The Basics of SAS Validation, a Primer on the Mechanics of Re-engineering Code and Documentation for Rapid Software Validation and Delivery

Edzel Cabanero, San Diego, CA

ABSTRACT

SAS software continues to be a great tool to bring about a full complement of information analysis solutions. Due in part to its flexibility and a cadre of self-taught practitioners, typical information systems software development life cycle practices are seldom implemented in the SAS world.

But like any IS concern, the SAS group is tasked with delivery dates. Quality is expected, and professionalism demanded. Yet, it is far more difficult to achieve these with SAS mainly because of the lack of know-how to organize work effectively.

INTRODUCTION

Consider this situation: As developers, we frequently encounter archives of existing code libraries. Often we are asked to maintain, reuse, or extend these libraries even as projects gain or lose favor with each change in management.

In the process, we make decisions as to the validity of certain code. We make informed guesses as to required updates, and we may wrestle with the specter of writing code which may already exist somewhere like needle in a haystack.

Even as we have to deal with legacy code, introducing process and discipline will go a long way to make sure that we ourselves don't exacerbate current problems as well as propagate bugs for future developers to discover.

The objective of this paper is to advance ideas whereby the reader's practices can be improved. It aims to provide the developer with the minimum outline sufficient to organize work to be able to ensure quality, professionalism, and guarantee delivery in SAS systems development.

DEFINING A PROCESS

We begin by defining a process:

a) create a stable environment
b) create a good team that communicates well
c) create standard techniques for development
d) establish simple validation methods
e) develop enough paper-trail to track progress (specifications and validation)
f) establish systems-level defenses to prevent clobbering
g) start over at step a)

The succeeding sections will define and clarify each step. (If there ever is a doubt about meaning, think simplicity as this is the author's one guiding concept when writing this paper.)

CREATE A STABLE ENVIRONMENT

This step is where you make sure that the environment does not prevent your developers from doing their work. This is the step where you create the necessary structure or foundation upon which all efforts can be built upon. A few examples of activities which belong here are:

a) choose your operating system carefully
b) write standard operating procedures (SOPs) and basic guidelines on programming design, use of constructs, validation, and change control
c) standardize directory structures
d) create derived datasets (no raw data file access for reporting)
e) use global formats similar in nature across projects (promote reuse)

CREATE A GOOD TEAM THAT COMMUNICATES WELL

If you have senior developers with leadership qualities, consider yourself lucky. You can rely on them to assist in building a team capable of high work output capacity. You can certainly help things along by keeping these in mind:

a) make a conscious effort to recruit talented developers
b) equalize skills or raise by training
c) get used to planning ahead (don't code down to the wire)
d) meet regularly to review hot-spots and general progress
e) share technical know-how
f) attempt code walk-throughs to share stylistic concerns you are trying to impart

CREATE STANDARD TECHNIQUES FOR DEVELOPMENT

The project leader approves the specifications and you are now ready to code. You wouldn't be here if you didn't already know what you are doing of course. *I still find that these issues* deserve a quick review to show how one can exert some discipline in the coding process (which we are bound to let go of every now and then):

a) are you working on approved specifications? Standard reporting packages may not require this
b) standardize the look of comments (via SOP)
c) comments are not the place to be a clown or write essays. Write enough comments to matter. If your comments are longer than the code, justify it or rewrite. Write comments with similar logic as the code; comments can be orphaned too!
d) read Code Complete by Steve McConnell
e) code to equal work produced by the team (if substantially different, it is hard to maintain.)

Here are some rules of thumb when re-engineering or updating code:

a) if the code can be extended without serious loss of readability, update and don't rewrite
b) if the code becomes a jumbled mess, think about rewriting rather than simple update
c) don't rewrite just because you think you can do it better
d) make standardization an end, efficiency and style the means
e) decide ahead of time not to introduce errors. This caution will make you a more careful coder
f) learn from your more senior developers

ESTABLISH SIMPLE VALIDATION METHODS

I've identified common SAS validation to be made up of four types: a) self-validation; b) simple exchange; and c) team validation; and d) code walk-throughs. The order determines the likelihood of occurrence, due in a large part to lack of established SOP's for validation or a commitment to quality.

There are three types of tests which are identified in the testing phase of the software development life cycle:

a) the unit test - single segments of independent code, can be a program
b) the function test - multiple units related, cohesiveness
c) systems test - multiple functions or the entire system across different platforms

The expectation here is that items a) through c) should be part of a test and validation plan. The quality of software produced often suffer if any of these are skipped due to scheduling concerns.

To get a handle on validation, you can take these actions: create a template document with at least 10 general areas to check; use this form to verify code and then re-verify after new updates are in place; you have to be able to trust the tester (train or hire a competent one); avoid the tendency to automate--intimate knowledge of program's internals will increase the problem-solving abilities of your group.

What does one look for? The author have encountered groups that practice zero tolerance--meaning no warnings, duplicates, or error messages in the SAS logs. Watch those duplicate values by verifying interim results. Verify the results of the code; don't worry about how it was designed (unless it is totally not in keeping with established coding practices). Use the specifications as your guide.

CREATE ENOUGH PAPER-TRAIL TO TRACK PROGRESS

As SAS developers, we typically will only see two documents—the specifications and then the validation form. As a minimum, these must be maintained and new segments added with each successive code update. The developer must not be allowed to sign off his own validation form for his code. An extra pair of eyes to double-check can help.

Guard against either extreme--not having enough and having too much documentation. An example of the former is not knowing what utilities you have in place, whether a certain piece of code exists, or not being able to find what you need when you need it. An example of the latter is when you go through a half-day process of retrieval and re-release for a one-line update.

ESTABLISH SYSTEMS-LEVEL PRACTICES TO PREVENT CLOBBERING

How often has a junior developer inadvertently deleted your code? What about multiple updates to the same code? Here are a few steps to ensure that you are able to protect your code in relative safety:

a) archive clean copy weekly, or daily if updates are being made daily
b) standardize backups
c) file documentation
d) establish read-write protection rules
e) educate

GAINING EXPERIENCE AND SUCCEED BY REPETITION

Ask anyone who has ever succeeded at something, this person will tell you that consistency and perseverance will win the day over luck and talent. It goes without saying that established procedures will only work if used over and over, with an eye towards learning from past experience.

CONCLUSION

If used properly, the process introduced by this paper will go a long way towards establishing a SAS development team capable of high output and growth. To reiterate, here again is the process:

a) create a stable environment
b) create a good team that communicates well
c) create standard techniques for development
d) establish simple validation methods
e) develop enough paper-trail to track progress (specifications and validation)
f) establish systems-level defenses to prevent clobbering
g) start over at step a)

Steps a) through e) defines a setup for simple validation. Steps c) through f) defines a process for re-engineering code and documentation. Steps a) through g) puts in place a simple-yet-effective method of implementing rapid software validation and delivery.

The author believes that a working process without bureaucracy empowers. The aim is to derive the greatest common benefit without hampering the developer's need to provide creative solutions.

That is why, despite the ambitious title, the author has basically only provided the two ends of a suspension bridge. He leaves it up to the reader to gain the knowledge to span the two ends, and thereby succeed despite himself or herself in the SAS world.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Edzel Cabanero
PO Box 927948
San Diego, CA 92192
Work Phone: 619 678-4128
Email: ed_cabanero@yahoo.com
Website: www.byte-symphony.com