PhUSE 2011

Paper AD05

Writing Standard Programs your People will Use and Love!

Jean-Marc Ferran
Consultant & Owner at Qualiance ApS
Copenhagen, Denmark

Abstract

Standard program implementation is an interesting exercise that requires software development insight, good code design adapted to company existing standards and processes and also an implementation methodology that will facilitate their use and most important end-user's adoption.

Although there is always a learning curve when using a new tool or program for the first time, it can be difficult to convince programmers to use a standard program if (they feel) they would have spent less time coding it themselves. Senior programmers often have their own undocumented and unvalidated although knowledgeful tool box that they develop and improve over the years and could probably do the job. There is nothing harder than selling programs to programmers and this is what standard programs are going up against in organisations.

This paper discusses different implementation strategies and important considerations regarding code design and documentation that will increase robustness, user-friendliness, programmers' quick adoption and program release success.

Contents

Introduction

Industry standards implementation is reaching a certain state of maturity and many companies have engaged on the path of developing suite of standard programs to speed up the process of producing derived datasets and outputs but how to get the benefits quickly and keep work challenging and rewarding for staff in the long run?

At McDonalds®**,** one of the most standardised companies in the world, the creativity is in the standardisation and development of associated processes. A Bigmac® tastes the same in every restaurant in the world. Working in a McDonalds restaurant requires discipline, strength and some essential cooking and customer service skills. Clinical Programmers expect more from their daily tasks and the way we are approaching standardisation and its implementation at a larger scale will be crucial in retaining the talents in our industry.

Standard programs must speed up production of standard outputs so there is more time for the challenging and specific non-standard analyses. Make sure the programmers still get to program if you don't want to loose them. Overall timelines are made assuming outputs comes fast with standard programs. If it doesn't in practice because they are difficult to use or because they take too long to update, it means there will be even less time for the rest with obvious consequences on both quality and staff satisfaction.

This paper describes different implementation strategies and important points to take into consideration in terms of processes, code design, documentation and support and training. There is no one-size-fits-all and concepts have to be adapted to company sizes, culture and outsourcing strategies.

This paper is based on the author's experience working with companies of different sizes and cultures on such assignments and reflects his ideas and opinions. It has also been written with a review of the available literature on the subject.

This paper addresses standard program development from both an organisational and operational point of view.

Overall Picture

Standardisation is a rather new concept in clinical research in comparison with other industries. Banks or telecom companies have to align certain standards so bank transfers or text messages can be exchanged across institutions and providers. The last thing drug maker companies would like to do is sending their data to competitors. With the emerging of Global standards such as CDISC together with regulatory requirements to submit data in a common format and the increasing partnership level between CROs and Sponsor companies, the pharmaceutical industry and biometrics in particular are taking a whole different approach to their processes. Benefits and savings in using standards (global or sponsor driven) are also an obvious reason to streamline the way we collect, store and analyse clinical data. At the same time, regulatory requirements are increasing as much as pressure from the generics on the innovative industry, which requires to deliver more and more data and results in a shorter timeframe to remain competitive.

Standardisation is a drastic change and involves a number of functions such as Trial Management, Data Management, Biostatistics and Medical Writing as a minimum (Input from Medical Science or Health Economics is also relevant). All those departments now have to align their processes further, to align terminology and rethink the way they work together. This change can of course be seen as intent of limiting staff creativity by enforcing more rigid processes and it does not come through without some resistance.

Organisations run such changes through massive projects leading to implementation of data warehouse type of systems involving cross-functional working groups defining necessary Protocol, CRF, Database and output standards, IT folks building the systems integrating with existing ones and functional experts from the different line of business involved fitting the new business processes. Other organisations take a more iterative approach and develop their standards together with the tools and pieces of technology over a longer period through their clinical research activities and align processes along the way.

No matter which approach is adopted, the definitions and robustness of standards is key to reaching the goals of reducing KPIs and increasing overall quality.

This paper focuses on the development, use and maintenance of standard programs based on defined standards of databases and statistical outputs. A guestimate on how much sponsor companies wants to achieve in terms of standardisation of databases, outputs and associated standard programs is usually around 80% of all programming activities per studies. This means this 80% consists of repetitive tasks and Statistical Programmers and Statisticians should be spending their time and efforts on the remaining 20% consisting of trial specific, new and challenging tasks.

One important point (although probably obvious) is that programmers like to program (vs. point-and-click or filling-macro-parameters type of work, writing or reading large piece of documents) and they have been used to being fully in control of the whole flow from raw datasets, derived datasets to statistical outputs. Introducing new tools in that context is a massive change and involving and training programmers on their development and use is key to the success of user’s adoption and rapid learning curve in order to deliver the benefits of standardisation.

Organisation set-up and Stakeholder analysis

Although the need for standardisation and development of standard programs becomes now obvious in our organisations, development of standard programs do not directly belong to the critical path of life-science companies where submissions or important trial milestones often drag resources at critical times. Organisations often struggle assigning and keeping resources for the time-consuming task of development of standard programs and adapt their structure and strategy to accommodate this process and their pipeline of clinical tasks.

A number of stakeholders are involved who do not limit to the only sphere of biostatistics. It is important to understand who they are and what are their needs and desires in order to shape the strategy and define adequate processes with respect to standard program development.

Stakeholders	**Desires and Needs**
Outputs consumers*	• Get outputs faster and of highest quality • Want to be able to twist the standards every so often
Statisticians	• Get outputs faster and of highest quality • Need “plug-and-play” tools
Clinical Programmers	• Need tools they can use, understand and debug easily • Save some time on repetitive tasks so they can do what they like: ▪ Program! ▪ Data crunching ▪ Solving problems
Standard Program Developer	• Need well-defined process • Optimized documentation to write • Find innovative solutions to general problems • Do not spend most of their time supporting end-users so they can develop more standard programs
QA	• Make sure there is an adequate process for standard program development • The process is followed
Standardization group	• Their standards are used and supported by good tools
Management	• Cater all projects with less resources • Internal staff and Externals** can use standard programs with minimum training and support

*: Medical Writers, Medical & Science, Health-Economics, Pharmacovigilance, Regulatory, Marketing, etc.
**: Contractors, CROs, BPOs, etc.

It all usually starts where cross-functional standardisation groups are formed with the mandate to develop high-level specs including output mock-ups or brief description of derived variables and options. Such documents are used to get feedback and involve less technical but not less important functions in the process such as Medical Writers or Clinicians. These deliverables serve then as basis for development of more detailed and technical specifications (Requirements) that can be used for development and validation of associated standard programs.

From that point, organisations’ strategies will vary from one company to another according to their size, culture and overall outsourcing capability. Companies indeed try to balance internal and external resources involved in standard program development between dedicated support departments and clinical projects. Staff in clinical projects represent the end-users of the standard programs and are at the same time working on the critical path of drug development with limited time to spend on other activities.

Here are different organisation set-ups used in the industry where pros and cons are each time highlighted.

One department developing all standard programs for the rest of the organisation
Pros:
- Clinical Projects can focus on their trials.
- Consistency across all developed standard programs.

Cons:
- Huge training needs.
- Possible gap in Expectations vs. Final Product.

One department responsible for the corporate level and processes and each Clinical Project develops their own tools in addition
Pros:
- Global standards are addressed centrally
- The burden is shared across the organisation
- More people are also involved and can directly contribute

Cons:
- Lot of work will be needed to align each clinical project development.
- Clinical Projects may develop tools that could have benefited other Clinical Projects.
- Risk having unfinished or obsolete standard programs due to conflicting clinical timelines due to development dragging over a longer period.

Requirements written and reviewed in-house and development outsourced, final product delivered with review rounds in between.
Pros:
- Commit less internal resources from both dedicated support department and clinical project.
- Consistency across all developed standard programs

Cons:
- The review time can be much bigger than you planed if your outsourcing partner doesn't know well your company standards, processes and expected quality.
- Requirements need to be much more detailed.
- Final product may look like a totally new tool the internal staff does not adhere to.

One department doing it in collaboration with representatives from the different Clinical Projects where prototypes are being developed.
Pros:
- Better data standards adherence
- Enhanced clinical staff involvement, feedback and buy-in
- Standards supported by prototype are exposed to feedback from output consumers
- Best existing programming concepts and tricks can be dragged from the clinical projects
- Limited training needed late on

Cons:
- Some of your best resource can be pulled out from some important clinical trial tasks.
- Clinical programmers can struggle to find enough time to spend on prototypes.

- Or at the opposite, it can be difficult for the programmers to leave the "creativity" loop.
- And timelines can be harder to predict.

Implementation model, sequential vs. recursive

From the different organisation set-ups described in previous section, 2 pattern of development arise. Sequential and Recursive. The challenge remains to find the right balance between keeping enough internal resources assigned to the clinical projects and involve end-users in the development of such tools so project standards can be considered further and end-users adoption can be enhanced through out the process.

Standard program development must follow a software development life cycle at the opposite of custom/one-shot programs that will be used once in a study. It goes from

- Requirements (what the program must do)
- Code (technical implementation addressing requirements)
- Program documentation (Technical description of code aiming at facilitating maintenance)
- User guide (Technical documentation aiming at helping end-users using the program)
- Test plan (Test cases checking that all requirements are met)
- Test report (Documentation of test case execution)
- User acceptance test (A number of scenarios are tested formally or informally before release)
- To Release in production.

Both sequential and recursive approaches can be adapted to the V-model that is widely used in software development so development and validation remain compliant.

Sequential

Requirements -> Program code and documentation -> Test plan -> Test report -> User acceptance test -> Release in production

It is easy to outsource but relies on detailed requirements and strong programming guidelines and well established program design practices. There is always the risk creating a toolbox that will sound totally foreign to in-house programmers in clinical projects that they will have to learn like a new application vs. using their own suite of undocumented and unvalidated of programs.

When outsourcing this, make sure you have an in-house programmer assigned as reviewer who will follow the development and will be able to become "super-user" on the given component and can take ownership for the success of its release inside the organisation. Requirements and programming guidelines need to be more detailed and examples of in-house developed "state-of-the-art" standard programs to refer as gold examples.

Recursive

Draft requirements + Draft code prototype -> Use as-is within one or more projects -> Refinement of requirements -> Upgrade of prototype into robust and standardised program + doc -> Test plan -> Test report -> User acceptance test (anecdotic) -> Release in production

A prototype-based approach will aim at getting as much feedback as possible from most senior programmers and users before formal implementation. It will also help the users to learn the program and create a feeling of ownership. Naturally programmers will see some components they have spent time in improving as their "babies" and will be first advocate and support for using them in the future.

The challenge here can be to leave the "creative" loop and call a given prototype "final" (for now) and ready to be upgraded to a standard program.

Small wins early will come quicker. Prototyping can almost replace some of the analysis work that would lead to design requirements (more fun for a programmer than going on the black board). Following this, the requirements can be written retrospectively and the whole V-model followed step by step ensuring full compliance. During this process the prototype is elevated to some more robust code following internal guidelines regarding standard program practices including documentation. During the prototyping period, programs can be used in the projects as guinea pigs and double programmed for the time being. A given double-program can be reused across studies if needed in order to minimize validation time till it is fully validated and released as a standard program.

"Real" test for standard programmes is when programmers use it in real settings (trial reporting, submissions, publications). The second real test is when study team (Medical Writers, Medical & Science, etc.) reviews the produced outputs and check that they fulfil expectations from the defined and agreed standard. You can have surprises there too and if some modifications are needed, it is always better to get this feedback early on before finalising documentation and testing.

In a recursive type of approach involving prototyping, it is crucial that individuals are made responsible and have the mandate to get each given components into production. This is to avoid prototyping dragging for too long or being left aside due to important clinical deadlines.

Supporting process and Documentation

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 and test data

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

Requirements

- 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

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

- 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 reengineer from the source code and document rational 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

- 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 important 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

- 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.

Other items

Other items to consider are Test Data and Standard Program Catalogue to support development and use of standard programs.

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.

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.

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.

Programming consideration

We spoke a lot about organisation set-ups, implementation strategies and documentations but less about programming consideration. The code is of course the central item and its quality both in terms of design and in requirements-adherence is key to the success of the individual program releases.

Different segments are discussed here and the list below is not exhaustive. Companies usually have their Standard Programming Practices that they develop over time based on the learnings they make. The segments discussed here are important in order to address easier code usability and debugging.

Robustness, error message and debug mode

Standard programs first appear as black boxes to end-users and it is important that they give necessary feedback to end-users when they first use them.

To a certain extent, user's passed inputs (through macro input parameters, global variables or metadata datasets, etc.) to the program must be checked for erroneous parameterization including input data structure that would lead the program to fail/crash. When this is identified, code execution must be stop and a clear error message must be return in the log suggesting if possible to the user upon how to adjust the call (Ex: Population ITF doesn't exist, please select a population defined for this study). By doing so, the log remains clean of any messages that are code specific and that cannot be understood by a programmer who is not familiar with the details of the code.

Defensive programming is a must in this context.

A debug mode keeping the intermediate library and printing additional messages as notes will help the user fixing a call without in depth knowledge of the standard program code. Together with the debug mode, it must be possible to pass additional debugging system options that are often fixed in the standard programs.

Keeping only necessary variables in all intermediate datasets also enables easier maintenance and debug if needed.

If a certain level of robustness together with return of clear error messages and debug mode are implemented, level of support to the user can remain limited.

Modular approach but iteratively!

A natural way of designing your suite of macros is to use a modular approach and encapsulate components into larger ones. The advantage is that they can be reused across standard programs and even for addressing specific functionalities in trial-specific programs and save programming and testing effort.

Modular design is good for breaking down work into many programmers and avoid code redundancy but can also lead to highly complex programs to debug and maintain.

Such approach should be built up gradually within organisations focusing first on highest-level integrated standard programs and only obvious small specific functions generalised into standard macros first (access and import study specific formats and folders, set-up, close and reset output destination attributes, robustness functions, basic calculations macros, look-up functions to metadata, etc.). This will result in highest- and lowest-level standard macros (integrated into the highest level one) developed first.

Standard programs are bound to be updated along the way (See change/impact analysis section) and the process of encapsulating macros into larger macros handling more functionalities can be done through out the successive releases. Programs based on less complex components will be easier to understand, give feedback to and debug in the first place by other programmers. This initial phase would enable to get more programmers on board and get familiar with the whole process of standard program development, documentation and validation on simpler components before engaging in developing more complex ones.

Later on, the intermediate standard macros (producing all set of statistics for continuous variables, producing report for events data, imputing missing data according to many different rules to be selected for example) can be developed integrating the lower-level components and being themselves integrated back into the highest ones.

Ergonomy / Usability

It also important to agree on naming conventions for parameters and global variable that are intuitive and that will be used consistently across components. Set as many default value to parameters as possible and evaluate carefully whereas a parameter would be more suited to be set through a global macro variable (control of debug mode for example), an input parameter or sourced from some program-specific metadata (set of related input parameters for example).

Meta-programming

For maintenance purpose, it is neat to be able to keep the code as short as possible (besides using a modular approach and encapsulating code into macros) and keep it focused to the core and difficult aspect of the problem it is addressing.

By using both global and program-specific metadata (Derived datasets specifications or Robustness checks description for example) and storing as much needed information into look-up tables with pre-defined structure vs. being handled in the code directly, the code becomes much simpler. Lengthy “if-else” structures can be avoided and the code remains enjoyable to read and easier to go through.

It is also easier to change metadata than changing the code which leads to new versions and therefore going through the Software Development Life Cycle again.

"Some of the laziest programmer writes the best code!"

Training and support

It is also crucial to have a contact person for each component or a group of persons (usually seating in the support organisation) that can act as a hotline.

Training/Demo presentations also help at least when the first components are released to get the user familiar with the important usability and design considerations and go through a couple of examples.

The training and support set-up will very much depend on the size and culture of your organisation and your user's community.

Change / Impact Analysis

Along the way, standard programs will have to be updated for various reasons and depending on the context, different items will be impacted.

	Requirements	**Code**	**Program doc.**	**User guide**	**Test plan**	**Test report**
Change of data model	M	H	L	L	M	H
Bug	L	M	L	L	L	H
Additional functionalities	M	H	H	L	M	H
Lower-level component updated	L	M	L	L	L	H
New software version	L	H	L	L	L	H

H: High impact, M: Medium, L: Low

This Change / Impact analysis gives a feeling on how the different items can be impacted. Test report on the far right hand-side is always highly impacted since any change in the code will imply that all test cases will have to be rerun and test report will have to rewritten. At the opposite, Requirements, Test plan and Test cases are usually not very much affected if test cases are reproducible.

Unless there is a major new release and there is a combination of many changes listed in the left column, most items (except code and test report) are only impacted at a Low and Medium level. The first release is usually the most time-consuming one and if well structured, should enable a fast update of most items

Conclusion

Even though standardisation is coming late, it is still harder in our industry when there is more than 1 million different concepts compared to Banking where a bank transfer can be described within 10 fields.

Companies rush into full-on standardisation, buying new technologies and developing expensive tools while they haven't done much about it for years and could still get drugs on the market. The backbone of success is definition of standards and their alignment across departments while prototyping will give the opportunity to see how processes, structures and technologies will support the best way possible the flow from Protocol to Statistical Outputs. Best is enemy of good and intending to get such complex data involving different department functions into a streamlined workflow requires time and patience. A Small-wins-early approach would be more beneficial in the long run. And this applies too at the single level of standard program development involving mainly biostatistics where an iterative approach would enable to drag the know-how from your best programmers, get their buy-in and provide a rewarding work environment for them.

Another good example of prototyping comes from Google® that for both gmail® and google+® (to mention some of their more famous products) gave an account to all their employees for a period of time to get their feedback before releasing them to a larger audience. We always have clinical trials or submission supported by experienced programmers that we can use to give us feedback on our tools in real settings.

The goal is to gain efficiency and minimize repetitive programming and validation and not to develop software in the first place. Simpler components developed first and used as prototypes within clinical projects enables to get quick wins on the KPIs and valuable feedback first on both program design and defined standard program development process before embarking on the journey of developing extensive suite of complex, flexible, robust and documented standard programs.

Let's remember that we work in the innovative industry and innovation should be promoted through out our work processes. Programming must remain challenging, fun and rewarding for programmers and we need to continue providing an environment where they can contribute and develop.

Contacts

Your questions and comments are most welcome!

Jean-Marc Ferran
Consultant & Owner, Qualiance ApS
Guldbergsgade 25, st th
2200 Copenhagen, Denmark
Tel: +45 6016 0456
Email: JMF@qualiance.dk
LinkedIn: http://www.linkedin.com/in/jeanmarcferran