonly combining Windows and Unix environments. There can be many occasions where code will work on one platform and not on another. Portability is more than just working across multiplatform environments, it is also about making programs easier to be used across projects. Below are some suggestions to address some of the most common impediments to portability.

- Use rounding in newly created variables (if applicable) in order to avoid different results e.g. from 64 bit operating systems to 32 bit systems. (However give careful consideration to doing this and round at the limit of precision as otherwise it may affect results. Where rounding is only required for presenting results, do so after calculations and derivations are completed.)
- Avoid explicitly defining file paths in libname, filename, and %include statements requiring platform specific syntax such as forward slash or back slash.
- Avoid the use of X commands to execute statements directly on the operating system.

Hard Coding

Hardcoding is the modification of the value of an item of source data within program code. Hardcoding should be avoided whenever possible in final code, and changes to source data should be done in data entry or capture systems which give better compliance to regulations such as FDA 21CFR11. Hardcoding may be done temporarily in order to get a program to run due to dirty data or correct for database inconsistencies. Permanent hardcoding to fix incorrect data values in a final database is strongly discouraged, but if it is unavoidable then it must be approved following a standard process (usually an SOP) and clearly documented using standard comments and PUT statements to the log to show what has been hard coded.

Defensive Programming

Defensive programming is an approach to programming intended to anticipate future changes of the data that might influence the coding algorithms. Ideally programs should be written in such a way that they will continue to work correctly in case of new or unexpected data values which did not exist at the time the code was developed. Analysis dataset and table programs are often developed in the early stages of a project or even when the only available data is test data. In these situations the data often does not contain all possible values of data points such as visits or time points, race values, and questionnaire responses, but the program must be able to handle those values when they do become present in the data at a later point.

Additional Pages			
Good Programming Practice Project Team	Good Programming Practice – Links to Useful Internet Pages for Pharmaceutical Programming	Good Programming Practice - Standard Program Process Supporting Documentation	The Metadata Lifecycle
Good Programming Practice	Overview Of Common SAS Macro Functions	Study on Good Programming Practices in Health and Life Sciences	Write Standard Programs Your People Will Use And Love!!

Resources
PharmaSUG 2017 - Paper IB02 Good Programming Practices at Every Level