Paper 22-27

Large-Scale System Development in Base SAS®

Craig Ray, Westat, Rockville, MD

ABSTRACT:
A system may be defined as a series of programs designed to run repetitively in a production environment. Generally, systems of programs perform many functions and are maintained over time. It is accepted practice that the development of large systems should follow a development 'life-cycle.' The life-cycle includes processes for initial design and construction of the system, followed by an iteration of new requirements and modifications of the code. This paper primarily discusses how to design, structure and test SAS code to support the full software life-cycle. The resulting approach demonstrates how to structure completely separate production and testing environments that enable the programmer to develop and test new functionality without impacting production jobs in the process. This presentation is structured as a 'how-to' for system development and is targeted to all levels of SAS programmers who are responsible for applications development. The only assumption is that the programmer is familiar with the concepts of the SAS macro language.

INTRODUCTION:
While good development practices are always desirable, they are not quite as vital for many routine SAS tasks, such as ad-hoc or exploratory data analysis. Rather, the importance of this topic emerges in the development of production systems, ones which run repetitively and must be supported over time with upgrades.

This paper describes the techniques employed to develop high quality software, software that can be supported over time. The foundation of good systems development is following a defined software life cycle process, which this paper will briefly describe. Particular attention will be given to the coding phase and how to structure SAS programs to enable them to be supportable. The only assumption is that the reader is familiar with the SAS macro language.

DEFINITION OF A 'SYSTEM'
A system may be defined as a series of programs designed to run repetitively in a production environment. The following are typical characteristics of a 'system':

1) Large
2) Perform many functions
3) May contain many different programs
4) May be read-only or may write/update a database
5) Possibly developed by multiple programmers
6) Maintained/changed over time.

The fact that a system may be 'large' and developed by multiple programmers implies the developer requires a structured way to design and document the design of the system, before coding begins. The fact that the system is maintained and changed over time necessitates that the system be implemented in such a way that it may be modified and tested while at the same time supporting on-going production.

THE DEVELOPMENT LIFE CYCLE
From a simplistic point of view, the software development life cycle for a large production system is a process consisting of the following steps:

1) Gather requirements
2) Design system
3) Develop code
4) Test system
5) Do the following forever
    6) Move to production
    7) Run in production
    8) New requirements and design
    9) Modify code while supporting production
    10) Test (against a test database)

The important point is that systems generally are modified and maintained over time. Once in production, the modifications must be developed and tested in an environment totally separated from production.

SOFTWARE DESIGN
It has been said that 'you **will** do system and program design.' All systems are designed, from the smallest to the largest, whether or not there is a formal design phase. If there is not a formal design phase, then the design is done either in the mind of the coders or in random discussions during the programming phase. As has been demonstrated repeatedly, however, when there is a formal design stage, the resulting system is ultimately less expensive to build, of higher

quality, and delivered sooner. Therefore, one can and should eliminate the urge to declare that 'we don't have time to do a design.' To that, one should respond that 'we don't have time to **not** do design.'

It is beyond the scope of this paper to cover the vast topic of software design. However, in discussing software development it is worth stressing the importance of producing a coherent software design **prior** to undertaking the programming of a system. While there are numerous techniques for software design, a sample design technique is presented here to serve as an example for this paper. *Illustrations 1 and 2* show the design of a fairly straight-forward SAS system. *Illustration 1* represents a sample *preliminary design*, which depicts the flow of physical files through various subsystems of the overall program. *Illustration 2* represents a further refinement, where the program is depicted as a modular hierarchy. Each module then becomes a macro.

SAS MACRO AND THE AUTOCALL FACILITY:

When implementing any SAS production system, the primary building block for code is provided by the SAS Macro facility. While the SAS Macro language is a powerful run-time code generator, its primary purpose for this discussion is to allow the construction of 'subroutines.' In general, when developing production systems, all code should be packaged as macros. As will be demonstrated, this will facilitate 1) the smooth migration of code from the development to the production environment, 2) modifying and testing code that is already in production without adversely affecting production, and 3) code reuse.

The Autocall facility is invoked with the MAUTOSOURCE option along with the macro search path specified in the SASAUTOS option. In order for the Autocall facility to work, the file name where the macro is stored must have the same name (with .SAS as the extension) as the SAS macro it contains. When using the Autocall facility, the physical location of a macro does not have to be explicitly specified. Rather, a macro is simply invoked within SAS code. If the macro has not already been compiled within this SAS session, then SAS searches for a file of the same name as the macro in the search path, searching in the order specified in the SASAUTOS option. When a file of the same name is found, then the macro is 'included,' compiled, and executed.

CODING – LIFE CYCLE ENVIRONMENTS:

In order to develop and test code to support the full life-cycle, i.e., support continual change after going into production, it is important to structure the code specifically for this purpose. It is important that the programs can be tested in an environment totally separated from production and the code can be easily migrated to production after testing. The following section outlines the structure of the production and development environments to enable this. One should design the production environment first, then construct a development environment to simulate production.

Characteristics of a good environment for development and production are:

1. the process of testing, tracking changes, and migrating code from development to production is well understood and followed by all project members,
2. code can be developed and modified without adversely affecting existing production runs during the development process,
3. code can be migrated to the production environment without having to be altered in any way nor be retested.

Production Environment:

The production environment consists of separate directories for:

SAS database - directory consists of the SAS database; this includes code tables in the form of SAS data sets for dynamic tables,

Sample Directory: \bigproj\prod\data

code - this directory consists entirely of Autocall macros,

Sample Directory: \bigproj\prod\code

drivers - the actual main programs which are executed; it is only the drivers which have any

knowledge of the physical environment, in this case, the production environment; it is in this directory only that .LOG and .LST files are maintained.

Sample Directory: \bigproj\prod\runcode

Utility macros - Company or division-wide generic macro library. Typical utility macros might include:

- %TESTPRNT - conditionally generate PROC PRINT code,
- %NOBS - return the number of observations in a SAS data set,
- %DYNFMT - dynamically generate PROC FORMAT code from the contents of a SAS data set.

Sample Directory: \company\prod\maclib

Development Environment:
The development environment is an exact replication of the production environment:

SAS database - the SAS data sets have the exact same structure as the production database, but generally will have a small sample of the data; the code tables are generally exact copies of the production code tables, periodically copied directly from production to development,

Sample Directory: \bigproj\test\data

code - this directory consists entirely of Autocall macros, but generally only the macros which are currently under development or maintenance; this maintains high integrity in the development environment and allows managers to more easily track the code that is being worked on.

Sample Directory: \bigproj\test\code

drivers - the actual main programs which are executed; it is only the drivers which have any knowledge of the physical environment, in this case, the development environment; it is in this directory only that .LOG and .LST files are generated and maintained.

Sample Directory: \bigproj\test\runcode

Utility macros - generally refers to the production environment only. Altering these macros is out of scope for any project specific development.

Sample Directory: \company\prod\maclib

DRIVER PROGRAMS:
Driver programs should be the only code which has knowledge of the physical environment. This allows for code (i.e., macros) to be migrated to different environments and only the driver program needs to be changed.

In general, driver programs:

- Contain all LIBNAME, FILENAME, and OPTIONS statements. With few exceptions, these statements are not allowed in other code
- Define all global macro variables
- Contain virtually no other code

Example Production Driver Program:

```
OPTIONS ls=160 ps=60
        mautosource mprint
        sasautos=('\bigproj\prod\code',
                  '\company\prod\maclib');

/* Global macro variable defined in all drivers */
%let environ = prod;

/* The Database */
LIBNAME data "\bigproj\&environ\data";

/* Code Tables */
LIBNAME tabldata "\bigproj\&environ\data";

/* ASCII Input Files */
FILENAME rawdata
\bigproj\&environ\data\trans.txt";

%main(infile = rawdata, inlib = data)
```

Example Development Driver Program: By design, the development driver is different from the production driver in that 1) it refers to test data sets, and 2) the Autocall search path includes the test code library before the

production code library. The latter fact is paramount. Code exists in the development code library only while it is being tested. During test runs, the code which is being modified is invoked from the development environment while code which is not being modified is invoked from the production environment.

```
OPTIONS ls=160 ps=60
mautosource mprint
sasautos=('\bigproj\test\code',
          '\bigproj\prod\code',
          '\company\prod\maclib');

/* Global macro variable defined in all drivers */
%let environ = test;

/* The Database */
LIBNAME data "\bigproj\&environ\data";

/* Code Tables */
LIBNAME tabldata "\bigproj\&environ\data";

/* ASCII Input File */
FILENAME rawdata
"\bigproj\&environ\data\trans.txt";

%main(infile = rawdata, inlib = data)
```

SAMPLE MAIN MACRO:

Systems of any substantial size should include a main macro which directs the overall flow of the logic. The flow of the main macro should correspond with the high-level design so that one could have a basic understanding of the system by analyzing this code.

A common mistake is to combine the main macro with the driver. This temptation should be avoided. It is important to separate the physical details of the driver from the logical details of the main macro. Except for code which is being actively altered, the only differences between the production and development environments are the drivers, themselves, which should 'live' in their respective environments and rarely change. With the driver separated from the main macro, then changes to the main macro can be migrated to production without having to change any code. If the two are combined, then when migrating changes to the main macro to production, the driver portion of the code must be altered. The code below depicts the main macro for the system illustrated in Illustrations 1 and 2.

```
%MACRO main(infile1 = , inlib =, tabldata =);
        /*  Utility macro from company lib */
        %dynfmt(tabldata = &tabldata )

        /* Project macro */
        %readraw(infile = &infile)

        /* Project macro */
        %process()

        /* Project macro */
        %update(inlib = &inlib)

        /* Project macro
        %reports()
%MEND main;
```

CODE MIGRATION PROCESS:

If programs are written once and run in the production environment forever, without modification, then there would be no purpose for this paper. This, however, this is rarely the case. It is generally true that the cost of the initial development is a very small fraction of the overall life-cycle cost for any particular module. Programs must be continually modified to adapt to shifting business requirements. Code modifications can take days, weeks, even months, during which time, the production system must remain operational. During this process, this changes to programs must be kept completely separate from the production environment until fully tested. This accentuates the importance of establishing separate but structurally equivalent production and development environments. The environments must be separate so code can be changed and tested without affecting production runs; the environments must be structurally equivalent so that tests in the development environment as nearly as possible simulate the production environment.

The development and production environments are depicted in *Illustration 3*. This also illustrates the process whereby code is 'checked out' of production, modified, and tested, and migrated back into the production environment.

Generally, to alter a macro:

1. COPY from production code library to development code library
2. Make any changes
3. Test with the EXISTING driver in development (generally, no changes to the driver need to be made)

When testing is complete, to migrate the macro to production:

1. MOVE the macro from development to production
2. Run the production system; generally, NO changes need to be made to either the production driver nor the macro.

MISCELLANEOUS TOPICS:

The following are advanced topics which may be useful when developing large production systems potentially with multiple developers.

Adding an Integration Test Environment:

Frequently more than one developer may be simultaneously working on a given release of a system. Each developer may individually test his/her own changes, but it is essential to integration test all changes as a complete unit. While an integration test could be performed in development environment, the code has to be 'frozen', precluding other development activities during the integration test. To remedy this, a separate integration test environment may be added. This environment will look almost exactly like the development and production environments already described. The fundamental difference will be the SASAUTOS paths for macro searching:

Development Environment

```
sasautos=('\bigproj\test\code',
          '\bigproj\int\code',
          '\bigproj\prod\code',
          '\company\prod\maclib');
```

Integration Environment:

```
sasautos=('\bigproj\int\code',
          '\bigproj\prod\code',
          '\company\prod\maclib');
```

Version Control and Configuration Management:

The need for version control and configuration management is well understood by all who have worked on large systems that may involve multiple developers and multiple installations of a system. SAS, however, does not explicitly support configuration management tools. This does not preclude the use of a 3rd party tool. The production macro code library can be placed under version control. By doing so, the macros can be write-protected whereby code can be migrated to production only by adhering to prescribed procedures. All previous versions of all macros are thereby recoverable. In addition, a configuration management system can record a 'snapshot' of a system at any point in time. This records the current version of all macros which allows for recreating a release of the system as it existed. This can be extremely useful for technical support of a system.

CONCLUSION:

Developing high quality production systems requires careful planning and preparation. This paper described the software development life cycle as well as the establishment of robust development and production environments. These combined support rapid development of professional systems that are supportble over time.

The author may be contacted at:
Westat
1650 Research Blvd.
Rockville, MD 20850
(301) 517-8077
RayC1@Westat.com

Illustration 1, Preliminary Design

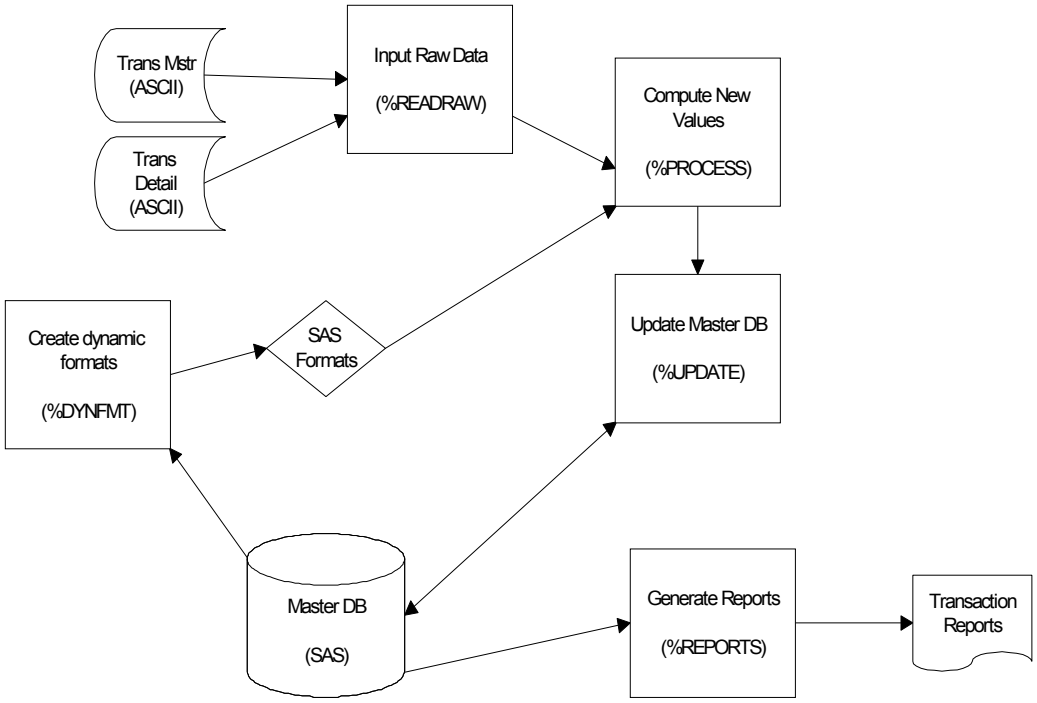

Illustration 2, Module Hierarchy

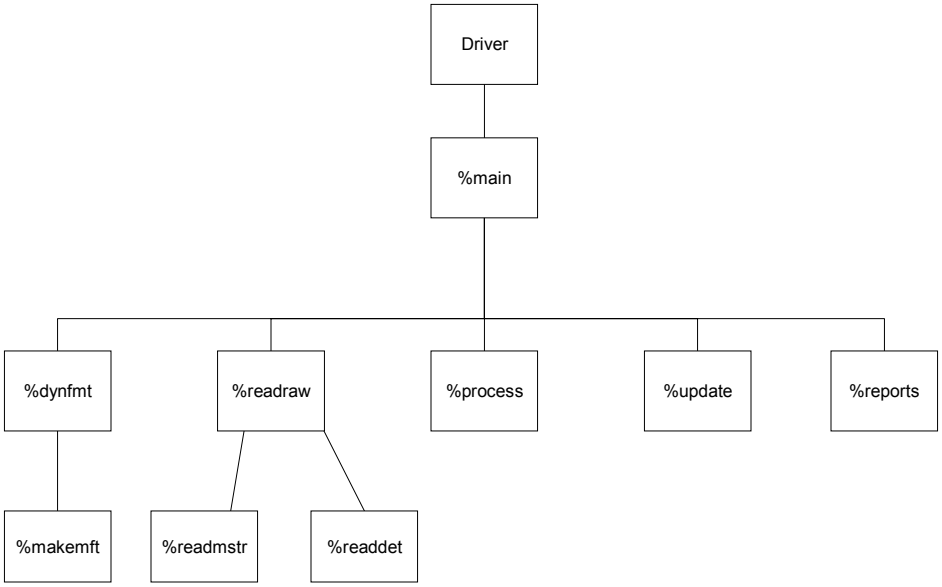

Illustration 3, Code Migration Process