Paper 867-2017

Quality Control Programming: A Lost Art?

Amber Randall and William Coar, Axio Research

ABSTRACT

Conferences for SAS® programming are replete with the newest software capabilities and clever programming techniques. But, discussion about quality control (QC) is lacking. QC is fundamental to ensuring both correct results and sound interpretation of data. It is not industry specific, and it simply makes sense.

Most QC procedures are a function of regulatory requirements, industry standards, and corporate philosophies. Good QC goes well beyond just reviewing results and should also consider the underlying data. It should be driven by a thoughtful consideration of relevance and impact. While programmers strive to produce correct results, it is no wonder that programming mistakes are common despite rigid QC processes in an industry where expedited deliverables and a lean workforce are the norm. This leads to a lack of trust in team members and an overall increase in resource requirements as these errors are corrected, particularly when SAS programming is outsourced. Is it possible to produce results with a high degree of accuracy even when time and budget are limited?

Thorough QC is easy to overlook in a high pressure environment with increased expectations of workload and expedited deliverables. Does this suggest that QC programming is becoming a lost art, or does it simply suggest that we need to evolve with technology? The focus of the presentation is to review the who, what, when, how, why, and where of QC Programming implementation.

INTRODUCTION

Quality control (QC) in programming is fundamental to good decision making. If businesses and scientists are willing to invest valuable time, money, and resources, then it follows that they would want to ensure that the results are correct. That may sound simple enough, but digging a little deeper uncovers a labyrinthine list of considerations and approaches. Elucidating what is best for any given project or team requires a critical review of many factors. We will explore methods to make sense of those options and propose a strategy for implementation.

We are presenting this exercise from the perspective of a Contract Research Organization (CRO) in the pharmaceutical industry. We provide statistical and programming services to a variety of clients including large pharmaceutical companies, government-funded research organizations, and non-profit research organizations. Our programming supports clinical study reports, clinical statistical consulting projects, data monitoring committees, and data transformation and reformatting into new industry standards. While we realize that our perspective is somewhat limited to standards and practices within pharma, we believe that good QC is a global programming concept and we will do our best to generalize these concepts as we go.

Life at a CRO can be a little unpredictable. Every project and client is different. We have very little control over the source data that we receive which makes the implementation of standard code and outputs very difficult. Timelines are often out of our control and, along with resource availability, are constantly shifting. The pharmaceutical industry benefits from detailed federal policies providing guidance for validation procedures and because of these high industry expectations, QC and validation can be very intensive. However, mistakes are still made which lead to an increase in the time required to complete a project. In addition, new rules have been recently implemented by the FDA which have forced the pharma industry to adopt defined data and documentation standards, collectively known as CDISC, for any clinical trial data submissions. While the implementation of these standards provide benefit to the industry in terms of predictability, efficiency, and interpretability, they also increase the total number of deliverables and overall workload required of the programming team. In addition to a greater number of deliverables, each is subject to strict requirements and documentation with a substantial learning curve in implementation. This can mean that a greater number of programmers are required per project to meet timelines and only raises the pressure to get more done with the same resources.

There is currently a trend in pharma as well as many other technical industries to move toward a leaner workforce and more expedited deliverables. Experienced SAS programmers are becoming harder to find and more costly to hire. Budgets are squeezed tighter and we find ourselves competing with lower priced offshore teams for contracts. In this environment, challenged to produce more with less, it is easy to want to pragmatically focus on development and deprioritize thoughtful QC and validation. But, as experienced professionals we know that is not a responsible tactic. We instead propose a new strategy that is perhaps a bit more thoughtful than a traditional approach.

WHAT IS QUALITY CONTROL?

We often use quality control and validation interchangeably. However, there is an important distinction between the two. According to dictionary.com validation is the process to *make valid; substantiate; confirm*. Quality control is a ***system*** *for verifying and maintaining a desired level of quality in an existing product or service by careful planning, use of proper equipment, continued inspection, and corrective action as required.* Validation shows that a specific program or output is correct. Quality control is a system that ensures that all output is validated in a systematic way. It describes what will be done, that it was done, and that it was done correctly. It can address a deliverable as a whole or smaller pieces such as individual programs.

QC in an environment that utilizes not only SAS Programming, but also other programming languages, to perform data analyses can be broken down into three key phases as shown in Figure 1 – input, processing, and output. Input verification examines the raw data collected. These validation activities generally happen long before programming begins, but are no less critical. SAS may indeed be used to assist in these activities which aim to answer the following questions. Were the appropriate data collected to answer the analysis question? Did the researchers choose to look at the appropriate metrics? Were the data collected appropriately? Was the sampling performed correctly? Were missing data points kept to a minimum? In clinical trials, these questions would most likely be addressed by a data management team, but it would be up to the statistical and programming teams downstream to deal with any data anomalies.

The second phase is the processing of the data which can be broken down into two main activities – analysis planning and programming. Were the appropriate analysis methods chosen and applied correctly? And, was the programming implemented correctly? This is where the bulk of the programming happens and probably the most intensive and time-consuming validation.

The final phase in the QC process is output verification. Most analyses are not simply reviewed as raw SAS output. Some sort of reporting function is likely used to make the output more user friendly. A thorough QC to assure that the correct outputs have been created, selected, displayed correctly, labeled appropriately, and packaged completely.

Without a thorough and thoughtful QC process, there is no guarantee that any analysis is representative of reality. For the purpose of the paper we will assume that the validation of the off-the-shelf statistical software being used has been performed and is reliable, but we don't want to overlook the importance of using properly validated software.

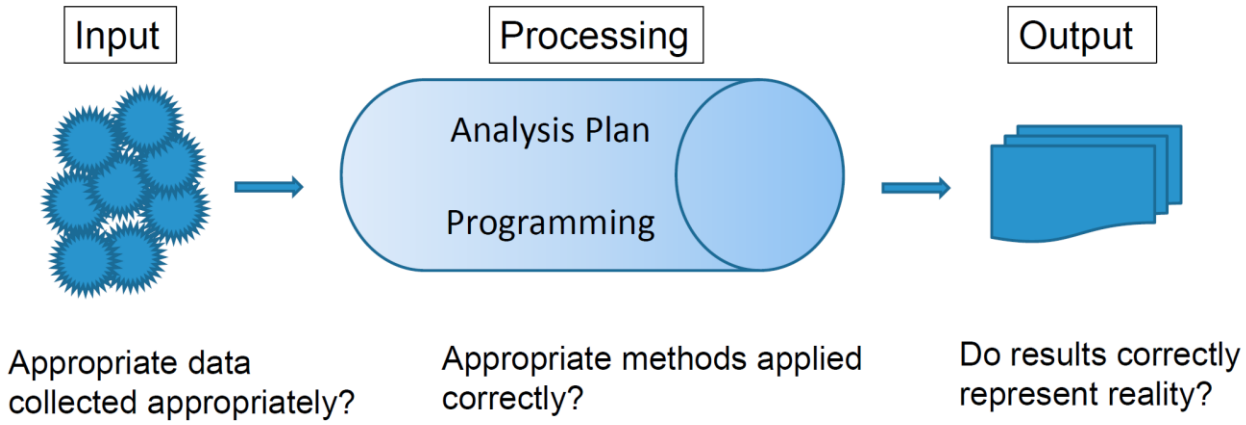

Figure 1. Schematic of General Programming Process as it Relates to QC

WHY DO QC?

The easy answer for why QC is so important could be condensed simply into the need to prevent mistakes. Data collection and programming can be both expensive and time consuming. It is reasonable to expect correct results, but let's delve a little deeper.

The types of errors encountered when performing statistical analyses can be visualized in a way conceptually similar to the standard statistical error table as demonstrated in Figure 2. The first occurs when reality is false, but the statistical analysis/programming shows truth. This would be a false positive and analogous to a Type I error. The second occurs when reality is true, but the statistical analysis/programming shows a falsehood. This would be a false negative and analogous to a Type II error.

	Reality is true	Reality is false
Programming finds true	Correct decision is made	Type I error (false positive)
Programming finds false	Type II error (false negative)	Correct decision is made

Figure 2. Error Types

THE IMPACT OF MISTAKES

What are the potential consequences of making these types of errors? Not all mistakes are created equal. Let's consider the difference in low impact mistakes and high impact mistakes.

Low impact mistakes do not lead critical consequences in the analyses. They may be caused by an error in a single data point or perhaps a mistake in the programming or reporting logic. While efforts are made in most QC processes to identify and correct as many errors as possible, the benefit of correcting low impact errors may not be worth the additional cost in time and resources. Of course, whether or not any errors can truly be considered low impact is dependent on the specific project. In pharma, a low impact error may be an accidentally truncated text field or perhaps a protocol violations table is mislabeled as

deviations. Maybe a handful of lab values out of tens of thousands were submitted in non-standard units and were not appropriately converted. Does it matter? It may, depending on the study, but even though the report would not be a full reflection of reality, none of those errors would likely impact the final outcome of the primary analysis. The greater impact may be felt in other aspects such as a loss in future business due to a loss of confidence in the programming team.

High impact errors are of greater concern and can have serious downstream consequences. These are errors that can lead to incorrect inference, affect the outcome of the entire project, and influence real-world consequences in profits, health, safety, and future decision-making. High impact errors can be a result of improper planning and data collection, inappropriate analysis plans, or incorrect programming. Sometimes errors that may seem minor can affect the overall interpretation of the results. It is important for the study team to have a good understanding of the true impact. If the key results do not reflect reality, then the whole analysis exercise is meaningless. In the majority of cases the benefit of fixing high impact errors is worth the additional investment in time and resources. Incorrect decisions can be made on flawed results that can lead to wasted time, wasted money, and missed business opportunities.

Let's consider this example of a high impact error from the pharmaceutical industry. NewDrugPharma has NewDrugA in clinical trials. In the pharmaceutical industry there is often something magical about a p-value of <0.05. Statisticians will insist that p-values do not tell the full story, but for investors, it often is a result that is considered worth funding. NewDrugPharma has limited resources available to test NewDrugA. Investors have already spent millions on the trial. Clinical trial subjects from a limited pool of people with the necessary disease characteristics have volunteered time and put their health at risk. Researchers from a limited pool of people with the appropriate knowledge are focusing their mental bandwidth on the development of NewDrugA.

What if a Type I mistake is made in the analysis process? A significant p-value is found that is not real due to an inappropriate study design, analysis plan or incorrect programming? An ineffective drug might be placed on the market. Money would be wasted prescribing an ineffective drug to patients and worse, the patients may be putting their health at risk by not taking an effective drug in its place.

What if a Type II mistake is made? What if a significant p-value is not found when there really is a positive effect? An effective drug may not make it to market. Billions may be lost by NewDrugPharma and patients may not get the important therapy that they need.

While these examples are fairly simplistic and maybe even extreme, they demonstrate that both types of errors (ie, mistakes), need to be considered.

CONSIDER THE FACTORS THAT INFLUENCE A QC PLAN

Validation can be very resource intensive and require a substantial amount of time, effort, and therefore money. In order to maximize the investment and ensure the most efficient process possible, a good QC plan is imperative. It is important to thoughtfully assess each step in your workflow and consider how the outcome of that step impacts the overall quality of the final product. Your plan needs to have the flexibility to consider both the risks and consequences of making a mistake at each step and for all associated tasks as a whole. There is no set model and no crystal ball for this effort. Each plan often needs to be project-specific and should also be considered a working, dynamic document and evolving along with the programming project progresses. For the purpose of this exercise, we will focus on the areas of a programming process that use SAS programming, but we do want to acknowledge that other non-programming activities may occur concurrently that affect overall quality.

Many factors need to be considered when developing a QC plan including when to start, who to employ, where validation should occur, what should be validated, and how to perform validation. Further we will explore how some of these concepts are interrelated. We recognize that some of the ideas presented here may be more appropriate to some industries than others, but we believe that some form of QC is relevant to all industries utilizing statistical programming and we will try to generalize where possible.

WHEN SHOULD VALIDATION START?

It is important to consider at what point in the life of a programming project validation should start. The answer depends greatly on the type of project. For a relatively small, short project, where both the

programming and validation can be completed relatively quickly, waiting until the initial programming is complete may be appropriate. For larger, long term projects, validation may need to start earlier and run in parallel with the programming process. There are both pros and cons to performing QC concurrently vs waiting until all programming is complete.

As mentioned earlier, my co-author and I approach the QC process from the perspective of a small data management and statistical CRO that supports clinical trials. Figure 3 illustrates our experience (though Figure 3 is applicable to many types of experiments). We are regularly required to recombine data from 15-25 raw datasets into several analysis datasets to support reporting of 20 to 200 different statistical tables, listings and figures. We sometimes work with data supplied by our data management team and sometimes receive data from our clients or a third party vendor. When receiving data from third parties, we have very little control over the structure and format of the data which limits our ability to create standard validation code.

In our programming process model, analysis programming often needs to start long before data collection is complete. At this stage data are very limited and often contain errors. It can be challenging to design both defensive programming and validation code with future data in mind. Because of the magnitude of the deliverables, QC generally occurs concurrently with the programming process and while data collection is ongoing. In addition, we are sometimes asked to produce interim reports on early data as well as final reports on complete data. Validation can't be left until the end. Because of the variety of deliverables, our QC approach changes based on the requirements of the project.

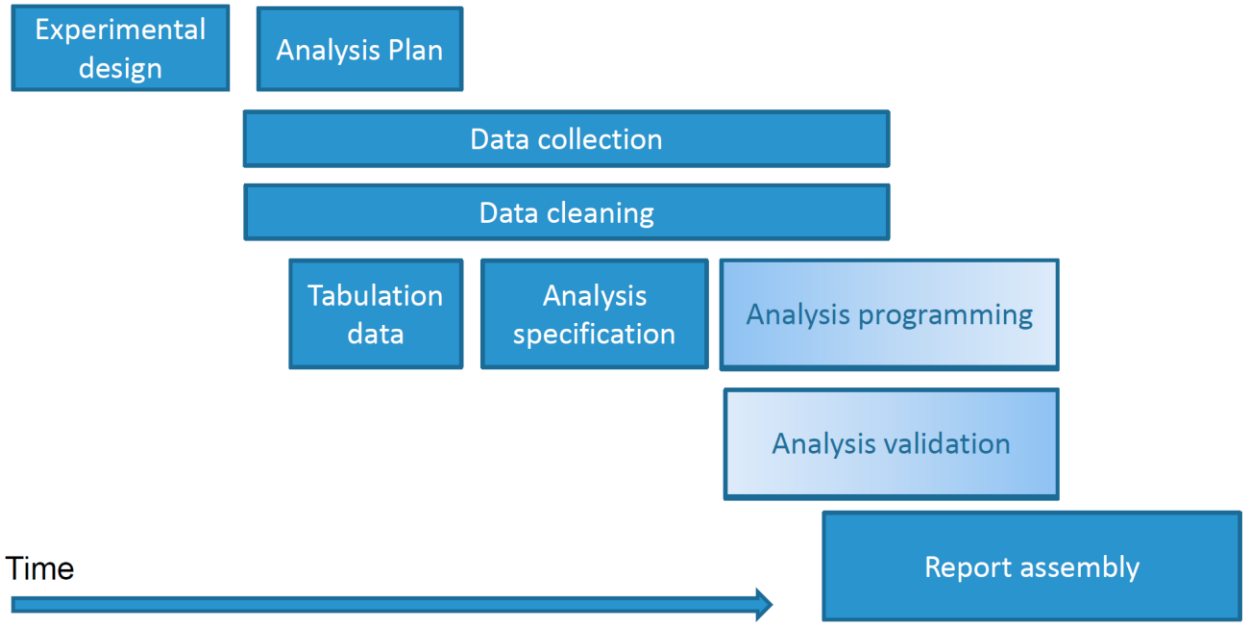

Figure 3. A Generalized Process Flow for a Clinical CRO

To further complicated our plan we also now need to consider conforming to new industry programming standards. Within the pharmaceutical industry, a set of data and programming standards has been mandated by the FDA to improve efficiency in both reporting and in the data review process as well as improve traceability from raw data to results. These standards, collectively known as CDISC, manifest as a substantial programming and validation effort. The current CDISC process flow starts with raw data which may or may not have been collected in a format supporting the CDISC standards. Raw data is then converted to the Study Data Tabulation Model (SDTM) per strict guidelines for the purpose of data review and submission. The SDTM data are then passed to the analysis team which recombine the datasets into the necessary analysis datasets to support the analysis requirements described in the analysis plan. These analysis datasets must be in the format known as Analysis Data Model (ADaM). The ADaM datasets are used to create the required tables, listings, and figures.

In theory this may sound like a straight forward process, but in practice it can be much more challenging as shown in Figure 4. Each piece of the CDISC process needs to be independently validated. As a CRO we are sometimes asked to perform interim analyses on early data or start programming a final analysis prior to the end of data collection. SDTM cannot truly be considered final until all data have been collected, yet we need to create our ADaM datasets from preliminary SDTM datasets that we know will change over time. All upstream changes in SDTM will likely require downstream updates and additional validation. It is difficult and time consuming to hit a moving target and in cases like these, it may make more sense to wait a bit longer before starting validation programming efforts. However, in real life, the optimal process is not always feasible. Depending on when interim reporting is requested, we need to adjust the timing of our validation programming.

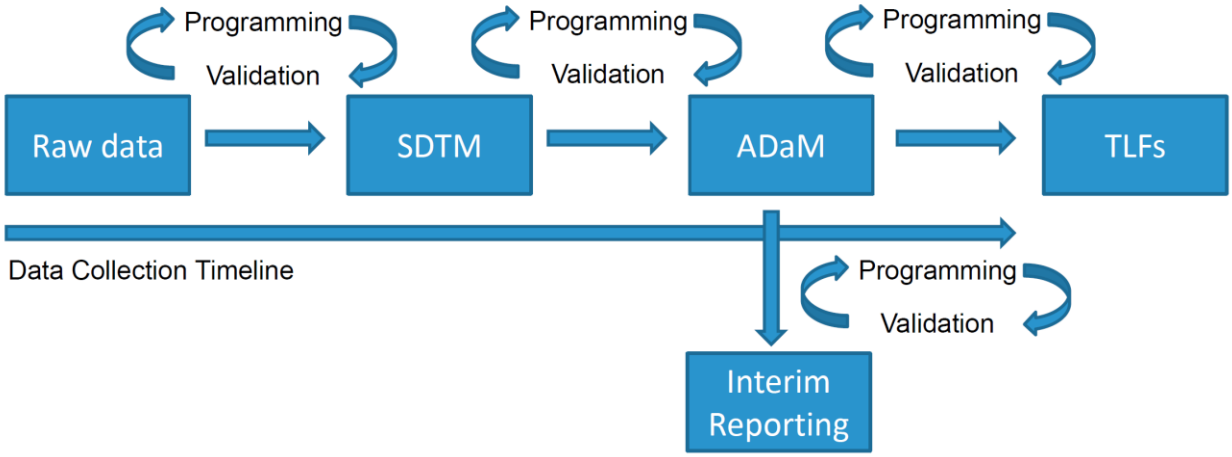

Figure 4. Validation in the CDISC Programming Process

In general early validation can be desirable because there is a lot more time to fix any programming or data mistakes that are identified. That additional time for correction can mean that all errors have the potential to be much lower impact. However, early data are often incomplete and contain errors. Both programming and validation can be challenging and need to be much more defensive and proactive. If they fail to account for unexpected data conditions, repetitive code updates become necessary which can lead to more hours spent overall. Further, there may be changes and updates to specifications as the project develops. It becomes increasingly difficult to track programming and changes and keep the associated validation code current and documented. Further, even if validation is started early, a final validation still needs to be performed on the locked database.

Validation started later in development can be more efficient because data are more complete and the planned analyses and deliverables are much less likely to change substantially. This means that there will likely be fewer updates in programming and therefore also fewer changes in the validation code. It also avoids potentially complex programming to accommodate rigid data standards in the presence of missing/incomplete data. That stability, though attractive, may come with a price as the consequences of errors found at a late stage may have a greater impact if such errors are related to design of data capture systems. There may not be as much time to fix the programming errors, nor to query for data cleaning. Sometimes data cleaning will no longer be possible. In addition, early results may have been distributed and program decisions may already be in progress.

While programming based on clean, locked data is more ideal for performing validation, the reality is that there will likely not be time to start most validation programming that late in the process. A thoughtful compromise needs to be made to balance efficiency, and impact on decisions made on interim results. The available options should be discussed with the whole study team prior to finalizing a data plan.

In summary, the decision of when to start validation programming for any project should consider several factors. There are many right approaches and a variety of opinions. The most important task is to find balance in the factors discussed below.

Consider the likelihood and impact of errors – A project with a higher risk of errors and/or one in which any errors could have a high impact should probably be considered for earlier validation programming. Rather, one that is less likely to have major errors or errors that will not affect major decision making may require very little or simply brief validation at the end. This may be appropriate in cases of iterative reporting when new data are simply run through previously validated programming. An appropriate QC process may be to focus on ensuring that all new data were included and that all programs ran correctly via a review of the final report and logs.

Magnitude of project – For a very large programming project, saving all validation until late in the process is probably not the best choice, especially if there is a solid delivery deadline. Validation can often take longer than the initial programming especially when the iterative nature of programming fixes and updates are considered.

Resource availability – Sometimes the decision of when to begin validation may be driven by resource availability. At a CRO especially, priorities, timelines, and workload are constantly shifting despite the best efforts at planning. Validation programming may need to be scheduled and occur when people are available even if that time is earlier than preferred.

Type of project – The type of project may drive when validation begins. Validation for a completely new report for a high visibility clinical trial would likely need to begin earlier in the process than validation for an iterative report that has already been run many times in the past. Early validation may also be necessary when interim results are needed prior to the release of final results.

Impact of the programming process on validation – A complicated, multi-step programming process may justify saving the validation programming for later in the process when programming may be more stable. As with the CDISC example mentioned earlier, when upstream changes can require many downstream programming updates, starting the validation programming too early can lead to a lot of additional work and expense.

Timeline – The overall deliverable timeline should also be considered. Projects with a long or lax timeline may be able to save validation programming for after all development programming is complete. Projects with short timelines would likely need to start validation programming earlier to meet expected deliveries.

WHO SHOULD PERFORM VALIDATION?

A statistical programming team is likely composed of some combination of the following – senior programmer, junior programmer, statistician, project manager, data scientist. All members regardless of position are inherently participating in the QC process. Beyond any formal policy, each individual has a basic responsibility to self-validate any work product that they generate before releasing it to the team. In regard to the formal QC and validation process, all roles have a unique and useful perspective and should be strategically employed for the optimal results. In choosing a lead for your validation programming process, you should consider the strengths and weakness of each role and team member.

Validation may often be thought of as secondary to development programming, but in reality it is the last line of defense before results are released so the decision of who should spearhead the effort should not be taken lightly. Even with a solid process in place, mistakes still happen for a variety of reasons. Specifications may be followed precisely by both the lead and validation programmers, yet errors may still exist because neither took the time to look for anomalies in the underlying data or did not consider whether the specifications actually made sense. When thinking about who should lead the validation team, you should considered who has the most experience in thinking strategically and anticipating areas where errors are more likely to occur, and who has a good global understanding of the project and goals. Perhaps it makes more sense on some projects to allow the least experienced programmers to perform the development programming and save the validation for the more experienced programmers who are more likely to take a critical look at both data and programming logic.

The decision may be made easier if a standard process already exists or if there is industry guidance dictating how validation should be performed. The FDA provides guidance for many tasks undertaken in the pharmaceutical industry. Different industries may have differential flexibility in the roles and experience of the validation team. In marketing, for example, the results of statistical analyses are more

likely to affect only the company that performs them rather than the health of many individuals as might occur in pharma.

The senior programmer on a project likely has the most extensive programming experience and will likely make the fewest mistakes in both programming and validation activities. She will also likely reduce the time spent on both. However, someone with this level of experience can be expensive. The cost of a senior person needs to be balanced with time saved as well as risk and impact of errors. In some cases, high risk or impact may be justification enough for the cost of employing a senior programmer in validation programming.

A junior programmer will likely be more affordable, but will also have less experience in both the industry and in writing code. This can lead to additional time required to complete validation tasks. A junior programmer may be more reliant on detailed specifications and may not catch subtle nuances in programming syntax and logic. He may also not have the confidence to question decisions made by a senior or lead programmer and may let errors go unreported. He may also have a more limited understanding of the industry and data. However, for companies working on multiple projects, junior programmers may be viewed as a way to supplement the efforts of senior programmers and may also be considered an investment in the future as they gain more experience and get promoted to senior positions.

A statistician is likely to have the best big picture understanding of the project and is a good choice for overall QC and any inferential statistics. However, not all statisticians are skilled programmers and may not catch subtle programming nuances or write flexible validation code that can elucidate subtle errors.

If you are working on a team that includes a data scientist, you are likely to have the benefit of both a statistician and skilled programmer. Data scientists tend to have a good big picture understanding of the project and a solid understanding of programming algorithms, but like statisticians, can be expensive. The expense would be balanced with the knowledge that they may work faster than less experienced staff, but it needs to be considered that they may have played too great a role in initial development to play an intensive independent role in validation.

Contractors can provide additional validation support when needed, but it should be recognized that bringing in any external resources adds another level of complexity to a validation plan. Contract services can come from independent individuals or contracting organizations. Good contractors can be difficult to find, vet, and secure. It is our experience that there are many individuals on the market that sell themselves as experts, but do not really have the skills advertised. Some experienced contractors can effectively provide an independent, expert, opinion for your team and will provide technical feedback if they see errors in specifications or further will be able to generate a product with very little instruction. Others will be completely reliant on very detailed and reviewed specifications and may program only what they are instructed to do with very little independent thought. Like employees, some experienced contractors work very well independently and others require daily management. If your team is considering the use of contractors, do not fail to add these considerations to your QC plan.

WHERE SHOULD VALIDATION BE PERFORMED?

While it may seem like less of an issue than some of the other factors presented earlier, where validation occurs should also be considered in a QC plan. It is generally most optimal to have the validation process occurring in the same physical location as the corresponding programming process. The validation and programming teams have easy access to communication and there is probably better group understanding of the project and data. If your team has decided to use contractors to help or if many of your employees work remotely, then it is possible that the programming and validation teams are in different physical locations. It has been our experience that this arrangement leads to a greater reliance on thorough specifications and more time spent in maintaining documentation than might normally be required unless you are working with a very experienced programmer doing the validation. Even with advances in technology, time zones and communication can be a challenge. If your team members are international, different holiday schedules and work norms can change a timeline. While not the most critical consideration in developing your validation plan, team location is important and should not be ignored.

WHAT SHOULD BE SUBJECT TO VALIDATION?

For the purpose of this paper in discussing what should be validated, we will focus on the programming and reporting aspects of the process as they relate to company and industry expectations and as they relate to end user expectations. We will assume that the data were captured appropriately, that the correct data were captured to address the primary question, that the data were entered into the database and cleaned sufficiently, and that the data are complete.

When thinking about what should be validated, you first need to consider your own process flow. A physical diagram can help with this task. Break your process down into large functional steps. Within each of those steps, identify individual tasks where SAS may be used and therefore where validation may need to occur. Let's think about how we might define a process to validate the outcome of a statistical programming workflow. Figure 5 shows six key steps – Planning, Data Collection, Analysis Specification, Data Manipulation, Analysis Programming, and assembly of a Final Deliverable.

At the Planning stage (1), confirm that the correct question is being asked, that the correct analysis population has been chosen, and that the correct data will be collected. SAS may be used at this step for tasks such as sample size calculations. At Data Collection (2), the data should be reviewed to limit the amount of missing and erroneous data. SAS may be used here to write data quality queries. When the analysis is designed and specified (3), a second statistician or statistical programmer should review to assure that the correct methods have been chosen and that the appropriate variables and data points are being called. This is especially true given the variety of options associated with statistical procedures. During data manipulation and analysis dataset creation (4), the programming for the analysis datasets should be validated. The programming for the final analyses and tables (5) should also be validated for correctness and completeness. Lastly, an overall validation of the final deliverable (6) should be performed to make sure that all desired elements are present, formatted as expected, and consistent and coherent within the report.

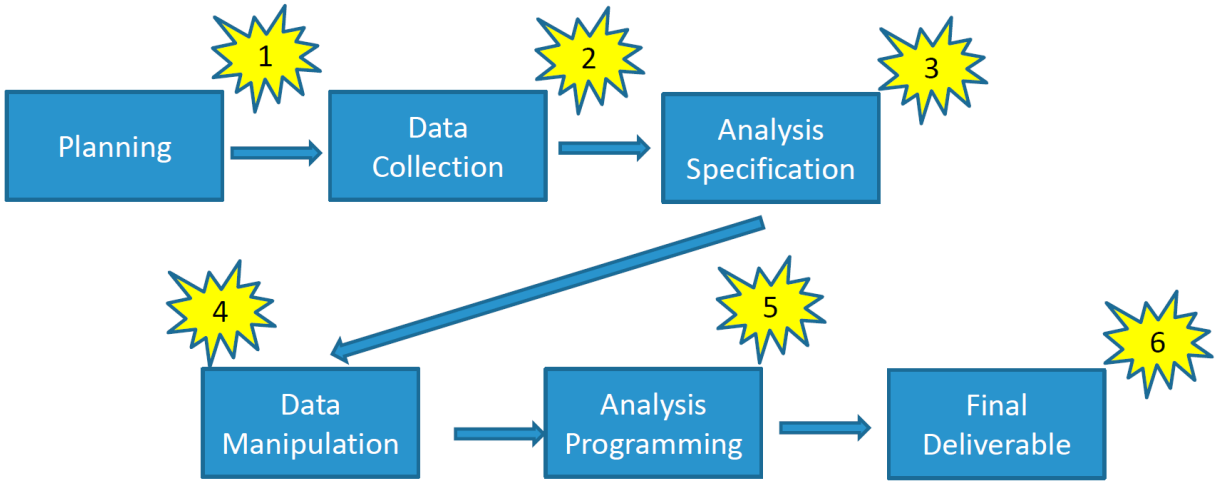

Figure 5. Generalized Statistical Programming Process

These validation tasks will be revisited later in this discussion.

HOW SHOULD VALIDATION BE PERFORMED?

The concept presented in Figure 6 is probably familiar to most individuals that work in a programming field. In validation, as with many things in life, conventional knowledge postulates that you can't have it all. You can safely choose two of the three concepts of speed, affordability and quality in any programming project. Achieving all three in any given task is likely out of reach. Your choice will be driven by budget, timeline, resources available, end product, and a variety of other factors. My co-author and I would like to challenge this assumption and propose that maybe it is possible in some cases to have all three if you are willing to assume a certain level of responsible risk. We fully admit that what can

be defined as responsible risk is clearly subject to opinion, but we would like to explore various approached to validation in hopes of identifying a path to the elusive complete triangle.

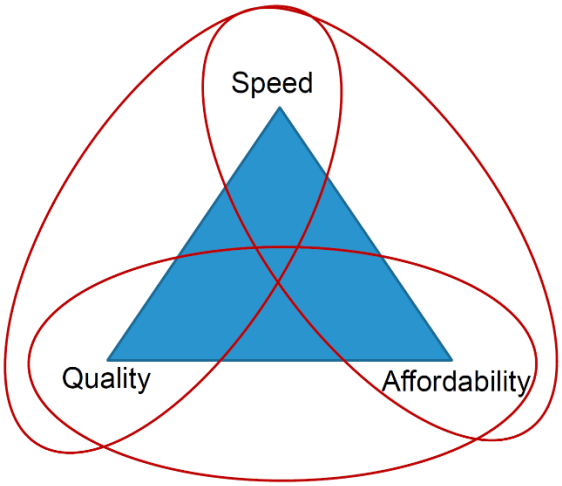

Figure 6. The Validation Triangle

A Bit of Philosophy

When considering a technical approach to validation, it is important to take into account your organization's internal philosophies. Is the preference to validate programs or to validate each individual output? Internal Standard Operating Procedures, Work Instructions, and Guidelines may provide guidance and direct the approach that you take. If the QC Plan focuses on programs, such as macros, with extensive and thorough testing and well defined parameters and definitions, then the output of that program can be assumed to be correct with very little further validation or the results. Care should be taken though, as it can be impossible to predict and test for every parameter and input. Uncleaned and incomplete data and lead to surprises with macros. An approach to instead design a QC plan to validate all results can enjoy less rigid programming standards, but must accept a lower level of efficiency. Your choice of which approach to choose may also be driven by whether the project is brand new and using early data, or established iterative reporting using updated data.

You should also consider whether you want to focus on validating statistical displays themselves or the underlying datasets used to create those elements. The validation of underlying datasets may allow for more automated testing while the validation of report elements will likely need to be done manually.

Some teams may prefer validation to be a very structured process with a very detailed list of step-by-step a tasks to be cleanly checked off as completed. We would caution that too many checkboxes may discourage validation programmers from thinking critically about the task. We would instead propose finding a good balance between structured guidance and free, critical thought. As discussed in this paper, both programmers and validators can program precisely to a specification and yet the outcome can still be wrong because the instructions were incorrect.

Regardless of your philosophy, the QC plan and validation process should be reassessed frequently and adjusted to fit current requirements.

Risk-based Validation

The concept of risk-based validation can be generally summed up as the idea of choosing a validation approach for both individual programs and for a project as a whole based on a perceived level of risk for that defined unit. You should first define the overall risk for the project as a whole and then use that to choose an acceptable level of risk for each programming unit. A higher risk task warrants a more rigorous technical approach than a lower risk task. The bulk of time and resources should be invested high risk tasks where the effort will be the most beneficial to providing a high quality deliverable. These perceived levels of risk can change over time as the project progresses and develops and should be examined along with the QC plan at regular intervals.

Once the perceived level of risk has been determined, the project team can decide which type of validation to apply. We will discuss a range of methods from one extreme to the other.

Double (plus) Programming

The most rigorous and thorough approach to programming validation is double programming in which two distinct programmers independently program an output from the same set of specifications and input data. They may use proc compare to electronically compare datasets and some combination of manual and/or programmatic review to compare output from tables, listings, and figures. This is an appropriate method to choose for results leading to high impact decision making. That level of rigor comes at a cost and it can be expensive and time consuming to have two different programmers replicate the same task. Further, even with this level of effort, it is not foolproof. Mistakes can and do still happen. In our experience these are often due to a variety of factors including errors in the specifications, too much reliance on a check list of test cases and not enough independent thought, failure on the part of the programmers to think critically about the specifications, or failure on the part of the programmers to thoroughly examine the data for anomalies and unexpected parameters or new data when producing iterative reports. Examples of failure in double programming highlight the fact that validation can never just be a fully defined process devoid of critical thought. All team members must stay vigilant to identify vulnerabilities in the system.

In our experience double programming continues to be industry standard, but perhaps that level of rigor is not necessary for every part of the deliverable. Perhaps, when considering a risk-based approach, that additional effort could be redirected into more efficient methods where appropriate and save overall programming time.

Code Review

Code review, in which a second programmer visually reviews the code of the lead programmer, but does not actually program, may be an appropriate approach for items that require a second look, but are not critical to decision making. One example would be listings of data. The validation programmer may check to make sure that the correct datasets and variables were chosen, that the expected number of records are present, that the formatting of the output is correct and that no text fields have been truncated. This method is more rigorous that self-QC and less time consuming that double programming, but should probably not be used for any high-impact programming and results.

Self-Validation

Self-validation is an acceptable method of validation in many instances where no high impact decisions will be made or where results will not be used for publication or some other public application. It is cheaper than double programming and code review and involves the lead programmer simply reviewing his or her own work for obvious errors. This is far less rigorous because it is highly likely that if the programmer made an error in interpretation the first time around, that the same error may be made in validation. At times when self-validation is employed, it is also likely that the lead programmer may be responsible for creating the specifications for the programming. Self-validation is not acceptable in all industries and should only be used when the level of risk is appropriate.

Automated Testing

Automated testing can provide the rigor of double programming with more efficiency, but only if time is invested upfront to develop the tools and processes. It can lower the potential of human error and be applied and executed faster than manual processes. The development of validated macros is one way to approach automated testing. Another real life example is Pinnacle 21, which has been developed to help programmers validate datasets for CDISC compliance. In order for these tools to be effective, they need to be developed with a stringent set of parameters and guidelines and they are only useful with the validation programmers understand the application and limitations.

Macros

Macros are just one tool that might be developed for an automated validation approach. They can effectively increase the efficiency of both programming and validation. The concept of developing a QC plan and validation process also applies to macros. With macros, the programming process is validated within a set of pre-defined criteria and in theory, a validated macro can safely be applied within its defined parameters without having to thoroughly validate the output. Further, sets of macros can be developed to capitalize on an automated validation process.

BRINGING IT ALL TOGETHER

We have shown that many factors need to be considered to assure accurate results for any project. Further, those factors often interact, cannot all be considered individually, and may evolve over the course of the project. There is much to be considered, and quality control can rapidly become overwhelming. Taking a structured approach to simplify QC at each step of the process can be used viewed as putting it all together through a QC Plan.

DEFINE YOUR QC PLAN

As has been emphasized, every project is different and a good plan needs to be designed specifically for that project. It also needs to be fluid and flexible and evolve as the project progresses. We can't design a plan for you, but we can propose some steps that you might take to get there.

Consider the generic process flow described by Figure 7 below.

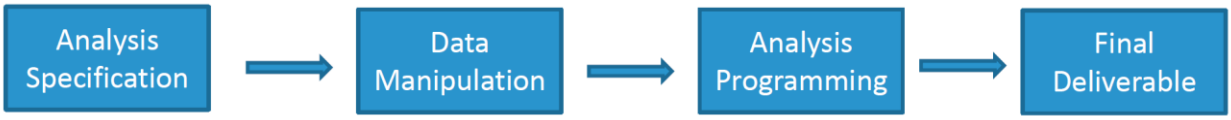

Figure 7. Generic Programming Process Flow

At each step of your process, evaluate **WHEN, WHO, WHERE, WHAT,** and **HOW**. Identify the SAS programs that are associated with the final deliverable at each step. Keep the validation triangle in mind.

Attention to detail in each individual SAS program as well as inherent dependencies become more important. Clearly define the risk of an error in each task as well as the benefit and cost of fully validating it. Assign a level of effort and QC approach based on this risk/benefit analysis. Reassess your plan frequently as the risks and benefits can change as the project progresses. A structured risk-benefit assessment may be helpful. Table 1 below is a sample risk/benefit assessment for the Analysis Specification step.

Key Tasks
• Review study documentation • Develop analysis plan • Create specifications and instructions for programmers

Events	Incorrect assumption made in choosing model in analysis plan
Risk	Results of analysis will not be real – could invalidate study
Impact	High
Time to QC	2 hours of independent statistical review
Cost to review	~$600 vs cost to fix later (substantial)
Value	High
Action	Review statistical plan
Method	Review by second statistician and double programming

Table 1. Example of Structured Risk/benefit Assessment for an Analysis Specification Step

At the analysis specification stage, SAS programs may be used to assess distributional assumptions to be used in the final analysis. They may be used by data management simply for data review, and by the statistician to become familiar with the existing data structures. In this example, it may be more important to focus on distributional assumptions for model selection. Note that this can be done in early stages of

the process rather than after data manipulation since the data manipulation stage is typically done to assist with streamlining analysis dataset programming. Having an independent statistician review the planned analysis (including distributional assumptions) may cost a few hours of his/her time. However, this may be worth the cost if additional ad hoc analyses need to be planned/programmed at a later time point. These additional analyses have both a cost implication as well as an implication on perception of results. If initial results are less impressive relative to those from ad hoc analysis, people may question the ad hoc results even if they are more appropriate statistically.

The process gets repeated for each program at each step. Table 2 shows that one need not wait until the final deliverable to have extremely consequential impact. In an experiment with a randomization component of the design, corresponding key information at the Data Manipulation step has a direct impact on final results.

Key Tasks
• Follow specs to recombine data and create analysis datasets • Provide feedback on data errors

Events	Randomization assignments could be incorrectly assigned to subjects
Risk	Results of analysis will not be real – could invalidate study
Impact	High
Time to review	1 hour of programmer/statistician review
Cost to review	~$200
Value	High
Action	Plan for additional review
Method	Double programming by second programmer

Table 2. Example of Structured Risk/benefit Assessment for a Data Manipulation Step

Here, observations (ie, patients in a clinical trial) are randomly assigned to groups in the beginning of the experiment (clinical trial). The randomization process is a fundamental part of the study design that leads to unbiased and interpretable results. It is critical that this key information be attached to the data correctly prior the analysis. As important as this step is, there have been times in the authors' experience where issues arose with this step causing delays in the final analysis. It is well worth the time and effort to verify the correctness of this step.

Once your risk/benefit analysis and plan is complete, you need to consider different approaches to validation that can be applied to each task. The SAS programming in Example 1 may not involve complete independent programming whereas Example 2 does. In all cases, some sort of independent review by someone with appropriate qualifications is necessary (see Marchand and Tinazzi, 2016).

It is clear that the risk/benefit analysis can become substantial for a complicated process flow. Fortunately, internal standards are often developed so that risk/benefit analyses are often already considered in their development. However, it is fair to say that a thoughtful risk/benefit analysis should occur more often. One approach to this is by encouraging the newer generations of statisticians and programmers to critically think more about the impact of their work on the end results. An alternative approach is periodic review (such as annual or bi-annual review). While little may chance with some parts of the process flow and risk/benefit assessment, the tools in which we use to streamline QC are likely to change.

Projects often evolve as work is done. The QC process also needs to be constantly reevaluated. When mistakes are made, the root cause of each needs to be examined to help identify why they happened and how to prevent the same mistakes in the future.

EFFORT VS QUALITY

One of the most important concepts to recognize in QC is that despite all efforts, you can't validate everything 100%. Further, it is our observation as shown in Figure 8 that at the later stages of the process, you begin to see diminishing returns on your efforts. In the beginning a relatively low amount effort can lead to a great increase in quality while later it make take a disproportional amount of effort for incremental gains. Given this, it is very important to define what really matters for your project. Use that to define an acceptable level of risk for each task and for the deliverable as a whole. Take some time to

define the ‘sweet spot’ for quality and its corresponding amount of effort. Keep in mind that resource availability may also influence your decision.

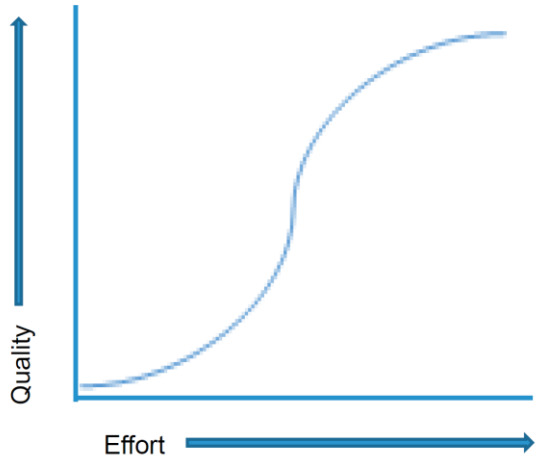

Figure 8. Diagram of the General Effect of QC Effort on Overall Quality

DOCUMENTATION OF QC

Though somewhat beyond the scope of this paper, one aspect of QC that should not be overlooked is how to document your process, specifications, steps that were followed, output proving your findings, traceability of activities, and final stamp of approval. This becomes especially important in industries that are highly regulated and may be subject to audit. Your approach may be governed by regulatory agencies, industry standards and guidelines, or internal SOPs. Many opinions exist. Some consider a detailed, structured checklist to be appropriate while others would rather place emphasis on the thought process of individual programmers.

CONCLUSION

We work in an age where increased output with fewer resources is expected, if not mandated. Thorough QC is easy to overlook in a high pressure environment with increased expectations of workload and expedited deliverables. The purpose of this paper is to revisit the concepts of QC and to evaluate things to be considered when you are expected to increase capacity without increasing staff.

The fundamentals of QC do not need to be complicated. Processes and programming may become complicated but the idea of preventing high impact mistakes is not. Keep in mind that you may not always be able to validate 100%. While there may be commonality among your projects, each deliverable has its own intricacies that set it apart from others. Develop a plan that accounts for this that considers the tradeoff between quality, speed, and affordability.

Other structured approaches exist. You may even have yours (which clearly is better suited for your needs). Breaking down your process and performing a risk benefit analysis is itself a form of process improvement. Furthermore, this can and should happen on a routine bases throughout the course of each project as it evolves in time. When done in a team environment it encourages team members to think more critically at each step since they are now aware of the thought process that goes into the plan.

QC should not be a lost art. Decisions are made every day based on statistical results from our SAS programs. Mistakes will happen, but we can work together so they happen less frequently and with lower impact.

REFERENCES

Marchand, C. and Tinazzi, A.. 2016 “What Auditors Want”, Proceeding of the PhUSE 2016 conference.

CONTACT INFORMATION

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

Amber Randall
Axio Research
amberr@axioresearch.com
http://www.axioresearch.com