Documentation and Accountability: Why Your SAS Programming Projects Can Benefit from Implementing a System Development Life Cycle (SDLC)

Jennifer Fulton
Biostatistical Programmer
Westat, Inc.

Stephen Black
Section Manager, Biostatistical Programming
Westat, Inc.

Introduction

Regardless of the industry in which a set of SAS programs is written and utilized, the process by which the programs are designed, developed, tested, implemented, and retired tends to be cyclical. For each new project the parameters and scope of what is to be accomplished with the programs must be explored, the programs are written and checked, and ultimately archived upon completion of the project. In some cases, the project and its SAS programs must be revisited and rerun with updates, so some sort of maintenance plan is needed to facilitate retrieval, updates, and re-archival of the programs with documentation of the changes. When a SAS programming project fits this cyclical paradigm, the organization, documentation, accountability, and maintenance can be substantially improved with the implementation of a System Development Life Cycle, or SDLC.

The Phases of the SDLC

Our company has adapted an SDLC based on a number of system development resources and texts. SDLC is most often applied to the development of large software systems, . but the idea can be extended to SAS programming projects. An SDLC breaks down a project into phases, with each phase requiring specific tasks and sign-offs by appropriate personnel indicating that the tasks have been completed and reviewed. The exact naming and components of the phases can vary and should be adapted to your company's needs. This is the approach that our company has taken:
Phase 0 – Validation Plan
Phase 1 – Requirements
Phase 2 – Design and Development
Phase 3 – Testing
Phase 4 – Implementation and Acceptance
Phase 5 – User and Training Guide
Phase 6 – Maintenance
Phase 7 – Retirement

These phases were defined generally in a company-wide SOP to allow implementation for different circumstances across departments. Our SAS programming department has recognized that for each project, the set of SAS programs is created, tested, implemented, maintained, and retired in a cyclical pattern similar to the development of large software systems. Hence, the SDLC can be applied specifically to SAS programming projects by adapting the requirements for each phase. For example, here is the approach we have taken:

Phase 0 – Validation Plan

The "Validation Plan" describes the background and scope of the SAS programming involved in accomplishing the project goals. It includes a section for the roles and responsibilities of the personnel anticipated to be working on the project, a section describing the documentation that will be required to consider the entire set of SAS programs truly "validated" at the end of the project, and sections describing plans for security and maintenance of the SAS programs. Some sections may be the same or very similar for all projects.

Phase 1 – Requirements

Our projects always include a statistical analysis of project data, and SAS programs analyze the data and summarize the results in tables, listings, and graphs. So the Requirements phase consists of three items: a statistical analysis plan (SAP), mock-ups of the anticipated output to give programmers and clients guidance on the format in which results will be presented, and a list of programs including program names, table numbers, and titles which can be linked to the SAP and mock-ups. Each of these items must be finalized and approved by appropriate management (such as the client or internal management) before programming can begin.

Phase 2 – Design and Development
This is the phase during which actual writing or "development" of the SAS programs takes place. Each program "developer" is responsible for properly testing the program and verifying the results, and will be held accountable with a sign-off page attesting that the appropriate procedures were followed.

Phase 3 – Testing
During this phase, a programmer who has had NO involvement in the development of the program performs his/her own independent checks and works out any discrepancies with the developer until both are satisfied that the program is functioning correctly and the results are complete and accurate. The tester also is held accountable with a sign-off page for the testing phase.

Phase 4 – Implementation and Acceptance
At this point, all SAS programs for a project have passed the Testing phase and the data has been finalized. The programs are run once more on the final data and results are verified once again. The Implementer signs off to indicate that the programs are final and the results are ready for delivery.

Phase 5 – User and Training Guide
If, for example, a client is to take possession of the programs and may one day run the programs on another system, a user guide may be provided with information such as the SAS version to use, the necessary directory structure for macros to function properly, and the order in which programs should be run.

Phase 6 – Maintenance
SAS programs for a project may need to be updated as issues are found during testing or implementation, or even after the initial delivery to the client. Issues may include unexpected data discrepancies, like missing values, or even changes in the scope of the project, such as format changes or the addition of several tables. A maintenance plan should facilitate accurate tracking of changes, including who made the changes, when, and why.

Phase 7 – Retirement
Retirement is the final phase of the SDLC, or the end of the cycle for the set of SAS programs. A plan should be defined and followed consistently from project to project for the archival and storage of the system at the end of each project. This should include appropriate documentation with all signatures, program files, format catalogs, program logs, and output files, as well as any external macros called by the programs.

The Traceability Matrix
"Traceability Matrix" is a high-level document containing an overview of the SDLC process. It allows each component defined in the Requirements phase to be followed or traced through the Development, Testing, and Implementation phases by using appropriate naming techniques and documentation. A "matrix" such as a spreadsheet or database can summarize this information in one place. For example, assume Table 1 is a summary of the first piece of information in the SAP, and a mock-up of Table 1 will be created with "Table 1" in the title. A list of all tables to be programmed for the project is created, and a program name is assigned to Table 1. An auditor should be able to identify the name of the program associated with Table 1 in the traceability matrix, along with the name of the developer, the date of development sign-off, and other pertinent information. The auditor can then trace the program for Table 1 through the Testing phase including the name of the tester, issues that were worked out during testing, and the date of testing sign-off. Finally, the auditor can trace the same program into the implementation phase and see the name of the programmer who ran and tested the program on final data, any problems and how they were resolved and pertinent dates. Clearly, such a matrix puts a great deal of information in one place and can decrease the time it takes to identify where problems occurred in the course of a project.

What are the benefits of SDLC?
If your company operates in a highly regulated industry, such as our clinical trials work, which is under the close scrutiny of the FDA, an SDLC of some sort is necessary to meet the stringent requirements of accurate documentation and accountability. In any industry, a well-defined SDLC offers the potential for improved tracking of costs resources, the ability to spot employee training issues, and improved levels of organization.

The Validation Plan and Requirements phases ensure that the project is not approached in a haphazard manner. SAS programmers on the project know what is expected and what the scope of the project is from the beginning. The more specifically the requirements are defined, the more quickly and independently the programmers can work while maintaining consistency across the project. This also minimizes the time spent down the line trying to coordinate results when deadlines are imminent. Auditors and management also like to see that a plan was clearly defined from the beginning, and to be able to track why the plan was not followed, should this occur. From a statistical standpoint, stating hypotheses and other analysis plans in advance reduces bias, and is preferred over performing data-driven tests.

A clear delineation between the Design and Development phase and the Testing phase will almost always reduce the error rate in your programs and result in improved output overall. An independent programmer with a "fresh set of eyes" who is properly trained will catch minor problems that the developer might have missed. Such a structure is also an excellent training opportunity for both the developer and the tester. Here are some of the testing techniques we have required our programmers to follow:

1. <u>Give the tester a separate copy of the program</u> – Ideally, the tester should test the program in an environment identical to that in which it was developed, but separate from the location of the original program. This allows the tester to "play" with the program and data sets in order to do a thorough job of testing, while protecting the integrity of the original program. The original program should not change during the testing phase until the tester issues change requests. If the program is updated in response to change requests, the new version should be copied to the testing area so the tester can check the changes as well.

2. <u>Check program documentation</u> – A descriptive program header and detailed comments throughout the body of the code are more than just good programming practice. The tester should not have to come to the developer repeatedly with questions like "Why did you do this?" or "How can I verify this hard code?". The tester's job is easier and more efficient if the program is self-explanatory. This also means that if the program is revisited in 3 years when both the developer and tester are long gone, the next user has a clear explanation of all of the steps in the program.

3. <u>Check the log</u> – The tester should search for more than just the word "ERROR" in the program log. Other keywords in the log can be indicators that the program is not functioning properly. Some keywords we search for include: Error, Warning, Uninitialized, Invalid, Missing, W.D, Repeats, Outside, Unreferenced, Resolved.

4. <u>Look at the statistical analysis plan (SAP)</u> – As mentioned in the description of the "Requirements" phase, a statistical analysis plan is often prepared before programming begins. The tester should verify that the program produces exactly the output described in the SAP, both by comparing the true output to the mock output, and by reading the text of the SAP to confirm that correct formulas were used, along with rules for handling missing values, etc.

5. <u>Bring the output into the word processor</u> – The tester should look at the output in the format in which it will be delivered to the client or other recipient. We bring our .lst files into MS Word and run a macro to format them appropriately. The tester should do this with the test output, and perform a spell-check, check page numbers, margins and page breaks as affected by line size and page size, and anything else that has to do with the look of the final product.

6. <u>Does the output make sense?</u> – By this we mean do the titles clearly describe the content of the output and differentiate between similar tables such as subset analyses? Do footnotes explain any odd values or the rationale that went into the creation of the table that is not obvious to a reviewer? Do column and/or row headers accurately and clearly describe the contents of the table? The tester should have the same perspective as the final recipient, in that the output should be self-explanatory and not require a phone call to the program developer in order to decipher it.

7. Check consistency – The tester should not confine him/herself to the one program he/she is charged with checking. The tester should also compare the program output to other output for the project, to make sure the output is consistent in both format and content, where appropriate. If a data listing is provided to support a summary table, the tester should also crosscheck these two items to ensure that the results are indeed supportive.

8. Look at the SAS code – Some would argue that a truly independent check of the program would require that the tester attempt to re-generate the program results without ever looking at the program code, so as not to be influenced by the developer's code. This is a good approach, but at some point the tester should look through the actual program code carefully to check for hardcoding or other less dynamic code that may yield incorrect results if the program is run again on updated data, especially handling of missing values and conditional branching (e.g. if sex=1 then sextxt='Male'; else sextxt='Female').

9. Check the results in the output – Last but far from least, the tester must "check the numbers" to make sure the output not only looks good but is an accurate summary of the raw data. The tester should look at all numbers, including numbers in a footnote or column header, percentages (was the correct denominator used when calculating the percentages?), and total columns or rows. In other words, don't assume any numbers are automatically correct!

The User and Training Guide phase makes more sense in the applications development area, where systems are delivered for use by clients, whereas we typically deliver only output to our clients. But again, depending on who the ultimate recipient of the output and/or programs will be, some documentation of the type of system on which the programs were created and run may be useful. Such documentation also can be useful if the programs are revisited long after the initial developers have moved on to other things or even left the company. The User and Training Guide may be documented as a template with only minor updates for each project.

A good plan for Maintenance is invaluable for client or internal audits, or for simply tracking down the source of future problems. A documented plan for the Retirement phase will result in consistency and better organization so that no crucial items are misplaced, and will make it easier to retrieve items in the future if necessary.

Issues to think about when setting up your SDLC

1. What level of documentation do you require?
   Do you work in a regulated industry such that you may have to produce names and dates associated with each program and any changes made during the course of the project? Or do you simply hope an SDLC will better streamline the SAS programming portion of your projects? Depending on your response, you should decide how you would document the completion of the various phases of your SDLC. You can create templates with signature lines, which are printed, signed, and filed as appropriate. You can keep track of the phases and the associated personnel in a spreadsheet or database or you can design a more complicated documentation system that minimizes paperwork by saving details about each program electronically.

2. How will you define the phases of your SDLC?
   Each company, and possibly each department within the company, will need to decide which phases are applicable to the approach you take to a project, and what the requirements will be for completion of each phase. We recommend that the Requirements, Development, and Testing phases remain as the "meat" of any SDLC for SAS programs. But phases may be combined or eliminated until you create the custom-designed SDLC that meets your needs. A clear guideline should be written listing exactly what tasks are to be completed, and who is authorized to complete each task, in order to consider a phase completed. For example, is the Development phase considered completed for a given program when the developer signs a form indicating that he/she has completed coding and tested the program? Or can the developer simply contact the tester to say that the program is ready for the Testing phase, and document this with an entry for the program in a shared spreadsheet?

3. Training and Enforcement

Implementing an SDLC will not meet all of the needs we've told you about unless all of the personnel who will have responsibilities during the different phases have been trained on both their individual tasks and the whole picture into which those tasks will fit. Particularly if programmers have previously worked fairly independently, keeping few records of what tasks were done and when, implementing an SDLC can mean a drastic change in the way a project is approached. If the phases are clearly defined and guidelines and other forms of training are provided, the transition will be smoother and enforcement of the new methods will be possible. Speaking of enforcement, it can be tempting to "cheat" and skip steps, especially during a crunch time. For this reason using an SDLC will be even more successful if the documentation system you develop is easy to use and takes into account the various types of projects your company or department encounters.

Conclusion

Our company is required, under FDA regulations, to implement some sort of documentation system that provides accountability for the end results our SAS programs provide for a project. We also want a system that meets our own internal requirements for checks and balances, so that we can feel confident that we have done everything we can to provide accurate results in an efficient manner. And we want a system that will allow us to identify problem areas and improve training, if needed, so that our work continues to improve. Your company can reap some or all of these benefits as well when you design and implement your own System Development Life Cycle for your SAS programming projects.