Validation and SAS® Programming: Benefits of Using The System Life Cycle Method

Russell Newhouse, Parke-Davis Pharmaceutical Research

Introduction

Formal program validation is something new to a lot of SAS programmers. Sure, we've always thoroughly tested our programs, but we normally wouldn't have created a well-detailed test plan before we started developing programs, nor would we have thoroughly documented all of the testing and checking that we did do. But thanks to changes in how the FDA and other drug regulatory agencies view computer systems (eg, programs), we need to start developing and testing our programs in a more highly structured, and highly documented, manner.

This need for a formal validation process led Parke-Davis to create the Validation Master Plan (VMP). The VMP fully embraces the System Life Cycle method of system development. System Life Cycle (SLC) methodology breaks program development down into many steps, with some documentation for each step. In general terms, those steps are:

1. collecting user requirements,
2. creating programming specifications,
3. creating a validation protocol,
4. system development and testing,
5. executing the validation protocol,
6. user acceptance,
7. system integration testing,
8. change control management, and finally
9. system retirement.

But the SLC method does more than just give you a lot of paper to file. It gives you a well-defined development methodology that offers a number of benefits.

This paper will discuss those benefits by comparing program development using the SLC method against how programs were typically developed at Parke-Davis. Not all of the SLC steps will be looked at, but we will look at those steps that seem to have the biggest benefits.

Requirements

The Old Method

What could be considered "requirements" were gathered in many different ways.

For such things as summary tables, a shell of how the table should look was quite often developed in WordPerfect, or by hand using pen and paper. Sometimes this shell would even contain some numbers to show how percentages should be calculated and numbers totalled. Usually, this shell was created by the Developer (the person responsible for the creation of the program) and not the User (the person, or persons, who asked for the program and will be using its results), based either on a conversation the Developer had with the User or on the Developer's hunch of what the User usually asks for.

For statistical analysis programs, the statistical analysis plan tended to be used as programming requirements. The analysis plan, however, just gave a general description of what was being done and would only give some details. It wouldn't always include specifics such as the model or options (or even which Procedure) to use.

The New Method

The User must now supply written requirements before programming can start. These requirements are an "English" description of what the program should do. This would include such things as describing any calculations that are needed using data collection terminology, instead of data base variable terminology ("study day is calculated as the date of the observation minus the date of the first dose of drug"), describing what the resulting summary table should include, identifying what denominator should be used to calculate percentages, or specifying what the analysis model should be.

Benefits

By having to create formal requirements, the User is forced to really think about and define what the question is that they want answered, and how it should be answered. It can also make the User think more about how to handle "strange" data.

The main thing requirements should at least limit, if not eliminate, is the "that's not quite what I want" syndrome, where the User sees the final output and decides that it doesn't quite address what they now see as the question of interest. This alone should reduce reprogramming, and reduce the total amount of programming time needed.

Programming Specifications

Old Method

True programming specifications were never really created under the old system. The Developer would just take the "requirements" and try to translate them into programming and data base terminology on the fly, while creating the requested program.

New Method

The Developer now creates programming specifications that take the requirements and translates them into programming and data base terminology ("SDOBS = OBSDATE - DAY1"). These specifications are then reviewed by the User to verify that the Developer is interpreting the requirements correctly. This is done before any coding is started by the Developer.

Benefits

Having the Developer create programming specifications should eliminate confusion about the requested program. If the User is doing a proper job of reviewing, the Developer shouldn't develop a program using the wrong variable or the wrong record, because the programming specifications which were approved by the User specify exactly which variable or record to use. Since the potential for confusion is eliminated before the Developer starts programming, it should reduce the need for reprogramming caused by misinterpretation of the User's needs.

Validation Protocol

Old Method

No formal test plan was developed before programming started. It was up to the Developer to check that the program was producing correct results before issuing results to the User. And it was up to the User to check the issued results for accuracy and consistency. So the level of checking done depended on how conscientious the Developer and the User were.

New Method

Each program needs a formal Validation Protocol, which should be written and approved before program development starts. The main purpose of the protocol is to describe the test plan and the test data that will be used to validate that the program meets all requirements. All of the tests need to be passed before the program can be considered to be validated. This testing includes checking that all data manipulations and calculations work properly, but does not include having to check that a SAS Procedure is performing properly. It is assumed that given the correct data, a Procedure is producing the correct results. We do not want to get that deep into testing the SAS system itself.

Benefits

The benefits of a Validation Protocol are twofold. The Validation Protocol lays out a more formal, and hopefully more thorough, test plan than previously done. It also produces a chance for the Developer and User to better understand what they want the program to do. So the Validation Protocol not only confirms that the program is designed to produce the desired results, but also aids in developing the program.

System Development and Testing

Old Method

Program development and testing was completely left up to the Developer. There were no programming standards and no testing standards. And quite often, program development was going on while the Developer was still figuring out what the program should do.

New Method

Programs are now developed using the requirements, the specifications, and the Validation Protocol as references. Programs are developed following formal programming guidelines, which are geared towards producing readable code. Finally, it is strongly emphasized that the Developer must thoroughly test the program and not rely on the Validator (the person responsible for independently executing the test plan described in the Validation Protocol) to find errors.

Benefits

Following programming guidelines produces code that is easier to maintain, both by the original programmer and by any other programmers. And using the requirements, specifications, and Validation Protocol as programming reference tools should reduce reprogramming, if not total development time.

Change Control Management

Old Method

The process to change a program that was considered to be “in production” was as haphazard

as the original development process. Usually a change was requested verbally or at best by a cryptic note from the User jotted down on the output. Very rarely was there any attempt to justify the need for a change that was beyond a simple bug fix.

New Method

Any changes to a validated program must follow a well-defined Change Control process. In this process, you document why a change should be made (is it a bug fix or a program enhancement), justify the need for the change, estimate the amount of time needed to make the change, and use this information to determine if there is a real need to make the requested change. Once the change is made, the program must be revalidated before it is put back into production.

Benefits

This is one case where the main benefit might just be the paper trail that is being created. It allows you to see the history of the program, and remember when and why a particular change was made.

This paper trail also gives you a chance to see if the other steps in the development process are being followed as closely as they should. If you are constantly having a number of "program enhancements", the User might not be doing a good enough job determining what their requirements are. And if you are having a large number of bugs, you might have reconsider how thorough your test plans are.

One additional thing that might happen is that the "threat" of the paperwork might make a User think twice about wanting a minor "enhancement".

Side Benefits of the Overall Process

Implementing a formal validation and development process seems to force a move towards two ideas that can change, and improve, how things are done. And that is a move towards standardization and creating reusable code. Sort of unintentional side benefits of the process.

Standardization

As people look at the amount of time, effort, and paperwork that is required (or appears to be required), they start to see the advantages in trying to standardize as much as possible. A large chunk of what is collected and reported in a drug study is actually quite similar from study to study, regardless of the drug or even the indication being studied. This "similarity" includes things like demographic information, vital signs, patient history, concurrent medications, safety information (adverse events), and laboratory data. Sure, some of the individual parameters being collected might vary, but the collection methods for the overall category are basically consistent. In other words, you may collect respiration rate in one type of study and not another, but vital signs as a whole tend to be collected in a similar manner. So why not come up with some consistent ways to collect, store, and report that data?

Traditionally, there has been a level of resistance to the idea of standardization. Some have felt that standardization limits their "scientific creativity" and their control over a project. They might agree that standardization may save some of the programmer's time and effort, but under the old development process, it didn't directly affect the amount of time they need to put into a project, so they didn't see any real advantages. However, under the System Life Cycle method of development, they are more directly involved in program development and validation, so they start to see that there are real advantages in having a

standard method and using a standard program which is validated once instead of over and over.

And a standardization effort doesn't have to be limited to just those parameters that can follow a company-wide standard. The one set of parameters that does vary greatly from indication to indication is efficacy parameters. What is collected to measure drug efficacy has to be based on what indication is being studied. So this can't be a one size fits all. But you can still try to standardize how you are measuring and reporting efficacy from study to study within an indication.

Reusable Code

The move towards standardization really triggers a similar move in program development. If you now have a standard way in which a parameter is to be reported, why have a separate program for each study to report that parameter? Why not simplify development and validation needs and create a general program that can be used by the majority of studies to report that parameter?

And as you think about it more, you realize that there is no need to limit this idea of creating "reusable code" to complete programs only. There are quite often a number of items, such as page numbering, which are done in programs in a similar manner regardless of the indication the program is being developed for. Why not turn that code into a standard macro (or a "module"), that only needs to be validated when it's developed? When you need to use the module in your program, you don't need to revalidate it, you just need to validate that it's being used how the requirements specify it should be used. Why keep reinventing, and revalidating, the wheel?

Adopting both of these "side benefits" should shorten overall development time and effort. Yes, initially the time and effort required will be the same as before, and will probably be more. It always seems to take longer than initially planned to develop general procedures and programs. But as you start to use them, you'll find that the amount of time required shortens. A lot of the work (including validation) will already be done. All the developer would have to do is determine which modules are appropriate to use, and develop a driver program to use them. The validation effort required for this new project then becomes just having to test the driver program. So spending some extra time early on can buy you a lot of saved time in the end.

Conclusion

Yes, the validation process is a lot of work. And yes, it can be intimidating. But it can add a lot of benefit if it is not seen as something that is being forced upon us, but is something that can be used as a tool for improving how you do business.

If you have any questions or comments, you may contact the author by writing to:
Russell Newhouse
Parke-Davis Pharmaceutical Research
2800 Plymouth Road
Ann Arbor, MI 48105

or by e-mail at: newhour@aa.wl.com