PharmaSUG 2018 - Paper LD-14

To Error is Human: An Overview of Human Errors in SAS® Programming and How to Mitigate Them

Nagadip Rao, Eliassen Group

ABSTRACT

Until a few years back, Clinical SAS® programming, used to be done by a team of programmers who would write programs based on the specifications provided by study statistician or lead programmer**.** However, in recent years, this has changed, programming task now involves coordination between multiple teams of programmers belonging to different organizations, spread across the globe.

The complexity of programming tasks, along with work being done by multiple outsourced service providers, increases the risk of human errors especially, if the pharmaceutical client is a large organization. To mitigate human errors current International Conference on Harmonization guidelines for Good Clinical Practice (ICH GCP) has included an addendum to its Non-compliance (5.20) section, which instructs: "When significant noncompliance is discovered, the sponsor should perform root cause analysis and implement appropriate corrective and preventive actions." Many RCA (Root Cause Analysis) and associated CAPA (Corrective and Preventive Action Plans) show that "human error", is the underlying root cause for most quality events. This paper provides an overview of human error(s) that can potentially occur in clinical SAS® programming supported with real life examples that we encountered, and discusses ways to mitigate them. A good understanding of the nature and types of human error(s) is essential for reducing errors by taking proactive measures and if needed in performing meaningful root-cause analysis and developing of effective CAPA (corrective and preventative action plans).

INTRODUCTION

Not very long ago statistical programming was done by a group of programmers along with guidance from statistician working from same location. It was not uncommon to have same set of programmers and statistician work on all phases of clinical study for a given drug. In that relatively simple setup, there were only three major types of errors one encountered

1. **Syntax Error**: Most common type of error, occurs due to not following programming language rules while writing a program code. A simple example is a missing or inappropriately placed semi-colon which prevent compilation of the program
2. **Runtime Error**: Runtime error occurs when properly compiled program encounters a situation that it is not equipped to handle. A classic example would be trying to divide a number by variable that contains "zero"
3. **Logic Error**: An error that is caused when program and specification does not match. These are more complex in nature and harder to detect. An example would be creation of treatment emergent adverse event flag not considering change in severity after a drug is administered

In today's world where clinical trial projects are lot more complex and are managed by multiple partners. It is not uncommon for one service provider to do data management, second do CDISC® mapping, and third do clinical study report programming, and fourth provide data pooling services for ISS/ISE (Integrated Summary of Safety/Efficacy) or RMP (Risk Management Plans). Complexity in SAS programming also increases when mergers and acquisition of pharmaceutical company takes place or acquisition of compound from one company by another, result in changing data standards and programming procedures - further increasing the complex nature of clinical trial programming and Increase the risk of error in reporting.

The increasing risk of error in various aspects of clinical trial conduct, which includes data analysis: programming and reporting, has led International Conference on Harmonization guidelines for Good Clinical Practice (ICH GCP) to include an addendum to its Non-compliance (5.20) section requiring the

sponsor perform RCA and implement Corrective and Preventive Action Plan (CAPA) which would assist in mitigation of future errors.

After examining multiple quality related deviations, re-work requests and RCA's, we have recognized that the root cause for many errors in clinical trial programming and reporting were the result of a person(s) making a mistake, or in other words, a Human Error, rather than system or process error. Human Error is defined as **a mistake made by a person rather than a machine** according to Oxford English Dictionary**.** This paper will provide an overview of human error and error mitigation methods as applicable to clinical trial programming and reporting in SAS®.

AN OVERVIEW OF HUMAN ERRORS

Human errors are quality deviations which are result of a person making a mistake. They can be classified into five categories based on underlying root causes as shown in Figure 1. A description of each type of error is in the following.

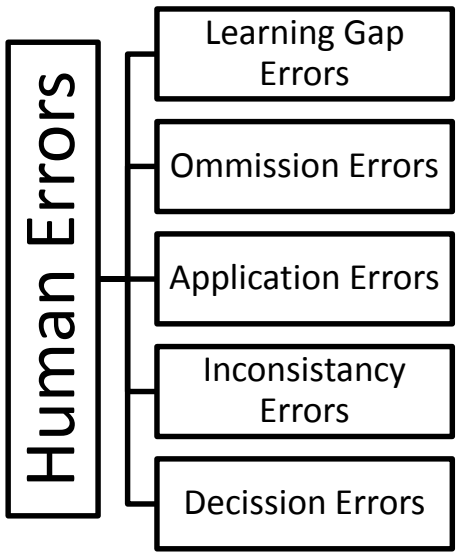


Figure 1: Human error classification in Clinical SAS® Programming

Learning Gap Errors

A programmer lacks functional knowledge or skills to perform an assigned task, this type of error is generally more prevalent in programmers who are less experienced. Errors caused by "Learning gap", might also occur in experienced programmers who are working on new projects or procedure about which they may lack in any previous experience. A good training program and mentorship may be key to minimize this type of error. In our experience these errors are relatively simple and are identified and resolved much earlier in the processes. For example, someone who is working under CDISC® standards for the first time and uses AESEV (Adverse event severity) for adverse table creation instead of ASEV (Analysis Adverse Event Severity) is a learning gap error.

Omission Errors

In this type of error, a programmer may have knowledge or required skills but misses a procedural step or a task. This error may also occur because of a programmer getting confused about requirements or in choosing correct procedural step. Omission errors are common among both experienced and novice programmers. Omission errors can be as simple as a programmer making updates and not checking in

code or may be more complex in nature, where it would be hard to trace source of error. Many other factors like competing priorities, resource inefficiencies, and work/non-work related psychological stressors can cause omission errors. An example of omission error that we actually encountered was related to response derivation in a clinical study related to infectious diseases. The response to therapy is derived using FDA's, MSDF (Missing, Switch, Discontinuation'=Failure (MSDF) Snapshot algorithm) algorithm, which assigns any subject who is missing primary endpoint infection data, switches to prohibited medications, discontinues from study or drug irrespective of the reason or dies as failure or non-responder. The pre-requisite to implementing MSDF algorithm is that: one must remove invalid values. In one such situation that we faced, a programmer omitted the step of removing invalid values before implementing MSDF algorithm which resulted in subjects being flagged as responders based on invalid values.

Application Errors

In this type of error, a programmer does have the required skill, knowledge and clear understanding of what needs to be done but makes a mistake at a point of application of knowledge. As we are aware, use of standardized general-purpose macros in increasing across the industry in order to improve quality and efficiency of programming processes. The application errors may occur because of a disconnection between programmers understanding of how a standardized macro works vs how it actually works, when he/she encounters a situation that is not very common. As we know, the Adverse Events (AE's) are collected for a pre-defined period after the end of study drug treatment to get a complete safety profile of a compound, which is also known as the lag period. Most of the drugs undergoing clinical trials will have AE data collection lag periods ranging from 30 days to 1 year and are defined in protocol. In one of the situation we faced, a few adverse events were collected after the lag period of 1 year and it was not included in final AE tables or listings. We found this issue post submission, when we performed RCA, it was found that that our standardized AE processing macro only included AE's for a period of one year after the last study drug dose date. The programmer had not made any updates to standard macro algorithm because the Statistical Analysis Plan (SAP) did not include explicit guidance to the programmers about including all AE data even outside lag period in AE tables and listings. Programmer assumed that Standardized macro includes all collected AE's.

Inconsistency Errors

Programmer believes he has applied the knowledge correctly but there is a lack of clarity in the standard or document. In one of the clinical study we worked, study drug could potentially increase risk of renal impairment. The renal impairment is assessed using creatinine clearance (CrCl) which is collected as a part of laboratory data. As per SAP (Statistical Analysis Plan), if there are multiple records in an analysis visit, one record was picked for summarizing labs. Programmers picked according to what they were summarizing even for creatinine clearance. However, all the records for CrCl needed to be included in assessment of renal impairment similar to how we handle potential clinical significance. This error was a result of incorrect assumption by programmers: that CrCl values needs to be handled in the same way as other laboratory values, due to lack of clarity in specification document (SAP), about the renal impairment derivation. Any inconsistency in the requirements, should be specified clearly to avoid inconsistency related errors.

Decision Errors

This type of error occurs when a programmer based on available information along with good intent makes decisions, but his/her decision leads to undesirable outcome. As we all know there is an increasing effort across pharmaceutical industry to develop medications for treatment of rare medical conditions. These drugs known as "Orphan Drugs" are relatively easy to get approval, but many of them need long term post marketing observational studies. A programming work on these long term observational studies involves downloading updated data at pre-defined intervals, re-run of tables and listings using pre-validated programs, self-validation and delivery of table and listings. In one such long term observational study on an orphan drug for treatment of rare genetic disease, a programmer who was working on it for couple of years realized, when he/she downloaded data and tried to re-run the tables using pre-existing programs a few

tables were failing because of change in data structure. Instead of escalating this matter to programming manager he/she updated previously validated program, thinking it would be a quick update, created tables and delivered to study team. After a few months, when study team was looking more closely into data reports they realized that some of the critical data on adverse events related to tumor and laboratory data was not represented appropriately in study reports. Upon further investigation, it was found that some of the source data structure had changed due to change in vendor. This change in data structure impacted programming assignment by changing it from simple re-run of validated programs to a more complex programming update which needed thorough independent validation. In situation like this, a programmer is expected to inform the programming manager about the change in data structure and they should have re-assessed programming assignment in this situation.

Latent Human Error

Any paper on human error would be incomplete without an overview of latent error. The latent errors are the hidden errors that are previously undiscovered or the flaws that can turn into active error under right circumstances alike to any of the previously mentioned error types. These errors are hard to recognize until they are activated. Latent human errors are generally a result of flaw in communication process, procedure or oversight in programming specifications. Identifying these latent human errors and developing a mitigation strategy needs to be an integral part of any CAPA. A quality event we encountered as a result of latent human error is described below.

Typical Case of Latent Human Error

The clinical laboratory testing is governed by GCLP (Good Clinical Laboratory Practice) which provides guidelines on quality laboratory testing. As per GCLP guidelines, normal ranges for laboratory tests could change over time for various reasons such as recalibration of instrument, chemical reagents used, analysis techniques, etc. As per GCLP, it is required to use laboratory normal ranges provided by the laboratory that collects and processes laboratory data. The source laboratory data for local laboratory and the related normal ranges can be in same or in different dataset depending on laboratory service provider. Laboratory normal ranges can change over a period for various reasons such as recalibration of instrument, chemical reagents used, analysis techniques, age of the subject etc. A given laboratory test value is flagged normal or abnormal depending upon which set of lab normal range we compare it against.

In one of the projects we worked on, standard macro used in pre-processing lab data for SDTM (Study Data Tabulation Model) mapping, laboratory data was captured in two separate datasets, one for local lab and another for central lab, a typical manner in which they are captured. Any change in laboratory normal reference range value was updated in central lab data but for local laboratory data it was sent in a separate third dataset. The way to incorporate updated local laboratory normal range data was to first merge local laboratory and updated local laboratory normal range data so that original laboratory normal range gets overwritten with updated normal range values.

During review of laboratory tables, the clinician found inconsistency between records flagged for laboratory abnormality in certain laboratory tests when they were compared with raw source data. When we were informed of this issue, we conducted RCA, during this investigation we found that local laboratory values were not being updated with changed local normal range data from secondary dataset. This particular laboratory pre-processing program was used multiple times before in other studies, without any issues but failed in this above-mentioned case. In other words, latent error in the laboratory pre-processing macro was undiscovered previously but got activated.

Latent human errors occur because of a system or a process containing error precursors that gets aligned with circumstances, in our case, the program was not able to process the laboratory normal limits updates. Latent errors can be difficult for the programmers to notice since the errors may be hidden in the algorithms of standardized programs they use regularly. We have also seen multiple times that programmers get used to latent errors and there is a tendency to work around it without actually fixing it at source. This act of working around an issue without fixing it, can many a times act as a potential error precursor.

Latent errors can be reduced by controlling error precursors, by standardization of programming process along with a more robust program algorithm design and its interface along with continuous improvement

over the period of time. It is unfortunate that most latent errors are only resolved after problems occur and by taking preventative actions to avoid it in future.

HUMAN ERROR PREVENTION AND MITIGATION

Human errors in Clinical SAS programming can be mitigated or even prevented through appropriate preventative action plans, involving a combination of training, standardization and putting proper well thought process in place. There are three different error mitigation and prevention strategies, classified based on effectiveness of each approach (Figure 2).

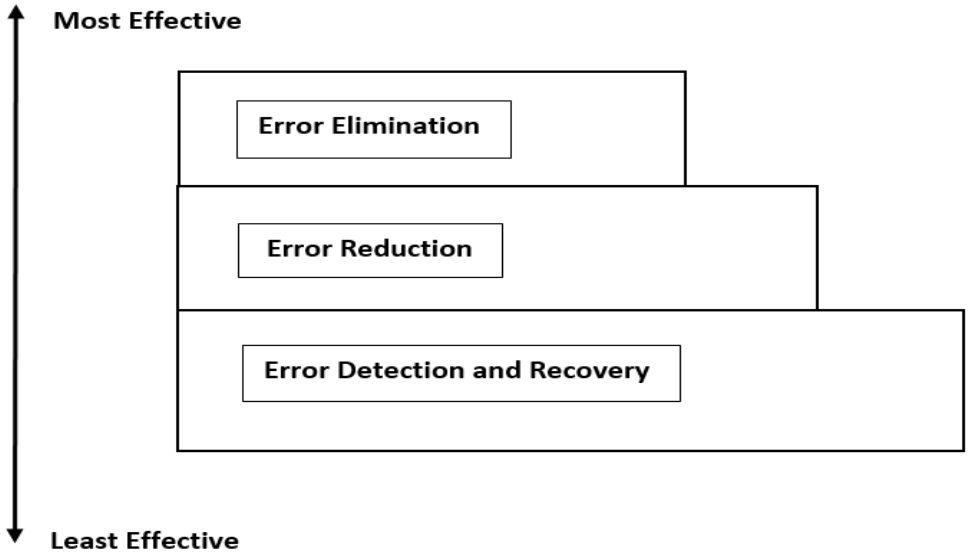


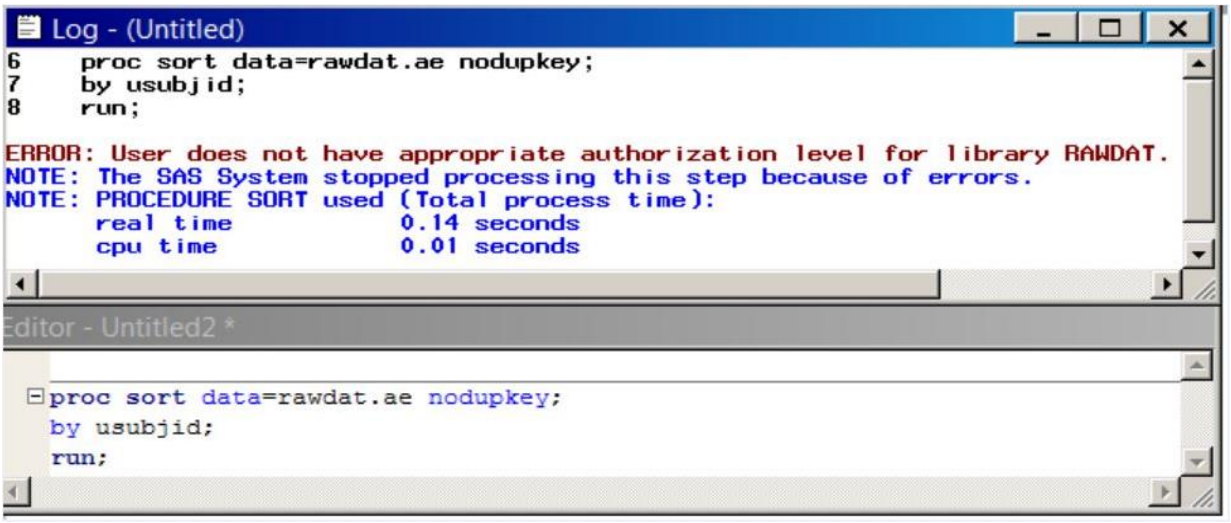
Figure 2: Error Prevention/Mitigation Methods and Relative Effectiveness

The basic principle of error prevention/mitigation strategy revolves around the idea that, “if we can eliminate an error we should eliminate it, if not reduce its occurrence, or else reduce consequence of an error”. Depending on complexity of the problem and effort needed, a suitable method needs to be chosen.

Error Elimination

Error elimination refers to eliminating possibility of making error by using automation or other system/processes solution. Error elimination is generally accomplished by reducing human involvement in a processes. A few good examples are using AUTOEXEC system option in pre-processing programs, to make sure programs use correct data and appropriate initial setup variables. Accidental modification of validated programs or permanent source datasets are prevented by using RCS (Revision Control System) and ACL (Accesses Control List).

The Display 1 shows how system prevents possibility of making an error when an attempt is made to overwrite adverse event source. A few other important error elimination methods that are built into programming are standardized macros that check for existence of required variables and their valid values before executing an analysis dataset or table algorithms.



Display 1: Error Elimination when an attempt is made to overwrite source AE data

It is to be noted that, from practical perspective, it is not possible to eliminate all errors though automation and other system level modifications.

Error Reduction

Error reduction refers to potentially reducing an error from occurring in the first place. This is proactive processes centric approach. Error reduction is achieved by increased standardization of processes and related documentations in clinical trial programming. A good example of process standardization is to have common templates for table shells, analysis dataset specification and list of tables with consistent table numbers across clinical studies in a company. A good way to reduce human error is to have programming related specifications finalized before start of actual programming. Uniformity in documentation is critical for achieving error reduction. A standard template to capture validation comments and resolutions of clinical data tabulations along with programmer assignment goes a long way in minimizing potential errors. Having a well-defined processes is more important when programming work transitions from one service provider to another or from service provider to client as after transition it is not easy to trace root of an error if it occurs. Standardizing programming processes is only useful if it is implemented at study level programming and formally reviewed at pre-specified milestones. To check how programming team is implementing existing process, pre-specified checkpoints are placed, also known as Quality Gates (QG). An overview of QG is given in the following section. Error reduction is an ongoing processes and lesson learnt from previous quality issues needs to be transitioned into programming processes in order to minimize potential future errors.

Error Detection and Recovery

Error detection and recovery involves detection of error and taking action before consequence of that error impacts. Error detection involves adding or strengthening error signals by incorporating algorithmic traps into programming e.g. Sample Code 1 shows how a program outputs message if supplemental domain is not found for existing SDTM (Study Data Tabulation Model) domain data. The purpose here is to make programmer aware that there is no supplemental data, allowing them to take further action if needed. Error signal is also strengthened by making layout of clinical study table shell more intuitive for reviewer e.g. AE tables that display number of AE preferred terms (Table Shell 1) let a programmer know if most severe AE preferred terms was selected for a given subject and also as if missing AE severity was correctly imputed.

```
proc sort data=alldat nodupkey;
             by memname;
             where index(compress(lowcase(memname)), 'supp') & (upcase(compress(memname)) in (&domn.));
             run;
   %end;

   *Added step to provide the summary report;
   data allsupp;set alldat;run;

   *Get macro variable list of memname;
   proc sql noprint;
         *Get List of memname;
         select distinct(compress(memname)) into :memname separated by " " from alldat;

         *Get count of memname;
         select count(distinct(memname)) into :n from alldat;
         quit;
      %let n=&n;%put N=&n.; /*Remove trailing blanks*/
   %if &n = 0 %then
   %do;
         %put ------------------------------------------------------------------------------- ;
         %put                  SUPPLEMENTAL QUALIFIER DOMAIN NOT FOUND                        ;
         %PUT CHECK SDTM DATASET LIBRARY AND ENTER CORRECT SUPPLEMENTAL QUALIFIER THAT EXIST  ;
         %PUT           E.G. for SUPPDM expecting DM or SUPPLB expecting LB etc              ;
         %put ------------------------------------------------------------------------------- ;
```

Sample Code 1: Program outputs message if supplemental data is missing

Most common way of accomplishing error detection and recovery in programming is via duplication of effort or in other words validation by double programming. Recovery processes involves rectification of detected errors. It is critical that errors and rectification method to be documented along with date and initial of programmer responsible for it.

Table 1x.x.x
<Drug name> Protocol xyz123

Incidence and Severity of Treatment-Emergent Adverse Events by System Organ Class and Preferred Term – Safety Analysis Set

Number of Subjects Evaluable for AEs	Treatment A (N=xx)			Treatment B (N=xx)		
Severity(a)	Mild	Moderate	Severe	Mild	Moderate	Severe
Number (%) of Subjects: by SYSTEM ORGAN CLASS and Preferred Term	n (%)	n (%)	n (%)	n (%)	n (%)	n (%)
With Any Adverse Event	Xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)
SOC term 1	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)
AE Preferred term1	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)
AE Preferred term2	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)	xx (xx.x)
Total AE preferred terms	***xx***	***xx***	***xx***	***xx***	***xx***	***xx***

If the same subject in a given treatment had more than one occurrence in the same preferred term event category, only the most severe occurrence is counted.

Table Shell 1: Intuitive table shell layout lets programmer know if most severe AE's were used incidence count

Quality Gates

Quality gates (QG) refers to checkpoint in a clinical study process including programming at pre-defined milestones to assess if all existing processes as defined in SOP (Standard Operation Procedure) are followed until that point. A QG involves formal review of pre-selected relevant documentations for sufficiency to gauge satisfactory execution of previous project milestone before team starts work on next milestone (Figure 3). In a clinical programming processes QG's are placed at three pre-defined project milestones. First programming QG (Gate 1) review occurs before start of programming, to make sure proper programming related initial set of documents (list of tables, table shells and SDTM logical data maps, analysis dataset specifications etc.) are created appropriately. Second QG (Gate 2) document review occurs about 10-15 days post LSLV (Last Subject Last Visit) assessing completion of programming code,

validation and blinded data test run reviews. QG reviewers expect updated QG1 documents along with test run comments and resolution logs. Final programming QG (Gate 3) review occurs after completion of final study reports post database lock. At QG3 programming documents are reviewed for satisfactory execution of programming processes before commencement of medical writing

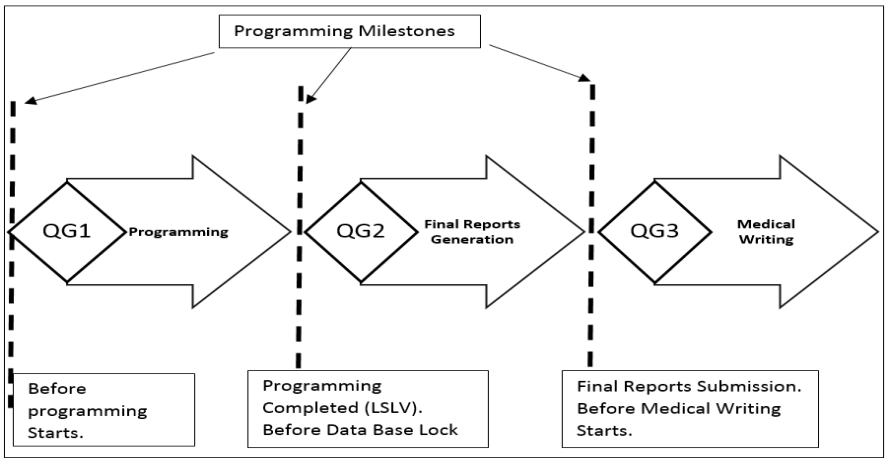


Figure 3: Clinical Programming Quality Gate Process

Quality gate ensures efficient and orderly execution of programming tasks by reducing risk of programmers getting ahead of themselves and working out of sequence (e.g. completion of analysis datasets before specifications are complete) potentially increasing risk of human errors and hence impacting programming efficiency. .

CONCLUSION

Clinical SAS® Programming processes has becoming a complex undertaking involving multiple partner organizations across the globe in today's world. Increasing complexity of programming task and also involvement of multiple organization increase risk of human error. RCA performed on programming related quality issues show that human error as the leading cause. This paper provides an overview of human errors and a framework of error mitigation strategies to help programming teams improve quality and efficiency of programming processes.

REFERENCES

1. E6(R2) Good Clinical Practice: Integrated Addendum to ICH E6(R1) https://www.fda.gov/downloads/Drugs/Guidances/UCM464506.pdf
2. What Errors Do We Miss in Clinical Trials? http://www.pharmexec.com/what-errors-do-we-miss-clinical-trials?pageID=1
3. Galin,D . 2003. Software Quality Assurance: From Theory to Implementation. London, UK : Pearson
4. Human Error : https://www.nopsema.gov.au/resources/human-factors/human-error/
5. A Guide to the Project Management Body of Knowledge (PMBOK®).
6. S. Schmitt, "Root Cause Analysis: Finding the Root of the Problem," Pharmaceutical Technology 40 (9) 2016

ACKNOWLEDGMENTS

Author would like to acknowledge Eliassen Group- Biometrics and Data Solutions for providing the opportunity to work on this paper.

CONTACT INFORMATION

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

Nagadip Rao
Eliassen Group: Biometrics and Data Solutions
400 Atrium Dr Somerset, NJ 08873
Email:Nrao@eliassen.com