PhUSE 2014

Paper PD02

Confessions of an obsessive compulsive programmer

Diana Stuart, Cmed, Horsham, UK

ABSTRACT

A lot of SAS programmers develop their own style of programming that can be as individual as their personalities, whilst being within the boundaries of good programming practice guidelines. With experience and knowledge, their programming style will change, eventually settling into something they rarely deviate from. Whether one programming style is "better" than another is subjective, however, some programmers develop a style that is not so much based on logic rather than habit, without questioning the reasoning behind certain practices.

As a reasonably experienced programmer, I have become aware that the way I program has become an almost obsessive pursuit of efficiency and logic. This paper covers why I have a preference for certain methods, with an emphasis on readability, reusability and simplicity.

INTRODUCTION

Obsessive compulsive disorder is serious. A condition for which there is plenty of ongoing research, and recent thought is that the habits might drive the obsession, rather than the obsession driving the habits. This means there is a distinct possibility I am in trouble. My first confession is that I am not really a sufferer of OCD, at least not to my knowledge, but I have habits, programming habits that are verging on the compulsive.

Most programmers develop a style. If you work within a programming team, and are exposed to each other's code, over time you will know who wrote a program without having to look at the header box.

The point of this paper isn't to impress upon you my style of programming as the only way of programming. Far from it. What I would like is for more programmers to consider their style, and to think about their habits.

Fully acknowledging that some people have a natural talent, they see the problem and immediately know how to get to the solution. If we all put our egos to one side for a minute, the reality is that this type of programmer is rare, extremely rare. We all know how to solve problems, but do we always know the best method without having to take the time for proper consideration of the problem? The answer is no. Good programmers are usually good because they work hard at their craft, try new things out and learn from their mistakes.

"Anyone who has never made a mistake has never tried anything new."
– Albert Einstein

"Success does not consist in never making mistakes but in never making the same one a second time."
– George Bernard Shaw

With the SAS programming job market being what it is; i.e. demand far outstripping supply, and in particular I am referring to the contracting market, there are a number of programmers who don't have the opportunity to learn from the shortcomings of their code. Principally this is because they are no longer working for the same team or company. This is not meant as a criticism of contractors, but rather an observation of why programmers who change their employment position frequently, may well be completely unaware of their legacy.

I once heard that it was common for architects to never go back to a building they had designed. In other words they have no concept of the impact their design has had on those who actually have to live or work in one of their designs, both the positive and the negative. What has this to do with code creation? It is one thing to write code to get the job done. It is another to write code that is appropriately efficient, easy to read and easy to amend.

Once you've had the challenging task of untangling someone else's unwieldy code to make updates, you may start to consider your own style more closely. Repeat that task many times over, and like me, you may find yourself becoming fastidious about indentation, program layout and styles favoured as more readable.

RETURNING TO THE SCENE OF THE CRIME, ALSO KNOWN AS; ……AND YOU WILL KNOW US BY THE TRAIL OF THE DEAD

Whilst we don't always have the opportunity to right our wrongs, it is a good thing that we sometimes learn a lesson or two from our bad code, as we improve and better ourselves. What we should all bear in mind is that the world of SAS programming, particularly in the pharmaceutical industry is small. Gaining a reputation for unusable code is obviously not a good thing. As a lead programmer, I want programmers on a project who I know will code in a way that another programmer can come along and pick up or make amendments with the minimum of fuss. This I am aware is overly idealistic, however, we can all strive to be better programmers. Just getting the job done is fine if you want to live in the world mid-table mediocrity, but if you want to be a "good" programmer, your code can't just do the job.

CONFESSIONS

The habits I have developed often are a result of frustrated trouble shooting of code taken over from another programmer, and occasionally even my own.
Indentation is probably in every good programming practice ever written. Am I going to impress upon you the virtues of indenting 2 spaces over 4? Absolutely not. In fact I'm not even going to criticise those crazy cats who indent 5 spaces. The point of indentation is to make code more readable. Readability of code, is like grammar in language. You can write all the words you like without any punctuation, the chances are that the reader will get the message of what is written eventually, punctuation means that the reader gets there quicker, usually on the first read through. Being able to easily identify the start and end of a datastep is good, being able to see the start and end of loops is good, this means lining them up and indenting the code inside them so it is easy what happens where. It also means that you are less likely to forget your end; quit; or run; statements.
A lot of good programming practice guidelines recommend using a style consistently throughout a program. This often refers to use of upper and lower case in code, and use of character or symbol operators in quantitative statements. In my experience consistency is rare. A lot of programmers like to use a combination of "=" and "NE" in their IF and WHERE clauses. My personal compulsion with this is that I like to use EQ, for the simple reason that wherever I see an "=" being used in my code, I know that I am creating a variable with a value.

Should you be unfortunate enough to sit near me in an open plan office, the most often cause of hearing me scream is KEEP and DROP statements hiding amongst the main body of code in a datastep. My personal obsession with these is having them stand out, and that means putting them in plain sight in the first statement of the datastep, or where the output dataset is created. Whilst SAS does colour KEEP and DROP a nice blue colour, it also colours many of the other statements in the body of the data step blue. When a datastep is short it's easier to spot, but the cause of my screams are usually when it is seemingly randomly placed amongst a selection of statements in a long piece of code.

For example:

```
data qc_a_ae;
  set final ae;
  if nmiss(aevsttd,trtstdtm) eq 0 then do;
     if aevsttd ge datepart(trtstdtm) gt . then aesttdy=aevsttd - datepart(trtstdtm)+1;
     else aesttdy=aevsttd - datepart(trtstdtm);
  end;
  if nmiss(aevstpd,trtstdtm) eq 0 then do;
     if aevstpd ge datepart(trtstdtm) gt . then aestpdy=aevstpd - datepart(trtstdtm)+1;
        else aestpdy=aevstpd - datepart(trtstdtm);
  end;
  drop aevmdra pt_term poc_term aevstdtm;
run;
```

Versus:

```
data qc_a_ae(drop=aevmdra pt_term poc_term aevstdtm);
  set final_ae;
  if nmiss(aevsttd,trtstdtm) eq 0 then do;
     if aevsttd ge datepart(trtstdtm) gt . then aesttdy=aevsttd - datepart(trtstdtm)+1;
     else aesttdy=aevsttd - datepart(trtstdtm);
  end;
  if nmiss(aevstpd,trtstdtm) eq 0 then do;
     if aevstpd ge datepart(trtstdtm) gt . then aestpdy=aevstpd - datepart(trtstdtm)+1;
        else aestpdy=aevstpd - datepart(trtstdtm);
  end;
```

```
run;
```

I am a big fan of using KEEP and DROP statements. I like to tailor my data as much as possible as I bring it into a datastep. I also restrict my observations with WHERE statements as much as appropriate, so that I only have the data I am interested in. Other programmers like to have, what they might refer to as, freedom of choice. To bring in everything and deal with it later in the data step or even further down in the program. Their preference is for having everything available, just in case. As previously stated there are no absolutes in this, only consideration of what works for the individual programmer and that there is logical thinking behind it.

MAKING AN IMPRESSION

Some programmers program as if no one else will ever see their code, the reality is that programs are adapted and reused. If you are a programmer and are under the misapprehension that you and your code live in your own bubble, then then following is probably news to you. When other people look at your code, like it or not they are forming an opinion.

> "What you wear is more important than what you say, and if your shirt's not pressed, impressions are instantly formed about you. That's why I wear nothing, I say nothing, and I network with nudists."
> – Jarod Kintz

An auditor may not know what the code does that they are looking at, but they know how to read a header box. A dataset delivered to a customer (either internal or external) will be checked against its specification. Code delivered to customers used to create analysis, could be reviewed. As a lead programmer having worked for both pharma companies and CROs, I am constantly looking for signs that give me confidence in what I am looking at. Attention to detail is one of those signs. Are the header boxes correct and up to date? How are variables ordered in a dataset? How much of the code I am looking at is complying with good programming practice? If there is a general feeling of chaos, how confident am I about the product of the code that I am looking at? I am not naive enough to believe errors don't happen in programs with a neat header box, but troubleshooting is much easier when programs are readable, version history is documented and there are sufficient comments. If the attention to detail is not there, I am going to be spending much more time checking the quality of the deliverable.

Not only are lead programmers and customers going to be looking at your code, your peers are too. Your programs are going to be amended, and it is important to understand the impression your coding style makes on your peers.

Returning to my obsession with tailoring of data, this is partway explained by my perception of the programmer's understanding and knowledge of the data that they are using. Repeated use of unnecessary data steps, in particular sorting data unnecessarily leads me to believe that the programmer may not have taken the time to understand the data they are bringing into the program, rather they are approaching the program with an attitude of chance. Some of these sorts of habits are born from bad experiences with bad data, this then moves the debate into whether habits that program around bad data are a good thing? There is a fine line between defensive robust code, and programming around bad data management. There is always going to be an ongoing dialogue between programming and data management about programming around issues, and inexperienced programmers will need to learn when to push back.

It is quite possible that something that looks like an unnecessary datastep is entirely necessary, but that is what commenting is for. Anyone who has troubleshot a program, maybe their own or another's, and has got lost in the code will understand the importance of commenting. There is an art to commenting; making comments helpful, using them to divide up code into useful sections and maybe not too much of stating the obvious, but helping the reviewer understand what is going and most importantly why.

CONCLUSION

A lot of us develop habits and compulsions, understanding the logic behind them is important. Being open to new ideas is important. Appreciating that perceptions are formed by others should prompt programmers to look at their code from an outsider's point of view, and be committed to creating readable and reusable code.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Diana Stuart
Cmed Clinical Services
Holmwood, Broadlands Business Campus, Langhurstwood Road
Horsham, West Sussex, RH12 4QP, United Kingdom
Work Phone: +44 (0)1403 755432
Fax: +44 (0)1403 755051
Email: dstuart@cmedresearch.com
Web: www.cmedresearch.com