Write Standard Programs Your People Will Use And Love!!

Table of Contents

Write Standard Programs Your People Will Use And Love!

Standard program implementation is an interesting exercise that requires software development insight, good code design adapted to the company's existing standards and processes and also an implementation methodology that will facilitate their use and most importantly, end-users' 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, toolbox 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 that will increase programmers' quick adoption and program release success. Industry standards implementation is reaching a certain state of maturity, and many companies have engaged in the path of developing a 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?

Introduction

At McDonald's ®, one of the most standardised companies in the world, creativity is in the standardisation and development of associated processes. A Big Mac ® tastes the same in every restaurant in the world. Working in a McDonald's 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 talent 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 lose them. Overall timelines are made assuming outputs come 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 their pros and cons with respect to the use of support organisations vs. clinical projects, internal vs. external resources and the use of sequential vs. iterative models. There is no one-size-fits-all, and concepts have to be adapted to company sizes, culture and outsourcing strategies.

Organisation set-up & Stakeholder Analysis

Although the need for standardisation and development of standard programs has now become obvious in our organisations, the development of standard programs does 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 with assigning and keeping resources for the time-consuming task of developing standard programs and adapting their structure and strategy to accommodate this process and their pipeline of clinical tasks.

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

Stakeholders	Desires & 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
Standardisation 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 o Contractors, CRO's BPO's etc
Example	Example

It all usually starts where cross-functional standardisation groups are formed with the mandate to develop high-level specs, including output mock-ups or a brief description of derived variables and options. Such documents are used to get feedback and involve less technical but no less important functions in the process, such as Medical Writers or Clinicians. These deliverables serve as the 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 highlighted each time.

One department is 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 is responsible for the corporate level and processes, and each Clinical ~Project develops its 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:

- A 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 of 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 fewer internal resources from both the dedicated support department and the clinical project
- Consistency across all developed standard programs

Cons:

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

One department is 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 the prototype are exposed to feedback from output consumers
- The best existing programming concepts and tricks can be dragged from the clinical projects
- Limited training is needed late on

Cons:

- Some of your best resources can be pulled from some important clinical trial tasks.
- Clinical programmers can struggle to find enough time to spend on prototypes.
- Or, on the contrary, 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 organisational set-ups described in the previous section, two patterns 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 involving end-users in the development of such tools so project standards can be considered further and end-users' adoption can be enhanced throughout the process.

Standard program development must follow a software development life cycle, contrasting opposite to 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 use 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 it relies on detailed requirements, strong programming guidelines, and well-established program design practices. There is always the risk of 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 programs.

When outsourcing this, make sure you have an in-house programmer assigned as a reviewer who will follow the development and will be able to become a "super-user" on the given component and can take ownership of 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 should be provided for reference.

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 the 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 improving as their “babies” and will be the first advocates and supporters 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 blackboard). Following this, the requirements can be written retrospectively, and the whole V-model can be 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 minimise 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 the study team (Medical Writers, Medical & Science, etc.) reviews the produced outputs and checks 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 component into production. This is to avoid prototyping dragging for too long or being left aside due to important clinical deadlines.

Conclusion

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 the 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 the 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 you 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 ®, which, 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 submissions 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 minimise repetitive programming and validation, and not to develop software in the first place. Simpler components developed first and used as prototypes within clinical projects enable getting quick wins on the KPIs and valuable feedback first on both program design and a defined standard program development process before embarking on the journey of developing an 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 throughout 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.

Author: Jean-Marc Ferran