Macroitis - A Virus or a Drug

Margaret K. Schrempf
G.D. Searle and Company

Abstract

In early May 1994, the disease which had been infiltrating SAS® programs for more than a decade was finally identified. Named Macroitis, the disease was classified, with results of preliminary research being presented at SUGI 19 in Dallas.

Since the discovery and initial analysis, the original research team has not only identified a possible cure but is now well advanced in Phase II clinical trials. In addition, guidelines are being drawn up to help the afflicted achieve the appropriate balance of good macro coding while avoiding Macroitis.

This paper will deal with the results obtained from the clinical trials. It will also propose a drug free therapy designed to avoid the disease altogether. Including coding standards and validation methodologies, the therapy also deals with the adverse effects of Macro Deprivation Syndrome (MDS).

This new approach has been shown to slow (and in some cases) eradicate Macroitis altogether, while improving coding practices in 100% of trial subjects.

Introduction

Macro coding is a good way to eliminate repetitive coding, however when macros are developed in a "black box" or when good coding practices are not followed, Macroitis begins to take hold. It is the purpose of this paper to establish Macro coding guidelines, macro grading scales and validation procedures for macros. Ways in which current macro libraries can be modified to use coding standards will also be illustrated.

Identification of Macroitis

In order to correct or prevent the disease we must first learn how to identify some of the signs or symptoms of MACROITIS. Our clinical trials have involved subjects over an eleven year period and have been able to identify and isolate the behavior exhibited by programmers infected with Macroitis.

Remember, even though it is possible to identify macroitis, the programmer who is infected must first admit that she/he exhibits some of the symptoms.

Macroitis Symptoms

1) *Grinning at the monitor.*
2) *Every 3rd or 4th sentence is "I've got a macro that will do that!"*
3) *Substringing a macro at least 3 times to obtain another macro.*
4) *Creating a macro to check other macros.*
5) *Strange ritual beliefs that untouched macros get better over time.*
6) *Cannot write a sentence or a report without a % or &.*

Macro grading scale

In order to identify the kind of macros that your organization is currently using, our Clinical trails have developed a simple macro grading scale to quantify the problem. We have found that most macros fall into three grades, Poor or obnoxious, Fair, and Excellent.

Excellent Macros are macros that a novice to experienced SAS® programmer can use without asking any questions. These are macros that are internally documented and do not contain any magic or black box processing. Macros that fit in this grade are those that simplify programming tasks, as in the following examples.

Report generation

MACRO's can save immense programming time and typing effort by simplifying tasks. For example, if you had to produce a report for 50 different study protocols each displaying race and sex, it would make sense to create a macro to automate the task.

Consistency

Each pharmaceutical or other type of company has adopted a specific way format to display information such as a customer number or patient number. It would make **SENSE** to have a macro available to use for reporting this information on a continual basis.. This will ensure that the patient or customer number is always handled in the same manner.

Calculations

Other logical uses are conversions for units of measure. Temperature conversions from Celsius to Fahrenheit or Fahrenheit to Celsius, pounds to kilograms, or inches to centimeters. A macro to effect these convertions makes **SENSE** because it insures that the conversions are being done EXACTLY the same way every time and that there is no room for programmer creativity.

Fair graded macros are those which can be run by SAS® programmers that have at least 3 years of experience and where some questions must still be asked of the author of the macro. These macros begin to show the early signs of the Macroitis virus but can be helped through Macro deprivation therapy techniques.

Some identifiable symptoms of these macros are:

1.) *No internal documentation within the macro.*
2.) *Passing unknown or undefined parameters.*
3.) *Passing over 50 parameters to one macro.*
4.) *Creation of a data set within the macro which will never be used again in the program.*
5.) *Commented code which no longer works.*
6.) *A block of code which is commented out, that was obviously put there for special processing.*
7.) *Macros that do not use "normal" naming conventions.*

Poor or Obnoxious macros are the plague of SAS® programmers and usually can only be run by the author of the macro. These are also the types of macros that will require a company to hire a macro specialist to de-macroize, when the author has left. They can, however, be classified and therefore can be dealt with when encountered. These classifications are as follows:

1.) *Unreadable macros.*
2.) *Macros that hide the process taking place in the program.*
3.) *Macros that obscure the process taking place in the program.*
4.) *Macros that create logicals(True -False) and do not use them.*
5.) *Macros that "save" CPU time.*
6.) *Macros that call macros, that call macros, that call macros, that call macros, that call macros.*
7.) *Macros that were originally part of a macro system, that are no longer used but still reside in the macro library.*

<u>Macro standards</u>

There is no excuse out there in SAS® land to not have some kind of documentation in place for any macros that your business uses. Macro code need not provide job security for programmers or consultants. Our clinical studies have uncovered several instances where poorly documented macro code has resulted in several months of rework.

This kind of rogue macro coder takes no prisoners, relies on a lack of supervision, and leaves the company with a guarantee of a return trip, all expenses paid. However we have discovered that macro documentation is the first line of defense in controlling what is now called "RAMBO" macro code.

Macro documentation is simple to put into place and is an essential starting point in this battle. Clinical trials have proved that sometimes a simple Header starts the documentation process.

```
%*************************************************************;
%* Program Name: REORDER.SAS                     Date: 01/1/91  *;
%* Author: Jane Doe                                             *;
%* Description:   This macro will reorder and keep only the     *;
%*        designated variables in the output file               *;
%*                                                              *;
%* Parameters:                                                  *;
%*          DSIN - data set name to be reordered. If none is used *;
%*          then the _last_ data set created is used.           *;
%*          DSOUT - data set name of the output data set if none is *;
%*          used then the _last_ data set name created will be used *;
%*          ORDSTR - String of variable names in the order      *;
%*          required for the output data set.                   *;
%*                                                              *;
%* Validator Name: Alex Smith                    Date: 01/31/91 *;
%* Validation Protocol Number: CP-91-0005                       *;
%*********  Revision History  ***************************;
%* Author Name: Tom Jefferson                    Date: 06/1/94  *;
%* Validator Name: John Adams                    Date: 06/30/94 *;
%* Validation Protocol Number: CP-95-0176                       *;
%* Description:   Upgrade for use on UNIX platform              *;
%*************************************************************;
```
Figure 1- Documentation Header

Documentation Header
The documentation header should contain both general information, and comments that are specific to the macro. General information contains the:
Name of program
Author
Date

The first section of specific information should start with the *description* or the *purpose* of the macro. For example, in Figure 1 the purpose of the reorder macro is to reorder and only keep the variables designated in the output file.

The second section of specific information should contain the parameters of the macro. Questions that should be answered here are:

1.) How is the macro used?
As part of a data step?
As part of a proc?
Does it just subset?
Can it run as a separate step?

2.) Are there any default parameters?
Uses last data set created?
Uses a data set within a library?
Are there variables created with default values?
Are there special data set names used?
Are there any SAS® options or macro options being used?

3.) What are the input parameters?
Are there parameters that the user must supply?
In what order should the user supply them?

Will the macro supply the parameters?
Can the user override the parameters?

4.) What are the output parameters?
Does the macro create an output data set?
Will the macro create variables that can be used within a current data step?
What are the variable names?

The third section of the documentation header should contain the validation information for the macro. The date, validator's name, and the Validation Protocol Number should appear in the header. Macro Validation will be discussed later.

The last section of the documentation header should contain any revision history of the macro. Each time a change has been made in a macro, it should be noted. In Figure 1, the macro was changed to run on a Unix platform. Any significant change in a macro should require that a revalidation of the macro take place. This will ensure that the macro performs as it originally did. Information that should be required is as follows.

1.) Name of the programmer making the change.
2.) Date of the change.
3.) Complete description of what change took place. If any parameters change, they should be noted at this point.
4.) Name of the Validator and the validation date.
5.) Validation Protocol Number if any.

Documentation within program code
There should be no difference between writing a program in SAS® and writing a macro in SAS®. The amount of documentation that is required for a program should also be required for a SAS® macro. The argument of too many comments in a program unfortunately has left us with far too few comments. A way to guide comment use can be fashioned in this manner.

Each <u>step</u> or <u>process</u> should be documented. This kind of style also permits a more modular approach to programming. Commenting each step or process also will allow the programmer to make sure that the macro is performing a specific task. In some cases the programmer or manager may want to split out the process performed into two or more macros. For example, if the macro will be selecting randomization for a specific protocol the comment could read:

```
%*************************************
     Select randomization file and subset data for
     the protocol number contained in &PROT
%************************************;
```

Comments can also be a source of information for where and when a particular calculation has been decided upon. An age calculation is pretty common, but can, in some cases, be complicated. The following comment clears up any questions regarding this calculation.

```
%*************************************
     Patient age is defined as the int(Date of
     Birth - Date of First Dose)/365.25 per
     Standardization Committee 4/12/88.
%************************************;
```

So here we have the first solid result from the clinical study: Commenting macro code within the program code is one way to start to get Macroitis under control. The macro mysteries start to fade away once the process of the macro is explained.

Macro Coding Standards
Macro coding standards should be part of any SAS® coding guidelines already in place in your business or institution. While some still consider SAS® code as AD-HOC or one-off programming, there still is a place for structured programming that will not "cramp" the style of your programmers.

Macro Coding standards should contain all of the same elements that general SAS® coding guidelines have. Some principles that should be contained in coding guidelines are as follows.

<u>Indentation of code</u>
All SAS® coding should follow some indentation standard within the structure of the program. There is nothing worse than trying to determine the flow of a program that is difficult to read. These are simple standards to set in place and that will provide the first steps in controlling Macroitis.

1.) Each DATA or PROC should end with a run.

```
PROC PRINT DATA=DATAIN.LABS;
   VAR SUBJECT TESTNAME TESTVAL;
RUN;
```

2.) All statements after a DATA or PROC should be indented by 3 to 5 spaces.

```
PROC PRINT DATA=DATAIN.LABS;
   ID SUBJECT;
   VAR  TESTNAME TESTVAL;
RUN;
```

3.) All blocks of IF-THEN-DO statements should have the END statement line up with the IF.

```
IF SEX='FEMALE'  THEN
DO;
   IF PRGTEST='P' THEN RABBIT='DOA';
   ELSE RABBIT='AOK'
END;
```

4.) No more than one SAS® statement per line.

<u>Realistic or reserved data set names</u>
Output data set names should reflect the data contained in it. Data set names should not reflect pets, children, spouses, or other names. FIDO may be a loyal dog, however, as an output data set name

it certainly leaves something to be desired in its description.

If your macro library will create data sets to be used in programs, then your coding standards should provide a list of "reserved" names so that programmers will not use them in their program. For example, if a macro subsets a Drug Code Dictionary for specific drugs listed in the macro, then the output data set is called DRUGCODE. This name now becomes a reserved name and should not be used in other programs or in other places in the calling program where it refers to a different data set.

Modular Programming

This is something that we all learn about, but most forget to practice. Coding Guidelines can be used to "force" modular programming within your business and institution. Indeed macros that are written to perform a particular function fit easily into modular programming design. Some of the programs that could fit the design of modular programming are programs that print the standard headers and footers, specific calculations, or read master files. These can be brought into any program and easily modified. There are many uses for macros, but we must make sure that they have been well written and documented.

Macro deprivation therapy

Our clinical trials have shown that sometimes the only way to regain control of macro coding is to institute Macro Deprivation Therapy(MDT). Our approach with this therapy is first to require that all macros used in programs, or systems be validated. This guarantees that the macros used are fully documented and have gone through a series of validation steps. MDT also ensures that the macros are used in a limited manner and that macros generated by consultants are controlled and managed.

Part of MDT teaches macro sense to the afflicted. Macros should only be used when it makes sense to use them.

1.) It must make SENSE to "macro it".

2.) The MACRO should bring consistent results and save time.

3.) The MACRO should actually do something.

4.) The MACRO must be maintainable by more than 1 person.

5.) The MACRO should provide an explanation of all parameters.

6.) The MACRO should provide a clear and concise summary of what it does.

7.) The MACRO should be able to be used several times.

8.) The MACRO must adhere to programming standards.

9.) The MACRO must be planned with an obtainable result.

10.) The maximum amount of parameters must be set.

11.) All MACROS should have a 5 day waiting period. If the programmer can still understand why he/she wrote it, then it can be used.

Item 11 seem humorous and sounds like something that you would expect from the Surgeon General. Humorous as it may seem, it is a way to require justification. As programmers, we get so involved with the task at hand, that we no longer, see any other way to resolve a programming problem, except by macroizing it. Sometimes this works. Sometimes, though, it exacerbates the problem, requiring a programmer to justify the use of a macro simply provides the programmer with another way to resolve the problem. It has been our experience that after allowing time for this thought process at least 80% of the solutions are non-macro. It also benefits the programmer to look at other options within the SAS® language.

Macro validation protocols

There are several books which are available on the topics of system and program validation which were invaluable to our clinical study team during their research. These results determined that validation must contain at least three core components.

Programmer Validation

The first step in any validation process is the use of a validation check list. The programmer should have a checklist of items that the program must have before it moves into the next step in the validation process. Some of the checklist items are listed below.

1.) Are there any warning messages in the log?

2.) Are there any unintialized variables in the log?

3.) Does the program use the standard header?

4.) Is the program written according to programming guidelines?

5.) Is there sufficient documentation throughout the program?

Although just a sample, these are critical elements in a programmer checklist. The second component is the peer review panel which should convene after the programmer is satisfied with the code.

Peer review
A peer review panel contains only programmers. Under no circumstances should this panel contain supervisory personnel, especially the supervisor of the programmer submitting the program. A Peer Review Panel is not intended to be the vehicle for a programmer's evaluation. That should take place under other circumstances

The panel should contain a variety of programming experience, from the most experienced to the novice. Such an approach allows a diverse look at the program. The programmer should provide the panel with the program, program log, and output if necessary at least three to five days before the panel convenes. This will allow an appropriate amount of time for the panel to review and prepare questions on the code.

When the panel convenes the programmer should present an overview of the program. The panel should then ask any questions regarding the program including its overall function, documentation, and use of the appropriate PROCS and DATA STEPS. This should not be an inquisition but a learning experience for all parties. After the panel has examined the program they will be able to report one of three conclusions.

1.) The program can proceed to validation testing without change.

2.) The program can proceed to validation testing with one to three minor change (usually in documentation).

3.) The program may not proceed. It must be modified per the Peer Review Panel requests, and returned to the panel.

The Peer Review Panel signs a document with their findings, and turns it in to the supervisor. The Peer Review Panel findings are final and binding.

Validation Test Plan
The last step in program or macro validation is the validation test plan. The test plan validates the program performance, the accuracy and consistency of correctness of the requested program functions, and that the output generated (if any) is consistent with departmental or company standards. This should not be confused with programming testing during program development. The validation test plan provides formal testing of the code according to defined specifications, and is not the ad-hoc programmer testing procedure.

Most companies or institutions should have a Validation Protocol available that will assist in writing a validation-test plan. In addition the test plan should be drawn from the design specifications of the program. Those items which should be addressed in such a plan include:

1.) How does the program handle missing data?
2.) Are there error messages that will notify the user of a problem?
3.) Will the error messages stop the program from processing.
4.) Can anyone other than the author use the program?

There are more specific items in the design specifications which should be addressed within the test plan.

After the test plan is written and completed, a person other than the author of the program, should do the testing. This will assure that anyone on the programming staff will be able to run the program. Finally once the program is tested, it should be placed in a program or macro library that can be used by everyone.

Conclusions

This paper provided the identification of the signs and symptoms of Macroitis. It has also provided the tools needed to control macros, such as Macro Depravation Therapy. It has also provided some guidelines to follow for better SAS® coding practices. Documentation and validation are tools that our industry has used for many years. It is time that we integrated those tools with SAS® programming. As figure 2 illustrates, it is time to take control of macro coding skills instead of it controlling us. It is

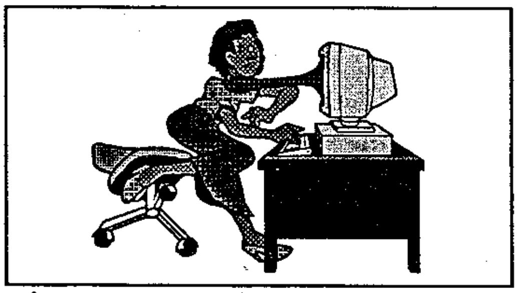
Figure 2

possible to ensure good programming guidelines within your business by adopting just a few of the processes discussed. For our part, we adopted many of these practices in our clinical study and we know they work!

Acknowledgments

I would like to thank **Billy G, Goat III** for his editorial acumen, **Donna Lucas-Mudd** for her attention to composition and structure, **Lynn Thome-Polingo** and **Joanne Laffer** for their technical expertise.