"Training Wheels" vs "Racing Wheels" for the Pharmaceutical Industry

Todd Case and YuTing Tian
Vertex Pharmaceuticals, Boston, USA

Abstract

Within the pharmaceutical industry there is a constant need for qualified and skilled and professionals at all levels. On one hand, there is never enough highly skilled senior Statistical Programmers; on the other hand, the industry is constantly changing so tasks that used to require senior programmers can more and more often be performed by junior programmers, leaving senior staff with more time to devote to the most complex problems. As the industry changes, senior managers must adapt to not only recruit quality programmers, but also create the environment that ensures these programmers are motivated to learn and advance both as individuals and as important team members.

This paper was written by an entry level programmer and a Director of Statistical Programming to demonstrate how to fuse the experience that Senior Level Programmers have with the talent, new ways of thinking and technical skills that Junior Programmers bring to the table. The origins of this paper go back to PharmaSUG 2019, when the Director of Programming sat in on a presentation by a student who was looking for a job in the pharmaceutical industry.

This paper has two goals. The first goal is to lay out a common perspective using a framework of three aspects that statistical programmers and experienced statistical programmer managers face. The second goal is to illustrate some example problems and examine how we try to solve these problems for statistical programmers across over different levels - from junior to manager.

Key Words: Pharmaceutical industry, statistical programmer, SAS rigorous program,
work with team, analysis

Introduction

This paper is divided into four sections which correspond to the steps outlined in our suggested process to build a logical picture of creating a blend of senior statistical programmers and managers that have a lot of clinical trial experience with junior level statistical programmers that have a lot of cutting edge ideas and more experience with new programming languages that may boost efficiency and innovation within the clinical statistical programming community.

- 1. Understanding the Industry
- 2. Developing rigorous code
- 3. Working with team
- 4. Recommendations

1. Understanding the industry

1.1 Case Report Forms (CRF)

Example of annotating the CRF quickly vs. understanding the raw database and the SDTM domains they map
Case Report forms are designed for a specific study to collect clinical trial data. The statistical programmer creates variable names in terms of index text designed on the CRF to help programmers map to SDTM domain. Therefore, the deeper you understand Case Report Form, the quicker you can annotate it with variables and the easier to create SDTM domains.

Examples of how to annotate a CRF (keeping SDTM in mind):

The CRF of vital signs shown below (Figure 1):

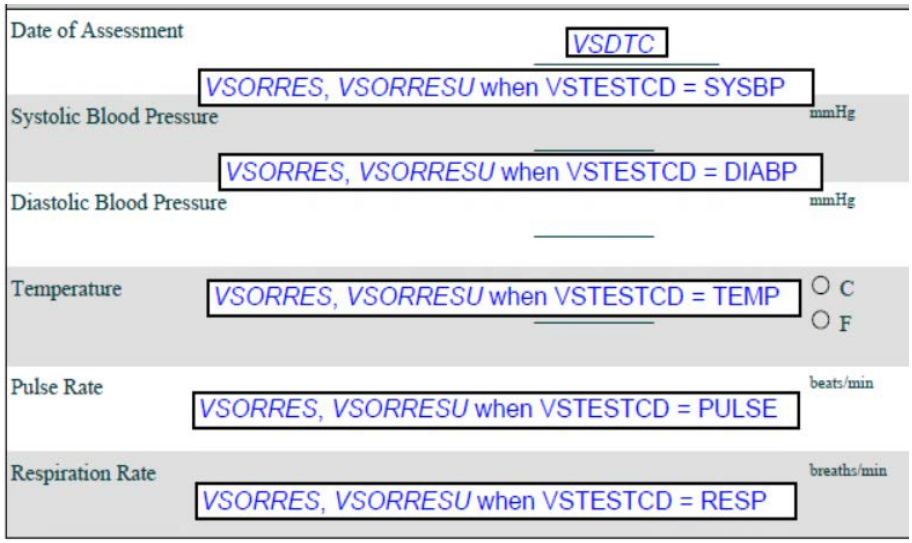

Figure 1

From the graph, we see those measurements the site will test, including date of assessment, systolic blood pressure, diastolic blood pressure, temperature, pulse rate and respiration rate. Statistical Programmers create variables to represent the test measurements. The VSDTC variable means date of assessment; when VSTESTCD's value is systolic blood pressure, we assign it to SYSBP; analogously, when index is Diastolic Blood Pressure, we set VSTESTCD as DIABP; temperature is set to TEMP, Pulse Rate is set to PULSE and respiration rate is set to RESP. In addition, with regarding to measurement value, we need to record measurement unit as well. In the dataset, we create VSORRES to represent units. For example, when we test Systolic and Diastolic Blood Pressure, the VSORRESU=mmHg; when we test Temperature, the VSORRESU separates to two units, C(Celsius) and F(Fahrenheit). When we test Respiration Rate VSORRESU is set to breaths/min.

1.2 Common Industry Terms and Basic Knowledge of CDISC (SDTM and ADaM)

There are some pharmaceutical terms and clinical trial processes we need to understand. As a junior statistical programmer, a primary role is to work with other programmers and use our technical and programming skills to enable clinical trial statisticians to perform their statistical analysis duties more efficiently. A simplified illustration of the general work processes of the statistical programmer shown on the below Figure 1:

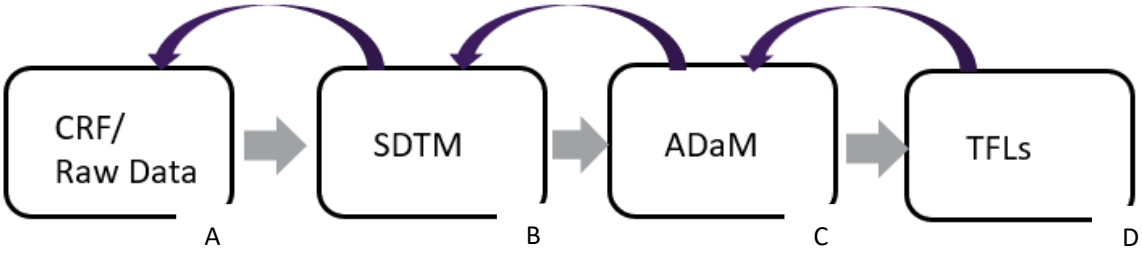

Figure 2

From Figure 2, we see the work process starts with CRF, designed for a specific study to collect clinical trial data. After the site management complete the CRF properly (using CRF Completion Guidelines), for each field entered on the CRF, a field in the database is created. After entering CRF data into a database (A), the data entry group has a standard operating procedure to ensure the data entry quality (e.g., such as double data entry). Then we create an SDTM domain (B) to group related information with common topics. Next we create ADaM domains (C) to support clinical trial statistical analysis and traceability among analysis results, analysis data, and data represented in the SDTM. Finally, we generate Tables, Listings and Figures (TFLs) which are included in the Clinical Study Report (CSR) to provide evidence to FDA about the study drug safety and efficacy.

1.3 Statistical Analysis Plan (SAP)

An SAP describes the planned analysis for a clinical data. It should be reviewed very carefully by statistical programmers who work on the project for clarity and comprehension to facilitate the creation of analysis datasets and prepare Tables, Figures and Listings. Some things to look for and thoroughly review and understand are:

a.) Visit Windowing (for safety and efficacy),

b.) Creating additional observations (e.g., an average for ECG triplicate measures),

c.) What the Primary efficacy endpoints and key safety analysis are,

- Complicated Endpoints (e.g., derived from diary/messy data, difficult statistical models, and composite endpoints (e.g., death, loss of function and organ failure))

d.) Last Observation Carried Forward (LOCF), if applicable,

e.) What is the treatment emergent and safety follow-up periods and how do they impact analysis

2. Developing More Rigorous SAS Coding Skills

2.1 SAS Programming Environments

As a programmer, we know the Microsoft Windows, Unix and Linux are client servers where your SAS session interfaces - working on the remote computer. While SAS will work on any of these operating systems, we often need programs that create our ‘setup’ environments (e.g., defining libnames, pointing to a global macro library, pointing to raw data, etc.), such as programs like autoexec.sas which are used to interface with such programs. It’s also very useful to have programs that define the conversion of original units to standard units, point to the current (and previous) versions of coding dictionaries such as WHODD and MedDRA, etc.

2.2 SAS Macro

Such as create more general datasets with start date and end date (come from different datasets); if we need date from DM, safety follow up, randomization date, loop much of different datasets.

SAS Macro is a powerful programming feature which allows programmers to avoid repetitive sections of code, and/or create dynamic code so that we don't always have to count the parameters or set the parameters to a fixed number in order to generate code. We can also use them again and again. As a statistical programmer, we should be aware of the situations in which we want to use SAS macro, when not to use, and how to create an intuitive macro that others will like to use.

The wide use of SAS Macros in the pharmaceutical industry includes examples such as: create dynamic code e.g., as referenced above, perhaps we want the macro to count how many datasets are in a library so that we can loop through them all without having to count the number each time – which can be done with existing metadata. Another situation when macros are incredibly efficient is when we use them to ensure corporate standards and consistency across different studies. For example: epoch means the interval of time in the planned conduct of study (each study team may use different approach and this code can be hundreds of lines long – a SAS macro used throughout the department can save thousands of hours in a single year!).

The code shown below is a simple way to create epoch variable when there is no missing time value in the dataset.

```
proc transpose data=se out=se_st(drop=_name_
_label_)
  prefix=st;
  id etcd;/*assign the value of etcd*/
  var SESTDTC;
  by USUBJID;
run;
proc transpose data=se out=se_en(drop=_name_
_label_)
  prefix=en;
  id etcd;
  var SEENDTC;
  by USUBJID;
run;
data cm2;
  length EPOCH $20.;
  merge cm1(in=a) se_st se_en;
  by USUBJID;
  if a;
  if CMSTDTC ne ""  then do;
    if stSCRN<= CMSTDTC <enSCRN then
EPOCH="SCREENING";
         if stPRE<= CMSTDTC <enPRE then
EPOCH="PRETREATMENT";
    if stTRT<= CMSTDTC <enTRT then EPOCH="OPEN LABEL
TREATMENT";
         if stFU<= CMSTDTC <enFU then
EPOCH="FOLLOW-UP";
  end;
 run;
```

Figure 3

From the Figure 3, we transpose Subject Element Domain(SE) first, then merge with Concomitant Medication Domain(CM) by subject id to create EPOCH variable in that specific domain.

However, when missing time values exist in the dataset, we need to change our code. In order to maintain code standard and data consistency, we use SAS Macro to create EPOCH and other complicated variables.

Example Macro Call to Create EPOCH Variable:

%***epoch***(indata =XX_input , datevar =XXSTDTC , outdata = XX_output);

In the pharmaceutical company, a macro we will call %epoch can assign a patient's EPOCH to study specific elements (SE). The SE domain contains variables USUBJID, EPOCH, SESTDTC and SEENDTC. Therefore, we only need to assign values for those specific parameters and, as such, EPOCH can be created efficiently for the subject XX. In the example macro call above, the parameter INDATA is an existing data set that will have the EPOCH column added; DATEVAR is the variable in the existing INDATA used to identify the epoch (DATEVAR is used to compare with the date in the SE domain in order to slot the visit to an EPOCH) and OUTDATA is the name of the final dataset containing the epoch.

One of the most useful (and powerful) macros used in clinical trials can apply all the dataset attributes (e.g., dataset and variable lengths, types and labels) can be used to apply the CDISC standards attributes to a every domain using the CDISC Implementation Guide (IG) values. For example, %applytrrib macro produces label and length of CDISC standard variables (and provides a warning if the type is not the same as in the CDSIC Implementation Guide) in the final CDISC domain automatically. This macro should first be run to read in the specifications spreadsheets and create analysis level metadata data sets containing the domain and variable information. For SDTM this macro will create SAS data domain_XX and var_XX in the XX sdtmXX folder in the spec folder; For ADaM, it will create SAS data sets domain_YY and var_YY in the adamYY folder. These data sets will be used to identify the variables to create in the output data sets and the attributes of those data sets.

Example Macro Call to Create CDISC Standard Variable Attributes:

%***applyattrib*** (data=xx, domain=xx);

Data=XX: The name of the SAS input data set that will be used to create the SDTM or Adam data sets
Domain=xx; The name of the SDTM or ADaM domain.

2.3 SAS code tips

Array statement

An array is a powerful construct in SAS® DATA step programming, allowing the application of a single process to multiple variables simultaneously. An array statement allows you to define which variables should be grouped together into an array. For example, we use array statement to avoid repeating blocks of code. Code show as below:

Stimulated Raw data:

```
data data1;
input height weight sysbp ;
datalines;
170 . 100
run;
```

Figure 4

The original code without using array statement:

```
data data2;
  set data1;
  length height weight sysbp    8;

  if missing(height) then height = 0;
  if missing(weight) then weight = 0;
  if missing(sysbp) then sysbp = 0;
run;
```

Figure 5

The code with array statement:

```
data data3;
  set data1;
  array orig(*) height weight sysbp ;
  do i=1 to dim(orig);
  put orig(*);
  if missing(orig(i)) then orig(i) = 0;
  end;
keep height weight sysbp;
run;
```

Figure 6

height	weight	sysbp
170	0	100

Figure 7

Figure 4 is a simulated dataset, three numeric variables height, weight, sysbp with values 100, missing, 170; the Figure 5 shows how do we create new three variables height height, weight, and sysbp without array statement; and replace blank values with 0. Figure 6 shows how the repeating codes can be replaced by array statement, all of the variables are the same type and length, thus the same code can apply to them. And Figure 7 shows the result of the code.

In addition, we use array statement to transpose data easily. For example: transfer data from horizontal to vertical. Figure 7 is our raw dataset in this case.

```
data data4 (drop=height weight sysbp);
  set data3;
  length paramcd $ 8;
  array value(2:4) height weight sysbp ;
  do paramn=lbound(value) to hbound(value);
  paramcd = vname(value(paramn));
  aval = value(paramn);
  if not missing(aval) then output;
  end;
run;
```

Figure 8

paramcd	paramn	Aval
height	2	170
weight	3	0
sysbp	4	100

Figure 9

From the Figure 8, we use the array statement to make sure the AVAL values match with raw dataset's values; PARAMCD match with the raw dataset's variables' names. Importantly, the order in which the variables are listed when defining the value array is important - specifying the lower and upper bounds in array statement can easily handle a string of consecutive PARAMN values that does not start with 1.

Within clinical trial data, PARAMN is the numeric representation of PARAM (the descriptive of the analysis variable), useful for ordering and programmactic manipulation. There must be a one-to-one relationship between PARAM and PARAMN within a dataset for all parameters where PARAMN is populated. For example, when PARAM is Weight (kg), PARAMN equals 2, when PARAM is Height (cm), PARAMN equals 3, when PARAMN

equals Systolic Blood Pressure (mmHg), then PARAMN equals 4.
In addition, Paramncd is the short term of the analysis parameter of PARAM. AVAL is the numberic analysis value described by PARAM.

Figure 9 shows the output of code from Figure 8.

Using Where Versus If/then

Where statement is applied before the data enters the input buffer, on the other hand, if/then condition statements are applied after the data enters the program data vector. Thus, the Where statement is faster because not all observations have to be read and it can be applied on variables that existed only in the input dataset. For example, in the data step:

Stimulated raw data

```
data raw;
input height weight
sysbp;
datalines;
170 .   110
120 50 70
run;
```

Figure 10

```
data data5;
   set raw;
   where weight>20;
run;
```

Figure 11

```
data data6;
   set raw;
   if weight>20 then output;
run;
```

Figure 12

height	weight	sysbp
120	50	70

Figure 13

In this case, we make up raw data to show in Figure 10, then we get the same result whether we use the where statement (Figure 11) or if/then statement (Figure 12). The final result shows on Figure 13 – using the where statement would be more efficient in the data step.

In addition, the where statement and if/then statement have their separate advantages and disadvantages as well. For example, the where statement cannot be used in input statement directly, because where requires variables from a dataset. On the other hand, the if/then statement can use in the Input statement. Where statement also can apply to the proc statement, but if/then cannot. There are some many situations when the where statement or if/then statement can be more advantages or practical. The one we choose to use applies to the specific conditions in a scenario.

Date and Time variables.

We check if there are missing values on variable reference (specific to clinical trial data, with the reference often being the start of first dose) date and time (RFSTDTC). Subsequently, we want to convert character format variable RFSTDTC to numeric format variable RFSTDT using the input function and format is8601dt.

We see the is8601dt just read ISO date(YYYY-MM-DD). Demonstrating the difference in these date/time formats, the is8601da. format is to read ISO datetime (YYYY-MM-DDThh:mm:ss).

```
data cm3;
  merge cm2(in=a) dm(keep=USUBJID RFSTDTC);
  by USUBJID;
  if a;
  if RFSTDTC ne " " then RFSTDT=datepart(input(RFSTDTC,is8601dt.));
  if CMSTDTC ne " " then CMSTDT=input(CMSTDTC,is8601da.);
  if CMENDTC ne " " then CMENDT=input(CMENDTC,is8601da.);
  if nmiss(RFSTDT,CMSTDT)=0 then CMSTDY=(CMSTDT-RFSTDT)+(CMSTDT>=RFSTDT);
  if nmiss(RFSTDT, CMSTDT)=0 then CMENDY=(CMENDY-RFSTDT)+(CMENDY>=RFSTDT);
run;
```

3. Working in a team environment

3.1 Ask questions!

Ensure your work is done and have time to review, review, review

Last, but not least, we need to assume anything we do can be done better (e.g., shorter code, easier to read, procedures that are not as complicated, etc.), particularly when we wear our Quality Control (QC) programmer hats. When performing QC, we need to assume the production programming is not correct, using our own independent QC program (based solely on the Protocol, Statistical Analysis Plan and Dataset Specifications) to validate.

3.2 Stretch yourself, ask your Study Lead if there is anything they need help with.

In addition to be a sophisticated SAS programmer, we also need to learn more about our respective industry (for the sake of this paper it's biotech and pharma, but each industry should follow the same principles), the company's purpose and how to help people.

3.3 Bring up issues you see or don't understand at the team meeting

There is a good chance that if you don't understand, someone else won't either! There is no such thing is a stupid question, is better to ask stupid question than stupid mistakes.

As a student, my success depends primarily on taking my own initiative. As a junior programmer, my success depends not only on my own initiative, but also on the overall team success.

In the clinical trial team environment, different team member are responsible for different roles. For example, the biostatistician creates the Statistical Analysis Plan (SAP, Table/Figure/Listing (TFL) shells and cross-functional data review plans; statistical programmer creates production and validation SDTM/ ADaM datasets, TFLs, and Data transfer timeline(s); other teams have other roles such as Medical Monitor, Medical Writer and so on.

4. Recommendations

Bringing on junior programmers requires some training investment in them, not only the hard skills of SAS programming but also soft skills related to integration into the company culture and a team environment. Whereas the SAS programming skills can be augmented through a variety of training methods, the soft skills require a stronger mentorship from senior programmers and managers. As companies build this mentorship attitude it will be embedded into the junior programmers so as they grow within the company they too will grow to be active mentors.

References

Using Macro Functions, Arthur L. Carpenter, California Occidental Consultants

https://support.sas.com/resources/papers/proceedings/proceedings/sugi25/25/aa/25p004.pdf

What's New in the SDTMIG v3.3 and the SDTM v1.7 ,Fred Wood, Data Standards Consulting Group A Division of TalentMine, Raleigh, NC

https://www.lexjansen.com/pharmasug/2019/DS/PharmaSUG-2019-DS-311.pdf

No Regrets: Hiring for the Long Term in Statistical Programming, Chris Moriak, AstraZeneca, Gaithersburg MD, Graham Wilson, AstraZeneca, Alderly Park, UK Elizabeth Meeson, AstraZeneca, Alderly Park, UK
https://www.pharmasug.org/proceedings/2015/MS/PharmaSUG-2015-MS04.pdf

Training Statistical Programmers on SAP Review Skills, Sascha Ahrweiler, UCB BioSciences GmbH, Monheim am Rhein, Germany

https://www.lexjansen.com/phuse/2011/is/IS01.pdf