

Quality Improvement in SAS Coding: Reducing Defects

Linda M. Quinn
Case Western Reserve University

ABSTRACT

How much time did you waste debugging your own code only to find a silly error? How much effort did you expend trying to understand another programmer's code only to write your own from scratch? Why not apply Quality Improvement techniques to SAS coding? The simplest concept to implement is the defect chart. Defect - an imperfection that impairs worth or quality. First, I will develop a defect list from my own code, experiences, and horror stories. Once established, I can show how to apply defect lists to improve programming quality and minimize debugging time. When applied to my own code, a defect list helps me to remember what I should or shouldn't be doing and makes debugging my code easier. When applied to another colleague's code (until now, unbeknownst to them), a defect list allows me to develop a profile of that programmer: how to read their code; where to look first for errors; where to improve individual training; and what priority / effort should be expended to find their errors. Lastly, when supporting a multitude of users, defect lists can be used: to determine which courses would be of greatest value; to develop company specific programming standards; and to chart internal quality improvements.

INTRODUCTION

Personal Quality Checklists were first introduced in *Quality is Personal* by Roberts and Sergesketter. Personal quality is the application of quality ideas to your own individual job (and even to your own personal life). Personal quality helps people to perceive, and remove waste at the levels of their own jobs. As a result they

can achieve both continuous improvement (repeated small gains that add up to substantial gains over time) and breakthrough (substantial gains in short periods of time). This can be done whether there is a quality management philosophy in your company or not. There are three potential gains to the individual: increased job satisfaction; education in quality management concepts and tools; and assessment of the usefulness of quality management concepts and tools based on direct experience rather than claims of others.

A quality checklist (or defect list) is a means of recording shortcomings or defects on a collection of key personal activities that an individual wishes to improve. What better to improve than SAS coding? Let's first start by improving our own coding; then, we can start to work on other's code.

DEVELOPING A CHECKLIST

The first step is develop a checklist. The idea is to choose items for your list that you wish to improve upon. Of course, sometimes the list could seem endless. Start by picking those that are most serious or those that you think you can easily improve upon.

SOME CHECKLIST RATIONALES / ITEMS

Program style refers to the general approach used when designing and coding. Guidelines are often imposed by an employer or client. In the absence of those, programmers usually develop their own

unique style. Consider some of the following:

- Structured programming
- Separation of tasks into different datasets or PROCs
- Reprogramming current SAS functions
- Extensive and unnecessary nesting

Editing style refers to the layout and readability of the program. Once again, in the absence of guidelines, consider some examples:

- Lack of indentation
- Multiple statements on the same line
- Statements broken across lines

```
DATA TESTCASE; SET TRIAL.TESTCAS2
TESTCASE; IF X=5 THEN DO; Y=8*X;
Z=9+Y*X; END;
```

Naming conventions can be the most important part of good documentation and programming. Choose appropriate variable and dataset names. If you have to continually look to see what your variables stand for then maybe you could find more appropriate names.

- Variable names like X152 may be less helpful than AGE.
- A variable called MALE that equals zero when male and one when female may not be the best choice of variable names.
- A dataset named DATA may not be as illuminating to the reader as one named LASTYR although it might be better than a dataset named SET.
- Remember certain letter and number combinations just shouldn't go together. Consider I & 1 (testI vs test1), O & 0 (testO01 vs test001), H & I (HHIHIIH), V & W (VWWWV), and sometimes Z & 2 (QWXZRT vs QWX2RT).

```
DATA SET;
MERGE SET DATA;
BY BY;
IF SEX='F' THEN MALE=1;
ELSE MALE=0;
TESTO1 = OOOO * WWWW;
```

Comments are an invaluable documentation tool. Use them appropriately and liberally. Comments are supposed to help a reader follow what you are doing, not confuse or hide what is going on.

- Nested commented code is dangerous to the uninformed.
- Comments can be an easy snare when combined with bad coding habits mentioned above.

```
* WHAT EXACTLY IS THIS CODE DOING?;
* SET UP THE DATA FOR; PROC REG;
* AFTER ADJUSTING FOR LACK OF FIT;
```

```
* AND HOW ABOUT THIS CODE?;
```

```
DATA TWO;
SET ONE;
X = Y + 150;
* NOW CREATE THIRD DATASET
DATA THREE;
SET TWO;
Z = X + Y;
```

Labeling all variables and datasets is also a nice documentation feature. It may be a good idea to require a label statement for each variable or dataset created. This may be time consuming at the first pass through a dataset yet worthwhile.

- Labels such as LABEL X152="VARIABLE 152"; may not be too useful.

Choice of statements, commands, and options can also be a valuable resource or a tremendous stumbling block.

FIGURE 1.

Area	Wk 1	Wk 2	Wk 3	Wk 4	Wk 5
Unmatched quote	/				
TABLES statement	//				
variable labels					
dataset labels	/				
can't figure out code	///				

- The DROP and KEEP statements can reduce dataset size and dataset management.
- The RUN statement can effectively separate data steps or procedures.

SPECIFIC ITEMS FOR A CHECKLIST

Depending on your level of expertise, operating system, and your improvement goals, the items on your checklist might include:

- Missing semi-colon
- Colon instead of semi-colon
- Misspelled word
- Unmatched quotation mark
- Unmatched comment
- Missing run statement
- Mishandling missing values
- Data statements out of place (in a Proc step)
- Bad variable or dataset name
- Lack of readability
- Treating character variables as numerics or vice-versa
- Re-using a variable name inadvertently (or purposely)
- Failing to label a dataset
- Using the same dataset over and over and over and over

Your checklist doesn't need to be long and exhaustive. Pick a few items at a time. Try to improve those first. Then improve other areas. That is the whole concept behind Continuous Quality Improvement (CQI). Let me give you my personal checklist, as of today.

- Unmatched quotation mark
- Use of VAR statement instead of TABLE statement on PROC FREQ
- No labels on created variables
- No labels on created datasets
- Lack of comment on code I went back to and couldn't figure out

WHAT TO DO WITH YOUR CHECKLIST ITEMS

Create a simple checklist form where you just tally the number of defects you have each day (Figure 1). You can keep it on an index card hanging by your terminal or in your pocket protector. The idea behind a checklist is to be a simple reminder of what you are trying to improve. If you have a system that is so elaborate that it takes a disproportionate amount of time to record your defects or is so difficult that you won't keep track of them, then it's not going to work. Simplicity.

Now you can measure how you are doing. Hopefully, just tracking your defects will cause you to make fewer. Are your defects going down over time? Which defect should you emphasize or try to fix?

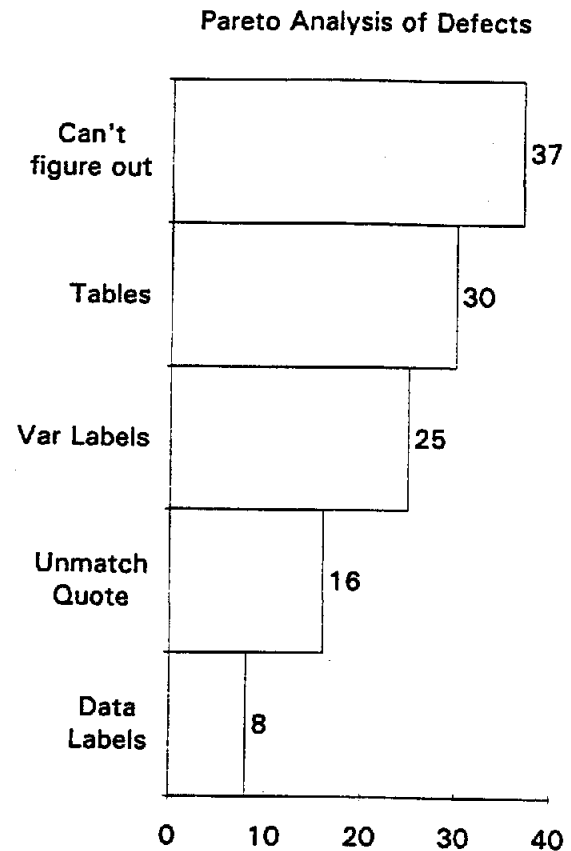
The first time through this exercise, I had a huge list that I wanted to pare down. I really had no idea which of my coding defects occurred most frequently or how prevalent any of the ones on my list were. Use a simple Pareto chart to determine which defects are the biggest problems. A Pareto chart is a bar graph that is organized such that bars are in decreasing frequency order. Obviously this puts a rank order on the defects that are your largest problems. Figure 2 shows that my worst problem area is labeling datasets.

Reduce your checklist to a few items. Now that you know what your biggest defect contributors are, try to reduce the defects in those categories. This may mean getting additional training in-house, taking a Programming Efficiency course from a SAS training vendor, or making a conscious effort to minimize those defects. Continue to track those items on your defect list. Sometimes just tracking them creates a pseudo-Hawthorne effect. Periodically check how you are doing. Figure 3 shows, using a simple run chart, that, thank goodness, my defects are coming down over time.

Hopefully, over time you are decreasing your defects. As they reach an acceptable level (near zero), refine your checklist. If you increase or remain the same, reevaluate why your current strategies to reduce defects aren't working.

The classic tool to monitor quality is the control chart. While this can be a very formal analysis, it can also be insightful. A c-chart for my defects could show points in time where I was out of control (so to speak). Looking back, I could pinpoint the

FIGURE 2.



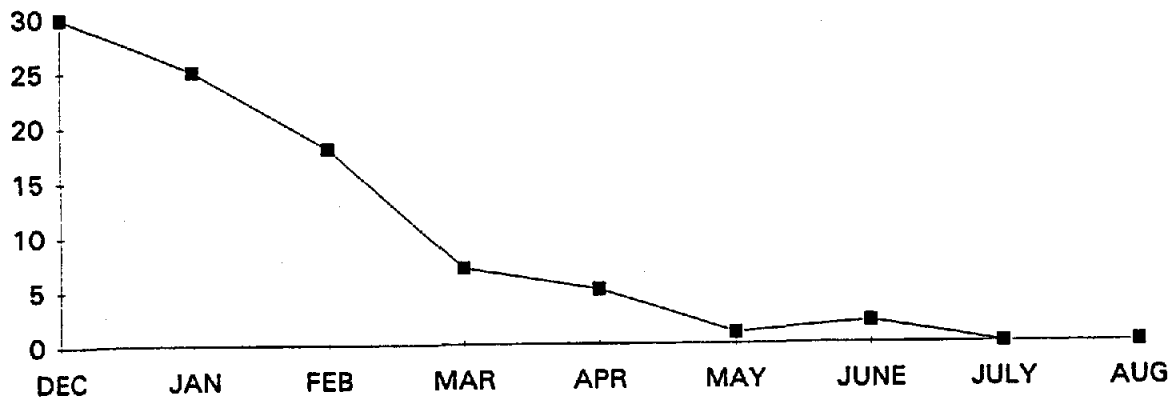
cause as abstract time. When abstract time hits here, everyone needs everything yesterday. It doesn't leave me much time to be responsive and a responsible coder. However, this graph should give me some ammunition to get my boss to leave me alone for a few days after crunch time to clean up code before I forget what I have done.

IMPROVEMENT

Checklists can achieve improvement by two distinct mechanisms. Checklists allow you to determine what the underlying causes of waste are. Behavior modification can be achieved by the psychological impact of carrying and tracking your defects. The first mechanism, analyzing underlying causes, is usually associated with

FIGURE 3.

Run Chart for the "TABLES" Defect over Time



continuing small gains that accumulate over time to large savings. The behavioral mechanism can be associated with breakthrough gains and re-engineering. The behavioral mechanism may not work for everyone and neither may the analytic mechanism. But you won't know unless you try.

TIPS ON USING MAXIMIZING USEFULNESS

Many people make progress within a short period of time. In general, some people are satisfied with short-term improvements and abandon their checklists after a few weeks or months.

Here are suggested tips to help make this exercise more effective:

- Take it seriously. I'm sure many of you think this is a light piece of TQM fluff, but it won't work unless you try.
- Simplicity. Don't wait until you have a long laundry list of items. Just start.
- A short checklist with one good standard beats a long checklist which will have little impact.
- Waste reduction is a principle component of the list.

- Don't have vague descriptions of your checklist items. You need to be able to identify a defect when it happens.
- Identify checklist items with causes not symptoms.
- If you don't want to fix it, don't include it.
- Checklist items are not New Year's Resolutions!
- Don't try to do everything at once. That's why it is called CQI.
- Don't delay until you have time to do it. You'll never find the time.

OTHER PROGRAMMERS

When I look at another programmer's code it is usually for one of two purposes: to understand and use it or to debug it. In either case, it helps to understand what defects the other programmer has. So why not develop a profile of each programmer?

Consider Elvis, a SAS novice who is trying very hard to learn. However, he is constantly in your office asking you questions about code that didn't work. First, he couldn't figure out what a SAS library was. Then, he blew away a SAS dataset without realizing it (good thing for backups) and was getting zero observations. Finally, he was creating

variables in PROC steps. You can't even count the number of times you went in search of missing semi-colons - the scourge of every novice.

How about the not so new coder who knows enough to be dangerous? Consider Mick. Mick learned about the DROP, KEEP, and IF-THEN-ELSE statements but doesn't quite understand them (although he thinks he might). He drops variables only to use them later in subsequent data steps. He comes to you with spaghetti code that would feed an army and a printout that has so much missing data that you are picturing black holes in space.

How about the lazy programmer, Diana, who never wants to look hard for her errors but knows you can find them in an instant. Almost always, they are the same type - misspelled variable or dataset names.

Then there is Aretha. She's been using SAS for 12 years and is very good at the data step. In fact, she usually figures out your problems. When she comes to you, you know it's not a routine semi-colon. So now I keep a list of defects for Elvis, Mick, Diana, and Aretha. After a quick Pareto analysis, I know that Diana's biggest defect is misspelled words (probably secondary to bad choice of variable and dataset names); Elvis's problem areas are missing semi-colons and erratic comment blocks; Mick has problems with variable definitions especially involving missing values. Aretha's problems don't seem to fall into any one area except that I can't read her code very well since she runs everything together.

Debugging. I have been asked by Elvis to look at some code that he wrote that isn't working. I can take a quick look at his code and see that he accidentally commented out the PROC MEANS statement and so his error about statement out of place is really

correct. Sometimes I even make him look take his log and highlight what he *thinks* are comments and I just look there.

I need to double check, or ask Mick to double check, his coding for a regression he presented with 43% missing observations. I'll ask which of the regression variables he created and which ones are original database variables. Before, I search for problems, Mick needs to run FREQS or MEANS on all the regression variables. Now I'm set to debug his code. In fact, last time I didn't have to ask; he already ran them and was bringing them to me for confirmation of what he thought was the problem.

When Diana whines that her dataset is empty, I can see that DATA NEW; SET T95AUG01; should really be: DATA NEW; SET T01AUG95; Thank goodness, SAS recognizes PORC as PROC or it would be a long day.

Aretha's questions are always time consuming since I really need to look hard at her code - so it's not something I can take a quick second to look at. Her problems aren't the simple one but her style makes finding them even more difficult. Usually I learn something though which makes trouble shooting her code fun.

Now that I know the weaknesses of my colleagues, I have noticed that my external debugging time has decreased.

Using and Understanding. How can profiles help if I am taking over code, either for it's support / modification or to use in a different application?

Well, I know that I need to walk through Elvis's code to make sure it was doing what he thought it was. Sometimes, I know it is easier and more time efficient to just start from scratch.

I need to look strongly at newly created variables in Mick's code. He has a habit of ignoring what should happen in missing cases or using faulty IF-THEN-ELSE constructs.

I need to examine the log of Diana's program to check for warnings that might have been ignored. It's quite possible the dataset didn't perform properly.

For Aretha, I need to spend time reformatting her code so I can read it.

WHAT NEXT

Remember that Quality Improvement is a continuous and on-going process. It doesn't stop when you've shown some improvement. However, if you are doing it correctly, the largest gains in quality are in the beginning when there is the most to improve.

How can all this information be helpful to you? First of all, it is easier to make informed decisions with data (although I have been in places where this is the exception to the rule). You can now show your supervisor that you have been improving yourself and your coding by working smartly, reducing defects, and improving your coding efforts. Not only can you show, you can quantify. This should directly relate to programming efficiency and improved documentation skills. But don't count on that being directly related to monetary rewards.

This may also show justification that a training course, SAS specific, may be appropriate. After all, you recognized that macros were one of your weaknesses; you tried a course of improvement; it isn't showing much success; maybe, a course is in order. After the course, you can continue to track to see if the course truly was beneficial.

SUPPORTING A GROUP

Tracking phone calls on the support line, number of questions on the same subject, repeat customers by using quality checklists is a way to be responsive to your customer while at the same time developing strategic plans for overall and long-range improvement. What company wouldn't like that? Now you can test how successful your in-house classes are. You can recommend which classes should be offered. You can recommend which people should take them.

FINAL NOTES

This approach to personal quality improvement is good preparation for organizational quality improvement projects using teams. This can become a foundation for developing team-wide or corporate-wide programming standards. If team members and upper management have tried waste removal at the personal level, they will be better prepared to detect organizational waste. Also, they will understand that quality improvement does not need to be slow and tedious.

Now I am sure some of you are skeptical that I actually do this. Truth be told, I think it is a great idea and I have made a good effort to do this for myself. My defect list is rather short and very similar to the one I showed earlier (not due to my lack of defects by any means). I have seen a decrease in the number of defects. Has it gotten me anywhere? I think the biggest advantage to following this approach is a reduction in debugging time, a decrease in the amount of time it takes me to go back to a project that I haven't worked on in a while, and a better ability to manage multiple programs simultaneously.

How has it helped me with my colleagues? After reading the abstract, my colleagues were a little concerned that I was *actually* doing this. They know me well enough to

know that I quantify and analyze everything. I control chart my gas mileage, I track the drive time to work, and I have data on canned food consumption at my house. I know how to teach SAS coding techniques to our novices. I know that I shouldn't let sloppy habits slide when I review other programmer's code, especially if I will be debugging or taking over their code. Do I keep defect lists for my colleagues? I need to keep my inquiring colleagues on their toes - while most aren't written, there are definitely mental lists. How's that for a vague answer. But I will tell you that I can find their errors pretty quickly and sometimes without getting out of my chair and looking at their code.

As you can tell, there wasn't too much high powered SAS code in this presentation. Maybe you just found it an interesting piece of humor (I guess those Cleveland Kominex-tary's are rubbing off, eight pages with one simple take home message). The take home message here is that everyone can improve their coding habits. Sometimes we don't even know what our bad habits are. Maybe the simplest way to find out is to get an outside, honest critique of some of your coding. It's a humbling experience, but if it makes you a better programmer...

REFERENCES

Roberts, HV and Sergesketter, BF (1993), *Quality is Personal: a Foundation for Total Quality Management*, Free Press.

PRODUCT INFORMATION

SAS is a registered trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

AUTHOR CONTACT

Linda M. Quinn
Research Associate
Case Western Reserve University
Division of General Internal Medicine and
The Program in Health Care Research
1286 Gary Blvd.
Brunswick, OH 44212-2912

Phone: 216-791-3800 x4814

Fax: 216-231-3476

e-mail: lmq2@po.cwru.edu