Writing Your SAS Program Right

Clarence Wm. Jackson, CSQA
City of Dallas

Abstract

This paper is for SAS Users who want to write programs right the first time, and it defines the steps needed to write quality SAS programs. These steps include how to define the problem to be solved and the goal of the programmed solution, defining input and output specifications, and other elements of the life cycle in implementing the program in the environment for use.

Introduction

The primary objective of any computer program is to solve a problem, and SAS is no different as it is well suited to solve most any programming problem. The problem can be defined in many ways and can be called many things, and the resulting SAS program is the solution. Requirements that are not defined will result in requirements not delivered, so it is always best to define the requirements before any programming is done. Many computer solutions fail because the problem isn't clearly enough defined to provide the correct solution, resulting in wasted effort and additional expenses in correcting the solution.

In the QA world, "requirements" are defined as the "wishes of the end user", and so, if "quality" is to be attained, then the requirements must be defined so that it can be measured in relation to the end user's "wishes being fulfilled". However, getting agreement on requirements has not been the easiest thing to do, much less documenting those requirements.

In a perfect world, getting properly documented requirements before beginning programming would result in near zero defects, but we don't operate in a perfect world. The U.S. National Institute of Standards and Technology estimates U.S. companies wastes of $27.5 to $59.9 billion each year in the total cost of software quality, of which includes bugs, ABENDS, patches, lost productivity, and other defects in developed software. Since we live and work in a 'wasteful' world, the best way to improve is to do it right the first time, one program at a time. And it starts with getting good requirements.

Background

Writing SAS programs can be fun and productive (for me anyway), especially after having written ALC, COBOL, VB, Notes, and other programs. SAS is a much easier and more powerful tool to use for a programmer than the others, and requires less time to learn and begin using. This means that one can write the right or wrong solution faster in SAS, which may or may not be a good thing.

Since "wrong" or defective solutions written in SAS can do more things to harm the organization faster, defects must be found and corrected sooner.

Making sure that the requirements are correct will reduce the number of defects in software development, and provide the solution with a minimum amount of rework and costs, while providing a more reliable solution for the end user. The best way to ensure that requirements are correct is to review development of the programs at various stages with the requirements to make sure defects are not embedded.

The cost of correcting defects to requirements grows with each stage of the life cycle, so it pays. Time spent on correcting programming problems could be spent on new development, so there are benefits to writing your SAS programs right. Adopting activities associated with a System (or Solution) Life Cycle methodology makes sense in reducing the costs associated with SAS software development.

Stages of the Solution Life Cycle

The stages of the solution life cycle are the same basic stages for any project or system development life cycle (SDLC). The Quality Circle of "Plan-Do-Check-Act" is represented in the steps of SDLC, as well as most successful ad-hoc projects. The basic stages are:

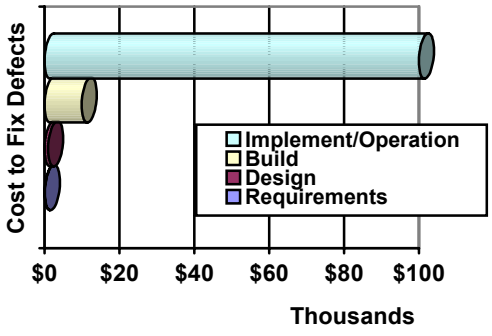
1. Feasibility and Requirements definition
2. Specification and Design
3. Code and Test (Build)
4. Accept, Install and Use (Implementation and Operations)

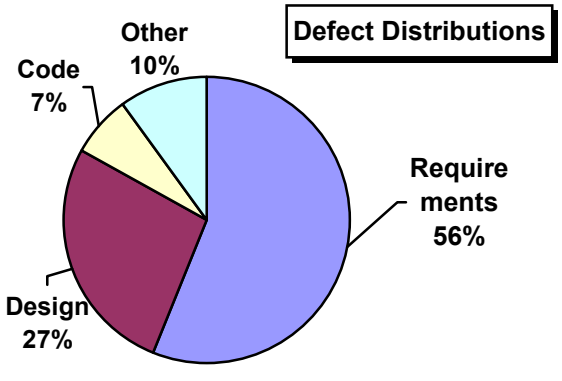
A more detailed review of each stage is later in this paper. These stages are usually documented for large projects, with check offs at defined points in the SDLC. But what relevance or benefit is there to applying a SDLC to writing a SAS program? Defect prevention, or in other words, less bugs and rework.

"Defects" are differences and/or problems in the delivered solution when compared to the requirements, so requirements definitions should be performed and monitored during the programming stage. It takes less time and resources to correct defects at the requirements definitions stage of the life cycle than to correct the defect later on in the coding, testing, implementation, and operations stages.

According to leading experts, the cost is about 10 times the cost to fix errors per stage of the SDLC, as indicated by the chart below. While the cost to fix errors per stage grows, these same errors can be traced back to the point of introduction. As the "Defect Distributions" chart illustrates, the majority of

defects can be traced back to the requirements stage, but usually will show up in later stages when the costs are higher.





The most costly of defects are those that become evident in the programming and later stages, so naturally, the best way to reduce defects is to focus on the requirements stage. This will reduce the total cost.

The development of a SAS program will follow these same stages, although the deliverables for each stage may not be documented as such. All programming benefits from ensuring that the requirements stage deliverable is well defined, so that each following stage will deliver the required solutions to the end user.

Everyday Ad-hoc SAS Programming

Realistically, not each and every SAS program will require the extensive steps outlined in this paper, but every SAS program will benefit from ensuring that basic elements from each step are done, such as, at least detailing the requirements before coding.

Most SAS programs are written by a single programmer, and this is usually the same person that collects information for the requirements, does the program design, coding, testing, implementation and initial program execution. Does it make sense for a one-person shop to adopt SDLC activities to write a program? Yes, if you want to do it right.

Let's examine a typical life cycle for an ad-hoc SAS program, since this is the area where most SAS programs start, and the need to document the life cycle appears to be less important. For everyday ad-hoc SAS programming, such formal SDLC activities may seem a waste of time, but for the programmer that wants to write that SAS program right, here is a tip: document your processes, especially the requirements gathering.

Depending on the task and the problem to be solved, only a note pad and pen will be needed to document the requirements of the solution, and note taking could be used to document the rest of the process. For large projects and

programs, more documentation may be required. It really depends on the shop requirements.

In one of my former job sites, the practice for requesting reports from our section was for the end user to come to our office, and talk about the problem and what they needed in the way of a report, extract file, or other simple tasks. These programs usually had quick turnaround times (2 hours or less), and had to be correct on delivery, or with minor defects that could be quickly corrected. We would use a note pad to write down what the user wanted, including a rough outline of the report and the data needed. Getting the requirements agreed to ahead of time was critical, so as not to waste time.

Since we always kept a current copy of the record layouts, data dictionaries, ER diagrams, system charts, data flow diagrams, and other charts handy, we would be able to gather most of the requirements in the 10-15 minute meeting. We could talk with the end user to ensure

(1) We understood what was needed and expected, and
(2) The end user understood what would be delivered, and
(3) There was agreement between us on what would be acceptable

The lessons I learned from the experience was the value of having updated documentation of the systems, files, and other tools available in order to develop the specifications and design while gathering the user requirements made coding the SAS program much easier and accurate, and met the user's expectations. We fulfilled the wishes of the end user.

Example "Fictional" Ad-hoc SAS Program Development

Let's make an example of Joe D. Newman, a new SAS programmer to the group at Dewey, Chethum, and Howe Marketing on Monday. His manager delivers to him his first SAS programming assignment, which is to solve the following problem for the Sales Group by noon Wednesday:

- Identify customers that are located in Texas, New York State, Florida, and California that haven't done business with the company in the last 12 months. Provide them the information of the salesman who they can contact. And let them know that we really want them to spend money with us.

The above represents the user's problem, with high-level requirements. Joe and his manager go to the client group's manager to verify the requirements, and after about 20 minutes and verifying the requirements, his manager delivers to him the specification and design of the solution as:

1. Create list of these customers. List customer ID, all contact info, as well as last activity date.
2. Send the customer a form letter with the contact info of the sales representative we want them to deal with. A sample letter was supplied in the meeting with the clients.
3. Create lists for the sales reps with the customer's contact info for cold calling and reference.
4. Create lists for the district managers of their sales reps and customers they should call on.
5. Create an export file that can be opened by spreadsheet applications that could be emailed later. The export file should include customer, sales rep, and district manager info.

The file specifications and field layouts for all files to be used for this project is provided by Joe's manager and DBA.

The requirements and the scope is defined, specifications and design for the solution's output deliverables are also defined. All the elements needed to complete this program are there. Joe and his manager feel that he has the info to provide the solution. (A sample of the start of the coded solution is at the end of this paper). Joe spends the rest of Monday afternoon and Tuesday morning coding the program.

After coding the program, Joe and his manager review the code while comparing the expected results to the requirements from the end user. After being satisfied with Joe's programming, he authorizes Joe to test the program (against test data, a copy of the production files), and then review the results of testing with his manager, then the end user. This is Tuesday afternoon.

Since he's using SAS, the coding almost documents the specification and design, and he uses the reports and output file for review with the end user. The end user wanted to take the test run's output for the final deliverable, but Joe and his manager insisted on rerunning the program in production to ensure that the correct and current data is used.

The program was installed into the production environment Wednesday morning, and executed without any changes or problems.

Joe and his manager used elements of all SDLC stages to write the program right. They made sure that the requirements were clear, there was file documentation, and the coding was almost mechanical. He was able to show the end user a sample report to verify, and obtained user acceptance. Although none of the stages of the SDLC were documented as such, all stages of the SDLC were followed.

The most important activity was to ensure that the end user's requirements were correct, and getting agreement on what those requirements were before coding avoided rework. Getting the end user involved in the development is a good thing, and will always help in writing your SAS program right.

Stages of the System (Solution) Life Cycle

A more detailed review of the life cycle is presented below, and is meant as a quick review of the more common items, tasks, and deliverables of the SDLC stages.

1st Stage: Feasibility and Requirements Definition

Depending on the scope of the problem, some level of a feasibility review of possible solutions is done. Before any SAS solution is decided, there must be something that is a problem, and a review of the impact. A proper feasibility review will at least:

1. Review current conditions, and the impact of the problem
2. Review possible solutions, including costs, impact on problem and current conditions
3. High-level requirements for the solution, including the intended deliverable.

Documenting the requirements, and getting agreements on these requirements with the end user BEFORE doing any coding will save time and resources later in the program development life cycle. This is the most important activity in the program development process. The requirements should at least be clear, concise, unambiguous, achievable, and testable. Basic inputs to requirements definitions are:

1. Problem definition
2. Defining the scope of solution
3. Environment variables for use
    a. Data stores
    b. Business rules
4. Documenting the criteria and conditions for acceptance
5. Documenting the intended deliverables

Defining the problem to be solved by the solution should be documented, and include the 'W's and 'how' of the problem. For example, the following questions could be asked in defining the problem:

- 'What' is the problem?
- 'When' did it become a problem?
- 'Why' is it a problem?
- 'Where' to apply the solution?

- 'How' should it be solved

Defining the scope of the solution uses the 'How' part of the question sets to set the parameters of the solution, and should state what the solution should be, how it will be applied, and high level conditions for use.

Environment variables for use includes:

- Operating systems
- Data stores and format of files, fields, and data
- End users
- Business rules for data
- Other processes that are impacted by the solution

Understand the operating system that the solution will be implemented on, and the location of data. Find out how the end users will be using the solution, and what problems they may have. Make sure that the requirements include these items, as they will affect the quality of the delivered solution.

Business rules for data need to be understood and defined. If the rules for data aren't clear, there is a chance that ETL or other processes depending on the data could be corrupted.

Documenting the criteria and conditions for acceptance is a very important part of the requirements stage of the life cycle. This is the stage that the end user or requirements' owner defines under what conditions the solution solves the problem, and how this condition can be measured for acceptance. For instance, a condition of acceptance could be that one deliverable of the solution will be a report of some specified nature, or data file with specific attributes that can be processed by another application. If user acceptance testing (UAT) will be done, then the test scripts to be used should be supplied at this time.

Documenting the intended deliverables at the requirements stage may seem premature, but it is necessary for the proper solution development. The deliverables for a SAS program solution can include:

- Reports
- Data for use by another application of process
- Applications for use by end user
- Other deliverables as defined by the requirements

2nd Stage: Specification and Design

Once the requirements have been defined, then the specification and design of the program can begin. The deliverables from the previous stage should be used to specify the solution parameters and design the program's logic to address the

solution. If the requirements are done correctly, then this stage should require less rework.

Before proceeding to actual coding and testing of the program, some review of the specification and design should be conducted with the end user or requirements owner, and any deviations to the requirements should be addressed. For instance, reviewing the specification and design may uncover problems with unstated assumptions held by the end user that the SAS programmer would need to resolve that is contrary to the solution. Examples include:

- Files to be accessed
- Handling missing data
- Timing and file update issues
- Performance issues
- Format of the deliverables
- Other technical and data management issues

Once the specification and design is agreed upon and issues resolved, only then should coding the solution begin.

3rd Stage: Code and Test

Not all test scenarios and techniques will apply to all solutions, and can sometimes be counter productive to the delivery of the solution, while some testing will be essential to the success of the solution. Testing the solution against the requirements should be defined in a documented test plan for large projects. The testing criteria should be defined as part of the requirements stage using attributes for acceptance, use, and other attributes relative to the solution.

This stage is where most SAS programmers want to be, writing that code, unit testing, and then making sure everything works. This stage is where the solution begins to take shape, when test reports and other deliverables can be seen, and the results are more apparent. But if the previous stages have not been properly followed, this could be the stage that requires more rework and time, and cost more money.

The code should be reviewed and compared to the deliverables from the previous stages to ensure that defects are not included in the code. Defects include:

- Missing features documented in the requirements
- Incorrect features, bugs, errors, and
- Additional features not documented in the requirements
- Performance problems

Various techniques exist for code review such as walkthroughs, manual reviews by peers and management, and automated reviews using software. Since the majority of software testing tools in use today target C, VB, HTML, XLM, and others "popular" development software, and not SAS, then some type of manual code review process should be used. It would be of benefit to have an experienced SAS programmer review the code of their peers, and document the review. All review issues should be resolved before doing testing.

The worst thing that could happen is to install a solution into production without testing to make sure it works. Testing the solution should include some or all of the following types of testing in a proper testing environment:

- Unit testing of the modules
- Regression testing of the modules against a standard test data set.
    - Changes to installed solutions should also be tested to make sure that previously verified features and functions still function properly.
- Integration testing
- Performance testing to simulate the solution in production
    - Stress test if the solution required will be highly stressed
    - Load test if the solution required will process high volumes of data
- Other testing relevant to the solution

All testing activities should be documented, and all defects in testing should be reviewed and resolved if possible. These defects should be tracked to resolution, even if the resolution is to allow the defect. In some cases, it may be desirable to allow a defect to be included in the solution because of cost to fix, or to correct a poor requirement.

4th Stage: Accept, Install and Use

OK, we are now at the point of presenting the solution to the end user for acceptance. The acceptance process should have been defined in the requirements stage, so there should not be any surprises at this point. It may have been defined that the end user will perform acceptance testing (UAT), which should have been included in the requirements. Depending on the size, scope, or other factors, the acceptance of the solution should be documented. The solution is not installed until it is accepted.

If the solution is not accepted, then the reasons should be documented for review and resolution. If the reason includes undocumented requirements or criteria not previously included, then in some cases, it will need to be decided if the new requirement will be an enhancement in a later version of the solution, or included in the current version. I recommend that such items be included in later versions of the solution, and to go ahead to the installation of the current

solution that meets the original requirements as defined, if the change will greatly impact the delivery schedule or cost of the solution

Once the solution has been accepted, it's time to install to production for use by the end user. The installation environment should have been discuss, and/or documented in the requirements, specification and design phases of the project.

Conclusion

The key to writing SAS programs right is to make sure to understand the end user requirements. When the requirements are "right" and are used to develop the SAS solution, then chances are that the solution will be accepted by the end-user. "Right" written programs require less attention in the form of user requested modifications. To do it right starts with the requirements and getting them right. The rest is a matter of making sure to include the requirements in the development of the program by review and verification.

Joe D. Newman's SAS Program **(a sample, not the complete solution …)**

```
filename customer "custfile";
filename salesrep "salesrep";
filename managers "managers";
filename xported  "exported-file";

data customers;
 infile customer;
 input custid custname custaddr custcity
        salerid state cusphone cusfax cusemail last_buy;
 if state in ('tx','fl','ny','ca')
  and last_buy lt (today()-365.25);
run;

proc sort; by salerid; run;

data salesrep;
 infile salesrep;
 input salerid salename saleaddr salephon salefax
       salemail divmanid;
run;

proc sort; by salerid; run;

data custsale norep;
 merge customers (in=a) salesrep (in=b);
 by salerid;
 if a then output custsale;
 if not b then output norep;
 delete;
```

```
 return;
run;

proc datasets; delete customers salesrep; run;
```

{Joe's coding continues. Since this is more for example, we know how this program should turn out, hopefully right!}

Author Contact info:
Clarence Wm. Jackson, CSQA
City of Dallas, Communication & Information Services
Manager, Change Management, Quality Assurance, and IT Disaster Recovery
1500 Marilla 4DS
Dallas, TX 75201
cljacks@ci.dallas.tx.us (work)
CJac@compuserve.com (home)