Paper 485-2013

Bridging the Gap Between SAS® Applications Developed by Business Units and Conventional IT Production

Thomas E. Billings, Union Bank, San Francisco, California
Euwell Bankston, Union Bank, Los Angeles, California

Abstract

Multiple factors are involved in the decision by an enterprise to decide whether to allow a business unit to run its own production versus having SAS® applications developed by business units run in conventional IT production. There can be a wide gap between the business unit view of "production-ready" programs vs. core IT standards for production systems. The nature of the gap is discussed here, and also the risks of business-run production. Specific suggestions are made regarding whether IT and business should have joint ownership of critical SAS applications vs. segregated roles, and when/how should SAS-based systems be migrated into a fully controlled IT production environment.

Note: *This article is a discussion paper and a work-in-progress. It should be interpreted as a starting point for discussions between the business and IT departments and not as the official policy or practices of Union Bank.*

Common issues/problems present in business-developed SAS application systems

Business units and developers may consider their SAS program, project, job, or system to be "production ready" when their solution has the crucial logic required, but is really only a prototype of the core processing requirements. Generally, these solutions can run reliably ***only*** under optimum conditions, and may require manual intervention or possibly a code change when presented with other less than optimal conditions. Many SAS solutions often assume that spreadsheets and other user generated files are OK to import into the production run. Under typically accepted production practices, user generated files require some form of automated validation be performed to ensure integrity. In addition, the file(s) being presented are usually not considered production quality if they are not automatically generated and presented via methods that preclude human interaction in the process

For example, business users may think a program is ready for production even when many of the following issues are present:

Manual processing (examples):

- Requires manual manipulation of the source code (e.g., to set file names, dates, etc.) before every run
- Requires any manual manipulation of spreadsheets or other input/output files during any segment of the process.
- Requires disparate and manual runs of a set of sequenced programs or where manual QA checks are required during the end to end process.

Unstable/dynamic source code, input components, and/or results/requirements:

- Uses input files – spreadsheets are the most common offenders – with uncontrolled structure. File structure may change frequently, usually for cosmetic reasons, and have the net result of causing production runs to fail. These files typically need to be reviewed prior to consumption.
- Code may require ongoing or frequent changes to accommodate changing conditions or adding new variables or values not previously communicated.

Disorganized:

- While the newer SAS Solutions are becoming more conformant to source control, veteran or older SAS developers often have many program versions available, and these may be poorly managed resulting in code that is decentralized and scattered across multiple directories (for no good reason). Also, in some cases, there is a lack of prior version backups; this may force the business to scramble to recover functionality after serious system failures.
- Often program changes are done "hot" (i.e., ad-hoc to fix production) and ignore change control processes used for coordination and auditing.
- Programs may use directories, formats, and other items such as compiled objects that are in isolated areas and are owned by specific users and not accessible by others, and/or may access these items via hard-coded physical addresses preventing migration and creating inadvertent security issues.
- In some cases, user directories may be deleted when a user leaves a company, often causing total loss of work product vital to the department.

Source code is often hard to maintain for some of the following reasons:

- Code achieves results but fails to provide adequate error handling and may possibly cause catastrophic events such as a critical file is made unusable and the file has not been backed up properly. This may require recreating the file and reruns of previous reports, with unexpected side-effects (e.g., the data no longer match), etc.
- User generated code, formats, and objects that have not been centralized and controlled are often overlooked (e.g., compiled macros) and may cause unexpected and possibly catastrophic results if called.
- User code is often written quickly to achieve results, and may be the victim of bad documentation practices, e.g., little to no code comments and/or external documentation.
- Output from the process has not been validated or qualified via a QA (quality assurance) process
- Usually does not adhere to standard SDLC (Software Development Life Cycle) methodologies including stress-testing. Sample tests might include running the report across critical cycle boundaries, or with empty or missing input files, etc.
- Fails to address safe backup and restoration, recovery and, if required, history management/rebuilds.

Business units are often pushed to produce results quickly. However, it is the lack of development follow-up that causes business developed systems to usually be incomplete per the traditional SDLC practices and/or IT production requirements, with the result that such systems are non-transferrable and unsupportable by IT.

Features of a good, fully-automated IT production system

In contrast to the above, most IT production infrastructures usually have many of the following features.

Fully automated, no manual processing required for standard runs:

- Runs in batch or via a sequence of command line scripts
- Required inputs are usually sourced from external control files/databases, eliminating user-input and making it automatable.
- Usually requires no pre- or post-run manual processing of files (by IT or other staff)
- Scripts and processes properly handle error detection to alert the automation agent to properly create alarms/alerts and halt if the process is a critical component to the flow.

Stable code and operational characteristics:

- Typically has been through reasonable QA and testing
- Can run be smoothly, reliably, and properly provides alerts of critical errors via a scheduler without human intervention.
- Code changes are usually maintained and protected via a change management and source control system
- Can be reasonably migrated though the typical IT promotion process (dev/test/production)

Minimal/low maintenance requirements:

- Structures of input files and/or data are stable
- Input data validation is in place to ensure that input data inconsistencies are detected and managed. This is *critical* for data that originates in spreadsheets or poorly controlled external sources, where user-driven “improvements” may break production systems. (Consider user entry via an input validation form into a database when user input IS required.)
- Has proper documentation and code notes for any/all support and programming personnel to triage and update code, if required
- When relevant, processes should ensure or include documentation of any objects used including external files (e.g., xml, csv), user-written formats and views, compiled macros and data steps, etc.
- Can be checked into source code systems for historical record and management of changes.

Additional nice-to-have features, and often required features, in an IT production system include the following.

Robust error handling for all steps requiring reloads if failure occurs:

- Handles errors gracefully to ensure data integrity is maintained
- Produces meaningful error messages and logs for support and issue resolution.
- Validates required input and objects existence prior to execution
- Performs sanity checks to input and control files and parameters prior to program execution.

- Provides “safe” default parameters that will prevent accidental running of program when required parameters are not passed to prevent damage to existing data.

Support for normal operations (and debug when things go wrong):

- Should allow for a “debug” level to be set for enhanced problem detection that will write additional information to logs
- When appropriate (should not include sensitive data) send relevant log snippets or log file location/information to support personnel via email to reduce time for troubleshooting issues.
- If not provided by the automation system, create appropriate notifications indicating starts, completions or other critical issues via email. This process should use an external configuration file that permits easy updating of email recipients and/or other useful information, to avoid the need for code changes.
- As appropriate, uses checkpoints to provide swifter recovery from processing errors; this may avoid full restoration when processing reruns are required.

Applications should include processes that handle:

- Application history rebuilds, if required.
- New or updated dataset generations required for program updates avoiding the need to copy dev or test datasets/tables/files into production.
- If required for stability, provide appropriate levels of data backups and management needed for checkpoints and/or historical archiving.

If a business fails to consider the above when designing new systems, then it is more than likely that additional development after-the-fact will be required to implement the functionality. This may require a significant effort if the target system already has a large number of existing programs. The bottom line is that the sooner a business embraces the production requirements while in design/development, the sooner the business will reduce its system maintenance overhead.

If your enterprise already has a significant investment in non conforming applications, consider engaging additional staff resources to enhance the existing code and/or create macro libraries to be leveraged, to support the production requirements. Start with the most critical components first and implement changes as feasible whenever code changes are required.

Business-run production

Is it appropriate for a company to allow a business unit to run its own production systems instead of the existing IT infrastructure? It is not unusual for a business to decide to create a new system to meet its needs. They often staff up and expect the IT teams to immediately provide resources for them to begin their missions. What is often overlooked is the need for IT to review the requirements before providing the resources needed. Business becomes upset and will often decide to purchase its own hardware and software. To further complicate this, they often buy standalone equipment that is housed outside of the core IT “safe” area that contains UPS, redundancy, etc.

The result of running outside an IT production environment is that business-run systems face additional risks that vary according to the setup used. Three setups are possible here; running on:

1. A dev server owned and operated by IT (Lowest risk)
2. A server owned by business, operated by IT inside the core infrastructure (Medium risk: split of responsibilities may be an issue)
3. A server owned and operated by business, outside of the IT core infrastructure (Highest risk).

In setups 2 & 3, the business may inadvertently create its own IT department. The downfall to this is that most businesses do not have properly trained staffing and the probability of a failure is thereby increased. The decentralized IT in setups 2, 3 may be reasonable if the application is not critical to the company’s operations, or where an interruption to its functions will not have a significant impact. If it is critical to company operations, then consider dealing with the more formal constructs of centralized IT and let the core teams assist in facilitating the process. Highly dynamic IT infrastructure teams may be able to create a hybrid solution that minimizes the core infrastructure costs while giving the business unit the flexibility needed for its work.

Some companies permit business units to run their own systems regardless of the purpose and are willing to accept the risks (or are unaware of the risks). Should IT not support this scenario, the business unit assumes ALL of the risks and responsibilities for the system’s operation and support that would have been performed by a traditional IT infrastructure.

The net result of the above is that the business unit is allowed to control its own destiny, good or bad, and can run all the programs and queries, ad hoc or scheduled, in a generally uncontrolled de-facto development environment. In most cases, these non-standard servers fail to adhere to SOX and ITIL (IT Information library) standards. The outputs from these systems, such as files and reports, may go into general distribution (outside the department) and be considered "the truth" by the other departments who are unaware that the reports are being generated outside of the IT core. The data may find its way in some form into reports consumed at higher levels or used as input into other production systems. Unfortunately, since there is no QA or other extensive testing performed, the slightest mistake to either the input data or a seemingly benign change to the program may result in erroneous data being generated and this may have a negative effect on the bottom line.

This model – business-run-production - in most companies that have formalized IT departments is usually against an organization's best practices/policies for this reason. Under this model, important items needed for sustainability are often overlooked. The following list is just a sample of items that are often overlooked:

- While some automation can be used, it still requires some degree of manual oversight to ensure that all the details pertaining to ensuring the stability and recoverability of the server are addressed.
- Departmental staff - including senior programmers, managers and/or other staff - may spend a significant amount of their time in supporting the manual processing required for regular "production" runs and stability/recoverability tasks.
- When executing the programs, the higher manual intervention required for input and operations can greatly increase the risk for human errors, causing delayed delivery by requiring reruns or worse, errors may go undetected.
- Lack of change control for source code can cause extensive reruns/rework or even loss of the original "working" source itself as well as an inability to detect what was changed from the previous run when troubleshooting errors.
- The phrase "pay me now or pay may later" may apply more frequently since the process typically circumvents the usual SDLC methods, resulting in a higher failure rate when cutover into production. Diversion of staff to fix things, i.e., crisis management, may make things worse.
- Non-traditional production setups can create a negative, high pressure, crisis-driven environment that impacts staff morale and may cause existing brain trust exodus. (Programmers generally don't like to perform extensive production-driven duties, especially under the crisis atmosphere of urgent production deadlines.)
- Another critical factor to the success of a self-controlled environment is regular backups. Often, vigilance for backup processes success begins to slack, and the lack of due diligence may mean that when a backup tape is needed, it might not be available (or correct). If this happens, the business unit may achieve the wrong kind of visibility by building a mission-critical solution outside the safety net provided by IT, and in some cases this can have a very negative impact on the careers of those involved.

Needless to say, a business unit is exactly that. It should not be burdened with server operations unless it opts to develop its own internal IT staff. Running its own production, long-term, can lead a BU into an unexpected **trap** -- a trap where the personnel efforts required to run and maintain the system(s) increase until it impacts departmental resource availability and prevents new development and program maintenance.

The figurative analogy here is that the department owned server(s) can become an alluring "cage" that the department has chosen to voluntarily "lock itself inside of." Ultimately, this type of trap can have a long-term, negative impact on the effectiveness (and profitability) of a business unit and possibly the company's bottom line. When this occurs, the business begins to look for a way to escape its cage and it may reach out to the internal IT department. If the proper diligence to SDLC practices and operational requirements has not been followed, the business may be presented with a large bill to bring in resources to upgrade the system to meet the current IT standards.

In most cases, businesses should consider avoiding creating their own IT department since most companies will benefit by reducing IT overall costs of operations via leveraging existing IT infrastructure and expertise. If there are specialized skills required to administer the area, it may be better to fund that position in IT rather than sequester it in the business unit. There are cases where the time to market for a particular need may make this seem implausible. In this case, due diligence to adhere to the ITIL and SDLC methodologies to ensure that the programs and processes that are created can be later automated and absorbed into IT, is required after initial development has completed.

Audit/regulatory requirements may force a system into formal IT production

Internal audits and/or external regulatory requirements may dictate the need for the business to place their SAS based system into a formal, change-controlled IT production environment instead of running it in a business run environment. Corporate IT policies may permit a system to operate outside the core infrastructure. Then, at a later time create a path to integrate it into the core infrastructure once the solutions are ready to incorporate into production so it can meet ITIL, SOX, and SDLC methodologies.

Inevitably, all work product generated by a business unit is consumed in some manner by others. When this occurs, protection of this process becomes paramount to the success of the enterprise. Business should expect that its system planning includes a life cycle path that places the final code and system in a controlled production environment. If business has been granted an exemption to run its own solution, this should still be the target to ensure minimal impacts to current reports while development and testing are occurring in other environments. At this point, it may find the traditional IT core infrastructure to be the optimal/best cost alternative for its success. Planning should include costs for implementing this secure end state. It should be noted that if IT is expected to provide services not included in its current skill set, the business may be requested to pay for the additional training required to support them.

Once inside a formal change control process, business provided solutions face the traditional reviews and approval processes used on all systems. These processes are designed to provide the organization with a clear understanding of changes and their validation. Change control takes time, but emergency changes or fixes can usually be implemented to fix urgent issues or handle issues that have a financial impact. In this case, some organizations may require change approval by senior management, so these are usually avoided.

Both business and IT should work together to negotiate a SLA (service level agreement) that includes the time and resources needed to do an adequate promotion and migration for SAS applications. Business should do everything in their power to make their SAS systems meet reasonable IT standards and IT should work with the business to coach them on change processes before production cutover and for future changes.

Bridging the Gap:
Good programming practices are the foundation for better SAS application systems

Whether a system is developed by IT, business, outside contractors, or a combination of these, large SAS-based systems should be developed in adherence to standard SDLC practices. This should include programming practices that yield higher-quality and more reliable production solutions that are easier to maintain, and are designed to support portability between SAS environments (e.g. promotion to another environment and/or to shorten disaster recovery on key systems).

The business and IT should collaborate in creating a uniform programming style that will assist developers in writing higher quality and more maintainable programs, macros, and systems. There are a number of SAS User Group papers that address this topic. A sample listing of such papers is provided in the References list; readers are invited to review those papers as an entry point into the topic.

Sample/suggested best practice programming practices that should be considered include the following:

- Source code should be peer-reviewed for functionality AND adherence to a joint IT-BU standard whenever possible (this should be a requirement for **all** systems subject to outside audit)
- Good documentation that includes meaningful comments throughout the code permitting third party reviewers to determine the intent of the subsequent code without needing to reference external documentation and to eliminate potential ambiguity on assumptions. It should also reference external documentation that provides a review/discussion of system processes.
- While advanced techniques can shorten coding, and in some cases may be required to improve performance, be sure to include adequate internal comments on the reasoning. Otherwise, coding to an average or medium level may be a better choice to avoid confusion.
- If your organization has advanced disk farms such as Teradata or Greenplum that have advanced SAS functionality, be sure to properly document the details of these both within the code and in the external documentation.
- When code relies on changing parameters, consider using external data sets, tables or parameter files that are updated appropriately prior to program execution. This would allow for a more controlled execution and prevent the need to touch a program every time a parameter changes. It may assist in increasing the portability of the code.

- Code should leverage metadata to improve security and assist in portability between environments. This is particularly true when more than one SAS environment exists on the same physical system, i.e., need to prevent accidental updating of the wrong datasets.
- Parameters such as mailing lists, error messages, etc., if not covered in the previous examples should also be identified at the top of the source code so that they can easily be changed as needed.
- File access and external/local libs should be protected, i.e., be made read-only, except when write access is absolutely required and items requiring write access should be located in a separate location.
- When implementing controls, thought should be given to its method. Should data be sent directly to the program or, instead, writing the controls to alternatives such as SQL and Oracle DBs that may have validation safeguards procedures such as stored procedures to provide input verification?

Many of the SAS User Group papers listed in the appendix address style in the context of user-written code, where it is most relevant. However, style also has relevance when working in SAS Enterprise Guide, SAS Data Integration Studio, and other GUI-based SAS solutions. A small investment in attention to style in GUI-based systems can make the relevant application systems much easier to maintain in the long-term.

Consistent adherence to good programming, design, and other SDLC practices is a challenge even for IT departments; business units are often less aware of standard best practices. This is an area where IT and business both need to be proactive and work to improve the quality and reliability of SAS application systems.

Bridging the Gap:
Who owns SAS application solutions: business or IT?

Many large organizations have internal politics that make cooperative implementation (of systems) a challenge. The question of solution ownership may be the biggest challenge. In some cases, business wishes to develop the core, but expects IT to implement, and in other cases, the business or IT wants to maintain the ownership. If business developed the SAS solution in-house, or actively managed the development of the application by outside contractors, then the business unit is the logical choice as owner of the SAS solution. Being the owner implies that the business has responsibility to maintain the solution, and IT acts in a supporting role, usually providing infrastructure support and advice on the production implementation process.

When core SAS updates are required, then program validation ownership can become a little unclear. This should be addressed early in the process so that an SLA can be developed and staffing levels can be addressed. This would include vetting the code against SAS maintenance updates and release upgrades.

There are several different operation models that may be used in an organization. We discuss several models below and what may or may not be possible in them.

1) Business Unit, Integration, and IT infrastructure administration teams.
   a) Business Unit is responsible for core logic creation
   b) Integration Team is responsible for standardization of BU developed code
   c) IT infrastructure is responsible for administrative functions and promotion/deployment of Integration team provided packages into other environments
2) Business Unit, Development, and IT infrastructure teams
   a) Business Unit creates requirements
   b) Development (includes Integration) team develops code per specification with BU validating results. This may not be the best choice since many iterations may be required if specifications are not adequate (often they are not adequate).
   c) IT infrastructure is responsible for administrative functions and promotion/deployment of Integration team provided packages into other environments
3) Business Unit and IT infrastructure team.
   a) Business Unit creates requirements, develops and provides IT conformant code. (This may require that a tertiary model validation or QA unit be employed to vet the code against requirements.)
   b) IT infrastructure is responsible for administrative functions and promotion/deployment of Integration team provided packages into other environments

What happens when a core SAS upgrade, hot fix, or major upgrade occurs? Who is responsible for the code validation at that point? Given the various models above, it should fall to the team responsible for integration and testing. This can be a sticky point as no one likes to “waste” resources on updates and upgrades. However this falls,

the team responsible should plan to have adequate resources to handle such events and the IT admins should be prepared to assist in emergency updates, etc.

When the business is the sole end to end provider of development and implementation, the short staffing levels will more than likely impede or prevent updates from occurring at all. This would also include the requirements for OS updates as well.

It should be noted that at this time (2012), most IT organizations have limited experience with SAS in an Enterprise production scenario since SAS, being primarily an analytical tool, was usually not used to feed other systems in the past. In recent years, SAS has grown and organizations are now using it similar to an ETL/ELT tool where the SAS system supports rapidly changing analytics needs. This area is dynamic and organizations are challenged to manage the evolving role of the SAS system.

The idea of joint ownership, a SAS solution that is owned and managed by both business and IT, is worth considering when/where it is both feasible and potentially appropriate. An example of such a system could be a large set of analytic results positioned in a database or series of data marts that are generated or updated through a set of SAS processes (developed and maintained by a business unit) and are consumed across the enterprise. Here business would be the owner of the SAS components, while IT might provide DBA (database administration) support for the relational database components. To avoid misunderstandings, a clear SLA and operational document defining the ownership and responsibilities of all the affected entities and stakeholders should be in-place before the plan is designed and before the migration to final production status.

Jointly-owned solutions, if they are large or complex enough, may also be governed by a committee, with representatives from all the disparate stakeholders that would include representatives from the business, various IT constituents such as data management, integrity, etc. and any consumers of the resultant data marts. If an organization allows the committee to become too large, quite often a (very!) bureaucratic and problematic committee is the result and this can have a negative impact on the system. In this case, it may require that rather than seeking an elusive consensus, decisions are made by either a majority vote or by key stakeholders. Decisions of these committees should be carefully documented and archived for audit concerns and the decisions should be published to the consumers with adequate time to allow them to plan for updates, etc.

An alternative to a Governance committee is to consider a user forum where the community submits their needs and lets a technical team comprised of the majority of non-consumer tertiary members and several key stakeholders represent the affected areas. The forum should not directly fall in any stakeholder's or software development group's direct control to prevent possible bias from occurring.

Bridging the Gap: How should maintenance of mission-critical SAS solutions be split between business and IT?

In most organizations, IT is responsible for installing the core SAS and vertical solutions on the servers and maintaining the infrastructure base, including administrative support. IT is usually the owner of the infrastructure, and business and/or software development is the owner of most of the solutions. If a software development group is the owner, then the business should be a major stakeholder in all changes to the SAS environment in which they operate. In general, the line of ownership will usually delineate the split of ongoing maintenance responsibilities.

The preceding section describes hybrid systems, SAS plus relational database components, and suggests a split for the maintenance duties. That example presents questions and scenarios that illustrate the underlying issues.

DB (Oracle) concerns

The development of the basic RDBMS database components (create, update) could be done by business in SAS, using pass-through SQL if/when necessary. However, there may be times when updates are required to Oracle data that do not need any of the functions of SAS. In this case, it may be prudent to allocate this function to other teams to use internally developed solutions where the operation can be run more efficiently. If the SAS program stream requires that this be performed in line with other SAS programs, then it may be wise to use an external scheduler such as Autosys or Tivoli to properly sequence the SAS programs in concert with the external programs.

However, if SAS is needed for analysis during the process of updating, then SAS code should be employed to perform the process. In the event that Teradata, Greenplum, or other systems that offer in-database SAS functionality are used, then SAS is the appropriate choice to leverage these enhancements.

Most organizations require a DBA to be engaged when certain DB functions are leveraged. They generally act as a sanity check and as a controller or "Second key" to review intended processes to reduce the possibility of destruction of key data and negative impacts on (other) systems. It follows that business and IT need to cooperate on systems that require joint maintenance by both departments.

Bridging the Gap: Migrating SAS systems into IT production environments

General Considerations

Almost all IT organizations rely on automation to achieve success. Often the SAS scheduler used in a development environment may not be appropriate for the production environment. This becomes particularly true when a SAS job relies on success of a predecessor non-SAS job. Before exploring SAS application promotion, it is appropriate to list the major possible SAS-based production environments:

- Running SAS code in batch
  - Relevant to user-written applications and SAS tools that can provide source code to support batch runs
  - May need to be part of a sequence with other non-SAS processing and may need to run under an automated tool such as Autosys.
- Stored process server
  - Requires that processes run unattended and this usually prevents the use of user-prompts.
- GUI Based runs
  - A production setup for a specific SAS solution using a GUI-based SAS solution requires significant manual effort, and this may be limited to business run scenarios rather than formal IT production running.

Note that many GUI-based SAS solutions provide methods to execute jobs in batch. Two examples: SAS Data Integration Studio can deploy source code which can be run in batch. The code produced for SAS Enterprise Guide project tasks can be harvested and compiled to produce a batch job; see the paper by Hemedinger (2012) in the reference list for information on this topic. However, it may be easier to run SAS Enterprise Guide projects in the form of stored processes, on a controlled production server.

Analyze degree of control required

To address the issue of migrating SAS systems into production, the first task is to assess each SAS application to understand the system attributes such as what, where, and how many users may opt to run them ad hoc and whether they may be run concurrently. If they are run concurrently, the jobs may need to generate unique interim files to prevent job contention and possible data corruption. System criticality or importance should also be considered when determining the best solution.

Form(s) of the system components, e.g., source code or metadata.

This determines the production environment(s) that the system can run in. SAS source code runs in batch, stored processes on a designated server, and jobs created using GUI-based SAS solutions can run in their environment and/or in batch, depending on the tool.

System products and user base:

- What does the system produce: a relational database, separate data files, reports, and/or other items?
- How many direct users will the system support at once?
- How many users are expected in the total user base?
- How many/which other departments will use the system?

Maintenance of the system
This is discussed above; it needs to be clarified before migration to avoid unpleasant surprises after cutover.

Importance of the system:

- Are the products of this system mission-critical? If yes, explain and justify.
- Will delays in producing the information have a negative impact on operations?
- Does the system support audit and/or regulatory requirements?

- How many downstream systems use/will use the products of the SAS application, and are any of those systems mission-critical?

Addressing the questions above will provide an idea of what the system does and how important it is. The information can then be used in determining how controlled the IT production environment needs to be, and the standards a SAS application should meet to be accepted into production.

Suggestions for deploying in production

The control level and strictness/number of IT standards that a SAS application needs to meet to be placed in production should depend on how mission-critical the application is, what the system does, how complex it is, and who develops, owns, and maintains the system. There should be a continuum of controls and standards, rather than a binary all-or-nothing approach. Some examples to clarify this are as follows.

Strict controls and standards:

- A large SAS-based database system that is mission critical should be migrated to IT production under full change management, and should meet the organization's IT standards for databases, including backup, disaster recovery, and history management. This applies regardless of who developed, owns, or maintains the system. Such systems may require the implementation of enhanced datasets.
- OLAP cubes, if mission critical, should be treated similarly to databases. Prior to moving into formal production, OLAP cubes may be operated by business (possibly in business run production) during a testing/proving period.

Moderate controls and standards:

- OLAP cubes used by one or a few departments that are not mission-critical. A compromise that could potentially be made here is for IT to provide the infrastructure while business is responsible for maintenance. Change management can be more relaxed in this case but should not be ignored, i.e., business can approve and make changes to the production systems but they have to document all changes. IT should be copied on change management paperwork.
- SAS applications that are department specific. An example of this would be programs that create derivative files (e.g., from enterprise databases) for analysis or reporting. Business units might run these applications in business-run-production so they may be outside the bounds of IT production. If moved into IT production, they should be maintained by business, with change management handled similarly to the preceding example. So long as these applications are used by only a single department and maintained by that department, IT production standards can be relaxed.

Limited controls:

- Reporting applications that create no permanent files (other than reports) used by other departments, that are not mission-critical, and are maintained by business. These reports may come under the production framework as a result of being used for regulatory filings or audit/compliance reasons. Here IT merely provides the infrastructure (server), and change management is fully under the control of business. Business should document all changes, but IT does not have to be in the change management loop.

Summary and epilogue

Many GUI-based SAS solutions, while simplifying and increasing productivity for the user, do not offer easy batch export capabilities. It should be noted that, traditionally, GUI based user-run tools often do not play well when they are migrated to production.

In many cases, IT organizations have not had the ability to control or manage SAS usage in their organization from the onset. As a result, when IT is tasked to migrate or assist in the migration of SAS-based systems to a secure and functional production environment, they often face a significant struggle resulting from a user community that has developed their own large solutions without adherence to IT standards or SDLC development techniques. Business often fails to appreciate that implementing corporate IT/SDLC standards at the project outset is the most effective way to ensure these solutions are migration ready compliant.

Once the programs have been developed, SAS development to upgrade and conform to production processes can require significant investment in short term personnel to realistically migrate the programs. To complicate the migration issues, the solutions may have been developed by personnel that are no longer with the company where little to no documentation was created. This may lead to time consuming analysis of the programs and creating the proper documentation needed to meet the standards of typical production ITIL and SDLC standards.

Often business requirements are outsourced to third party contractors and the requirements do not reflect operational requirements or common IT standards. When these applications are turned over to full time personnel for support, the program may be functional, however, such programs are often poorly written and fragile making the maintenance difficult (and trying to retrofit them into a production scenario is even more difficult).

Applying IT production standards originally developed for conventional computing/standard programming languages to SAS application systems can present challenges. The suggestions made here include business striving to develop more robust and better application solutions that meet the relevant needs and appropriate IT production operational requirements, and for IT to be more flexible in supporting SAS applications in production environments.

Finally, both IT and business should recognize that this is a learning process for all parties involved. It will take time – and significant effort - for an organization to develop expertise in migrating complex SAS applications into IT production.

References: General

Hemedinger, Chris. "Not Just for Scheduling: Doing More with SAS® Enterprise Guide® Automation" *Proceedings of SAS Global Forum 2012.* http://support.sas.com/resources/papers/proceedings12/298-2012.pdf

References: Sample SAS User Group papers on programming style

Nelson, Gregory S. and Zhou, Jay. "Good Programming Practices in Healthcare; Creating Robust Programs" *Proceedings of SAS Global Forum 2012.* http://support.sas.com/resources/papers/proceedings12/417-2012.pdf

Lal, Rajesh and Ranga, Raghavender. "How Readable and Comprehensible Is a SAS Program? A Programmatic Approach to Getting an Insight into a Program" *Proceedings of SAS Global Forum 2012.* http://support.sas.com/resources/papers/proceedings12/001-2012.pdf

Hamilton, Jack. "What Do You Mean, Not Everyone Is Like Me: Writing Programs For Others To Run" *Proceedings of SAS Global Forum 2012.* http://support.sas.com/resources/papers/proceedings12/229-2012.pdf

Polak, Leonard. "It's Now Your Project—Clean It Up and Make It Shine" *Proceedings of SAS Global Forum 2012.* http://support.sas.com/resources/papers/proceedings12/043-2012.pdf

Carpenter, Art and Henderson, Don. "Macro Programming Best Practices: Styles, Guidelines and Conventions Including the Rationale Behind Them" *Proceedings of SAS Global Forum 2012.* http://www.sascommunity.org/wiki/Macro_Programming_Best_Practices:_Styles,_Guidelines_and_Conventions_Including_the_Rationale_Behind_Them

Paper presentation history:

Versions of this paper are being or were presented at the following SAS User Group conferences:

- Western Users of SAS Software, 2012, Long Beach, California
- SAS Global Forum 2013, San Francisco, California.

Contact Information:

Thomas E. Billings
Union Bank
Basel II - Retail Credit BTMU
400 California St.; 9th floor
MC 1-001-09
San Francisco, CA 94104
Phone: 415-765-2567; Email: tebillings@yahoo.com