Planning for a Smooth Transition to Release 6.06 of the SAS® System In Production Support or
"What Do You Mean, 'Not Transparent'!!"
Andrew T. Kuligowski, Nielsen Media Research

In a Production Support environment, our focus is to keep existing application systems up and running. As such, our excitement over the new features accompanying the release of Version 6.06 of the SAS System was tempered by the knowledge that a few compatibility issues had already been identified by the SAS Institute, and the fear that additional compatibility issues were still waiting to be discovered. This paper will describe the methodology used by the Production Support team at our site to examine new software releases, with special emphasis on our plans to look into Version 6.06 of the SAS System.

**STEP 1: PLANNING**

The first task, before the first line of code is run or even examined, is to assemble a list of all known compatibility issues between Versions 6.16 and 6.06 of the SAS System. The initial reference source is the Course Notes from the SAS Institute's "Making the Transition to Release 6.06 of the SAS System" class. While the primary emphasis of the class is to introduce the new features accompanying Version 6.06 of the SAS System, many chapters also include a list of known compatibility issues, categorized by topic. The next sources you should consider are your co-workers. Many of them are also probably looking into Version 6.06 of the SAS System for their own purposes; ask them to record any problems that they discover. In addition, if your area has a local SAS User Group, consider trading information with the other members. Most of them are undergoing the same conversion efforts that you are. The Florida Gulf Coast SAS User's Group, for example, is facilitating the effort by including a "Conversion Tips and Traps" column whenever possible in its bi-monthly newsletter.

The next step is to set up or recycle test beds for your existing applications. These test beds, or test environments, should simulate your production environment as much as possible. Real-world considerations must be taken into account, but differences need to be kept to a minimum. For example, DASD limits at many sites will necessitate reducing the size of your test database. It is important to make note of any difference between your test bed and your production environment for future reference. Once your site has officially converted, those differences should be your primary suspects if any problems develop.

The third step is to write a test plan for every application that utilizes the SAS System. The plan should be as descriptive as time allows. We plan on executing coverage tests, or complete production runs with normal volume and conditions, for every system in our area. Some routines are so flexible that it would be impossible to test every combination of options in a reasonable period of time. In these cases, it will be necessary to test a subset of combinations. This subset should consist of two categories. The first should be made up of the combinations run most often in production. The second should consist of those combinations of options which have caused problems in previous releases of the routine in question. (Often, these categories overlap.) This process can be facilitated by examining historical test plans. Ideally, test plans should be updated to reflect modifications to planned tests and results of all completed tests, then filed away for future reference upon completion. Do not forget that some routines are executed multiple times in a system, or are included in more than one application. These "features" must be considered when developing your test plan.

The final step in planning is to divide your systems into two groups; those that should run without further modification, and those that have known compatibility issues. (For example, within a Version 6.06 DATA step, the "BY" statement must now immediately follow its associated "SET" or "MERGE" statement. Under Version 5.18, the "BY" statement could be anywhere within the DATA step.) These lists should prove to be dynamic in nature, as further compatibility issues will likely become apparent as tests are run. It is likely that not every routine in the systems with known issues is problematic. Most of the time, however, the routines will need to be run in a fixed order, with results of early routines being fed to successive ones. Therefore, individual routines should be kept in the general conversion category that their corresponding system was placed in.

**STEP 2: TESTING**

We are finally ready to begin our testing. The systems that are believed to have no compatibility issues are given directly to the testers, who will execute the test plan for those systems. Each routine will be run against Version 6.06 of the SAS System, and its results will be compared against a similar run using Version 6.16. It will save time and computer resources to use actual production run data whenever possible, rather than having to rerun a routine under most extraneous conditions. Environmental differences should be noted, and the tests rerun, to ensure that the problems in the test environment did not mask an actual software problem. If a problem is attributed to Version 6.06, that problem should be added to the list of conversion issues. All untested software should be visually or electronically scanned for the condition, if feasible. If not, the newly-found compatibility issue should at least be kept in the forefront, and routines checked as their turn for testing comes up.

One advantage to having separate testers and coders is that the testers can jump right in and test the systems that are believed to be directly transportable to Version 6.06 of the SAS System. The coders are then free to make corrections to those routines that have already been identified as requiring change. One hard-and-fast rule should be that no Version 6.06 innovations should be added to open routines unless that is the only mechanism to correct the known issue. Otherwise, the test effort becomes two-fold; verify that the routine still runs under Version 6.06, and verify that the change(s) made did not adversely affect the routine. The two efforts can hamper each other; we have had occasions of a programmer making two serendipitous errors that cancel each other out, except under most extraordinary conditions! A separate project, or series of projects, can be set up at a later time to utilize the new features of Version 6.06.

Communication is very important in a project of this nature. Problems found during testing are often stumbling blocks that
require an immediate resolution in order for the test team to confirm their efforts. From the other side, compatibility issues can be unexpectedly encountered by either testers or programmers. Time can be wasted on re-investigating issues if these are not communicated.

The testing team is also responsible for determining when regression tests are necessary, and carrying out those tests. Regression tests, designed to re-verify cleaned routines, would primarily be required when a fix is made that may have an effect on a routine that has already been cleared, or when a new issue is found that may be present, although dormant, in routines that have already been tested. The test team is responsible for determining how much regression testing should be done before resuming planned tests.

STEP 3: WATCHING FOR (AND MINIMIZING) SURPRISES

Once the effort is solidly underway, there is only one guarantee—unexpected things will turn up, most likely at the most inopportune times. Even though we cannot predict the nature of the unexpected, we can plan for it. First of all, sufficient time must be included in the budget for unexpected problems. The amount of time added to the budget for the unexpected should vary depending on experience. Project teams that have already undertaken similar projects should look back on the time estimated vs. time actually spent on those projects. The budget for the unknown can be extrapolated from that comparison. For a new team, the time allotted for unforeseen events should be high. 25% should be the absolute minimum added to the budget, although other such teams routinely pad their estimates by 50% and higher.

The second thing that can be done to minimize surprises is to look back on similar conversion projects, and include planning for those problems that occurred. For example, the following problems have occurred during previous conversion and enhancement projects we have been involved with:

'UNOFFICIAL' PRODUCTION ROUTINES & CLONES
One of the hazards of allowing End Users to have personal disk space is that they will store production routines on it. (All of us are guilty of this to some extent. Without exception, the more experienced of us, who should know better, are the biggest culprits.) Some use an existing routine as a starting point and modify it to their own uses, while others write their own routines from scratch. In our environment, where separate libraries are set up for development, testing, and production, we occasionally find "production" routines that are regularly run from the test and even the development library. There is no way that the project team can locate all of these routines, nor should they have to. Early in the project, the team should send a generic notice to all computer users in the building describing the nature of the conversion effort. The notice should inform the readers that it is their responsibility to test, and, if necessary, modify their routines. (The team can and should volunteer their expertise and assistance, time permitting. They should also make their test finding readily available to the individuals doing their own conversion, to avoid the rediscovery of the same problems.)

CONCURRENT DEVELOPMENT ON ROUTINES
The second situation that routinely crops up is that the company cannot stop development while the conversion team does its job. Now routines have to be written, and existing routines need to be modified to correct problems and add functionality. There are two approaches to this situation, each having its own benefits and drawbacks. The first is that the development team can be made responsible for conversion testing and modification. This adds additional responsibility to the development team, who may not be as experienced with Version 6.06 of the SAS System. The alternative is to add the new and modified routines to the conversion team's list of routines. This may mean that the same routines must be tested multiple times. At our shop, we routinely keep the conversion effort with the conversion team, even if that involves testing the same routines more than once. This practice can be negotiated on a case-by-case basis, depending on the expertise and schedules of both teams. On rare occasions, we even delegate the responsibility of making the modifications to the conversion team. Each shop must determine the ground rules for these situations in advance, but should allow circumstances to determine each specific decision.

CODEJCL GENERATORS
One of the reasons that code and JCL generators came into vogue is their flexibility under diverse circumstances. This same flexibility often makes maintenance of these generators a nightmare. Tests must be run under a multitude of conditions, and the experienced tester will still maintain a paranoia that some critical condition was left untested. This can be combated by reviewing previous test efforts on the generators, and noting which conditions caused problems in the past. An additional concern in this conversion effort is that not every code generator is written in the language of the code it generates. (This is the rule with JCL generators, but also common with code generators.) It is entirely possible that your site has at least one SAS code generator written in some other language. The conversion team will be charged with finding and testing these, as well. Do not forget JCL Generators in this effort, since the DDNAMEs are being changed with Release 6.06. (For example, the SAS Log which used to be written to FTP1F091, is now put to SASLOG.)

DRIVERS
All routine drivers need to be included in tests. Our site, for example, routinely invokes SAS routines from within REXX routines. Often, these REXX drivers include FILEDEFs, parameter passing, and even code cloning. As a corollary to Murphy's Law, "The likelihood of a Driver Routine causing problems is inversely proportional to the amount of testing done on it."

The final item on our list of considerations must be discussed at the very beginning of the project—potential failure. Even the most competent project team can be left to the mercies of the unknown. The project plan must consider what steps can be taken if Version 6.06 of the SAS System is officially brought in as the production version of the product, only to find out that some critical problems were left undiscovered. The failure will be compounded if there is no method of recovery in place. The SAS Institute appears to have eased this possibility by permitting the new release to read and write datasets in Version 5 format. This will allow Version 6.06 to be implemented in production without forcing the immediate conversion of SAS datasets.

CONCLUSION

The success of any project is largely dependent on the planning that goes into its Initial phases. Careful planning for the implementation of Version 6.06 of the SAS System will pay off in a smooth transition. By identifying and correcting compatibility issues before Version 6.06 is implemented, you will eliminate the panic that
will occur if these problems show up in a production run, and cause praise to be showered upon you and your team.

BIBLIOGRAPHY / FURTHER REFERENCE:


NOTICE

SAS and SAS Communications are registered trademarks of SAS Institute, Inc., Cary, NC, USA.