PharmaSUG2011 - Paper AD13

Beyond Double Programming
----SAS® Programming By Design (PBD) with Soop

Laiju Zhang, MDCI, MA, USA

Double programming has been the golden standard in the pharmaceutical industry. However, it has 3 inherent vital flaws: difficulty in implementation, confrontational working environments, and inefficiency and low productivity. Programming By Design, the proposed alternative programming paradigm in the paper, focuses on the systematic designing of the initial programming, the consistent testing and checking in the production period, and the repeated refactoring of code during the review period. The quality of the SAS programs will be enhanced, the productivity of SAS programmers will be increased, not as a result of comparison of two programmers' independent activities, but as a cooperation and mutual learning among programmers, statisticians, and other project members. Thus achieved is a deep understanding of the data in terms of the statistical analysis plan and the more efficient and correct realization of the analysis plan.

The PBD paradigm is illustrated by a systematic tool called Soop (pronounced as “Soup”), which is a software realization of SAS Object Oriented Programming guided by PBD principles.

Borrowing ideas from Java, C++, and C#, Soop specifies that a SAS program be organized in terms of methods, classes, and interfaces by using the available SAS macro facilities and by implementing DSL (Domain Specific Language) in the backend. With well designed SAS programs, it writes SAS programs based on the programmers' specs input via excels (or other interfaces); it then checks the programming rules to be implemented before submitting to SAS for processing, and it finally analyzes and summarizes the SAS log files after submission.

On the language tool level, Soop is built out of SAS compiled macros, VBScripts, Java, Perl, and Groovy language. In practice, the use of the tool only requires the user to know SAS basic and Excel basics. The important message it carries is that SAS programming activities can be fun and can be enhanced by learning from the general software development industry, not only on the language level, but also in the deep aspect of paradigm change - such as Extreme Programming, Programming by Contract, etc.

Key words: macro, SAS, OOP, Soop

PRESENTING MACRO SOOP

This section presents the SAS macro Soop . The macro reads an Excel file that carries the instructions for writing a specific SAS program. It then outputs a complete, ready-to-run SAS program. The output SAS program is no different from any regular SAS program. However, it leaves hooks as a convention that marks different steps for processing, implements a chain of tests at each step which results in either a pass or a fail. In the background, Soop cooperates with another facility (which can be implemented in SAS or in other languages) to collect the results of tests, make decisions about further actions, and issue commands to carry out these actions by programmatically modifying the SAS program.

The generation of SAS programs can be done either in a single mode, or in a batch mode. In a single mode, one program is generated according to the specification; in the batch mode, a series of SAS program is produced according to the instructions from the spreadsheet.

See %SASWriter.

BABY STEP (BS)

Baby Step (BS) refers to a block of SAS code on which a single test is conducted. It can be the simplest SAS step. But not all SAS steps are qualified BS. For example, a data step that only merges two datasets can be BS, but the

data step with additional data conversions and manipulations cannot. This approach may be against traditional programming wisdom emphasizing simplicity (less code is always better) and may invite criticism of too much coding than necessary. However, since most of coding is conducted by Soop automatically, more code does not necessarily lead to extra burdens on programmers in practice. By isolating each task from a comprehensive, overwhelming data step, an automatic test on a BS can be conducted and results can be easily analyzed and utilized for further code refactoring.

As one example for illustration, consider the following code snippet:

```
data subj_2;
  set subj_1;
  if HUNIT=1 then HGT_CM = HGT / 0.3937007874;
  else  HGT_CM = HGT;

  if WUNIT=1 then WGT_KG =WGT * 0.454 ;
  else      WGT_KG = WGT;

  label  HGT_CM = "Height (cm)"
        WGT_KG = "Weight (kg)"
                        ;
  all = 1;
  trt = 1;

  rename agerng = agegrp;

run;
```

According to SAS, there is one step involved – data subj_2. According to Soop, there are 7 steps involed: 1 set statement..2 if else statements; 2 assignments; 1 label statement, 1 rename statement. If we combine each of the above mentioned step with data statement, we can form a stand-alone data step. This is the unit for further testing.

The job of macro **%SoopBabyStep** is to read in the existing SAS file and split into these stand-alone SAS steps, ready for testing.

See %SoopBabyStep.

KNOWLEDGE ZONE (KZ)

To understand how Soop conducts its necessary testing, we need to explain the concept Knowledge Zone (KZ). The idea is about the general philosophy of establishing the criteria for writing each test. And it is very simple: pass what you know, fail what you do not know. Take race variable for example, in converting a char variable into a numeric variable with a format, you know from aCRF there are a limited number of possible values (“White”, “Black”, ...). In your logic, you let the test pass if the actual data are within the values list; otherwise you fail the test and send the results to your test results collector. This list of known values is called “Knowledge Element”. In addition to known values list, a knowledge element can be a known pattern. Take height for example, if you detect a pattern of digits combined with dot, then you can be sure that its unit is either cm or inch and you can use unit information to make necessary conversions. However, if the value is something like 5’8”, the pattern will be outside your known pattern (or the specified pattern in your program). Here, a known pattern is a Knowledge Element for your test. Based on a collection of different types of Knowledge Elements (values list or patterns), you can build more complex known elements set (called Knowledge Zone) to establish your test pass zone. Anything that that falls within KZ will be passed; any thing that falls outside KZ will fail. Note that in most cases, KZ only consists of one Knowledge Element; in rare cases, KZ can include many Knowledge Elements. In the best practice, it’s advisable to use single element KZ. In the process of defining a Knowledge Zone and “pass what you know”, you will be able to separate the problematic parts from the rest of your program.

As a result of applying KZ concept, SAS’s default handling of automatic conversion from numeric to character and character to numeric is not permitted in Soop. The default conversion is against KZ because what SAS does in the

back is not explicit. Thus, all notes such as “Character values have been converted to numeric” will disappear from log files and will be flagged as not permissible if any SAS program is coded to produce such notes.

%SoopTesting

PASSIVE PROGRAMMING (PP)

By passive programming, I refers to an approach by which you do not actively change your current program in order to use another one, a macro, for example. On the contrary, you let the macro you want to use to find your program and make decisions according to what your program is.

Take %output macro for example. In each SAS file, I like to place a %output to control the final product of my program: SAS dataset, a rft table, an xml file, statistic analysis results, etc.

The traditional approach is to pass a parameterized **outType** to control the type of the product, such as “ %output(outType=1.).” If nothing needs change after you’ve done your initial coding, every thing would be fine. When later you wanted to change the output type, however, you will feel the pain – imagine you had 50 or more files to open for editing!

An alternative approach is to only write a simple %output without specifying any parameter in the macro call. Within %output macro, you place necessary logic: if the SAS file starts with L, you do something; otherwise you do something else. Notice the paradigm change. In the old way, our Ms. Output is actively adjusting herself in order to produce the desired results. In the new way, our Ms. Output is just passively sitting there, accepting what comes out of the macro production. The behavior of output macro is not decided by the macro call, rather it is decided by the Type of the SAS file that makes the macro call.

Then we have the question: how to specify the Type of a SAS file? In other words, how to decide what a SAS program is? How does macro %output collect the SAS file Type information and behave accordingly?

1) File names – L_, D_, T_, F_ can be used to categorize the top level classification.
2) Maro variable information: purposely assign each file a Type for a particular use.
3) Contents of the SAS files (what datasets they use, what tests they conduct).
4) Meta data center – hard coded in an excel file or xml file or web page.

The great benefit is the program maintenance. When you later want to change the output type, you no longer need to edit the 50 SAS files that call the output macro. Rather, you only need to edit output macro and update instructions about what it needs to do when it comes to each of the SAS files. The passive programming is applied for all tests in Soop – all tests are implemented by macros, and macros can be used by many different tests and will behave differently according to different file types and other identifiers.

In summary, this paper presents an approach by which SAS programmers design the whole programming process, test each step as the process goes along, and constantly uses the feedback from the testing results to make adjustments. By doing so, doubling programming becomes a additional layer of testing for confirming result, not as a plausible requirement for quality control. Programming becomes a fun of creative thinking, not a constant thread of discrepancy from another programmer’s results.

STRUCTURE OF SOOP MACRO SYSTEM

To summarize, here is the table to list all the programs involved in Soop:

Macro name	**Function**	**Notes**
%**Soop**Main	start SOOP	

%**Soop**Writer	Write a program	
%**Soop**BabyStep	Split the program into baby steps for testing	
%**Soop**Testing	Test each baby step.	
%**Soop**Report	Report the test results.	
%SASReWrite	Re-write each step and combine each step back into one single program.	
%**Soop**Log	Logging each step.	Can be implemented using other languages.

Key Concepts

Abbreviation	**Full Name**	**Comments**
BS	Baby Step	
KZ	Knowledge Zone	
PP	Passive Programming	
Soop	SAS Object-Oriented Programming	

REFERENCES

(1) Example of 1*variable http://www.pauldickman.com/teaching/sas/char_to_num.php.

(2) Source Code

Laiju Zhang, PhD

Sr. Manager of Biostatistics & SAS programming/Sr Biostatistician
Medical Device Consultants, Inc.

49 Plain st, North Attleboro, MA 02760
(508) 316 7163

(508) 643-2237 Fax
LZhang@mdci.com
http://www.mdci.com