A Quality Control Approach for Statistical Computer Programs: How to Save a Million Dollars

Darryl V. Creel[1]
[1]RTI International

Abstract

There is information that provides very good high-level guidance for different aspects of a statistical process. This high-level guidance is typically implemented by computer programs. Consequently, statistical projects can be viewed as a software development process. This paper provides a cost effective quality approach for translating high-level guidance into well-documented and understandable statistical computer programs. The quality approach emphasizes communication and consists of three types of interrelated documentation: flow charts, text files, and computer programs. All three types of documentation communicate the overview, specific tasks within the overview, and detailed steps within the specific tasks to facilitate understanding the statistical process and ensuring quality control. The flow charts provide a graphic representation of the statistical process, and the text files provide a written description of the statistical process. The computer programs provide the detailed implementation of the statistical process through a self-documenting statistical program using variable naming conventions, commenting, and program structure. The interrelated nature of the documentation promotes quality statistical programs, provides opportunities for the development of junior staff through technical mentoring, and encourages standardization of routine processes to reduce the potential for errors as well as make the work more efficient.

Keywords: Quality Assurance, Quality Control, Software Development, and Statistical Programming

1. Introduction

"Although it might seem that the best way to develop a high-quality product would be to focus on the product itself, in software quality assurance you also need to focus on the software-development process." – Steve McConnell[1]

This paper provides a cost effective quality approach for translating high-level guidance into well-documented and understandable statistical computer programs by focusing on the software development process. **That is, this approach can save money and produce higher quality statistical programs by facilitating understanding, reducing errors, and minimizing the amount of time spent reworking code.** The goal of this approach is to find errors as early as possible in the software development process through continuous communication with clients and other stakeholders. Consequently, communication with a focus on quality should infuse every aspect of the project, starting before the project begins and continuing until after the project is complete. Understanding that improving quality reduces development costs "depends on understanding a key observation: the best way to improve productivity and quality is to reduce the time spent reworking code, whether the rework arises from changes in requirements, changes in design, or debugging."[2]

Our quality approach is an iterative refinement approach that takes high-level guidance and translates it to specific detailed tasks through continuous communication. Our quality approach consists of three interrelated types of documentation that facilitate quality through continuous communication:

(1) a flowchart showing what is to be accomplished;

[1] Steve McConnell, *Code Complete, Second Edition* (Redmond, WA: Microsoft Press, 2004), 466.
[2] Ibid., 474.

(2) a text file explicitly stating what is to be accomplished that contains a summary and a detailed section for each component to be accomplished; and
(3) a computer program that uses computer programming techniques to facilitate the understanding and quality control, e.g., variable naming conventions, commenting, and program structure that result in a self-documenting statistical computer program.

The flowcharts and text files are developed before the programming begins. They start with an overview and are iteratively refined into smaller more detailed steps that are further refined into even smaller and more detailed steps. For tasks in a very large statistical process, the statistical programming would begin after this iterative refinement process is essentially complete. This ensures that the programming does not start until the specific task is finalized and documented, resulting in better quality control. Since we know what should be programmed, there will be significant savings compared to starting the programming, making a change or finding an error in the statistical process documentation, and then having to go back and start the programming over again. It is important that you do not start programming without a plan.

Section 2 examines the importance of understanding the high-level guidance before programming is even considered. Section 3 shows the graphical representation of a statistical process by use of a flowchart. Section 4 describes making the written representation of a statistical process understandable and consistent with the flowcharts, and refining the text associated with the flowcharts to pseudocode. Section 5 provides an example of computer code using the techniques discussed in this paper.

To illustrate the statistical process, or software development process, we will use an example based on SAS that implements post-data collection adjustments to the survey design weights, i.e., the inverse of the probability of selection for the sampled cases and zero for the non-sampled cases. For this example, the post-data collection adjustments are an unknown eligibility adjustment, a nonresponse adjustment, and poststratification.

2. High-level Guidance

"Many of the problems encountered in the software development are attributable to shortcomings in the processes and practices used to gather, document, agree on, and alter the product's requirements." – Karl Wiegers[3]

The high-level guidance, or requirements, provided to the programmer could be something as simple as the need to weight the data to something as complex as a detailed specification for each step of the process. Whatever the programmers receive, at a minimum, they must have the following information:

- What is needed to perform the task?
- What exactly is to be done?
- What is to be produced?

After reviewing the high-level guidance provided, and when programmers feel they know the answers to these questions, the answers should be explicitly documented and reviewed, preferably by the person providing the guidance. It is critical to have a complete understanding of what is to be done and why it is to be done to identify any potential problems before the programming begins.

Finding a problem at this stage is much better than discovering it after the coding is complete. "The longer the defect stays in the software food chain, the more damage it causes further down the chain. Since requirements are done first, requirement defects have the potential to be in the system longer and to be more expensive. Defects inserted into the software upstream also tend to have broader effects than those inserted further downstream. That also makes defects more expensive."[4]

[3] Karl Wiegers, *Software Requirements* (Redmond, WA: Microsoft Press, 1999), 4.
[4] McConnell, *Code Complete*, 29.

3. Flowchart

"According to requirements authority Alan Davis, no single view of the requirements provides a complete understanding of them (Davis 1995). You need a combination of textual and graphical requirements representations to paint a full picture of the intended system and to help you detect inconsistencies, ambiguities, errors, and omissions. These graphical representations, or analysis models, improve your understanding of the requirements. Diagrams communicate certain types of information among project participants more efficiently than text can, and they help bridge language and vocabulary barriers among different team members." – Karl Wiegers[5]

There may be many flowcharts due to an iterative refinement of complex task within the statistical process. These flowcharts start with an overview flowchart that illustrates the entire statistical process at a high level. During each refinement a more detailed flowchart is created to reflect the refined process for tasks and steps within tasks.

During the creation of the flowcharts, junior statisticians or statistical programmers should have their peers informally review them. Once the major steps of the program are adequately documented, the person providing the high-level guidance and at least one other reviewer who is uninvolved in the project should review the flowcharts.

Figure 1 is a simplified example of a flowchart for the overview of the survey process. The tasks in the survey process are the survey design, frame development, sample selection, sample monitoring, post data collection weighting, analysis, and report writing.

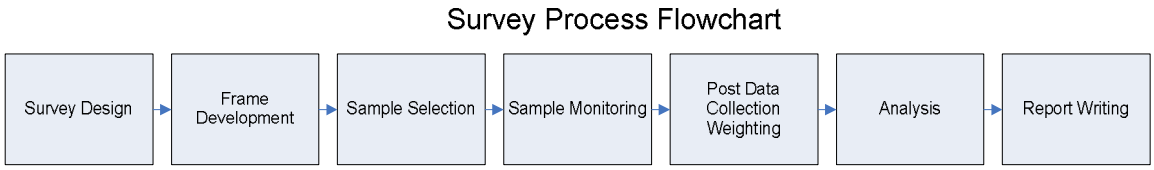

Figure 1: Overview of the Survey Process

Figure 2 (on the next page) refines the post data collection weighting task. The steps in the post data collection weighting task are an unknown eligibility adjustment, a nonresponse adjustment, and poststratification.

The post data collection task is further refined into the steps required to complete the task. Figure 3 (on the next page) shows the refinement of the unknown eligibility adjustment. Using the household status codes, the known eligibility and unknown eligibility households are determined. The sum of the weights for the sample households is the numerator, and the sum of the weights for the eligible households is the denominator. The unknown eligibility adjustment factor is calculated and multiplied by the design weight to create the unknown eligibility adjusted weight for the known eligibility households. The unknown eligibility adjusted weight is set to zero for the unknown eligibility cases. If the steps were too complicated or too big, they could be further refined.

[5] Wiegers, *Software Requirements*, 174.

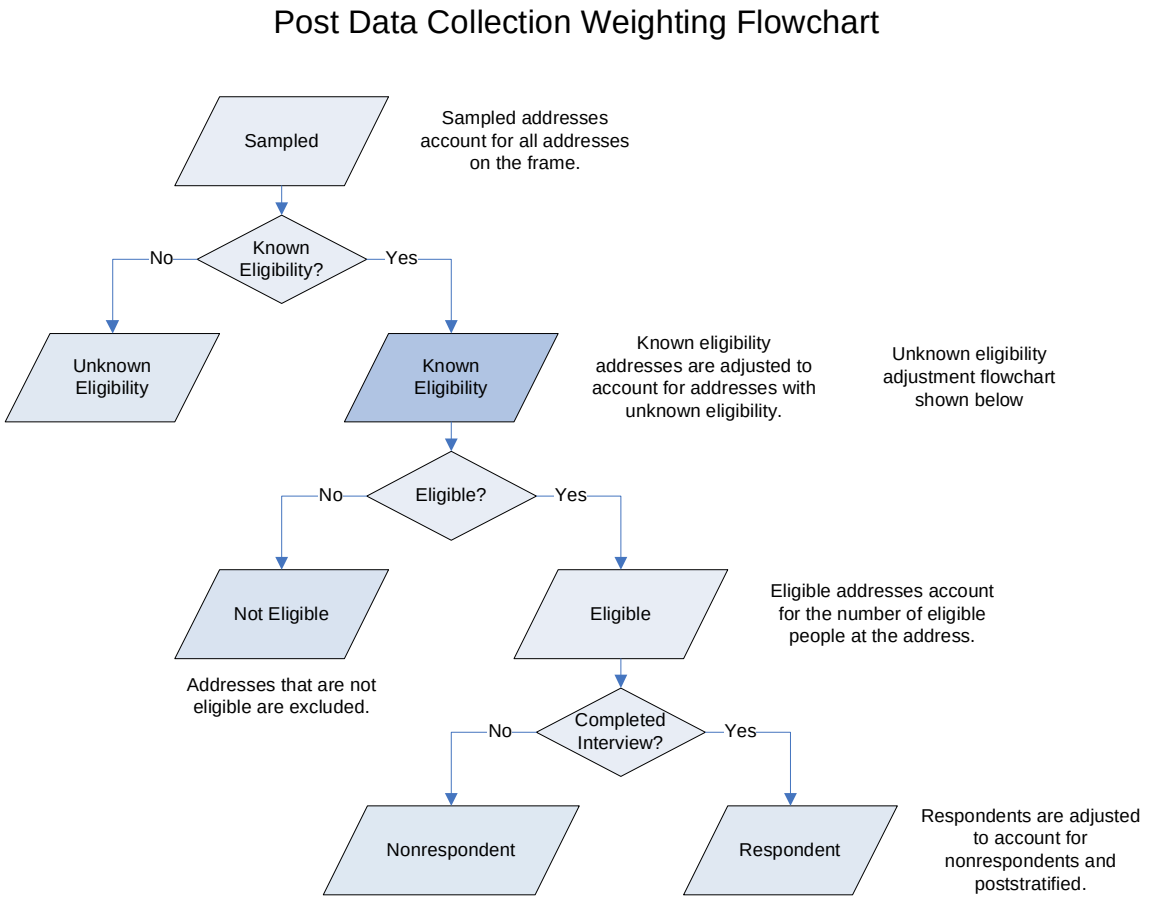

Figure 2: Post Data Collection Flowchart

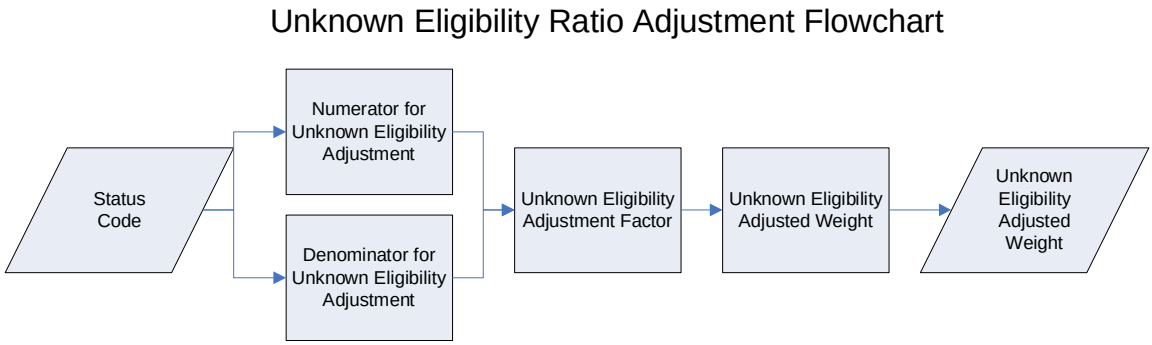

Figure 3: Unknown Eligibility Ratio Adjustment Flowchart

4. Text files

"If you can't explain something to a six-year-old, you really don't understand it yourself." – Albert Einstein[6]

[6] McConnell, *Code Complete*, 45.

The text files complement the flowcharts. They document the overview of the statistical process, the tasks within the statistical process, and the steps within the tasks. In addition, the detailed steps of the statistical process are further refined into pseudocode. "The term 'pseudocode' refers to an informal, English-like notation for describing how … a program will work."[7] "Here are the guidelines for using pseudocode effectively:

- Use English-like statements that precisely describe specific operations….
- Avoid syntactic elements from the target programming language….
- Write pseudocode at the level of intent….
- Write pseudocode at a low enough level that generating code from it will be nearly automatic…."[8]

While writing the pseudocode, junior statisticians or statistical programmers should have their peers informally review it. Once the pseudocode is completed, the person providing the high-level guidance and at least one other person who is uninvolved in the project (preferably the same person who reviewed the flowcharts) should review the pseudocode. "Once the pseudocode is written, you build the code around it and the pseudocode turns into programming-language comments."[9]

Here is sample pseudocode for the refinement of the unknown eligibility ratio adjustment step. The actual SAS code can be easily written following the pseudocode.

- *Unknown Eligibility Adjustment*
  - *Overall Adjustment Factor: Unknown Eligibility*
    - *Sum of the Address Design Weights for Sampled Households Overall*
    - *Sum of the Address Design Weights for Known Eligibility Households Overall*
    - *Calculate Overall Adjustment Factor*
      - *Merge the sum of the sampled weights and sum of the known eligibility weights data set.*
      - *Calculate the overall adjustment factor as the sum of the sampled weights divided by the sum of the known eligibility weights.*
  - *Stratum Adjustment Factors: Unknown Eligibility*
    - *Sum of the Address Design Weights for Sampled Households by Stratum*
    - *Sum of the Address Design Weights for Known Eligibility Households by Stratum*
    - *Calculate Adjustment Factors by Stratum*
      - *Merge the stratum sum data sets and add the overall unknown eligibility adjustment factor so we can check the stratum adjustment factors can be checked.*
      - *Merge the sum of the sampled weights and sum of the known eligibility weights data set by stratum so the stratum adjustment factor can be calculated.*
      - *For the known eligibility households, calculate the unknown eligibility adjusted factor as the sum of the sampling weights divided by the sum of the known eligibility weight by stratums. Otherwise, set it to zero.*
      - *Check the stratum adjustment factors to see if they are greater than 1.5 times the overall adjustment factor or to see if they have less 30 known eligibility households in non-certainty strata.*
      - *Create the unknown eligibility adjusted weight.*
      - *Merge the survey design weight and adjustment data set so that the unknown eligibility adjusted weights can be calculated.*
      - *For the known eligibility households, calculate the unknown eligibility adjusted weight as the product of address design weight and unknown*

[7] Ibid., 218.
[8] Ibid.
[9] Ibid.

eligibility adjustment factor. For the unknown eligibility households, set the unknown eligibility adjusted weight to zero. For any other households, set the unknown eligibility adjusted weight to missing. There should not be any with a missing value.

- *Make sure all of the households get an unknown eligibility adjusted weight.*

- *Check Adjusted Weights for Unknown Eligibility by Stratum*
  - *Sum sampling weights by stratum.*
  - *Sum unknown eligibility adjusted weights by stratum.*
  - *Merge the data sets containing the summed weights.*
  - *Create indicator variable so show when the sum of the sampling weights do not match the sum of the unknown eligibility adjusted weights.*

5. Computer Code

"Any fool can write code that a computer can understand. Good programmers write code that humans can understand." – Martin Fowler[10]

There are a few basic ideas that can make a program easy to understand. They are:

- Keep the program simple. Reviewers and other programmers who look at the code have to understand it. Make it as easy for them as possible.
- Use informative names for everything, e.g., programs, libnames, data sets, variables, and macros. This will make the code clear and conceptually understandable.
- Use the layout of the code to illustrate the program's logic. Break up the code into manageable steps with the use of white space and indent to emphasize the logical structure.
- Use comments. Even if the code looks relatively simple, write comments that describe the intent of the code to enhance understandability.

While writing the program, junior statisticians and statistical programmers should again have their peers review the program. Once the program is written, the person providing the high-level guidance and at least one other person uninvolved in the project (preferably the same person who reviewed the flowcharts and the pseudocode) should review it.

Here is part of the SAS program code created from the pseudocode.

```
title2 "Unknown Eligibility Adjustment";
title3 "Overall Adjustment Factor: Unknown Eligibility";
title4 "Sum of the Address Design Weights for Sampled Households";

proc means data = surveyDesignWt noPrint;
  var addressDesignWt;
  where sampled = 1;
  output out = unkElgNumerOverall ( drop = _type_ _freq_ )
       sum = unkElgNumerOverall;
run;

proc print data = unkElgNumerOverall;
run;

title4 "Sum of the Address Design Weights for Known Eligibility Households";

proc means data = surveyDesignWt noPrint;
```

[10] Ibid., 732.

```
  var addressDesignWt;
  where knownEligibility = 1;
  output out = unkElgDenomOverall ( drop = _type_
       rename = ( _freq_ = denomCountOverall ))
       sum = unkElgDenomOverall;
run;

proc print data = unkElgDenomOverall;
run;

title4 "Overall Adjustment Factor";

data unkElgAdjustmentOverall;
  *** Merge the sum of the sampled weights and sum of the known eligibility
  weights data set. ***;
  merge unkElgNumerOverall
        unkElgDenomOverall;
  *** Calculate the overall adjustment factor as the sum of the sampled weights divided by the sum of the
  known eligibility weights.
  unkElgAdjFacOverall = unkElgNumerOverall / unkElgDenomOverall;
run;

proc print data = unkElgAdjustmentOverall;
  var unkElgNumerOverall
      unkElgDenomOverall
      denomCountOverall
      unkElgAdjFacOverall;
run;

title3 "Stratum Adjustment Factors: Unknown Eligibility";
title4 "Sum of the Address Design Weights for Sampled Households by Stratum";

proc sort data = surveyDesignWt;
  by stratumV;
run;

proc means data = surveyDesignWt noPrint;
  var addressDesignWt;
  by stratumV;
  where sampled = 1;
  output out = unkElgNumer ( drop = _type_ _freq_ )
       sum = unkElgNumer;
run;

proc print data = unkElgNumer;
  sum unkElgNumer;
run;

title4 "Sum of the Address Design Weights for Known Eligibility Households by Stratum";

proc means data = surveyDesignWt noPrint;
  var addressDesignWt;
  by stratumV;
  where knownEligibility = 1;
  output out = unkElgDenom ( drop = _type_
       rename = ( _freq_ = denomCount ))
```

```
        sum = unkElgDenom;
run;

proc print data = unkElgDenom;
  sum unkElgDenom;
run;

title4 "Stratum Adjustment Factors";

proc sort data = unkElgNumer;
  by stratumV;
run;

proc sort data = unkElgDenom;
  by stratumV;
run;

data adjustment;
  *** Add the overall unknown eligibility adjustment factor so we can
  check the stratum adjustment factors can be checked. ***;
  if _n_ = 1 then do;
    set unkElgAdjustmentOverall ( keep = unkElgAdjFacOverall );
  end;
  *** Merge the sum of the sampled weights and sum of the known eligibility
  weights data set so the stratum adjustment factor can be calculated. ***;
  merge unkElgNumer
        unkElgDenom;
  by stratumV;
  *** For the known eligibility households, calculate the unknown
  eligibility adjusted factor as the sum of the sampling weights divided by
  the sum of the known eligibility weights. Otherwise, set it to zero.***;
  if knownEligibility = 1 then do;
    unkElgAdjFac = unkElgNumer / unkElgDenom;
  end;
  else do;
    unkElgAdjFac = 0;
  end;
```

6. Conclusion

This paper focused on producing a quality product with reduced costs through continuous communication, detailed documentation using iterative refinement, and identifying errors as early in the process as possible. The approach is easily understood and easily implemented. Additionally, it provides enhanced opportunities for technical mentoring of junior staff and may encourage software standardization for routine processes to reduce potential errors and to work more efficiently. The approach is appropriate for any size statistical programming project but is more beneficial as projects increase in size and complexity.

References

McConnell, Steve (2004). *Code Complete, Second Edition*. Redmond, WA: Microsoft Press.

Wiegers, Karl (1999). *Software Requirements*. Redmond, WA: Microsoft Press.