

Good Programming Practice - Standard Program Process Supporting Documentation



Introduction

Documentation is always a killer for programmers; they don't like it although this is necessary in software development type of work. If they did, they would be Medical Writers or would submit even more papers to [PHUSE](#). Programmers like data crunching and problem solving and are not naturally inclined to the sequential and documented process of software development.

Standard program development involves a whole chain of items that are time-consuming to produce (for the first release at least) and involves some documentation at every stage. The key here is to have processes and templates promoting documentation fit for purpose at all levels. Too much and too detailed documentation will be out-dated quickly, is cumbersome to maintain and often does not fulfil its purpose (people reading them...hopefully like this paper). When speaking about fit for purpose, the 5 purposes here are:

1. Develop
2. Test
3. Use
4. Debug
5. Maintain

The following sections are detailing some good documentation practice and tricks keeping in mind the purpose they each serve:

- Requirements
- Code comments
- Program documentation and User Guide
- Test plan / Test cases
- Test report
- Other items such as standard program catalogue, test data and folder structure & environments

The process and documentation practices must make each of the 5 purposes fast and easy to address

Requirements

Section content first added by: Jean-Marc Ferran

- Support coding and validation
- They describe what the program must do
- Must be unambiguous and testable
- Must be easy to update

Standard program development usually follows a simplified [V-model](#). One explanation is that statistical macros are simpler than software. User Requirement Specifications (URS), Functional Design Specifications (FDS) and Technical Design Specifications (TDS) are usually concatenated into 1 set of requirements. TDS can be detailed or not or be directly supported by company's Standard Programming Practices for the general ones. In order to keep creativity floating, it could be beneficial to keep them loose and see what the developers can come up with. This can always be adjusted then through code review.

The source for the requirements is usually the high-level derivation and output standards document produced by the standardisation groups. Those high-level specs can be seen as Business Requirements in Software Development/V-model terms.

While URS and FDS describe what is expected to be produced in terms of result and functionalities, TDS describe how this will be technically met and can be addressed directly in the code.

Writing requirements that are complete, structured and testable will optimize test plan and test cases writing. Requirements author must always have the testing in mind when writing the requirements.

It can be beneficial to split your requirements in different sections such as for example:

- Functionalities & Selections
- Statistics & Computations
- Exceptions
- Layout and Display

A pre-requisite section is also necessary to have prior to this listing for example:

- Which version of the database standards is supported
- Which software version the program is developed for

Requirements can be written using a logical numbering following requirements sections (01, 02, 03) and that leaves space for adding new requirements in later version or even during the process (01-010, 01-020,...,02-010, etc.). It is not uncommon to realise that a requirement is missing or inadequate during coding or even testing phase.

Code Comments

Section content first added by: Jean-Marc Ferran

Code is

- the technical solution addressing the Requirements
- Must be easy to use and debug
- Must be easy to maintain

First of all, code comments must reflect and separate the overall structure of the code. A program header must also details without too much details (space reasons) program parameters.

Code comments can be numbered in a logical manner too (10.10, 10.20, etc.) and related documentation can use same numbering so reader who wants to get familiar with overall logic and structure of the code can then use numbering to look up the related piece of code. Code comments may also be sufficient to document program design if well written.

It can also be neat when feasible to add requirements numbers to the comments code where they are met in the implementation.

Comments must explain the rational of using a given macro or plain piece of code (Update a macro variable/load a given style/derive a variable /Filter on certain values/etc.) rather than being descriptive (we can all see that some sorting is being done on a call of proc sort) and guide the user or the developer who needs to use/debug or update/maintain the standard program.

Complex pieces of code such as use of regular expressions often deserve descriptive comments on top of explanations of the rational.

Standard program code must follow sponsor's Standard Programming Practices and code must be both reviewed in that respect but also to check that technical solutions used by the developer (that are not covered by the Standard Programming Practices) are adequate. This process is often the opportunity to share knowledge and best practices.

Program Documentation & User Guide

Section content first added by: Jean-Marc Ferran

- Support Maintenance
- Support Use & Debug

Most programmers do not like writing program documentation or user guides and also do not like reading them unless forced too.

Having Program Documentation is a business decision since code comments may also be sufficient to document program design if well written.

Program documentation documents are rather technical and would have the aim to support another programmer updating the standard program or a user who needs to debug a particular use of it and understand how all program pieces (input parameters, global variables, local variables, metadata and intermediate datasets) are linked together.

Program documentation can contain only a roadmap linking the different piece of code with input and output to each other. This is usually difficult to see when reading the program code and it does add value rather than just being descriptive. Program documentation must communicate what is central and subtle that cannot be reengineered from the source code and document rationale for important design decisions.

A second document, User Guide, would simply help the user using the program quickly. This document should be as short as possible and focus on the essential (pre-requisites, input parameters) and come with full examples that the user can simply copy and paste, modify and run.

Such documents can be supported by Wiki type of technology or hyper-linked html documents instead of stand-alone pdf/word file.

User guide can also be used for posting along the way found bugs and work-arounds till they are fixed in next version or adding an example addressing a special case that was found after release or caused queries.

User guides must contain “ready to copy and paste” call examples the programmers can just get started with. It is even better if those code examples can use test data available to all programmers so any one can run and modify such call examples before trying it out on real data.

Test Plan/Test Cases

Section content first added by: Jean-Marc Ferran

- Must be unambiguous for tester to execute
- Must reflect that all requirements are tested
- Test cases must be reproducible

Programmers can have first shot on Test plan, test cases and scripts and a reviewer can review and challenge test plan according to Requirements and Code. The programmer of a given component knows it inside-out and have an idea of which corners of the code or which dependency of parameters data is of high importance to test. A separate independent code review is also essential and must be performed by a third person.

Test plan can follow same structure than Requirements and test cases can be written using a logical numbering following requirements structure and that also leaves space for adding new test cases. Every requirement must be addressed by at least one test case. Traceability matrix will help checking this and documenting it.

Test Report

Section content first added by: Jean-Marc Ferran

- Document execution of test plan and possible deviations and actions.
- Must be easy to review

Test report usually uses the test plan as a template where each test case is flagged as passed or not passed.

Having one section for deviations and actions (vs. several spread across sections) enables to get a quick overview of the status of testing.

Test Data

Section content first added by: Jean-Marc Ferran

A good set of test data is important (Different TAs/Clinical Projects, different trial design, parallel, dose-escalation, cross-over, compound repository, different data size for testing performance and program specific test data to test exceptions, e.g. testing robustness of a Subject disposition table on a trial with no screening failures) but doesn't replace the real set-up of a trial where your standards and options are regularly challenged by end-users and outputs consumers.

Standard Program Catalogue

Section content first added by: Jean-Marc Ferran

When writing a new standard program or looking for a standard program to use for addressing an established standard, it is important to know quickly what is available. A standard program catalogue including both linking to output and derivation standards and dependency appendix of the standard program library components will enable to find the needed program or macro.

While your suite of standard program is growing, a list of dependencies between macros will help assessing quickly the impact of update of each component.

Folder Structure & Environments

Section content first added by: Jean-Marc Ferran

Finally, it is also important to have several environments with similar folder structures to support Development, Testing and use in Production of standard programs from standard program libraries.