Reliably Robust: Best Practices for Automating Quality Assurance and Quality Control Methods

ABSTRACT

An often objective of SAS development is the generation of autonomous, automated processes that can be scheduled for recurring execution or confidently run with push-button simplicity. While the adoption of SAS software development best practices most confidently predicts programmatic success, robust applications nevertheless require a quality management strategy that incorporates both quality assurance (QA) and quality control (QC) methods. To the extent possible, these methods both ensure and demonstrate process success and product validation while minimizing the occurrence and impact of environmental and other exceptions that can cause process failure. QA methods include event handling that drives program control under normal functioning, exception handling (e.g., error trapping) that identifies process failure and may initiate a remedy or graceful termination, and post hoc analysis of program logs and performance metrics. QC methods conversely identify deficits in product (e.g. data set) availability, validity, completeness, and accuracy, and are implemented on input and output data sets as well as reports and other output. QC methods can include data constraints, data structure validation, statistical testing for outliers and aberrant data, and comparison of transactional data sets against established norms and historical data stores. The culmination of any quality management strategy prescribes the timely communication of failures (or successes) to stakeholders through alerts, report generation, or a real-time dashboard. This text describes the advantages and best practices of incorporating a comprehensive quality management strategy into SAS development, as well as the more challenging transformation of error-prone legacy code into a robust, reliable application.

INTRODUCTION

Robustness reflects the ability of software to overcome environmental and other obstacles and generally is regarded as a tenet of “quality” software development[i]. Thus, software should be tenacious in its intent and ability to execute successfully by anticipating, identifying, and navigating around challenges through prescribed mechanisms and, when all remedies have failed, by terminating gracefully and immediately communicating failure to users and dependent processes. Risks to program and process success should be identified during project planning and as part of continuous quality improvement (CQI) throughout development. During execution, these risks are mitigated most efficiently through intelligent, automated processes that can drive program control, discriminating between successes and failures.

If you can predict it, you can prevent it…one of the hard fast rules in firefighting, loss prevention, and safety

To be clear, not every bit of code needs this attention to detail for quality management. There is a balance, and you do not want to gold-plate a solution. At the very least, even if you never intend to re-use the code, you should enumerate quickly the risks to process success and data validity. For example “code will break if file is not named test.txt or if used after the end of the calendar year. Just as macros are not applicable for every environment, some very short projects do not necessitate a sophisticated quality strategy. But be considerate to others, and no matter what project, code with a modular intent that someone else will be utilizing and modifying your code In your absence.

Also, to be clear, robustness is just but one tenet of good software. Extensibility, modularity, documentation. blah blah blah…and that dynamic code is also more robust, for example, utilizing macro variables and reading configuration files so the code does not have to be stopped for editing. These are those best practices that are alluded to above.

CONTRASTING SAS WITH OBJECT-ORIENTED LANGUAGES

SAS practitioners inherently represent a diverse, multi-disciplinary array of professions. They do include classical software engineers and developers, but also data scientists, statisticians, methodologists, psychologists, biologists, business analysts, clinical research associates, epidemiologists, econometricians, intelligence analysts, and other professionals. In these latter roles, because SAS practitioners often simultaneously are cast as subject matter experts (SMEs) and developers, they may be responsible for both code development and data analysis of the resultant products. While these environments espouse flexibility and efficiency through a skilled class of “developer-analysts” who can modify code on the fly to overcome deficits or add functionality, they also can facilitate poor coding through

insufficient documentation, lack of risk identification and remediation, and the absence of an automated quality management processes.

Object-oriented programming (OOP) languages such as Java and Python incorporate intrinsic exception handling through “try-except” structuring that is designed purposefully to handle failures efficiently. Base SAS, a procedural 4th generation language, conversely lacks native exception handling, relying instead on macro statements and conditional logic to alter program flow. More than presenting a technical limitation, this deficit engenders a less responsible coding philosophy because SAS processes often are assumed to succeed, without an explicit test for failure. Complementing this philosophy is the fact that SAS development often occurs in support of business analytics for which the developer simultaneously is the user of the code and its data products. while applications development—contrasted as development of stand-alone, executable programs for an extra-developer class of users—typically employs OOP

Comparing %GOTO statements to C++, Java, and Python try-except paradigms

Mention that exception handling and event handling are utilized throughout both QA and QC to provide program control.

Example contrasting OOP approach to SAS procedural approach; mention other benefits of having inheritance and propagation of errors with OOP

Also contributing to the lack of quality in SAS appropriate error handling strategies is the fact that they are often saliently missing from SAS exemplar programs, even those produced by established SAS masters. This is not to assert that any of these SAS gurus are unaware of these processes, only to demonstrate that because of the lack of a straightforward and concise try-except model (that CAN be added to OOP exemplars without adding confusion), SAS cannot add its more garrulous exception handling without completely detracting from the readability of code. Thus, when SAS best practices and examples are presented in books and white papers, QA/QC methods may be lacking because of a belief that they might detract from understanding the substantive pith of the technical point that is being made.

The difference between SAS SCL language, and the rationale why some of the advances and techniques for OOP are applied there, but don’t seem to have been applied—at least through literature—in Base SAS.

These unfortunate practices occur when code never really is completed because developer is not required to relinquish control when he is also the user. In these environments, code is asserted to have run correctly after a manual, visual inspection of the log. This is valid quality assurance (QA) technique, but it is a manual process that is not preferred, as a developer should not be expected to page through thousands of lines of log. A common quality control (QC) technique employed in SAS environments is a visual inspection of the data set or other product (e.g., HTML report) that is produced as a “sanity check” to determine if the results prima facie appear to be correct. Again, this implies a tremendous amount of attention to detail, can be painfully slow, and is a solution not scalable to big data and other complex environments.

This is not to say that every project requires a comprehensive, automated approach to quality. So-called “one-off” activities that are intended to answer a straightforward business question or otherwise create a process that will never be operationalized often do not need this added overhead. But, all too often, over time these “one-off” projects transform over time into convoluted, poorly planned extract transform load (ETL) architectures that are relied upon by numerous dependent processes and products. Another possibility is that, if code is formed well and is extensible, it is more likely to be incorporated in other processes. In these cases, it is much easier to incorporate QA/QC methods into the original code before it is extended to further applications and uses.

DEFINITIONS AND DISAMBIGUATION

The beginning of knowledge is the definition of terms. And, suffice it to say, the QA/QC arena has a number of terms that are used in ambiguous, even contradictory fashions. Disambiguation of quality assurance in software development from QA in software execution

Disambiguation of event handling and exception handling—what should be handled and what should not? Various perspectives of what each is called, when to use them, and when to let the SAS compiler simply handle the errors as prescribed (possibly in a division-by-zero scenario.)

Observable errors and unobservable errors (logic errors) that do not produce compilation or run-time warnings or errors

Disambiguation between bugs, errors, faults, defects, deficits, and failures

If quality control is the inspection of your final product, then some software methodologists will interpret that since the final product is quality, working software, then quality control would be the inspection of that software. I on the other hand view code as the process, because it is primarily being used internally, rather than shipped off elsewhere.

QA/QC FROM SCRATCH

How to implement a quality management risk management structure when one has never been in place. First make a list of the issues and all the technical debt. Also make a flowchart of the data. But really, this all starts with what are you producing, and work backwards from there. What adds the most business value. And what are the products that the process is creating?

In Agile environments, ensure that the “definition of done” also includes references to QA and QC and other testing.

risk management – the risk is that shit will break and fail, and riskier yet that you will not realize this

You must first draw a process diagram (flowchart) of even the simplest process, which includes all inputs, outputs, and dependencies, and thus define all the ways in which accesses could be broken, including a server failing or other anomalies (exceptions).

QUALITY CONTROL

PMBOK states that “Prevention over inspection”, thus an initial QC check can prevent errors that will occur throughout, marring up the Q/A system.

Processes often rely on external, third-party transactional data sets that, whether ingested as SAS data sets or in some other format, may not have been scrubbed properly. I always spend 80 percent of my time cleaning data, to the delight of analysts who work downwind of my prodigious efforts, but never seem to be the recipient of the same observance of data quality.

Sometimes it’s very easy to add a quality control measure to check if an errors has occurred, but it’s very difficult to actually prevent the error. In this case, go for the low-hanging fruit, and implement the Q/C method.

QC - system OPTIONS for ensuring data constraints (possibly through an automated data dictionary?)

All QA methods will seek to determine if some event or exception has occurred. At a minimum, this information could be sent to the log for parsing during the QC phase. More appropriately, however, the true benefit is being able to both identify and resolve or handle these events in real-time.

quote about 80% of analysis being data cleaning and staging. How much I would love to be the analyst downwind of my work, because it would be a dream…that unfortunately has never occurred.

QUALITY ASSURANCE

Forgiveness vs permission…what happens if you only have the former, only have the latter, or have two much or not enough of one or both?

QA - system OPTIONS for whether to terminate SAS when the first error is encountered.

QA – system OPTIONS for the level of process logging to utilize

Use of event handling to demonstrate validation of parameters submitted when SAS code is run from command line

The determination must be made at what level(s) or granularity logging and report generation will occur, based on who should be reading the errors, and also what should occur and the type of error. Is it recoverable? Or should the entire process be terminated?

Nested exception handling, and how to gain modularity in macros akin to that desired by OOP.

What is a quality assurance process checklist?

Demonstrate QA remediation processes (either wait and try again, delete an element, modify an element, alert of the failure, terminate part of the program, or terminate the entire program.) For a data element, this could involve deleting the value, the observation, the data set, or performing some type of modification there.

Consider a configuration file that is also external to the code and which prescribes the level of error logging that occurs, so that one can switch from a user- to a developer-oriented view only through modification of the CFG file, not through modification of the code.

QA/QC TOGETHER AT LAST – AND AN APPEAL FOR BOTH

A diagram that seeks to describe the various dichotomies and disambiguations present in errors.

Demonstrate how much time it takes (on a large file, or when crunched repetitively) to have data constraints

For those who consider a quality management strategy that does NOT include a quality control validation of the product of your process, please consider that without this validation, you may be requiring additional quality assurance (data constraints) work by analysts and developers who rely on your dependencies.

COMMUNICATION OF FAILURES AND SUCCESSES

What does Project Management Framework say about the monitoring and controlling phase of a project.

performance measurement

Message dissemination that can occur through outlook automated messages, or reports that are posted to a portal. A dashboard can also be utilized to demonstrate both QA and QC methods that show process and product success and failures.

POST HOC QUALITY MANAGEMENT IMPLEMENTATION – TECHNICAL DEBT

What are the three factors that are important in deciding what defects to fix first? 1) what happens when the error occurs, 2) the frequency of occurrence, and 3) the likelihood that the error will be detected when it occurs. How can these three principles help drive QA and QC policies??? The PARETO principle in examining what causes the majority of process failures, and go from there.

Walk the reader through both a Data step example and a macro example of the three types of QA and two types of QC procedures that can be utilized.

Consider that for manual modifications made through event handling, that a drop zone lookup table can be utilized so that it does not all need to be coded explicitly in the code. This deviation, by allowing the dynamism to occur in a separate file or table as opposed to in the code also greatly improves the robustness of the code. Another approach to this, is to have a separate table prescriptively driving operations (rather than descriptively), such as the dynamic Catalog tables that can be used, which determine how to process warnings and errors that are encountered.

The black box concept

Creation of a risk register that prioritizes 1) identified risk, 2) response (remediation) possibility, 3) severity, 4) what will happen if uncorrected (most likely and most deadly), 4) difficulty or measure of time to complete resolution, 5) dependencies or what must be fixed before this deficit can, 6) likeliness to occur, and 7) likelihood of going undiscovered.

[i] robustness a facet of quality software