

Paper PO-170
Lessons learned from Integrating SAS® Applications
C. Jenny Shiao
USDA-NASS, Washington, DC

Abstract

National Agricultural Statistics Services conducts Agricultural Resources Management Survey annually. The Survey is large and time we are allowed to work on is limited. Due to the demand in distributed database processing, increasing need of flexibility and productivity, interactive GUI features, and reducing printing cost, staffs from the business units and IT worked together to find ways to meet these challenges for this survey. The goal was to bridge the batch oriented mainframe processing and more recently developed interactive systems. Through several iterations of enhancements, the system life cycle development process of planning, design, implementation, and support guides us to realize this goal. One of the major applications the business process utilizes runs on various platforms, mainframe, UNIX, and Window, that involves Base SAS, SAS AF, SAS Connect, etc. This paper discussed how we came about the solution, how we were challenged with the system complexity, what we had learned to focus on through the system development life cycle process.

Keywords: BASE STAT FUNCTIONS

1. Introduction

National Agricultural Statistics Services conducts Agricultural Resources Management Survey (ARMS) since 1980's. Its process involves Base SAS and large reports. The process has been reliable, mature and stable. In recent years, a separate SAS AF system was developed, which provided business communities an interactive process with graphical user interfaces. IT was challenged to integrate the two processes so our users can utilize functions that work well in each of the two environments.

The integrated ARMS application was first implemented five years ago. The original vision was simply to build a connection between two large systems. Just imagine drawing a line between two circles. (See Figure 1.) Today, the integrated application utilizes various SAS products, including Base SAS, SAS/AF, SAS Connect on various platforms. The green color parts depict the legacy system. The blue color items depict the new system. The purple parts are the interfaces. (See Figure 2.) The process involves various data base management systems and transactions in addition to SAS datasets. For the past five years, the lessons we learned each year became the guiding post for implementing enhancements for the following year. At each of the iteration of the System Development Life Cycle (SDLC), we focus our resources on area that need enhancements in system/network/application, and strengthen our administrative procedures. With this, we were able to stop printing large reports and help users with their business needs more efficiently.

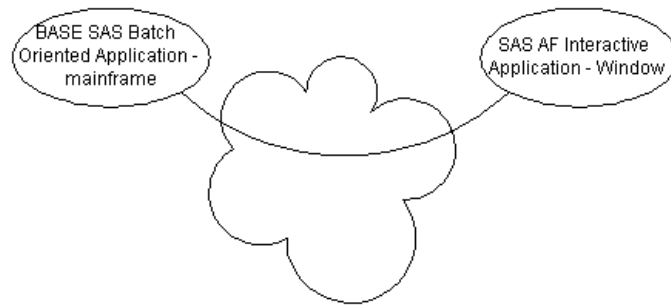


Figure 1. – Vision

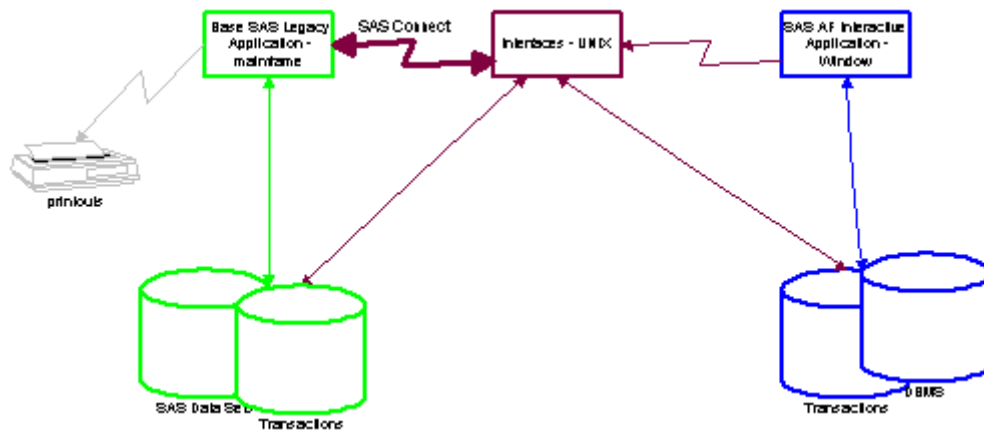


Figure 2. – Integrated Application

To start the SDLC process, we first wrote Software Requirement Specification (SRS) document. It established the baseline of the end to end processes. During the SDLC process, all issues we were concerned about were divided into seven categories. They are (1) Administrative, (2) Data, (3) System Environment, (4) Planning, (5) Process, (6) Software/Specification, and (7) User Coding/Meta Data. At the beginning of the each iteration, we analyzed the cases of concerns, implemented the solutions, and continued to track where and how problems occurred. The Standard Operating Procedures (SOP) provides useful instructions in handling data processing. At the latest iteration of the SDLC process, we finally earned a great deal of efficiency from all aspects of the application.

2. Software Development Life Cycles

The primary focus of the development, main cases of concerns and lessons learned, selected SAS coding examples are described in each of the SDLC iterations. We kept

track of number of cases of concerns and the improvement relative to the first Iteration. Due to the scope of the paper, I could not include all details.

2.1. Iteration 1

Primary Focus:

- Prototype the connectivity using SAS CONNECT across various platforms.
- Plan and setup per SRS and SOP for the quality and assurance of our system/network/application.
- Develop and put in production sub-systems with clearly defined calling parameters, return codes, exception handling, remote “**R****S****U****B****M****I****T**”, “**P****R****O****C****S****Q****L**” processes, DBMS interfaces, and SAS macro variables passing across various platforms. Enhance upon Text and Images sub-system capabilities.
- Strengthen the efficiency and correctness of User Coding and Meta Data.
- Users learn the interactive capabilities and the new operation procedures.

Main Cases of Concerns and Lessons Learned:

- The four categories that accounted for more errors than others area were User Coding and Meta Data, Specifications and Software, System Environment, and Data. The User Coding and Meta Data caused a lot of abends early on due the volume of data and changes in business logic. The Specification and Software was complicated because the application operates in a heterogeneous environment.
- SAS CONNECT was heavily incorporated in the computer operation along with other connectivity capabilities. It was difficult to diagnose a connectivity problem. It could be physical cable, network, firewall, system admin, space, memory, DBMS, system locking, access/rights, open client setup, ODBC, SAS CONNECT, etc. The list goes on. It’s imperative that the problem is identified ASAP and appropriate IT staffs get notified.
- We learned how to code SAS macro to pass macro variables among platforms. Watch for those ‘&’ for the symbolic replacement. Depending on what language the driver modules use and how they are implemented, you will need to test whether ‘&’ or ‘&&’ will be needed
- .
- Mainframe environment use large caps. UNIX and Windows are mostly case sensitive. It is a good practice to set up a standard for each of the platform. If lower case is selected as the standard, you may use ‘%**lowercase**’ to force the

entire execution statement to be in lower case as shown in the example.

- Many people write SAS codes in a sequential manner. To ensure the code conforms to a specific condition, a return code status should be validated. Use **sysrc** in the example to capture return code. It is good practice to establish a common set of standard return codes.

SAS Coding Examples:

```
/* A SAS MACRO syslputmacro that passes macro variables to the remote
server */

%macro syslputmacro(macvar,macval,remote=);
options nosource nonotes;
%let stra=%str(rsubmit &&remote ;options nosource;)
%nrstr(%let)
%str(&&macvar = &&macval;options nosource;endrsubmit);
&&stra ; options nonotes nosource;
%mend syslputmacro;

/* Pass macro variable Exec to the remote server ServerName */

%syslputmacro(Exec , &&Exec      , remote=ServerName );

/* Force an execute statement &&Exec... to be in lower case. */

%lowercase( &&Exec &&Job &&parm1 &&parm2 );

/* Catch the system return code sysrc from the previous SAS statements
*/

%sysrput retcode = &&sysrc ;
```

2.2. Iteration 2

Primary Focus:

- Focus on Specifications/Software, and User Coding/Meta Data areas so the application process is protected from un-expected conditions. For example, the application may need to wait for a period of time so various processes it depends on could be synchronized. Another example is to enforce the operational time so the risk of contention and locking be kept at the minimum.
- Again, improve the ability to plan and set up for quality and assurance for all categories. Test changes fully. Keep logs so we could trace back activities in order to resolve problems.

Main Cases of Concerns and Lessons Learned:

- The two categories that accounted for more errors than other areas were Data and System Environment. The two categories that had improved most since the first Iteration are Planning and User Coding/Meta Data areas.
- When problems occurred, there was not enough time to investigate the cause of the problems but to get users back on track. We learned to coordinate and control changes. We learned to be more proactive to the system space utilization and memory demand. Our users learned to follow the operating procedures, report problems through proper channels, and to ensure the process is complete successfully.

SAS Coding Examples:

```
/* A sleep function allows the system to wait. The following
sleep(#) statement put the system in the wait mode for a
defined number of '#' time units. Caution: '#' is the time
unit of the OS. Different platforms have different time units.
*/
```

2.3. Iteration 3

Primary Focus:

- Our system environment went through much needed consolidation. The goal was to make sure DBMS and SAS perform better in the new environment. We wanted to make sure patches and new releases were tested fully before they were put in production. We have converted some sub-systems to using SAS 9.1 on one of our platforms and kept others at SAS 8.2. As more data get added, we enforced the clean up process and the automatic system monitoring process. It was critical that resources were wisely used since large computing needs arose from many business units.

Main Cases of Concerns and Lessons Learned:

- System Environment and Data continued to account for more cases of concerns than other areas. The two categories had the least improvement were Administrative and Data. The category that showed higher improvement since 2004 was User Coding/Meta Data area.

2.4. Iteration 4

Primary Focus:

- Designed and implemented an important feature that enhanced the system

performance tremendously. We used data to determine whether or not to update, append, or delete DBMS transactions. This feature saved a lot of processing time and received high remarks from our users.

Main Cases of Concerns and Lessons Learned:

- The two categories that continued to account for more errors than others were System Environment and Data. The two categories need more improvement than other areas were Data and Process.

SAS Coding Examples:

```
/* It can be cost effective if the data is streamline before it
gets into the database management system. A simple SAS compare
proc statement gives you the differences between two datasets.
From the differences, you could decide upon the update, append, or
delete transactions. */
```

```
proc compare base=currentdata compare=oldedata
      out=result outnoequal noprint ; run;
data _null_ ;
  %if &sysinfo >= 64 %then
  %do;
    abort return 10 ;
  %end;
```

2.5. Iteration 5

Primary Focus:

- Strengthened the quality and assurance of the application especially in the security practice area.
- Coordinated and controlled changes in timing needs, network, security, operating system, and DBMS to ensure users were least affected and data integrity maintained.

Main Cases of Concerns and Lessons Learned:

- The two categories that continued to account for more errors than others area were System Environment and Data areas.
- The two categories needed more improvement were Administrative and Data areas. The category that showed large improvement since 2004 is Specifications/Software instead of User Coding/Meta Data. This was due to a late

change in User Coding caused many jobs to abends. Data and System Environment issues had also caused problems toward the end. We learned no matter how late or how small the changes are, it's imperative we test the changes and keep on guard until the final process is done.

3. Conclusions

The software development life cycle is an iterative process. At early iterations, the emphasis is on building a solid foundation. At latter iterations, the focus shifts toward quality and assurance.

During the first iteration of the software development life cycle, we spent a lot of energy getting the software and logic to work properly in our system/network environment. At the second iteration, many cases of concerns were in the system environment and data services area. At the third iteration, we were concerned about Administrative and Data areas but we also realized we had to improve the performance to make sure our administrative procedures and processes strengthened. At the fourth iteration, we finally realized some hard earned efficiency from all aspects of the system. At the fifth iteration, we were still challenged by operations in the system/network and data area; however, all categories of cases of concerns were more leveraged and users were much less affected.

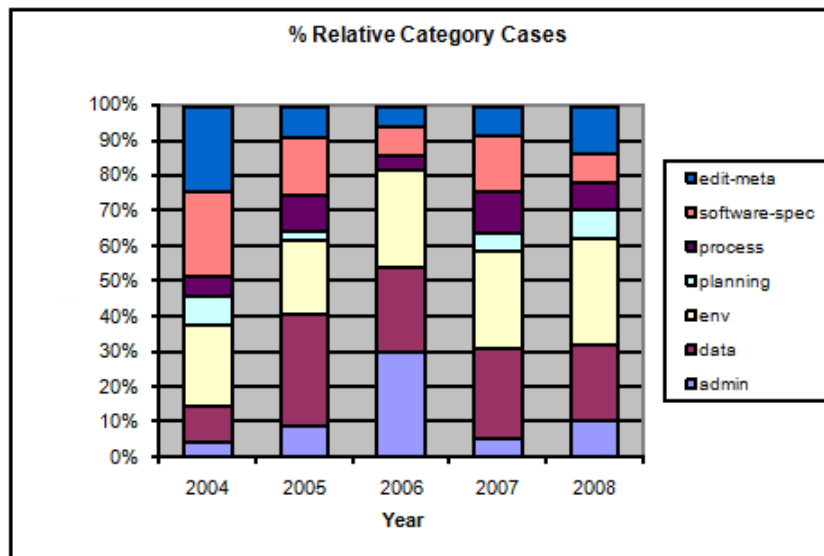


Figure 3.

In terms of the number of cases of concerns, we saw a dramatic drop from 2004 to 2008. We were challenged to improve the Administrative and Data areas. This indicates that we need constantly monitoring our systems. Though we still need to pay attention to Specification/Software, and User Coding/Meta Data areas, but the improvement in these

two areas rose 67% and 44% respectively. The lessons we learned and the efforts we put forth continue to refine our computing solutions.

Contact Information:

C. Jenny Shiao
NASS-USDA, Mail Stop 2031
14th and Independence Ave. SW
Washington, DC 20250
Tel: 202-720-8158
E-mail: jshiao@nass.usda.gov

Acknowledge:

I would like to acknowledge the NASS management for their support of the ARMS project and the team work from all team members. I would also like to thank the SAS tech support, and all internal and external IT support on behalf of the ARMS project.

Reference:

SAS Institute Inc. (1999) , *SAS OnlineDoc® Version 8.*, SAS Institute Inc., Cary, NC: SAS Institute Inc.

Software Engineering Institute, Carnegie Mellon, <http://www.sei.cmu.edu/publications/>

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.