

Effective strategies for managing SAS applications development

Christopher A. Roper, Qualex Consulting Services, Inc. , Apex, NC

Abstract

The SAS System is a powerful tool for developing applications and systems to solve the demanding information needs of the business world. As a result, SAS applications can, and frequently do, become very large and complex. As the size and complexity of these applications grow, it becomes increasingly apparent that good application development management techniques are critical to the successful implementation of the application or system. The sooner these techniques are applied the easier the task of managing the development and maintenance becomes, and the better the application can meet the demands of the information consumer.

This paper will introduce some applications development management techniques that will simplify the development and maintenance of any SAS application. General techniques for setting up development, testing, and production environments will be discussed. As will the purpose of each of these environments and the role developers and information consumers play in each of these environments. The goal of this paper is to help instill good application development habits in the reader, and in the long run provide better, more easily maintained applications.

Introduction

A proper discussion of effective strategies for software and application development would take more time than is allotted for in this paper, but hopefully enough information

will be presented within this paper to introduce some of the fundamental concepts. In broad concepts, this paper will discuss design concepts, quality control, and the roles of application or system environments play in effective application design. These concepts and practices are the foundation for good software development and proper application of these concepts can make the difference between success and failure for an application development project.

In the context of this paper, the term client will refer to the person or business entity for which the application is being developed. At the various stages of development this may or may not be the same person or group. Likewise, the term developer may also refer to a single person, or a team of developers.

The Design Process

One very important thing to understand about application design, is that while it is by necessity the very first part of the application's development process, it is never completed. While the design specifications for an application should be very detailed and complete, the design process itself must remain fluid and adaptive. Application development is an evolutionary process in which the developer and the client embark on together to reach a common goal, the completed application. For this reason, communication between the client and the developer must be extremely open. Good communication is critical so that both parties understand the expectations of the project. It is the responsibility of the client to understand and articulate to the

developer the purpose and intended functionality of the application. It is not sufficient for the client to state the goals for the application to meet, but rather to pass to the developer an understanding of the business rules the application must support and how those business rules and objectives are expected to be supported by the application. It is the responsibility of the developer to be able to communicate effectively and honestly to the client the limitations of the software and of the developer. This is sometimes referred to as “Managing the client” or “Managing expectations”. However it is termed, this is critical to the success of any project. The developer then, should fully understand the business rules behind the design specifications. With this understanding, the developer can effectively apply the design of the application to meeting the business rules. In short, the design process is a continuous, collaborative effort between the business client, who knows what the application needs to do, and the developer, who knows how to build the application. Through the marriage of these two pools of knowledge, a successful application can be completed.

The Environments

There are three basic environments for an application development project; Development, Testing, and Production. All three environments should be structured similarly, with the same directory structures and file naming conventions. If the environments are structured correctly, the process of migration from Development to Testing, and from Testing to Production can be greatly simplified. In the simplest case it can be as simple as changing the assignment of a macro variable in the application autoexec. For example, assume the following is the autoexec for an application:

```
%let root=c:\develop\sesug99;
```

```
libname system "&root\system";  
libname parmsdl "&root\parmsdl";  
libname sasdl "&root\sasdl";
```

```
dm 'AF c=system.windows.main.frame' continue;
```

Migrating this application to the Test environment could be as simple as changing the assignment of the macro variable ROOT. For example:

```
%let root=c:\test\sesug99;
```

```
libname system "&root\system";  
libname parmsdl "&root\parmsdl";  
libname sasdl "&root\sasdl";
```

```
dm 'AF c=system.windows.main.frame' continue;
```

Granted, usually there is more involved in the migration of the application through the three environments, but this demonstrates one technique that can be built upon to simplify the migration process.

Each of the three environments has a very specific and distinct role in the development process, and failure to keep these environments separated can lead to serious problems as the application nears milestones and completion.

Development Environment

The first of the environments is the Development environment. This is the “workbench” for the application developer. It is usually littered with scraps of code and programs that don’t really belong in the application, but are leftovers from trying out ideas to see if they might be useful. Many of these code fragments are eventually incorporated into the application, but the housekeeping of the development environment is usually lacking. This is usually alright as long as the mess is confined to one place, a library, catalog, or

perhaps just an entry in a catalog. Think of it as the “rec. room” of application development. The important thing to remember is that this clutter should not wind up in the Test or Production environments.

While the structure and naming conventions of the Development environment should mimic Test and Production as close as possible, the data source for Development doesn’t have to be particularly representative of the Production data. While it might initially sound like a good idea to have an exact duplication of the various data sources, both internal and external, that will be utilized by the application, in reality this just does not work. Duplicating the Production data would be very time consuming, place an onerous burden on the developer to maintain and upkeep, and greatly reduce the ability of the developer to make reasonable progress. What the developer really needs is a small, fairly representative sample of the Production data. If a trade-off between size and representativeness must be made, size wins. The important consideration is that the developer must not be slowed by having to push lots of data through the application. This is for Testing to do. The primary role of the Development environment is to be the origin for all modifications to the application. This is a key point. As issues are identified in the Test or Production environments, there is sometimes a temptation to modify the application in place, and migrate the change back down to Development. This is a mistake. The development process must be a one-way street, starting with the Development environment and migrating through Test to Production, with no detours or short cuts. Failure to follow this path will lead to serious version control problems where bug fixes or enhancements that have been implemented in one environment will be

overwritten and will disappear. These fixes and enhancements must eventually be incorporated into Development anyway, so the best practice is to always start in Development.

Test Environment

The Test environment can be thought of as the ‘Beta’ release of the application. Most of the bugs should have been identified and fixed in the Development environment. The Test environment should discover issues with scalability, not functionality. In practice, there are always functionality bugs found during the Test environment, but they should be relatively infrequent. The Test environment should verify that all the required functionality is present in the application. This is best accomplished by defining and utilizing a testing script. The testing script is a document that outlines in precise detail the actions the tester is to take in navigating the application and the expected result of the navigations. Any unexpected results should be recorded in a corrective actions repository, or issues log. This log should describe the problem, how it was identified, what the expected result was, what the observed result was, the date and time the issue was recorded. This issues log can be a simple Microsoft Excel workbook, or Access database, it doesn’t need to be anything complicated. The important thing is that it communicates the issues to be addressed to the developer in a brief and clear manner.

The Test environment should mimic the Production environment as closely as possible, including the data sources. This is not always possible, especially with applications dealing with very large data sources. Therefore, it may be necessary to approximate the data source. This approximation should be large enough to test how well the application will respond to

large amounts of data, plus the data should be representative of the breadth of the Production data. This means that the Test data should be pulled from the Production data as much as possible, and not just a few records repeated thousands of times. The application's ability to handle data should be stress tested by incrementally increasing the amount of data the application processes until it either processes all of the test data acceptably or it fails, and is sent back to the developer. Similarly, the work load capability of the application must be tested by incrementally increasing the number of users accessing the application until the application passes the established acceptance requirements. The end result of the Test environment should be that the client and the developer have a high degree of confidence that the application will perform acceptably in the Production environment.

Production Environment

The Production environment is the hallowed ground of the application development project. Nothing goes here that does not belong. There are two cardinal rules concerning moving application code from the Test environment to the Production environment:

1. Be absolutely sure that what is about to be moved works correctly and does not cause problems with any other components in Production.
2. Don't ever believe that you did rule number one correctly. Backup Production prior to ANY application changes.

Rule number 1 is easy enough to understand, if a little difficult to actually implement. It is a dictate the client expects to be

absolutely adhered to, and this is not unreasonable. The developer must have tools at their disposal that enables them to know, with certainty, what they are about to implement. There are precious few tools in the SAS applications development world to enhance the ability of the developer to manage the application migration process. One is the SAS Institute's Software Configuration Management (SCM). It can help with version control, will provide version labeling, and perform some of the functionality of a code librarian. SCM can handle code check-in/check-out, file authorizations, and a few other helpful functions. It does not contain any functionality useful for the exploration of application code. Cataloger, a freeware tool by Qualex Consulting Services, Inc. does not provide any of the version control capabilities of SCM, but does have excellent exploratory capabilities for one or more catalog entries. In short, SCM is a strategic tool useful for managing and tracking the process of application development, whereas Cataloger is a tactical tool useful for understanding the contents of an application and verifying that the intended changes have been made to an application. By utilizing these tools, the developer can have a reasonable amount of confidence that the migration to the Production environment will be done correctly. This is where rule #2 comes in. Never, ever implement a change to Production without first it backing up. Even if there is no way the change could possible cause a problem. It only takes one mistake to cause untold horror, and there is no valid excuse for not having a backup. Do not rely on the network backups, always have a separate, independent backup ready to restore the system should something go wrong.

Documentation

Documentation is often thought of as the grunt work of the application developer. This is because it is the grunt work of the application developer. But the fact remains that proper, professional application development demands that this documentation be provided. So what is complete documentation? There are four categories of documentation that should be provided. They are

- Inline or Imbedded
- Application
- System
- Historical.

Imbedded documentation is the placement of comment blocks throughout the application code. Some developers resist writing imbedded documentation because they believe their code is self-documenting. There is no such thing as self-documenting code. There is well written code that is easy to understand, but this is not self-documenting. Imbedded documentation should be notes to the developer to explain the logic that follows. These are very detailed notes, and typically address very specific actions the code is attempting to perform. Therefore, it is necessary to have these notes appear very frequently in the application code. They may only address a very few lines of code, even as few as one line of code. The purpose of these notes to allow a developer to understand the code months or years after it has been developed and the original person that worked on the code is not around anymore to explain it. The pitfall with this documentation is that it is frequently not updated as the code is updated. The result being that the documentation can be worse than no documentation at all. Because the code might have been updated but the

documentation had not, the next developer reading the documentation might be misled about the functionality of the code and this can lead to bad consequences. Therefore, it is very important to maintain the imbedded documentation, and conversely, do not trust that the imbedded documentation you discover in a legacy application has been maintained. Verify that the code performs as the documentation states. If not, correct the code, or the documentation, or both.

Application documentation is the printed hard-copy of the application itself. This includes screen prints, application code, and support files (i.e. autoexec, configuration files, etc.). The purpose of this documentation is to provide to the client a written record of what has been accomplished. This isn't always required by clients, but it can be very useful as a tool for reviewing the application design and code. The client can mark-up the screen dumps for future design changes, and the developer can mark-up the application code. In the end, it is an physical record of the application.

System documentation is a broader, more strategic form of documentation. It includes schematics and flowcharts to illustrate how all the pieces of the application fit together. The purpose of System documentation is to explain how the application should work. This entails illustrating the designed application flow, i.e. the navigation through the frames or windows. Additionally, the data sources are explained and the data flows are illustrated. This entails a data dictionary, schematics illustrating where the various data sources are utilized by the system, and where the data generated by the application are created and ultimately stored. The System documentation will not explain how to use the application, but it should explain how the application works, what it needs to work properly, and what it is

expected to accomplish. Additionally, the system documentation should document any external needs of the application, the hardware necessary to support it and any network support requirements.

Historical documentation is the record of the design and development of the application. This is a written record of the application development process. This is to record the important changes that impacted the development process. This can be accomplished without too much effort if the developer is disciplined enough to record the historical events as they happen. One technique to accomplish this is to use a two tiered approach. The first step is to use a standard header block on all application code that includes a section for the history of that entry. This could be something as simple as the following:

```
/* **** */
/* Programmer: CAR */
/* Date: 10SEP1999 */
/* Modification: Changed the number */
/* of items presented by the popmenu */
/* presented by the widget SELECT */
/* from 2 to 5 to support the addition */
/* of the three new values for the var. */
/* HUBS in the dataset A.CHOICES */
/* **** */
```

This would be added to the top of an entry, and a section similar to this would be added each time the entry was changed.

The second tier of this approach would be to repeat this information, but in a central history file. This could be a text file that is kept in the same location as the application code, or in the case of an AF catalog, it could be a source entry kept in the same catalog, named HISTORY.SOURCE. This entry would look just like the header block version, except it would also have the name of the entry, something like the following:

```
/* **** */
/* Entry: Lib.Cat.Entry.Frame/SCL */
/* Programmer: CAR */
/* Date: 10SEP1999 */
/* Modification: Changed the number */
/* of items presented by the popmenu */
/* presented by the widget SELECT */
/* from 2 to 5 to support the addition */
/* of the three new values for the var. */
/* HUBS in the dataset A.CHOICES */
/* **** */
```

The advantage of having the history file kept with the catalog or in the same directory as the application code is that as the code is saved and archived, the history file is also saved and archived along with it.

Conclusion

This paper has focused on a few issues regarding strategies for application development, especially within the SAS System. The roles of the Development, Testing, and Production environments was discussed, as well as the importance of good communication between the client and the developer. The types of documentation and the purposes of each type was introduced. There are many more issues involved with good application development strategies, but there is just not enough space in this paper to cover them all. For a further discussion and more in depth discussion of these concepts, an excellent text is "Code Complete: A Practical Handbook of Software Construction" by Steve McConnell, published by Microsoft Press.

Author Contact

Christopher A.Roper
25 R Way
Hillsville, Va 24343
Chris.Roper@qlx.com
<http://www.qlx.com>

Acknowledgements

SAS is a registered trademark and trademark of the United States and other countries. Other products are trademarks of their respective companies.